JavaScript 函数

目录 Contents

1. 函数的概念

在 JS 里面，可能会定义非常多的相同代码或者功能相似的代码，这些代码可能需要大量重复使用。

虽然 for循环语句也能实现一些简单的重复操作，但是比较具有局限性，此时我们就可以使用 JS 中的函数。

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。

目录 Contents

2. 函数的使用

函数在使用时分为两步：声明函数和调用函数。

2.1 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. 函数的使用

函数在使用时分为两步：声明函数和调用函数。

2.2 调用函数

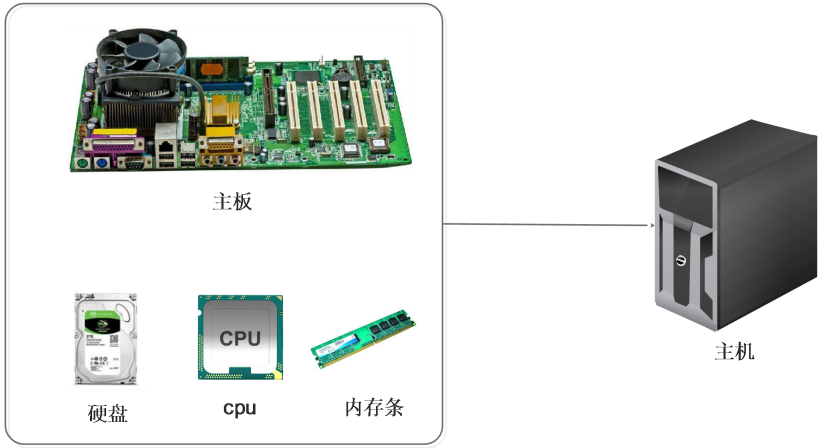
```
// 调用函数
函数名();  // 通过调用函数名来执行函数体代码
```

- 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。

2.3 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
- 简单理解：封装类似于将电脑配件整合组装到机箱中（类似快递打包）



2. 函数使用

案例：利用函数计算1-100之间的累加和

```
/*
    计算1-100之间值的函数
*/
// 声明函数
function getSum(){
  var sumNum = 0;// 准备一个变量，保存数字和
  for (var i = 1; i <= 100; i++) {
    sumNum += i;// 把每个数值 都累加 到变量中
  }
  alert(sumNum);
}
// 调用函数
getSum();
```

2. 函数使用

2.4 pink老师提问

1. 函数是做什么的（作用）？
2. 声明函数用什么关键词?
3. 如何调用函数?
4. 封装是什么意思?

目录 Contents

3. 函数的参数

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。

3. 函数的参数

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**。

```
// 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...);
```

3. 函数的参数

案例：利用函数求任意两个数的和

```
function getSum(num1, num2) {
    console.log(num1 + num2);
}
getSum(1, 3); // 4
getSum(6, 5); // 11
```

3. 函数的参数

3.2 函数参数的传递过程

```
// 声明函数
function getSum(num1, num2) {
    console.log(num1 + num2);
}
// 调用函数
getSum(1, 3); // 4
getSum(6, 5); // 11
```

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

3. 函数参数

3.3 函数形参和实参个数不匹配问题

参数个数	说明
实参个等于形参个数	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);              // 形参和实参个数相等，输出正确结果
sum(100, 400, 500, 700);    // 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。

3. 函数的参数

3.4 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

目录 Contents

4. 函数的返回值

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

4. 函数的返回值

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。

例如，声明了一个sum()函数，该函数的返回值为666，其代码如下：

```
// 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
```

4. 函数的返回值

案例 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));
```

4. 函数的返回值

案例 2：利用函数求任意一个数组中的最大值

求数组 [5,2,99,101,67,77] 中的最大数值。

4. 函数的返回值

案例 2：利用函数求任意一个数组中的最大值

```
//定义一个获取数组中最大数的函数
function getMaxFromArr(numArray){
    var maxNum = 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);
```

4. 函数的返回值

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
```

4. 函数的返回值

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
```

4. 函数的返回值

案例：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);
```

4. 函数的返回值

4.4 函数没有 return 返回 undefined

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

4. 函数的返回值

4.5 break ,continue ,return 的区别

- break： 结束当前的循环体（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

4. 通过榨汁机看透函数

榨汁机　他们俩的功能都是实现某种功能　**函数**

输入原料

内部处理

输出果汁


```
function fn(参数1，参数2..){
    函数体;
    return 返回值;
}
```

输入参数

内部处理

返回结果

作业

① 写一个函数，用户输入任意两个数字的任意算术运算（简单的计算器小功能），并能弹出运算后的结果。

② 写一个函数，用户输入任意两个数字的最大值，并能出弹运算后的结果。

③ 写一个函数，用户输入任意三个不同数字的最大值，并能弹出运算后的结果。

④ 写一个函数，用户输入一个数判断是否是素数，并返弹出回值(又叫质数，只能被1和自身整数的数)

目录 Contents

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：

- 具有 length 属性
- 按索引方式储存数据
- 不具有数组的 push , pop 等方法

5. arguments的使用

案例：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));
 console.log(maxValue(12, 4, 9));
```

目录 Contents

6. 函数案例

案例 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);
```

6. 函数案例

案例 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;
}
```

6. 函数案例

案例 3： 判断闰年

要求：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));
```

6. 函数案例

函数可以调用另外一个函数

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。

6. 函数案例

函数可以调用另外一个函数

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。

```
function fn1() {
  console.log(111);
  fn2();
  console.log('fn1');
}
function fn2() {
  console.log(222);
  console.log('fn2');
}
fn1();
```

6. 函数案例

案例 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

目录 Contents

7. 函数的两种声明方式

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();
```

- 因为有名字，所以也被称为命名函数
- 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

7. 函数的两种声明方式

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，所以也被称为匿名函数
- 这个fn 里面存储的是一个函数
- 函数表达式方式原理跟声明变量方式是一致的
- 函数调用的代码必须写到函数体后面

传智播客旗下高端IT教育品牌