

JavaScript 基础第四天

函数







❷学习目标

Learning Objectives

- 1. 掌握函数的基本使用,让代码具备复用能力
- 2. 理解封装的意义,能够具备封装函数的能力





- ◆ 函数
- ◆ 综合案例





函数

- 为什么需要函数
- 函数使用
- 函数传参
- 函数返回值
- 作用域
- 匿名函数



```
cyle>
    .left {
       width: 30px;
       height: 30px;
       text-align: center;
       line-height: 30px;
       background-color: pink;
    .right {
       width: 30px;
       height: 30px;
       text-align: center;
       line-height: 30px;
       background-color: ■purple;
</style>
```

抽取-封装

```
.btn {
   width: 30px;
   height: 30px;
   text-align: center;
   line-height: 30px;
.left {
   background-color:  pink;
.right {
   background-color: ■ purple;
```





- 1. 99乘法表,页面需要打印多个怎么办?
- 2. 体验函数的魅力
 - ▶ 代码复用



1.1 为什么需要函数

目标: 能说出为什么需要函数

● 函数:

function,是被设计为执行特定任务的代码块

● 说明:

函数可以把具有相同或相似逻辑的代码"包裹"起来,通过函数调用执行这些被"包裹"的代码逻辑,这么做的优势是有利于精简代码方便复用。

比如我们前面使用的 alert()、 prompt()和 console.log() 都是一些 js 函数,只不过已经封装好了,我们直接使用的



□生成外链播放器

難 我的滑板鞋

歌手: 约瑟翰庞麦郎

所属专辑: Panmalon Jon作品集













(55749)

作曲:约瑟翰庞麦郎

作词:约瑟翰庞麦郎

有些事我都已忘记

但我现在还记得

一段播放《我的滑板鞋》的代码

在一个晚上我的母亲问我

今天怎么不开心

我说在我的想象中有一邓清板鞋

与众不同量中间或舞肯定棒

整了城市找遍所有的街都没有

她说将来会找到的时间会给我答案

星期天我再次寻找依然没有发现

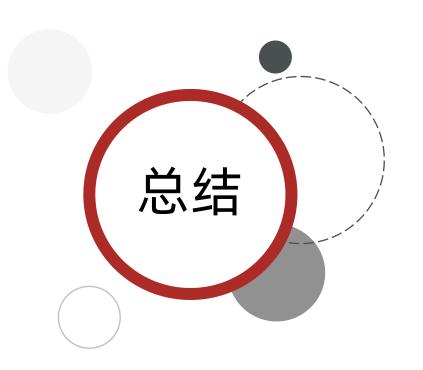
一个月后我去了第二个城市











- 1. 为什么需要函数?
 - ▶ 可以实现代码复用,提高开发效率
- 2. 函数是什么?
 - ➤ function执行特定任务的代码块





函数

- 为什么需要函数
- 函数使用
- 函数传参
- 函数返回值
- 作用域
- 匿名函数



目标:掌握函数语法,把代码封装起来

• 函数的声明语法

```
function 函数名() {
  函数体
}
```

● 例

```
function sayHi() {
   document.write('hai~~')
}
```



- 函数名命名规范
 - ▶ 和变量命名基本一致
 - ▶ 尽量小驼峰式命名法
 - ▶ 前缀应该为动词
 - ▶ 命名建议:常用动词约定

function getName() {}
function addSquares() {}

动词	含义	
can	判断是否可执行某个动作	
has	判断是否含义某个值	
is	判断是否为某个值	
get	获取某个值	
set	设置某个值	
load	加载某些数据	



• 函数的调用语法

// 函数调用,这些函数体内的代码逻辑会被执行 函数名()

注意: 声明(定义)的函数必须调用才会真正被执行,使用()调用函数

• 例

// 函数一次声明可以多次调用,每一次函数调用函数体里面的代码会重新执行一次 sayHi() sayHi()

● 我们曾经使用的 alert() , parseInt() 这种名字后面跟小括号的本质都是函数的调用



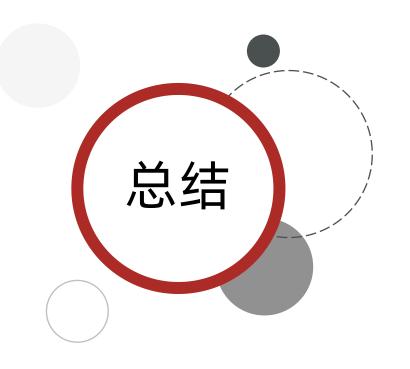
● 函数体

函数体是函数的构成部分,它负责将相同或相似代码"包裹"起来,直到函数调用时函数体内的代码才会被执行。函数的功能代码都要写在函数体当中。

```
//打招呼
function sayHi() {
  console.log('嗨~');
}
sayHi()

zayHi()
```





- 1. 函数是用那个关键字声明的?
 - > function
- 2. 函数不调用会执行吗? 如何调用函数?
 - ▶ 函数不调用自己不执行
 - ▶ 调用方式:函数名()
- 3. 函数的复用代码和循环重复代码有什么不同?
 - ▶ 循环代码写完即执行,不能很方便控制执行位置
 - ▶ 随时调用,随时执行,可重复调用





• 函数课堂练习

需求:

- 1. 写一个打招呼的函数 hi~
- 2. 把99乘法表封装到函数里面,重复调用3次





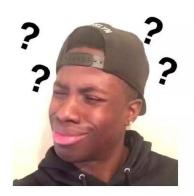
函数案例

需求:

- 1. 封装一个函数, 计算两个数的和
- 2. 封装一个函数, 计算1-100之间所有数的和

灵魂拷问:

这些函数有什么缺陷?







函数

- 为什么需要函数
- 函数使用
- 函数传参
- 函数返回值
- 作用域
- 匿名函数



● 思考:这样的函数只能求 10 + 20, 这个函数功能局限非常大

```
function getSum() {
    let num1 = 10
    let num2 = 20
    console.log(num1 + num2)
}
getSum()
```

- 解决办法: 把要计算的数字传到函数内
- 结论:
 - ▶ 若函数完成功能需要调用者传入数据,那么就需要用有参数的函数
 - ▶ 这样可以极大提高函数的灵活性

用户决定放什么原料





● 声明语法

```
function 函数名(参数列表) {
 函数体
}
```

- 例
 - ◆ 单个参数

```
function getSquare(num1) {
    document.write(num1 * num1)
}
```

◆ 参数列表

- ▶ 传入数据列表
- ▶ 声明这个函数需要传入几个数据
- 多个数据用逗号隔开

◆ 多个参数

```
function getSum(num1, num2) {
   document.write(num1 + num2)
}
```



• 调用语法

函数名(传递的参数列表)

● 例

getSquare(8)

getSum(10, 20)

● 调用函数时,需要传入几个数据就写几个,用逗号隔开

```
function getSum(num1, num2) {
    document.wr/lte(num1 + num2)
}
getSum(10, 20)
```

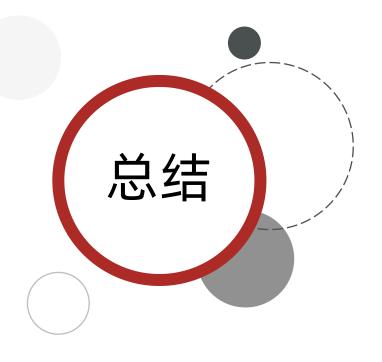
```
// 类似执行了如下代码
num1 = 10
num2 = 20
```





- ▶ 形参: 声明函数时写在函数名右边小括号里的叫形参(形式上的参数)
- ▶ 实参:调用函数时写在函数名右边小括号里的叫实参(实际上的参数)
- ▶ 形参可以理解为是在这个函数内声明的变量(比如 num1 = 10)实参可以理解为是给这个变量赋值
- ▶ 开发中尽量保持形参和实参个数一致
- ▶ 我们曾经使用过的 alert('打印'), parseInt('11'), Number('11') 本质上都是函数调用的传参





- 1. 函数传递参数的好处是?
 - ▶ 可以极大的提高了函数的灵活性
- 2. 函数参数可以分为那两类? 怎么判断他们是那种参数?
 - ▶ 函数可以分为形参和实参
 - ▶ 函数声明时,小括号里面的是形参,形式上的参数
 - ▶ 函数调用时,小括号里面的是实参,实际的参数
 - ▶ 尽量保持形参和实参的个数一致
- 3. 参数中间用什么符号隔开?
 - ▶ 逗号







• 函数封装求和

需求: 采取函数封装的形式: 输入2个数, 计算两者的和, 打印到页面中



1.3 函数传参-参数默认值

形参: 可以看做变量,但是如果一个变量不给值,默认是什么?

> undefined

但是如果做用户不输入实参,刚才的案例,则出现 undefined + undefined 结果是什么?

> NaN

我们可以改进下,用户不输入实参,可以给 形参默认值,可以默认为 0,这样程序更严谨,可以如下操作:

```
function getSum(x = 0, y = 0) {
    document.write(x + y)
}
getSum() // 结果是0 , 而不是 N
getSum(1, 2) // 结果是 3

Reference(I, 2) // 结果是 3
```

说明:这个默认值只会在缺少实参参数传递时 才会被执行,所以有参数会优先执行传递过来的实参,否则默认为undefined





函数封装-求学生总分

需求: 学生的分数是一个数组,计算每个学生的总分

分析:

①: 封装一个求和函数

②: 传递过去的参数是一个数组

③: 函数内部遍历数组求和





函数

- 为什么需要函数
- 函数使用
- 函数传参
- 函数返回值
- 作用域
- 匿名函数



1.4 函数返回值

提问:什么是函数?函数是被设计为执行特定任务的代码块

● 提问:执行完特定任务之后,然后呢?

把任务的结果给我们

● 缺点:把计算后的结果处理方式写死了,内部处理了

● 解决:把处理结果返回给调用者

● 有返回值函数的概念:

> 当调用某个函数,这个函数会返回一个结果出来

▶ 这就是有返回值的函数



```
function getSum(num1, num2) {
    document.write(num1 + n 厨子
}

getSum(10, 20) 点餐人
```



1.4 函数返回值

• 其实我们前面已经接触了很多的函数具备返回值:

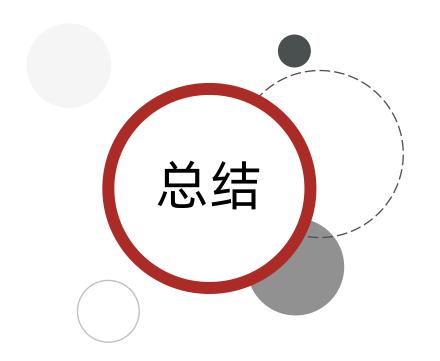
```
let result = prompt('请输入你的年龄?')
let result2 = parseInt('111')
```

- 只是这些函数是JS底层内置的. 我们直接就可以使用
- 当然有些函数,则没有返回值

```
alert('我是弹框,不需要返回值')
```

● 所以要根据<mark>需求</mark>,来设定需不需要返回值





1. 函数很多情况下需要返回值





1.4 函数返回值

- 当函数需要返回数据出去时,用return关键字
- 语法

return 数据

return 20

• 怎么使用呢?

```
function getSum(x, y) {
    return x + y
}
let num = getSum(10, 30)
document.write(num)
```

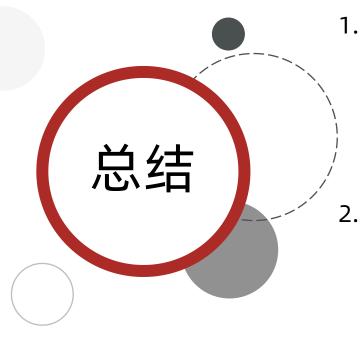


三. 有返回值的函数

● 细节:

- > 在函数体中使用 return 关键字能将内部的执行结果交给函数外部使用
- ▶ return 后面代码不会再被执行,会立即结束当前函数,所以 return 后面的数据不要换行写
- ➤ return函数可以没有 return,这种情况函数默认返回值为 undefined





1. 为什么要让函数有返回值

- ▶ 函数执行后得到结果,结果是调用者想要拿到的(一句话,函数内部不需要输出结果,而是返回结果)
- ▶ 对执行结果的扩展性更高,可以让其他的程序使用这个结果
- 2.函数有返回值用那个关键字? 有什么注意事项呢?
 - ➤ 语法: return 数据
 - ➤ return后面不接数据或者函数内不写return,函数的返回值是 undefined
 - ▶ return能立即结束当前函数, 所以 return 后面的数据不要换行写





• 函数返回值练习

- 1. 求任意2个数中的最大值,并返回
- 1. 求任意数组中的最大值并返回这个最大值
- 2. 求任意数组中的最小值并返回这个最小值

断点调试:

进入函数内部看执行过程 F11



思考: 如何返回多个数据?



1.4 函数细节补充

- 两个相同的函数后面的会覆盖前面的函数
- 在Javascript中 实参的个数和形参的个数可以不一致
 - ▶ 如果形参过多 会自动填上undefined (了解即可)
 - ▶ 如果实参过多 那么多余的实参会被忽略(函数内部有一个arguments,里面装着所有的实参)
- 函数一旦碰到return就不会在往下执行了 函数的结束用return



思考: break的结束和return结束有什么区





函数

- 为什么需要函数
- 函数使用
- 函数传参
- 函数返回值
- 作用域
- 匿名函数





Oncaught ReferenceError: i is not defined

```
function fun() {
   let num2 = 20
}
fun()
console.log(num2)
```

console.log(num2)

▶ Uncaught ReferenceError: num2 is not defined



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<mark>可用性的代码范围</mark>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。

全局作用域

全局有效

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



局部有效

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。



在JavaScript中,根据作用域的不同,变量可以分为:



函数外部let 的变量

全局变量在任何区域都 可以访问和修改



函数内部1et的变量

局部变量只能在当前函 数内部访问和修改

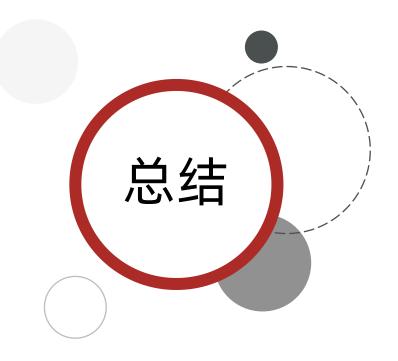


变量有一个坑, 特殊情况:

如果函数内部,变量没有声明,直接赋值,也当全局变量看,但是强烈不推荐

但是有一种情况,函数内部的形参可以看做是局部变量。





- 1. JS 中作用域分为哪2种?
 - ➤ 全局作用域。函数外部或者整个script 有效
 - ▶局部作用域。也称为函数作用域,函数内部有效
- 2. 根据作用域不同,变量分为哪2种?
 - ▶ 全局变量
 - ▶局部变量
- 3. 有一种特殊情况是全局变量是那种? 我们提倡吗?
 - ➤ 局部变量或者块级变量 没有let 声明直接赋值的当全局变量看
 - ▶ 我们强烈不提倡
 - ▶ 还有一种特殊情况,函数内部的形参可以当做局部变量看





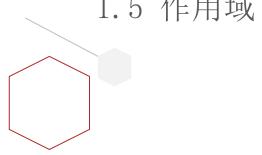
1. 在不同作用域下,可能存在变量命名冲突的情况,到底改执行谁呢?

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



变量的访问原则

- 只要是代码,就至少有一个作用域
- 写在函数内部的局部作用域
- 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
- 访问原则: 在能够访问到的情况下 先局部, 局部没有在找全局



3. 变量访问原则

案例 1: 结果是几?

```
function f1() {
  let num = 123
  function f2() {
     console.log( num )
  f2()
let num = 456
f1()
```



案例 1: 结果是几?

```
function f1() {
      let num = 123
      function f2() {
        let num = 0
        console.log(num)
      f2()
let num = 456
f1()
```

作用域链:采取就近原则的方式来查找变量最终的值

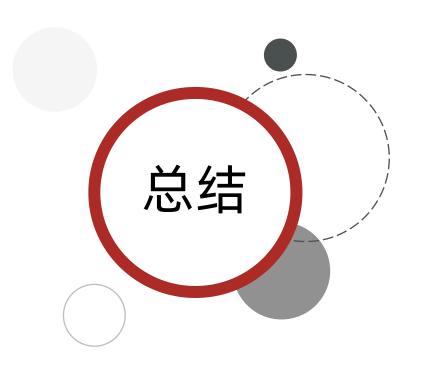


3. 变量访问原则

案例 3: 结果是几?

```
let a = 1
function fn1() {
   let a = 2
    let b = '22'
    fn2()
    function fn2() {
        let a = 3
        fn3()
        function fn3() {
            let a = 4
            console.log(a) //a的值 ?
            console.log(b) //b的值 ?
fn1()
```





- 1. 变量访问原则是什么?
 - ➤采取就近原则的方式来查找变量最终的值

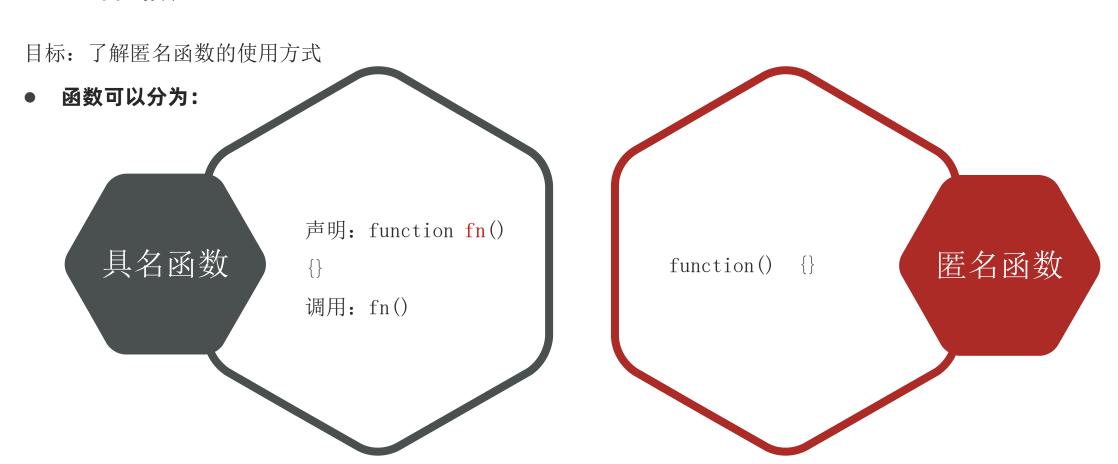




函数

- 为什么需要函数
- 函数使用
- 函数传参
- 函数返回值
- 作用域
- 匿名函数







匿名函数

没有名字的函数,无法直接使用。

使用方式:

- ▶ 函数表达式
- > 立即执行函数



1. 函数表达式

将匿名函数赋值给一个变量,并且通过变量名称进行调用 我们将这个称为函数表达式语法:

调用:

```
fn() // 函数名()
```

其中函数的形参和实参使用跟具名函数一致。



使用场景

目前没有, 先认识



● 后期 web API 会使用:



- 1.6 匿名函数
- 2. 立即执行函数

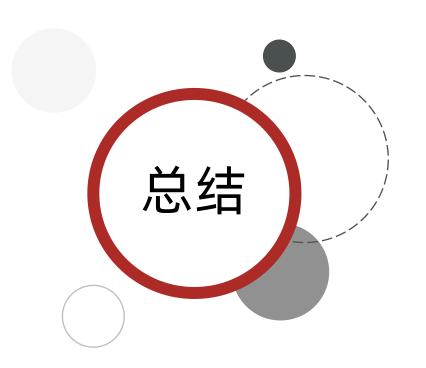
场景介绍: 避免全局变量之间的污染

语法:

```
// 方式1
(function () { console.log(11) })();
// 方式2
(function () { console.log(11) }());
// 不需要调用,立即执行
```

注意: 多个立即执行函数要用;隔开,要不然会报错





- 1. 立即执行函数有什么作用?
 - ▶ 防止变量污染
- 2. 立即执行函数需要调用吗? 有什么注意事项呢?
 - ▶ 无需调用,立即执行,其实本质已经调用了
 - ▶ 多个立即执行函数之间用分号隔开





- ◆ 函数
- ◆ 综合案例





转换时间案例

需求: 用户输入秒数,可以自动转换为时分秒

应用 ■ 双元总调	此网页显示 输入总的秒数:		≫ □ 阅读清
	1000 I		
	*	确定 取消	
			J



1 案例

转换时间案例

需求: 用户输入秒数,可以自动转换为时分秒

分析:

①: 用户输入总秒数 (注意默认值)

②: 计算时分秒(封装函数) 里面包含数字补0

③:打印输出

计算公式: 计算时分秒

小时: h = parseInt(总秒数 / 60 / 60 % 24)

分钟: m = parseInt(总秒数 / 60 % 60)

秒数: s = parseInt(总秒数 % 60)



逻辑中断

开发中,还会见到以下的写法:

其实类似参数的默认值写法



1. 逻辑运算符里的短路

● 短路:只存在于 && 和 || 中,当满足一定条件会让右边代码不执行

符号	短路条件
&&	左边为false就短路
I	左边为true就短路

● 原因:通过左边能得到整个式子的结果,因此没必要再判断右边

● 运算结果:无论 && 还是 || // 运算结果都是最后被执行的表达式值 // 一般用在变量赋值





```
(function flexible(window, document) {
  var dpr = window.devicePixelRatio || 1
}(window, document))
```

```
(function flexible(window, document) {
   // window.devicePixelRatio 获取当前设备的 dpr
   // 获取不到,则默认取值为1
   // 移动端 获取为2,则执行2
   var dpr = window.devicePixelRatio || 1
}(window, document))
}(mindow, document))
```



2. 转换为Boolean型

显示转换:

1.Boolean(内容)

记忆: "、0、undefined、null、false、NaN 转换为布尔值后都是false, 其余则为 true

```
console.log(false && 20) // false
console.log(5 < 3 && 20) // flase
console.log(undefined && 20) // undefined
console.log(null && 20) // null
console.log(0 && 20) // 0
console.log(10 && 20) // 20</pre>
```

```
console.log(false || 20) // 20
console.log(5 < 3 || 20) // 20
console.log(undefined || 20) // 20
console.log(null || 20) // 20
console.log(0 || 20) // 20
console.log(10 || 20) // 10</pre>
console.log(10 || 20) // 10
```



2. 转换为Boolean型

隐式转换:

- 1. 有字符串的加法 "" + 1 , 结果是 "1"
- 2. 减法 (像大多数数学运算一样)只能用于数字,它会使空字符串 ""转换为 0
- 3. null 经过数字转换之后会变为 0
- 4. undefined 经过数字转换之后会变为 NaN

```
console.log('' - 1)
console.log('pink老师' - 1)
console.log(null + 1)
console.log(undefined + 1)
console.log(NaN + 1)
```

```
> '' - 2
< -2
> null + 3
< 3
> undefined + 3
< NaN
> null == undefined
< true
> null === undefined
< false</pre>
```

```
-1
NaN
1
NaN
NaN
```



今日复习路线

- 晚自习回来每个同学先必须xmind梳理今日知识点 (md 笔记也行)
- 需要把今天的所有案例,按照书写顺序写一遍,特别是综合案例。
- 开始做今日测试题. 移动端扫码, PC端: https://ks.wjx.top/vj/wV1PSk1.aspx
- 独立书写今日作业 , 见附件: 06-作业
- 每日一句鼓励自己的话:



将来的你一定会感谢现在拼命的自己



传智教育旗下高端IT教育品牌