

JavaScript 第四天

函数

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



目录

Contents

- ◆ 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



思路

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。



生成外链播放器

单曲 • 我的滑板鞋

歌手: 约瑟翰庞麦郎

所属专辑: Panmalon Jon作品集

播放 + 收藏 分享 下载 (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
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双滑板鞋
与众不同最时尚跳舞肯定棒
整个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

我的滑板鞋 约瑟翰庞麦郎 魅力之都

00:00

一段播放《我的滑板鞋》的代码



总结

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

1.2 函数使用

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~~')  
}
```

1.2 函数使用

目标：掌握函数语法，把代码封装起来

- 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

1.2 函数使用

目标：掌握函数语法，把代码封装起来

- 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()  
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

1.2 函数使用

目标：掌握函数语法，把代码封装起来

- 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()  
sayHi()
```

函数体

1.2 函数使用

目的：通过这段代码 **封装** 成函数，体会函数的好处

```
<style>
  .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;
}
```



总结

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

案例

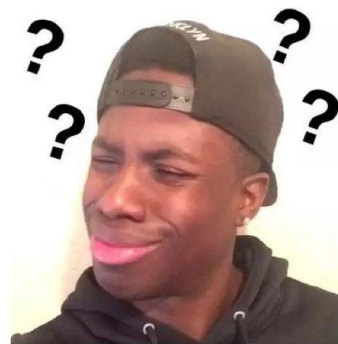
函数案例

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

1.3 函数传参

1. 为什么要有参数的函数

- 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



2. 有参数的函数声明和调用

- 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)  
}
```

1.3 函数传参

2 有参数的函数声明和调用

- 调用语法

函数名(传递的参数列表)

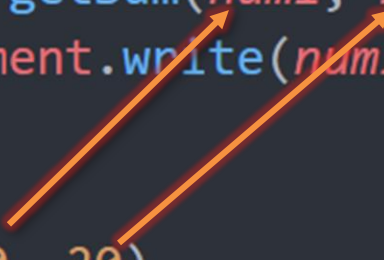
- 例

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

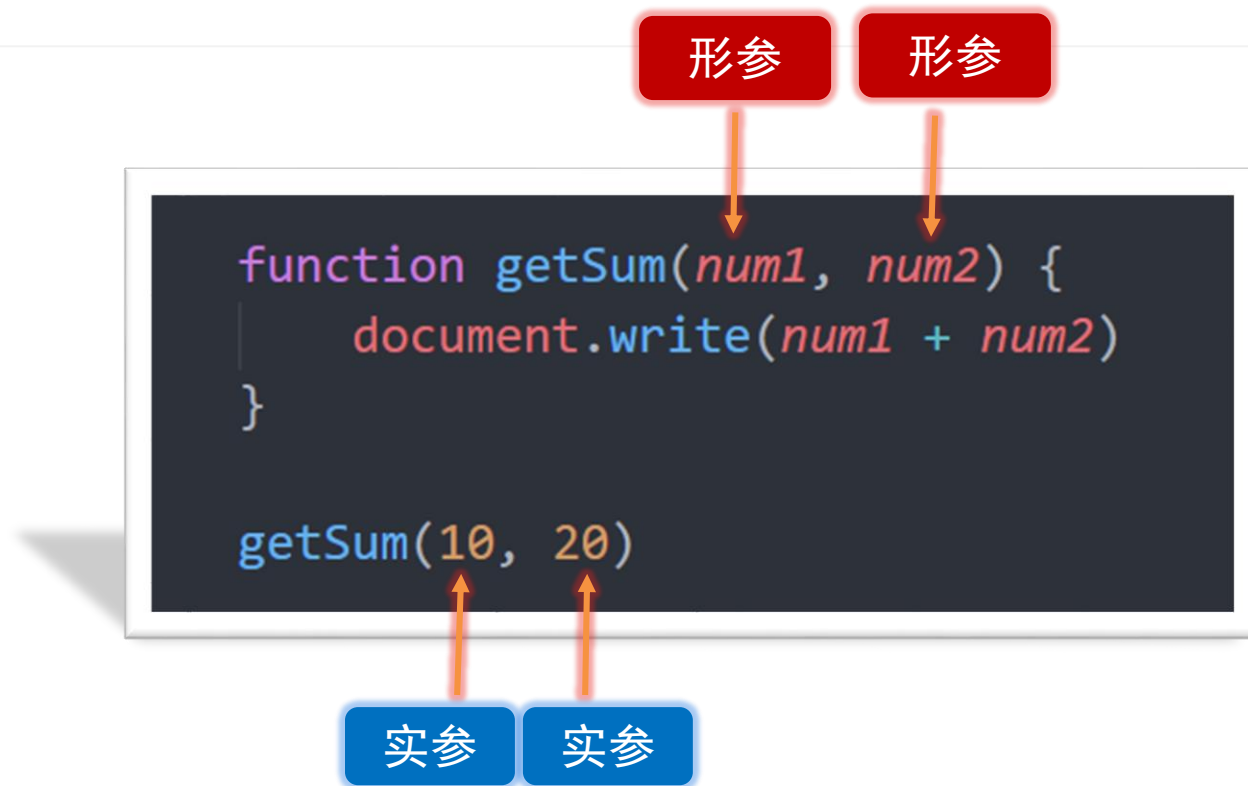
```
function getSum(num1, num2) {  
    document.write(num1 + num2)  
}  
  
getSum(10, 20)
```

Two orange arrows originate from the arguments '10' and '20' in the function call 'getSum(10, 20)' and point to the parameters 'num1' and 'num2' in the function definition 'function getSum(num1, num2)'. This illustrates how arguments are passed to parameters.

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

1.3 函数传参

3 形参和实参



- 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

总结

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



案例

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

兑现承诺：

合理利用**逻辑中断**

- 形参如果不被赋值，就是undefined

```
function getSum(x, y) {  
    x = x || 0  
    y = y || 0  
    console.log(x + y)  
}  
getSum(1, 2)
```

```
getSum(1, 5)
```




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

1. 为什么要让函数有返回值

- 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 + num2)  
}
```

厨子

```
getSum(10, 20)
```

点餐人

1.4 函数返回值

1. 为什么要让函数有返回值

- 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')
```

- 所以要根据需求，来设定需不需要返回值

总结

1. 函数很多情况下需要返回值

发票信息

开企业抬头发票须填写纳税人识别号，以免影响报销

普通发票

个人

商品明细

修改

使用优惠/礼品卡/抵用

6 件商品，总商品金额:

¥5919.10

运费:

¥0.00

返现:

应付总额:

¥5919.10

提交订单

1.4 函数返回值

2 用return返回数据

- 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)
```

3.2 用return返回数据

- 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
- 函数内部只能出现 1 次 return, 并且 return 后面代码不会再被执行, 所以 return 后面的数据不要换行写
- return会立即结束当前函数
- 函数可以没有 return, 这种情况函数默认返回值为 undefined



总结

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
3. 求任意2个数中的最大值, 并返回

断点调试:

进入函数内部看执行过程 F11



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

```
for (let i = 0; i < 3; i++) {  
    document.write('怎么回事呢? ')  
}  
console.log(i)
```

console.log(i)

✖ ▶ Uncaught ReferenceError: i is not defined

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

console.log(num2)

✖ ▶ Uncaught ReferenceError: num2 is not defined

1. 作用域概述

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。



2. 变量的作用域

在JavaScript中，根据作用域的不同，变量可以分为



全局变量

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



局部变量

函数内部let的变量

局部变量只能在当前函数内部访问和修改



块级变量

{ } 内部的let变量

let定义的变量,只能在块作用域里访问,不能跨块访问,也不能跨函数访问

3. 变量的作用域

变量有一个坑，特殊情况：

如果函数内部或者块级作用域内部，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。



总结

1. JS 中作用域分为哪三种？

- 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
- 块级作用域。{} 内有效

2. 根据作用域不同，变量分为哪三种？

- 全局变量
- 局部变量
- 块级变量

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

思考

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()
```


3. 变量访问原则-作用域链

- 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又可以诞生一个作用域
- 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

3. 变量访问原则-作用域链

案例 1： 结果是几？

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

3. 变量访问原则-作用域链

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-作用域链

案例 2： 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()
```



总结

1. 变量访问原则是什么？

➤ 作用域链：采取**就近原则**的方式来查找变量最终的值



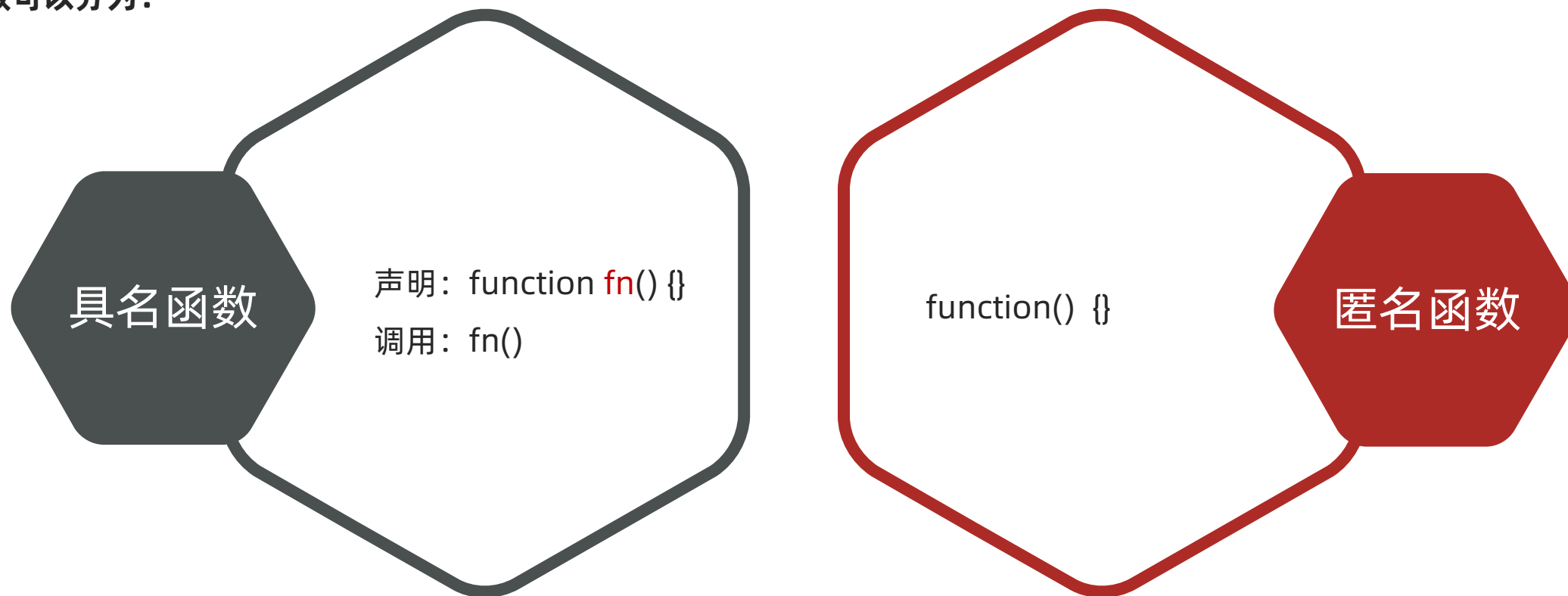
函数

- 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

1.6 匿名函数

目标：掌握匿名函数的写法

函数可以分为：



1.6 匿名函数

目标：掌握匿名函数的写法

1. 匿名函数

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：

```
let fn = function () {  
    // 函数体  
}
```

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。

1.6 匿名函数

目标：掌握匿名函数的写法

使用场景

目前没有，先认识



后期 web API 会使用：

```
<body>
  <button>点击我</button>
  <script>
    let btn = document.querySelector('button')
    btn.onclick = function() {
      alert('我是匿名函数')
    }
  </script>
</body>
```

```
<body>
  <button>点击我</button>
  <script>
    let btn = document.querySelector('button')
    btn.addEventListener(function() {
      alert('弹出')
    })
  </script>
</body>
```

点击我

我是匿名函数[

确定

1.6 匿名函数

目标：掌握匿名函数的写法

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



总结

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



目录

Contents

- ◆ 函数
- ◆ 综合案例

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



The screenshot shows a web browser window with a modal dialog box. The dialog box has a title bar with '应用' (Application) and '双元总课' (Dual Unit Total Course). The main content of the dialog is titled '此网页显示' (This web page displays) and contains the text '输入总的秒数:' (Enter the total seconds:). Below this text is a text input field containing the value '1000'. At the bottom right of the dialog are two buttons: '确定' (Confirm) and '取消' (Cancel). The background of the browser window shows a sidebar with '应用' (Application) and '双元总课' (Dual Unit Total Course) and a main area with a '阅读清单' (Reading List) section.

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
- 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： $h = \text{parseInt}(\text{总秒数} / 60 / 60 \% 24)$

分钟： $m = \text{parseInt}(\text{总秒数} / 60 \% 60)$

秒数： $s = \text{parseInt}(\text{总秒数} \% 60)$

1. 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)
2. 需要把今天的所有案例，按照书写顺序写一遍。
3. 独立书写今日作业
4. 练习下前面3天每天的综合案例
5. 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