

流程控制







- 1. 掌握常见运算符,为程序"能思考"做准备
- 2. 掌握分支语句,让程序具备判断能力
- 3. 掌握循环语句, 让程序具备重复执行能力





- ◆ 运算符
- ◆ 语句
- ◆ 综合案例





运算符

- · 赋值运算符
- 一元运算符
- 比较运算符
- 逻辑运算符
- 运算符优先级



1.1 赋值运算符

目标: 能够使用赋值运算符简化代码

- 赋值运算符:对变量进行赋值的运算符
 - ▶ 已经学过的赋值运算符: = 将等号右边的值赋予给左边,要求左边必须是一个容器
 - ▶ 其他赋值运算符:
 - **>** +=
 - > -=
 - > *=
 - > /=
 - > %=
- 使用这些运算符可以在对变量赋值时进行快速操作



1.1 赋值运算符

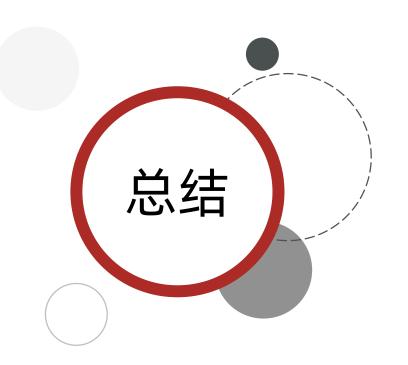
我们以 += 赋值运算符来举例

1. 以前我们让一个变量加 1 如何做的?

2. 现在我们有一个简单的写法啦~~~

提问:想变量加3怎么写?





- 1. = 赋值运算符执行过程?
 - 》 将等号右边的值赋予给左边, 要求左边必须是一个容器
- 2. += 出现是为了简化代码, 比如让 let num = 10, num 加5 怎么写呢?
 - > num += 5





运算符

- 赋值运算符
- 一元运算符
- 比较运算符
- 逻辑运算符
- 运算符优先级



目标: 能够使用一元运算符做自增运算

众多的 JavaScript 的运算符可以根据所需表达式的个数,分为一元运算符、二元运算符、三元运算符

- 二元运算符:
 - ➤ 例:

$$let num = 10 + 20$$

● 一元运算符:

▶ 例: 正负号

● 问题: 我们以前让一个变量每次+1,以前我们做的呢?

```
let num = 1
num = num + 1
```



- 我们可以有更简便的写法了~~~
- 自增:
 - ▶ 符号: ++
 - ▶ 作用: 让变量的值 +1
- 自减:
 - ▶ 符号: --
 - ▶ 作用: 让变量的值 -1
- 使用场景:经常用于<mark>计数</mark>来使用。 比如<u>进行10次操作,用它</u>来计算进行了多少次了



- 自增运算符的用法:
- ◆ 前置自增:

```
let num = 1
++num // 让num的值加 1变 2
```

- ▶ 每执行1次,当前变量数值加1
- ▶ 其作用相当于 num += 1

前置自增和后置自增单独使用没有区别

◆ 后置自增:

```
let num = 1
num++ // 让num的值加 1变 2
```

- ▶ 每执行1次,当前变量数值加1
- ▶ 其作用相当于 num += 1



● 自增运算符的用法:

前置自增和后置自增如果参与运算就有区别:(难点,但是了解即可)

◆ 前置自增:

▶ 前置自增:先自加再使用(记忆口诀: ++在前 先加)

```
let i = 1
console.log(++i + 2) //结果是 4
// 注意: i是 2
// i先自加 1,变成2之后,在和后面的2相加
```

◆ 后置自增:

▶ 后置自增:先使用再自加(记忆口诀: ++在后 后加)

```
let i = 1
console.log(i++ + 2) //结果是 3
// 注意: 此时的 i是 1
// 先和2相加,先运算输出完毕后,i再自加是2
```

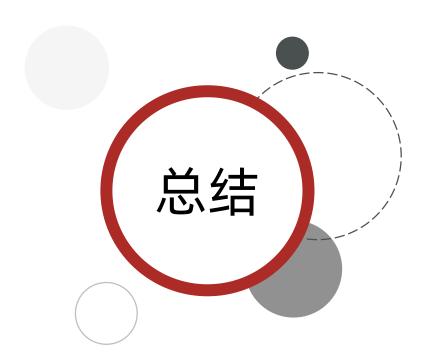


自增运算符的用法:

but:

- 1. 前置自增和后置自增独立使用时二者并没有差别!
- 2. 一般开发中我们都是独立使用
- 3. 后面 i++ 后置自增会使用相对较多,并且都是单独使用





- 1.只需要一个表达式就可以运算的运算符叫一元运算符
- 2. 自增运算符也是为了简化写法,每次自加1,使用场景是什么?
 - ▶ 经常用于计数来使用。用来计算多少次
- 3. 实际开发中,我们一般都是单独使用的,后置++ 更多





面试题:

```
let i = 1
console.log(i++ + ++i + i)
```





运算符

- 赋值运算符
- 一元运算符
- 比较运算符
- 逻辑运算符
- 运算符优先级



目标: 能使用常见的比较运算符进行比较运算

学习路径:

- 比较运算符的介绍
- 比较运算符的使用
- 比较运算符的细节



● 比较运算符的介绍

▶ 使用场景:比较两个数据大小、是否相等

▶ 实际运用例:





● 比较运算符:

▶ >: 左边是否大于右边

▶ <: 左边是否小于右边

▶ >=: 左边是否大于或等于右边

▶ <=: 左边是否小于或等于右边

▶ ==: 左右两边值是否相等

▶ ===: 左右两边是否类型和值都相等

▶ !==: 左右两边是否不全等

▶ 比较结果为boolean类型,即只会得到 true 或 false

● 对比:

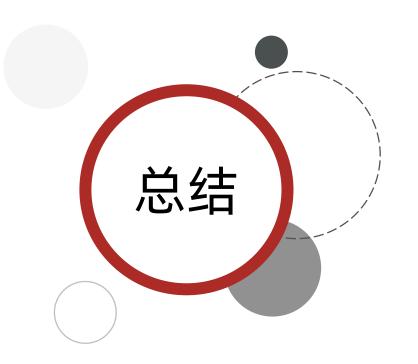
- ▶ = 单等是赋值
- ▶ == 是判断
- ▶ === 是全等
- ▶ 开发中判断是否相等,强烈推荐使用 ===



- 字符串比较,是比较的字符对应的ASCII码
 - ▶ 从左往右依次比较
 - ▶ 如果第一位一样再比较第二位,以此类推
 - ▶ 比较的少,了解即可
- NaN不等于任何值,包括它本身
 - ➤ 涉及到"NaN "都是false
- 尽量不要比较小数,因为小数有精度问题
- 不同类型之间比较会发生隐式转换
 - ▶ 最终把数据隐式转换转成number类型再比较
 - ▶ 所以开发中,如果进行准确的比较我们更喜欢 === 或者!==







- 1. = 和 == 和 === 怎么区别?
 - ▶ = 是赋值
 - > == 是判断 只要求值相等,不要求数据类型一样即可返回true
 - > === 是全等 要求值和数据类型都一样返回的才是true
 - ▶ 开发中,请使用 ===
- 2. 比较运算符返回的结果是什么?
 - ➤ 结果只有2个, true 或者 false





运算符

- 赋值运算符
- 一元运算符
- 比较运算符
- 逻辑运算符
- 运算符优先级



1.4 逻辑运算符

目标: 掌握逻辑运算符, 为程序"能思考"做准备

学习路径:

- > 逻辑运算符的介绍
- > 逻辑运算符的使用



1.4 逻辑运算符

● 提问:如果我想判断一个变量 num 是否大于5且小于10,怎么办?

➤ 错误写法: 5 < num < 10</p>

● **使用场景:**逻辑运算符用来解决多重条件判断

➤ 正确写法: num > 5 && num < 10</p>

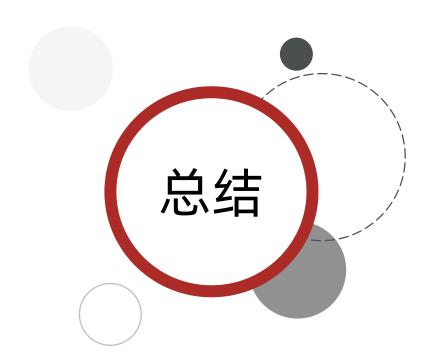


1.4 逻辑运算符

● 逻辑运算符:

符号	名称	日常读法	特点	口诀
&&	逻辑与	并且	符号两边都为true 结果才为true	一假则假
	逻辑或	或者	符号两边有一个 true就为true	一真则真
!	逻辑非	取反	true变false false变true	真变假,假变真





- 1. 逻辑运算符有那三个?
 - ▶ 与(&&) 或(||) 非(!)
- 2. 判断一个变量 num 是否大于5且小于10怎么写?

num > 5 && num < 10





• 判断一个数是4的倍数,且不是100的倍数

需求:用户输入一个,判断这个数能被4整除,但是不能被100整除,满足条件,页面弹出true,否则弹出false分析:

①:用户输入

②:判断条件,看余数是不是0,如果是0就是能被整除,余数不是0,则不能被整除







运算符

- 赋值运算符
- 一元运算符
- 比较运算符
- 逻辑运算符
- 运算符优先级



1.5 运算符优先级

目标: 掌握运算符优先级, 能判断运算符执行的顺序

优先级	运算符	顺序
1	小括号	()
2	一元运算符	++ !
3	算数运算符	先*/% 后+-
4	关系运算符	> >= < <=
5	相等运算符	== != === !==
6	逻辑运算符	先 & & 后
7	赋值运算符	=
8	逗号运算符	,

- 一元运算符里面的逻辑非优先级很高
- 逻辑与比逻辑或优先级高



练习

```
let a = 3 > 5 && 2 < 7 && 3 == 4
console.log(a);
let b = 3 <= 4 || 3 > 1 || 3 != 2
console.log(b);
let c = 2 === "2"
console.log(c);
let d = !c || b && a
console.log(d);
```





- ◆ 运算符
- ◆ 语句
- ◆ 综合案例





语句

- 表达式和语句
- · 分支语句
- 循环语句



2.1 表达式和语句

目标: 能说出表达式和语句的区别

● 表达式:

表达式是可以被求值的代码, JavaScript 引擎会将其计算出一个结果。

```
x = 7
3 + 4
num++
```



目标: 能说出表达式和语句的区别

● 语句:

语句是一段可以执行的代码。

比如: prompt() 可以弹出一个输入框,还有 if语句 for 循环语句等等



目标: 能说出表达式和语句的区别

区别:

表达式: 因为表达式可被求值, 所以它可以写在赋值语句的右侧。

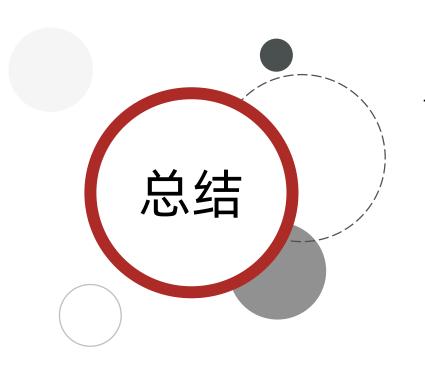
▶ 表达式 num = 3 + 4

语句: 而语句不一定有值, 所以比如 alert() for和break 等语句就不能被用于赋值。

➤ 语句 alert() 弹出对话框 console.log() 控制台打印输出

某些情况,也可以把表达式理解为表达式语句,因为它是在计算结果,但不是必须的成分 (例如continue语句)





1. 表达式和语句的区别

- > 因为表达式可被求值,所以它可以写在赋值语句的右侧。
- ➤ 而语句不一定有值,所以比如 alert() for和break 等语句就不能被用于赋值。





语句

- 表达式和语句
- 分支语句
- 循环语句



2.2 分支语句

目标: 掌握流程控制, 写出能"思考"的程序

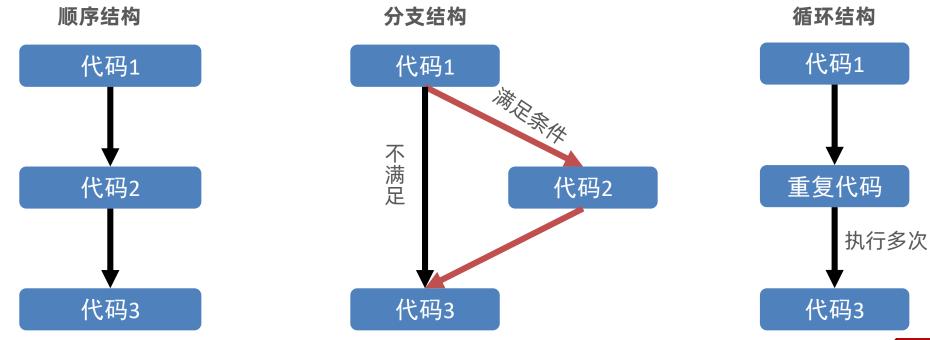
学习路径:

- 程序三大流程控制语句
- ▶ 分支语句



程序三大流程控制语句

- 以前我们写的代码,写几句就从上往下执行几句,这种叫顺序结构
- 有的时候要根据条件选择执行代码,这种就叫分支结构
- 某段代码被重复执行,就叫循环结构





2.2 分支语句

目标: 掌握流程控制, 写出能"思考"的程序

学习路径:

- 程序三大流程控制语句
- ▶ 分支语句



2. 分支语句

- ◆ 分支语句可以让我们有选择性的执行想要的代码
- 分支语句包含:
 - ▶ If分支语句
 - ▶ 三元运算符
 - > switch 语句





目标:能使用if语句执行满足条件的代码

1. if语句

- if语句有三种使用:单分支、双分支、多分支
- 单分支使用语法:

```
if (条件) { 满足条件要执行的代码 }
```

- ▶ 括号内的条件为true时,进入大括号里执行代码
- ▶ 小括号内的结果若不是布尔类型时,会发生隐式转换转为布尔类型
- ▶ 如果大括号只有一个语句,大括号可以省略,但是,俺们不提倡这么做~





if语句

●单分支课堂案例1:用户输入高考成绩,如果分数大于700,则提示恭喜考入黑马程序员



● 双分支if语法:









判断用户登录案例

需求:用户输入,用户名:pink,密码:123456,则提示登录成功,否则提示登录失败

分析:

①: 弹出输入框,分别输入用户名和密码

②:通过if语句判断,如果用户名是pink,并且密码是123456,则执行if里面的语句,否则执行else里面的语句。

此网页显示
请输入用户名:
a T
a TM
1. 啊 2. 阿 3. 吖 4. 嗄 5. 锕 ◀▶ ☑ 黑马程序 员 www.itheima.com





判断闰年案例

需求:让用户输入年份,判断这一年是闰年还是平年并弹出对应的警示框

分析:

①:能被4整除但不能被100整除,或者被400整除的年份是闰年,否则都是平年

②:需要逻辑运算符

此网页显示 请输入年份:				
200	I			
		确定	取消	



2.2.1 if语句

● 多分支if语法:

使用场景: 适合于有多个结果的时候, 比如学习成绩可以分为: 优良 中差

```
if (条件1) {
   代码1
} else if (条件2) {
   代码2
} else if (条件3) {
   代码3
} else {
   代码n
```

释义:

- ▶ 先判断条件1, 若满足条件1就执行代码1, 其他不执行
- ▶ 若不满足则向下判断条件2,满足条件2执行代码2,其他不执行
- ▶ 若依然不满足继续往下判断, 依次类推
- ➤ 若以上条件都不满足,执行else里的代码n
- ▶ 注:可以写N个条件,但这里演示只写2个





输入成绩案例

需求:根据输入不同的成绩,反馈不同的评价

注:

①:成绩90以上是优秀

②: 成绩70~90是 良好

③: 成绩是60~70之间是 及格

④: 成绩60分以下是 不及格



2. 分支语句

- ◆ 分支语句可以让我们有选择性的执行想要的代码
- 分支语句包含:
 - ▶ If分支语句
 - ▶ 三元运算符
 - > switch 语句



目标: 能利用三元运算符执行满足条件的语句

● **使用场景:** 其实是比 if 双分支 更简单的写法,可以使用 三元表达式

● **符号:**?与:配合使用

● 语法:

条件? 满足条件执行的代码: 不满足条件执行的代码

● 一般用来取值





判断2个数的最大值

需求: 用户输入2个数, 控制台输出最大的值

分析:

①:用户输入2个数

②: 利用三元运算符输出最大值

此网页显示 请输入第一个数			
	I		
		确定	取消





数字补0案例

需求: 用户输入1个数,如果数字小于10,则前面进行补0,比如0903等

分析:

①: 为后期页面显示时间做铺垫

②: 利用三元运算符补0计算

22:00 点场 距结束

00:02:18



2. 分支语句

- ◆ 分支语句可以让我们有选择性的执行想要的代码
- 分支语句包含:
 - ▶ If分支语句
 - ▶ 三元运算符
 - > switch 语句



2.2.3 switch语句

目标:能利用switch执行满足条件的语句

```
switch (数据) {
   case 值1:
       代码1
       break
   case 值2:
       代码2
       break
   default:
       代码n
       break
```

释义:

- ▶ 找到跟小括号里数据全等的case值,并执行里面对应的代码
- ➤ 若没有全等 === 的则执行default里的代码
- ▶ 例:数据若跟值2全等,则执行代码2

注意事项

- 1. switch case语句一般用于等值判断,不适合于区间判断
- 2. switch case一般需要配合break关键字使用 没有break会造成case穿透





简单计算器

需求:用户输入2个数字,然后输入+-*/任何一个,可以计算结果







简单计算器

需求:用户输入2个数字,然后输入+-*/任何一个,可以计算结果

分析:

①:用户输入数字

②:用户输入不同算术运算符,可以去执行不同的运算 (switch)





语句

- 表达式和语句
- 分支语句
- 循环语句



2.3 循环结构

目标: 掌握循环结构, 实现一段代码重复执行

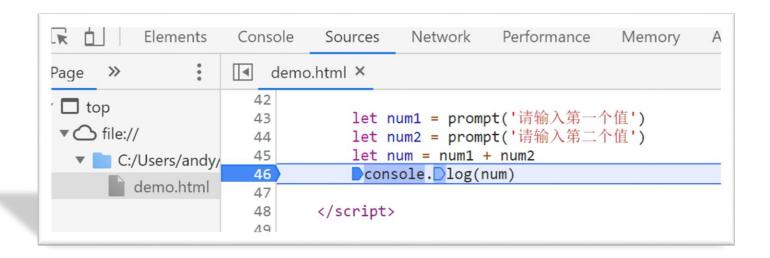
学习路径:

- 1. 断点调试
- 2. while循环



2.3.1 断点调试

- 作用: 学习时可以帮助更好的理解代码运行,工作时可以更快找到bug
- 浏览器打开调试界面
 - 1. 按F12打开开发者工具
 - 2. 点到sources一栏
 - 3. 选择代码文件
- 断点:在某句代码上加的标记就叫断点,当程序执行到这句有标记的代码时会暂停下来





2.3 循环结构

目标: 掌握循环结构, 实现一段代码重复执行

学习路径:

- 1. 断点调试
- 2. while循环



2.3.2 while 循环

目标:掌握while循环语法,能重复执行某段代码

循环: 重复执行一些操作, while: 在....期间, 所以 while循环 就是在满足条件期间, 重复执行某些代码。

比如我们运行相同的代码输出5次(输出5句"我学的很棒")

路径:

> while 循环基本语法

➤ while 循环三要素



2.3.2 while 循环

● 1. while 循环基本语法:

```
while (循环条件) {
    要重复执行的代码(循环体)
}
```

释义:

- ▶ 跟if语句很像,都要满足小括号里的条件为true才会进入循环体执行代码
- ▶ while大括号里代码执行完毕后不会跳出,而是继续回到小括号里判断条件是否满足,若满足又执行大括号里的代码,然后再回到小括号判断条件,直到括号内条件不满足,即跳出



2.3.2 while 循环

2. while 循环三要素:

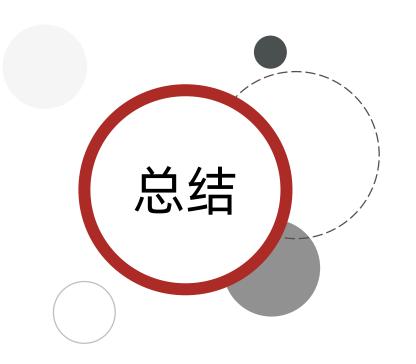
循环的本质就是以某个变量为起始值,然后不断产生变化量,慢慢靠近终止条件的过程。

所以, while循环需要具备三要素:

- 1. 变量起始值
- 2. 终止条件(没有终止条件,循环会一直执行,造成死循环)
- 3. 变量变化量(用自增或者自减)

```
let i = 1
while (i <= 3) {
    document.write('我会循环三次<br>')
    i++
}
```





- 1. while循环的作用是什么?
 - ▶ 在满足条件期间,重复执行某些代码
- 2. while循环三要素是什么?
 - ▶ 变量起始值
 - 终止条件(没有终止条件,循环会一直执行,造成死循环)
 - ▶ 变量变化量(用自增或者自减)





在页面中打印输出10句"月薪过万"

需求:使用while循环,页面中打印,可以添加换行效果





能不能改进,让用户输入打印输出的个数呢?



a 练习

While 练习

需求:使用while循环,页面中打印,可以添加换行效果

- 1. 页面输出1-100
- ▶ 核心思路: 利用 i,因为正好和 数字对应
- 2. 计算从1加到100的总和并输出
- ▶ 核心思路:
 - ▶ 声明累加和的变量 sum
 - ▶ 每次把 i 加到 sum 里面
- 3. 计算1-100之间的所有偶数和
- ▶ 核心思路:
 - ▶ 声明累加和的变量 sum
 - ➤ 首先利用if语句把 i 里面是偶数筛选出来
 - ▶ 把筛选的 i 加到 sum 里面



2.3 循环退出

目标: 能说出continue和break的区别

循环结束:

> continue: 结束本次循环,继续下次循环





2.3 循环退出

● 循环结束:

▶ break: 跳出所在的循环

● 区别:

> continue 退出本次循环,一般用于排除或者跳过某一个选项的时候,可以使用continue

▶ break 退出整个循环,一般用于结果已经得到,后续的循环不需要的时候可以使用





1 案例

页面弹框

需求:页面弹出对话框,'你爱我吗',如果输入'爱',则结束,否则一直弹出对话框

分析:

①:循环条件永远为真,一直弹出对话框

②:循环的时候,重新让用户输入

③:如果用户输入的是: 爱,则退出循环 (break)





- ◆ 运算符
- ◆ 语句
- ◆ 综合案例





简易ATM取款机案例

需求: 用户可以选择存钱、取钱、查看余额和退出功能

就应用□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□	此网页显示 请选择您的操作: 1. 取款 2. 存款 3. 查看余额 4. 退出			>> │ Ⅲ 阅读清单
		I	确定 取消	





简易ATM取款机案例

需求:用户可以选择存钱、取钱、查看余额和退出功能

分析:

①:循环的时候,需要反复提示输入框,所以提示框写到循环里面

②:退出的条件是用户输入了4,如果是4,则结束循环,不在弹窗

③:提前准备一个金额预先存储一个数额

④: 取钱则是减法操作, 存钱则是加法操作, 查看余额则是直接显示金额

⑤:输入不同的值,可以使用switch来执行不同的操作



今日复习路线

- 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. 扫码完成今日检测题,必须全对,一次考不过,继续考试,直到考过为止。
- 4. 独立书写今日作业
- 5. 明天上午复习今天内容, 下午预习后天的内容(for循环+数组)
- 6. 手机扫码测试题: pc端地址: https://ks.wjx.top/vj/h46xYbn.aspx





传智教育旗下高端IT教育品牌