JavaScript基础第02天

1 - 运算符（操作符）

1.1 运算符的分类

运算符（operator）也被称为操作符，是用于实现赋值、比较和执行算数运算等功能的符号。

JavaScript中常用的运算符有：

- 算数运算符
- 递增和递减运算符
- 比较运算符
- 逻辑运算符
- 赋值运算符

1.2 算数运算符

- 算术运算符概述

  概念：算术运算使用的符号，用于执行两个变量或值的算术运算。

运算符	描述	实例
+	加	10 + 20 = 30
-	减	10 - 20 = -10
*	乘	10 * 20 = 200
/	除	10 / 20 = 0.5
%	取余数(取模)	返回除法的余数 9 % 2 = 1

- 浮点数的精度问题

  浮点数值的最高精度是 17 位小数，但在进行算术计算时其精确度远远不如整数。

```
var result = 0.1 + 0.2;    // 结果不是 0.3，而是：0.30000000000000004
console.log(0.07 * 100);   // 结果不是 7，   而是：7.000000000000001
```

  所以：不要直接判断两个浮点数是否相等！

- 表达式和返回值

  表达式：是由数字、运算符、变量等以能求得数值的有意义排列方法所得的组合

  简单理解：是由数字、运算符、变量等组成的式子

  表达式最终都会有一个结果，返回给开发者，称为返回值

1.3 递增和递减运算符

- 递增和递减运算符概述

  如果需要反复给数字变量添加或减去1，可以使用递增（++）和递减（ -- ）运算符来完成。

  在 JavaScript 中，递增（++）和递减（ -- ）既可以放在变量前面，也可以放在变量后面。放在变量前面时，我们可以称为前置递增（递减）运算符，放在变量后面时，我们可以称为后置递增（递减）运算符。

  注意：递增和递减运算符必须和变量配合使用。

- 递增运算符
  - 前置递增运算符

```
++num 前置递增，就是自加1，类似于 num =  num + 1，但是 ++num 写起来更简单。

使用口诀：先自加，后返回值
```

```
var  num = 10;
alert(++num + 10);   // 21
```

  - 后置递增运算符

    num++ 后置递增，就是自加1，类似于 num = num + 1 ，但是 num++ 写起来更简单。

    使用口诀：先返回原值，后自加

```
var  num = 10;
alert(10 + num++);  // 20
```

1.4 比较运算符

- 比较运算符概述

  概念：比较运算符（关系运算符）是两个数据进行比较时所使用的运算符，比较运算后，会返回一个布尔值（true / false）作为比较运算的结果。

运算符名称	说明	案例	结果
<	小于号	1 < 2	true
>	大于号	1 > 2	false
>=	大于等于号 (大于或者等于)	2 >= 2	true
<=	小于等于号 (小于或者等于)	3 <= 2	false
==	判等号（会转型）	37 == 37	true
!=	不等号	37 != 37	false
=== !==	全等 要求值和 数据类型都一致	37 === '37'	false

- 等号比较

符号	作用	用法
=	赋值	把右边给左边
==	判断	判断两边值是否相等（注意此时有隐式转换）
===	全等	判断两边的值和数据类型是否完全相同

```
console.log(18 == '18');
console.log(18 === '18');
```

1.5 逻辑运算符

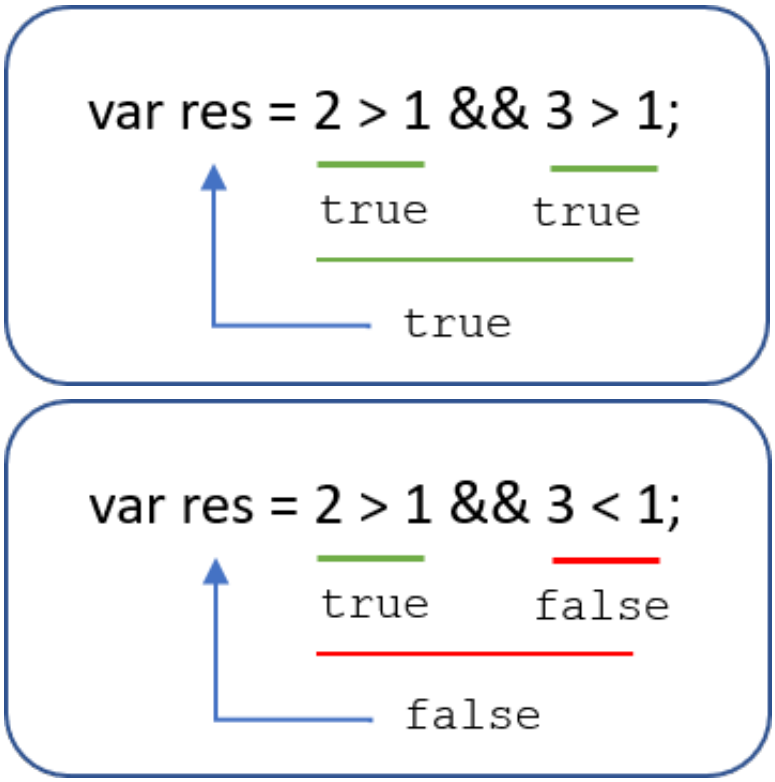
- 逻辑运算符概述

  概念：逻辑运算符是用来进行布尔值运算的运算符，其返回值也是布尔值。后面开发中经常用于多个条件的判断

逻辑运算符	说明	案例
&&	"逻辑与"，简称 "与" and	true && false
||	"逻辑或"，简称 "或" or	true || false
!	"逻辑非"，简称 "非" not	! true

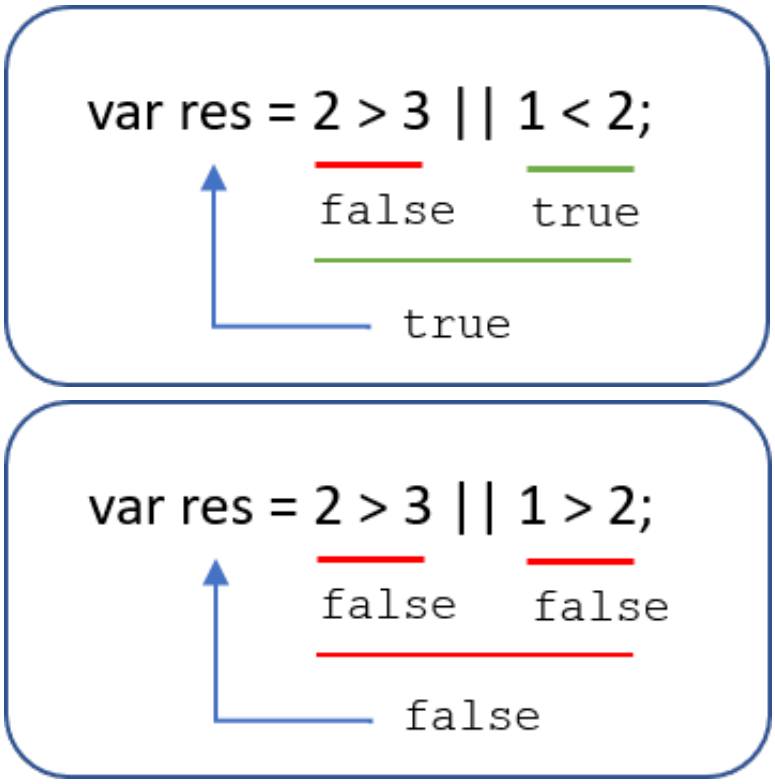
- 逻辑与&&

  两边都是 true才返回 true，否则返回 false



- 逻辑或 ||

  两边都是 true才返回 true，否则返回 false



- 逻辑非！

  逻辑非（!）也叫作取反符，用来取一个布尔值相反的值，如 true 的相反值是 false

```
var isOk = !true;
console.log(isOk);  // false
```

- 短路运算（逻辑中断）

  短路运算的原理：当有多个表达式（值）时,左边的表达式值可以确定结果时,就不再继续运算右边的表达式的值;

  - 逻辑与

    语法： 表达式1 && 表达式2

    - 如果第一个表达式的值为真，则返回表达式2
    - 如果第一个表达式的值为假，则返回表达式1

```
console.log( 123 && 456 );          // 456
console.log( 0 && 456 );            // 0
console.log( 123 && 456&& 789 );    // 789
```

  - 逻辑或

    语法： 表达式1 || 表达式2

    - 如果第一个表达式的值为真，则返回表达式1
    - 如果第一个表达式的值为假，则返回表达式2

```
console.log( 123 || 456 );          //  123
console.log( 0 ||  456 );           //  456
console.log( 123 || 456 || 789 );   //  123
```

1.6 赋值运算符

概念：用来把数据赋值给变量的运算符。

赋值运算符	说明	案例
=	直接赋值	var usrName = '我是值';
+=、-=	加、减 一个 数 后在赋值	var age = 10; age+=5; // 15
*=、/=、%=	乘、除、取模 后在赋值	var age = 2; age*=5; // 10

```
var age = 10;
age += 5;  // 相当于 age = age + 5;
age -= 5;  // 相当于 age = age - 5;
age *= 10; // 相当于 age = age * 10;
```

1.7 运算符优先级

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

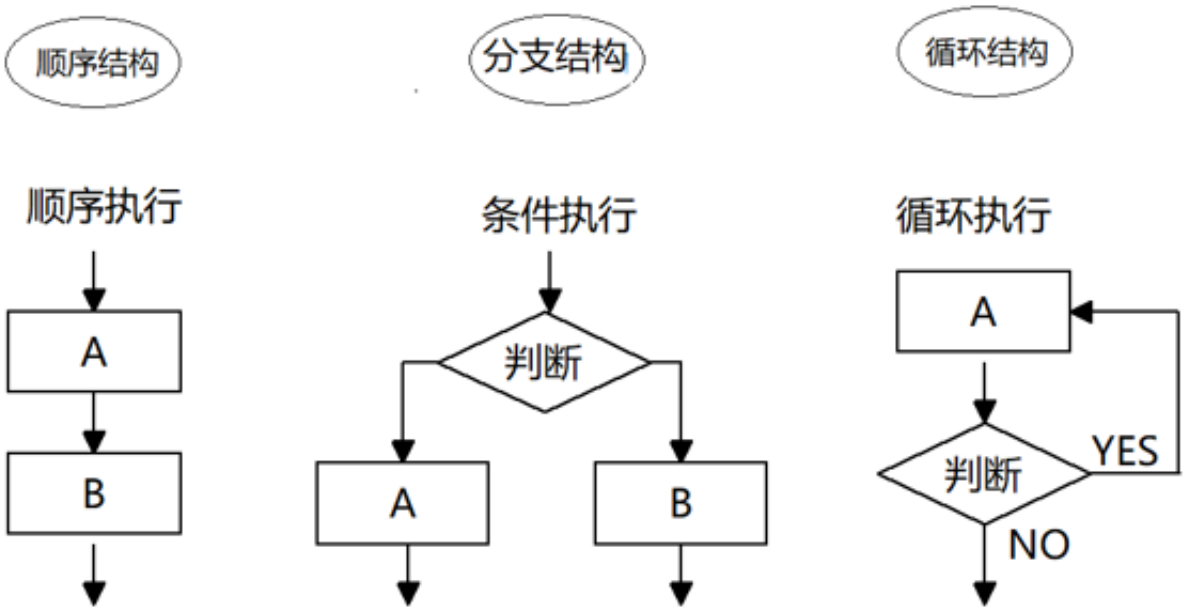
2 - 流程控制

2.1 流程控制概念

在一个程序执行的过程中，各条代码的执行顺序对程序的结果是有直接影响的。很多时候我们要通过控制代码的执行顺序来实现我们要完成的功能。

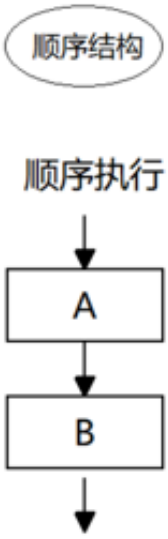
简单理解：**流程控制就是来控制代码按照一定结构顺序来执行**

流程控制主要有三种结构，分别是**顺序结构**、**分支结构**和**循环结构**，代表三种代码执行的顺序。



2.2 顺序流程控制

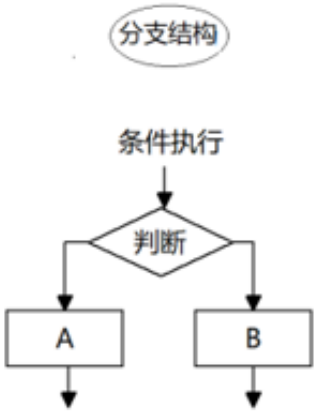
顺序结构是程序中最简单、最基本的流程控制，它没有特定的语法结构，程序会按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。



2.3 分支流程控制

- 分支结构

由上到下执行代码的过程中，根据不同的条件，执行不同的路径代码（执行代码多选一的过程），从而得到不同的结果



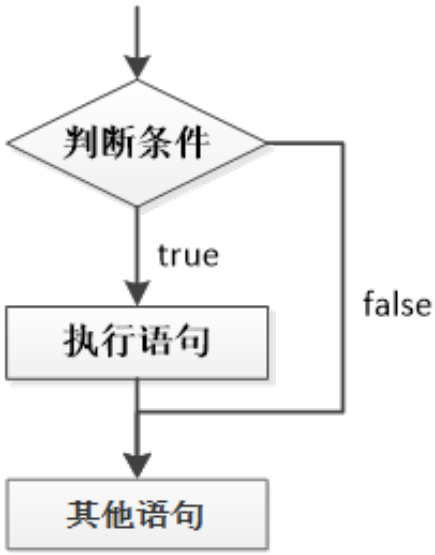
JS 语言提供了两种分支结构语句：if 语句、switch 语句

- if 语句
  - 语法结构

```
// 条件成立执行代码，否则什么也不做
if（条件表达式）{
    // 条件成立执行的代码语句
}
```

语句可以理解为一个行为，循环语句和分支语句就是典型的语句。一个程序由很多个语句组成，一般情况下，会分割成一个一个的语句。

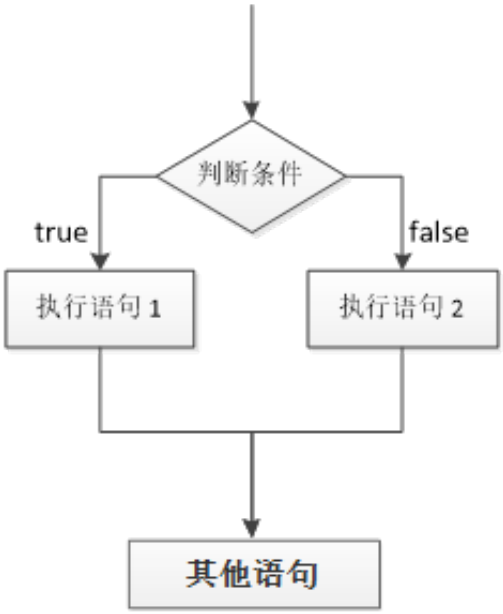
  - 执行流程



- if else语句（双分支语句）
  - 语法结构

```
// 条件成立  执行 if 里面代码，否则执行else 里面的代码
if (条件表达式) {
    // [如果] 条件成立执行的代码
} else {
    // [否则] 执行的代码
}
```

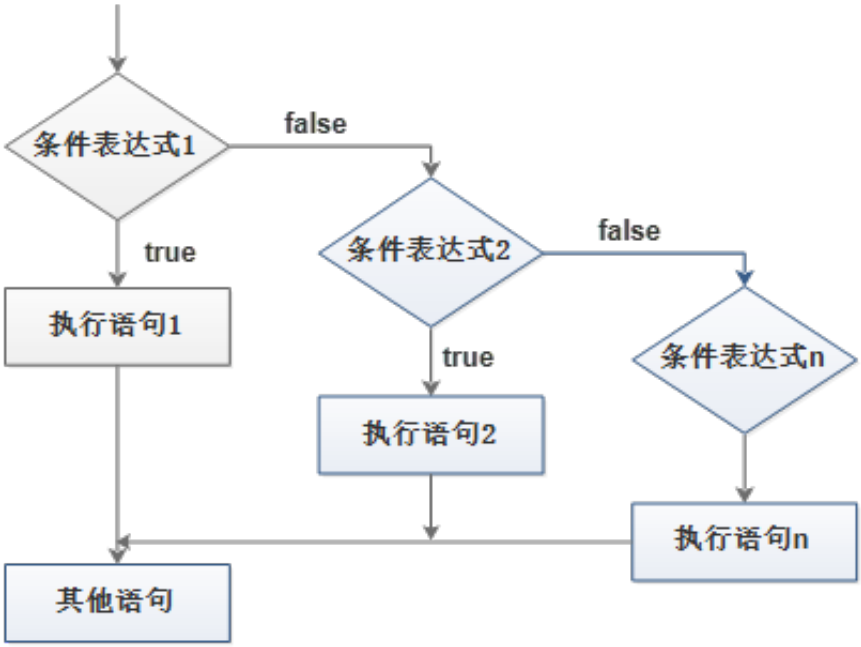
- 执行流程



- if else if 语句(多分支语句)
  - 语法结构

```
// 适合于检查多重条件。
if (条件表达式1) {
    语句1;
} else if (条件表达式2)  {
    语句2;
} else if (条件表达式3)  {
   语句3;
 ....
} else {
    // 上述条件都不成立执行此处代码
}
```

  - 执行逻辑



2.4 三元表达式

- 语法结构

```
表达式1 ? 表达式2 : 表达式3;
```

- 执行思路
  - 如果表达式1为 true ，则返回表达式2的值，如果表达式1为 false，则返回表达式3的值
  - 简单理解： 就类似于 if else （双分支） 的简写

2.5 switch分支流程控制

- 语法结构

  switch 语句也是多分支语句，它用于基于不同的条件来执行不同的代码。当要针对变量设置一系列的特定值的选项时，就可以使用 switch。

```
switch( 表达式 ){
    case value1:
        // 表达式 等于 value1 时要执行的代码
        break;
    case value2:
        // 表达式 等于 value2 时要执行的代码
        break;
    default:
        // 表达式 不等于任何一个 value 时要执行的代码
}
```

  - switch： 开关 转换 ， case： 小例子 选项
  - 关键字 switch 后面括号内可以是表达式或值， 通常是一个变量
  - 关键字 case , 后跟一个选项的表达式或值，后面跟一个冒号

    - switch 表达式的值会与结构中的 case 的值做比较
    - 如果存在匹配全等(===) ，则与该 case 关联的代码块会被执行，并在遇到 break 时停止，整个 switch 语句代码执行结束
    - 如果所有的 case 的值都和表达式的值不匹配，则执行 default 里的代码

      注意： 执行case 里面的语句时，如果没有break，则继续执行下一个case里面的语句。

- switch 语句和 if else if 语句的区别
    - 一般情况下，它们两个语句可以相互替换
    - switch...case 语句通常处理 case为比较确定值的情况， 而 if...else...语句更加灵活，常用于范围判断(大于、等于某个范围)
    - switch 语句进行条件判断后直接执行到程序的条件语句，效率更高。而if...else 语句有几种条件，就得判断多少次。
    - 当分支比较少时，if... else语句的执行效率比 switch语句高。
    - 当分支比较多时，switch语句的执行效率比较高，而且结构更清晰。