JavaScript常用的操作语句

JS判断操作语句

> 通过一系列的逻辑判断，来完成特定的事情

`if 、else if、else`

```
if(条件1){
    //=>条件1成立执行的操作
}else if(条件2){
    //=>条件1不成立,条件2成立执行的操作
}else if(条件3){
    //=>上面条件不成立，条件3成立执行的操作
}
...
else{
    //=>以上条件都不成立执行的操作
}
```

案例剖析

```
var num = 10;
if (num < 5) {
    num++;//=>num+=1  =>num=num+1  在自身基础上累加1
} else if (num >= 5 && num <= 15) {
    //=>&&: 并且,所有条件都成立整体才成立
    //=>||: 或者,只要有一个条件成立整体就成立
    num += 2;
} else {
    num--;
}
console.log(num);//=>12
```

```
//=>当在判断的操作中,很多条件都是符合的,执行完成第一个符合的条件后,后面的条件不管是否符合都不在处理了
var num = 10;
if (num <= 10) {
    num++;
} else if (num >= 5) {
    num--;
}
console.log(num);//=>11
```

三元运算符

语法：条件?条件成立执行:条件不成立执行;

三元运算符是对if(){}else{}这种简单判断处理的简写，即使不使用三元运算符,if else也能处理

```
var num = 10;
//------------------
// if (num >= 10) {
//     num++;
// } else {
//     num--;
// }

//------------------
num >= 10 ? num++ : num--;
console.log(num);//=>11
```

如果条件成立或者不成立的情况下，我们需要处理很多操作，那么需要把处理的事情用 `小括号` 包起来，每一个处理事情之间使用 `逗号` 分隔

```
var num = 10;
//------------------
// if (num >= 10) {
//     num++;
//     num = num * 10;
// } else {
//     num--;
//     num = num / 10;
// }

//------------------
num >= 10 ? (num++, num = num * 10) : (num--, num = num / 10);
console.log(num);//=>110
```

对于复杂的一些判断操作，使用 if else会更加的清晰明了，此时慎用三元运算符

```
var num = 10;
//num >= 0 ? (num <= 10 ? num++ : num--)
: (num <= -10 ? num++ : num--);

//->改写成 if else
if (num >= 0) {
    if (num <= 10) {
        num++
    } else {
        num--;
    }
} else {
    if (num <= -10) {
        num++;
    } else {
        num--;
    }
}
```

> 如果条件成立或者不成立的情况下，我们不需要做任何事情，可以在三元运算符中使用null或者void 0(undefined)来占位即可

```
var num = 10;
// if (num == 10) {
//     num++;
// }

//num==10?num++:; //=>Uncaught SyntaxError: Unexpected token ;

num == 10 ? num++ : null;//=>放undefined也可以(void 0)
num == 10 ? num++ : void 0;
```

switch case

> 也是if else某种特定情况的简写(另外一种写法)

```
switch(变量或者值){
    case [value1]:
        如果switch中的变量或者值和当前case后面的值相等，则执行这些操作;
        break; //=>每一种case情况结束都需要加break，达到条件成立处理完成，跳出当前判断
    ...
    default:
        以上值都不满足，执行这里的操作，最后一个就不需要加break
}
```

案例剖析

```
var num = 10;
// if (num == 0) {
//     num++;
// } else if (num == 5) {
//     num--;
// } else if (num == 10) {
//     num = num * num;
// } else {
//     num = 0;
// }

switch (num) {
    case 0:
        num++;
        break;
    case 5:
        num--;
        break;
    case 10:
        num *= num;
        break;
    default:
        num = 0;
}

console.log(num);
```

在switch case中我们可以利用case后面不加break的特点（不管后面条件是否成立都会继续执行，直到遇到break为止），实现当变量等于某几个值的时候，我们做相同的事情

```
var num = 5;
switch (num) {
    case 0:
        num++;
        break;
    case 5:
    case 10:
        num *= num; //=>只要当前的NUM等于5或者等10都去做同样的事情
        break;
    default:
        num = 0;
}
console.log(num);
```

每一种case情况的比较都是使用`===`进行比较的：绝对相等

`=`：赋值，变量`=`值
`==`：比较，值`==`值
`===`：绝对比较，值`===`

如果左右两边比较的值是相同类型的，那么直接比较内容是否一样即可；如果两边值的类型不一样，`==`和`===`是有区别的：
`===`类型不一样，最后的结果就是false，更加的严谨
`==`类型不一样，浏览器首先会默认的把类型转化为一样的，然后再比较内容，相对松散一些

```
1. '10'==10: true   浏览器会把'10'->10然后再比较
2. '10'===10: false
```

数学运算中的一些细节知识点
在JS中`* / -`都是数学运算，遇到非数字操作，浏览器也会转换为数字进行操作；但是`+`不一定是数学运算，如果遇到了字符串，属于字符串的拼接

```
1. '10'*10 =>100   浏览器会把字符串转为数字然后再
   运算（数学）
2. '10'/10 =>1
3. '10'-10 =>0
4. '10'+10 =>'1010'   字符串拼接
```

腾讯面试题

```
var result=10+null+[]+undefined+'zhufeng'+null+[]+undefined;
console.log(result); =>'10undefinedzhufengnullundefined'

/*
 10+null：数学运算,先把null变为数字0 ->10
 10+[]：数学运算,先把[]变为数字,[].toString()变为'',10+'',变为字符串拼接 ->'10'
 '10'+undefined：字符串拼接 ->'10undefined'
 '10undefined'+'zhufeng'：字符串拼接 ->'10undefinedzhufeng'
 '10undefinedzhufeng'+null：字符串拼接 ->'10undefinedzhufengnull'
 ...
 */
```

```
var result = 10+false+true+null+undefined+null+'zhufeng'+null+true+undefined;
=>'NaNzhufengnulltrueundefined'

//->分析步骤
10+false ->10
10+true ->11
11+null ->11
11+undefined ->NaN
NaN+null ->NaN
NaN+'zhufeng' ->'NaNzhufeng'
'NaNzhufeng'+null ->'NaNzhufengnull'
'NaNzhufengnull'+true ->'NaNzhufengnulltrue'
'NaNzhufengnulltrue'+undefined ->'NaNzhufengnulltrueundefined'
```

JS中的循环语句

> 重复做相同的事情就是循环：在真实项目中只要我们想重复做一件事件就要用到循环
>
> - for循环
> - for in 循环
> - while循环
> - do while循环
> - ...

`for循环`

```
for(设置初始值;设置循环执行的条件;步长累加){
    //->条件成立，执行循环体中的内容(循环体中存放的就是我们需要重复处理的事情)
}
第一步：设置初始值
第二步：验证(设置)循环能够执行的条件
第三步：条件成立，执行循环体中的内容，不成立直接结束循环
第四步：每一次执行完成循环体中内容，为了能够执行下一次的循环，做一下步长的累加
```

案例剖析

```
for(var i=0;i<5;){
    console.log(i);
}
//=>上述代码是死循环：i永远是0，条件永远成立，循环会一直执行下去
```

```
for(var i=0;i<5;i++){
    console.log(i);//-> 0 1 2 3 4
}
console.log(i);//->循环结束执行这个操作 =>5
```

```
for(var i=0;i<5;i+=3){
    console.log(i);//=> 0 3
}
console.log(i);//=> 6
```

```
for(var i=1;i<=5;i+=2){
    i<3?i++:i--;
    console.log(i);//=>2 3 4
}
console.log(i);//=>6
```

在循环的循环体中经常会出现两个关键词：
continue：结束当前本轮循环，继续执行下一轮循环
break：结束整个循环

所谓结束本轮循环：其实就是让循环体中continue后面的代码不在执行，直接的去进行步长累加，开启下一轮的循环
所谓结束整个循环：其实就是当循环体中遇到break，break后面的操作语句都不在执行，步长累加也不再执行，所有和循环有关的都结束了

```
for(var i=0;i<5;i++){
    continue;
    i+=2;
}
console.log(i); //=>5
```

```
for(var i=0;i<5;i++){
    break;
    i+=2;
}
console.log(i); //=>0
```

腾讯面试题

```
for(var i=0;i<10;i+=2){
    if(i<=5){
        i++;
        continue;
    }else{
        i--;
        break;
    }
    console.log(i);
}
console.log(i); //=>5
```

实战案例