JS 基本语法

《JavaScript 教程》作者：阮一峰

内容总结：

1. 语句以分号结尾；表达式一定返回值。
2. 变量名区分大小写，变量提升，动态类型，多次声明；变量赋值?
3. 合法标识符，以 `$_A-Za-z` 开头。
4. 单行注释；多行注释。
5. 代码块；选择语句三个；循环语句三个；break，continue只针对当前循环。
6. 标签，可用于改变代码结构，跳出多层循环。

---

1 语句

JavaScript 程序是按行执行的。一般情况下，一行就是一个语句。

语句 以分号结尾，一个分号就表示一个语句结束。

例如：

```
var a = 1 + 3;
```

这条赋值语句先用var命令，声明了变量a，然后将表达式 `1 + 3` 的运算结果赋值给变量a。

多个语句可以写在一行内。例如，

```
var a = 1 + 3 ; var b = 'abc';
```

语句和表达式的区别：

1. 语句一般情况下不返回值；表达式则一定会返回值。
2. 语句以分号结尾；表达式不需要以分号结尾。

2 变量

1. 概念

   变量就是给 “值” 起的名字，引用这个名字，就等同于引用这个值。

注意，JavaScript 的变量名区分大小写。

变量的声明和赋值，对于 JS 引擎来说，是分开的两个步骤，但用的时候一般使用声明并赋值的方式：

```
var a = 1; // 声明并赋值

var b; // 声明
b = 1; // 赋值
```

声明

JavaScript 可以同时声明多个变量。

```
var a, b;
```

1. 如果只是声明变量而没有赋值，则该变量的值是 undefined。

```
var a;
a
// undefined
```

2. 如果一个变量没有声明就直接使用，JavaScript 会报错，告诉你变量未定义。

```
x
// ReferenceError: x is not defined
```

3. 使用var重新声明一个已经存在的变量，是无效的。

```
var x = 1;
var x;
x
// 1
```

4. 如果第二次声明的时候还进行了赋值，则会覆盖掉前面的值。

```
var x = 1;
var x = 2;

x
// 2
```

赋值

JavaScript 是一种动态类型语言，也就是说，变量可以随时更改类型。

```
var a = 1;
a = 'hello';
```

一般使用变量需要声明并且赋值。但是在函数内，可以不声明直接赋值，这样的变量是全局变量。

2. 变量提升

JavaScript 引擎解析代码的时候，先获取所有被声明的变量，然后再一行一行地运行。这会造成所有变量的声明语句，都被提升到代码的头部，即变量提升（hoisting）。

例如，使用 var 声明的全局变量：

```
console.log(a);
var a = 1;
```

由 JavaScript 引擎进行变量提升后，变成：

```
var a;
console.log(a);
a = 1;
```

因此，上述代码在执行时不会报错。控制台将输出 `undefined` 。

又如，使用 var 声明的函数内变量：

```
function foo() {
       console.log(a);
       if(1===2){
           var a = '小马哥';
       }
   }
foo();
// undefined
```

在函数 foo 中，尽管变量 a 的声明在 if 语句中，也会被提升，函数 foo 相当于：

```
function foo() {
               var a;
       console.log(a);
       if(1===2){
                       a = '小马哥';
       }
   }
```

所以调用 foo 函数，控制台将输出 `undefined` 。

注意， 函数内使用 var 声明的变量，只在函数执行期间存在。

函数内的全局变量

```
function foo() {
       console.log(a);
```

```
        if(1===2){
                        a = '小马哥'; // 此处 a 为全局变量
        }
    }
foo();
```

运行上述代码，控制台会报错：`Uncaught ReferenceError: a is not defined`。说明函数内定义的全局变量不存在变量提升。

3 标识符

标识符（identifier）指的是用来识别各种值的合法名称。最常见的标识符就是变量名和函数名。

标识符有一套命名规则，不符合规则的就是非法标识符。JavaScript 引擎遇到非法标识符，就会报错。

```
// 合法的变量名
arg0
_tmp
$elem
```

```
// 非法的变量名
1a  // 第一个字符不能是数字
23  // 同上
***  // 标识符不能包含星号
a+b  // 标识符不能包含加号
-d  // 标识符不能包含减号或连词线
```

标识符命名规则如下：

- 第一个字符，可以是任意 Unicode 字母（包括英文字母和其他语言的字母），以及美元符号（$）和下划线（_）。
- 第二个字符及后面的字符，除了 Unicode 字母、美元符号和下划线，还可以用数字0-9。

  JavaScript 有一些保留字，不能用作标识符。

4 注释

单行注释:

```
// Support: Firefox 64+, Edge 18+
// Some browsers don't support the "nonce" property on scripts.
```

多行注释:

```
/*!
 * jQuery JavaScript Library v3.5.1
```

```
 * https://jquery.com/
 *
 * Includes Sizzle.js
 * https://sizzlejs.com/
 *
 * Copyright JS Foundation and other contributors
 * Released under the MIT license
 * https://jquery.org/license
 *
 * Date: 2020-05-04T22:49Z
 */
```

5 代码块

JavaScript 使用大括号，将多个相关的语句组合在一起，称为“区块”（block）。

```
{
module.exports = global.document ?
        factory( global, true ) :
        function( w ) {
                if ( !w.document ) {
                        throw new Error( "jQuery requires ..." );
                }
                return factory( w );
        };
}
```

6 条件语句

1. if 语句

```
if (exp)
  statement;

// 或者

if (exp) {
       statement;
} else if (exp) {
       statement;
} else {
       statement;
}
```

2. switch 语句

```
switch (fruit) {
  case "banana":
       // ...
       break;
  case "apple":
```

```
        // ...
        break;
  default:
        // ...
}
```

将变量 fruit 的值与 case 进行比较。如果所有case都不符合，则执行最后的default部分。

需要注意的是：

1. 每个case代码块内部的break语句不能少，否则会接下去执行下一个case代码块，而不是跳出switch结构。
2. switch语句内部采用的是“严格相等运算符”（===），这意味着switch语句后面的表达式，与case语句后面的表示式比较运行结果时，不会发生类型转换。

3. 三元运算符 ？：

```
(条件) ? 表达式1 : 表达式2
```

如果 `条件` 为 `true`，则返回 `表达式1` 的值，否则返回 `表达式2` 的值。

7 循环语句

1. while 循环

```
while (条件)
  语句;

// 或者

while (条件) {
  语句;
}
```

2. for 循环

```
for (初始化表达式; 条件; 递增表达式)
  语句

// 或者

for (初始化表达式; 条件; 递增表达式) {
  语句
}
```

for语句的三个部分（initialize、test、increment），可以省略任何一个，也可以全部省略。

3. do...while 循环

```
do
  语句
while (条件);

// 或者
do {
  语句
} while (条件);
```

不管条件是否为真，do...while循环至少运行一次，这是这种结构最大的特点。另外，while语句后面的分号注意不要省略。

4. break 语句和 continue 语句

   break 用于跳出循环；continue 用于跳到下一轮循环。

   for 循环也可以使用 break 语句。

```
for (var i = 0; i < 5; i++) {
  console.log(i);
  if (i === 3)
        break;
}
// 0
// 1
// 2
// 3
```

   如果存在多重循环，不带参数的break语句和continue语句都只针对最内层循环。

5. 标签

   JavaScript 语言允许，语句的前面有标签（label），相当于定位符，用于跳转到程序的任意位置。

```
label:
       语句
```

   跳出双层循环

   标签通常与 break 语句和 continue 语句配合使用，跳出特定的循环。

```
top:
  for (var i = 0; i < 3; i++){
       for (var j = 0; j < 3; j++){
         if (i === 1 && j === 1) break top;
         console.log('i=' + i + ', j=' + j);
       }
  }
// i=0, j=0
// i=0, j=1
```

```
// i=0, j=2
// i=1, j=0
```

上面代码为一个双重循环区块，break命令后面加上了top标签，满足条件时，直接跳出双层循环。

跳出代码块

```
foo: {
  console.log(1);
  break foo;
  console.log('本行不会输出');
}
console.log(2);
// 1
// 2
```

进入下一轮外层循环

continue语句也可以与标签配合使用。

```
top:
  for (var i = 0; i < 3; i++){
       for (var j = 0; j < 3; j++){
         if (i === 1 && j === 1) continue top;
         console.log('i=' + i + ', j=' + j);
       }
  }
// i=0, j=0
// i=0, j=1
// i=0, j=2
// i=1, j=0
// i=2, j=0
// i=2, j=1
// i=2, j=2
```

上面代码中，continue命令后面有一个标签名，满足条件时，会跳过当前循环，直接进入下一轮外层循环。