# 基础语法

## **语句**

JavaScript 程序的执行单位为行（line），也就是一行一行地执行。一般情况下，每一行就是一个语句。

语句（statement）是为了完成某种任务而进行的操作，比如下面就是一行赋值语句。

var a = 1 + 3;

这条语句先用var命令，声明了变量a，然后将1 + 3的运算结果赋值给变量a。

1 + 3叫做表达式（expression），指一个为了得到返回值的计算式。语句和表达式的区别在于，前者主要为了进行某种操作，一般情况下不需要返回值；后者则是为了得到返回值，一定会返回一个值。凡是 JavaScript 语言中预期为值的地方，都可以使用表达式。比如，赋值语句的等号右边，预期是一个值，因此可以放置各种表达式。

语句以分号结尾，一个分号就表示一个语句结束。多个语句可以写在一行内。

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

## **变量**

### **2.1 概念**

变量是对“值”的具名引用。变量就是为“值”起名，然后引用这个名字，就等同于引用这个值。变量的名字就是变量名。

var a = 1;

上面的代码先声明变量a，然后在变量a与数值1之间建立引用关系，称为将数值1“赋值”给变量a。以后，引用变量名a就会得到数值1。最前面的var，是变量声明命令。它表示通知解释引擎，要创建一个变量a。

注意，JavaScript 的变量名区分大小写，A和a是两个不同的变量。

变量的声明和赋值，是分开的两个步骤，上面的代码将它们合在了一起，实际的步骤是下面这样。

var a;

a = 1;

如果只是声明变量而没有赋值，则该变量的值是undefined。undefined是一个特殊的值，表示“无定义”。

var a;

a // undefined

如果变量赋值的时候，忘了写var命令，这条语句也是有效的。

var a = 1;// 基本等同

a = 1;

但是，不写var的做法，不利于表达意图，而且容易不知不觉地创建全局变量，所以建议总是使用var命令声明变量。

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

上面代码直接使用变量x，系统就报错，告诉你变量x没有声明。

可以在同一条var命令中声明多个变量。

var a, b;

JavaScript 是一种动态类型语言，也就是说，变量的类型没有限制，变量可以随时更改类型。

var a = 1;

a = 'hello';

上面代码中，变量a起先被赋值为一个数值，后来又被重新赋值为一个字符串。第二次赋值的时候，因为变量a已经存在，所以不需要使用var命令。

如果使用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

上面代码中，变量x声明了两次，第二次声明是无效的。

但是，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;

// 等同于

var x = 1;var x;

x = 2;

### **2.2 变量提升**

JavaScript 引擎的工作方式是，先解析代码，获取所有被声明的变量，然后再一行一行地运行。这造成的结果，就是所有的变量的声明语句，都会被提升到代码的头部，这就叫做变量提升（hoisting）。

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

上面代码首先使用console.log方法，在控制台（console）显示变量a的值。这时变量a还没有声明和赋值，所以这是一种错误的做法，但是实际上不会报错。因为存在变量提升，真正运行的是下面的代码。

var a;console.log(a);

a = 1;

最后的结果是显示undefined，表示变量a已声明，但还未赋值。

## **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，以及后面要提到的函数名。JavaScript 语言的标识符对大小写敏感，所以a和A是两个不同的标识符。

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。

## **区块**

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

对于var命令来说，JavaScript 的区块不构成单独的作用域（scope）。

{

var a = 1;

}

a // 1

上面代码在区块内部，使用var命令声明并赋值了变量a，然后在区块外部，变量a依然有效，区块对于var命令不构成单独的作用域，与不使用区块的情况没有任何区别。在 JavaScript 语言中，单独使用区块并不常见，区块往往用来构成其他更复杂的语法结构，比如for、if、while、function等。

## **条件语句**

JavaScript 提供if结构和switch结构，完成条件判断，即只有满足预设的条件，才会执行相应的语句。

### **6.1 if 结构**

if结构先判断一个表达式的布尔值，然后根据布尔值的真伪，执行不同的语句。所谓布尔值，指的是 JavaScript 的两个特殊值，true表示真，false表示伪。

if (布尔值)

语句;

// 或者

if (布尔值) 语句;

上面是if结构的基本形式。需要注意的是，“布尔值”往往由一个条件表达式产生的，必须放在圆括号中，表示对表达式求值。如果表达式的求值结果为true，就执行紧跟在后面的语句；如果结果为false，则跳过紧跟在后面的语句。

if (m === 3)

m = m + 1;

上面代码表示，只有在m等于3时，才会将其值加上1。

这种写法要求条件表达式后面只能有一个语句。如果想执行多个语句，必须在if的条件判断之后，加上大括号，表示代码块（多个语句合并成一个语句）。

if (m === 3) {

m += 1;

}

建议总是在if语句中使用大括号，因为这样方便插入语句。

注意，if后面的表达式之中，不要混淆赋值表达式（=）、严格相等运算符（===）和相等运算符（==）。尤其是赋值表达式不具有比较作用。

var x = 1;

var y = 2;

if (x = y) {

console.log(x);

}// "2"

上面代码的原意是，当x等于y的时候，才执行相关语句。但是，不小心将严格相等运算符写成赋值表达式，结果变成了将y赋值给变量x，再判断变量x的值（等于2）的布尔值（结果为true）。

这种错误可以正常生成一个布尔值，因而不会报错。为了避免这种情况，有些开发者习惯将常量写在运算符的左边，这样的话，一旦不小心将相等运算符写成赋值运算符，就会报错，因为常量不能被赋值。

if (x = 2) { // 不报错

if (2 = x) { // 报错

至于为什么优先采用“严格相等运算符”（===），而不是“相等运算符”（==），请参考《运算符》章节。

### **6.2 if...else 结构**

if代码块后面，还可以跟一个else代码块，表示不满足条件时，所要执行的代码。

if (m === 3) {

// 满足条件时，执行的语句

} else {

// 不满足条件时，执行的语句

}

上面代码判断变量m是否等于3，如果等于就执行if代码块，否则执行else代码块。

对同一个变量进行多次判断时，多个if...else语句可以连写在一起。

if (m === 0) {

// ...

} else if (m === 1) {

// ...

} else if (m === 2) {

// ...

} else {

// ...

}

else代码块总是与离自己最近的那个if语句配对。

var m = 1;

var n = 2;

if (m !== 1)

if (n === 2)

console.log('hello');

else console.log('world');

上面代码不会有任何输出，else代码块不会得到执行，因为它跟着的是最近的那个if语句，相当于下面这样。

if (m !== 1) {

if (n === 2) {

console.log('hello');

} else {

console.log('world');

}

}

如果想让else代码块跟随最上面的那个if语句，就要改变大括号的位置。

if (m !== 1) {

if (n === 2) {

console.log('hello');

}

} else {

console.log('world');

}// world

### **6.3 switch 结构**

多个if...else连在一起使用的时候，可以转为使用更方便的switch结构。

switch (fruit) {

case "banana":

// ...

break;

case "apple":

// ...

break;

default:

// ...

}

上面代码根据变量fruit的值，选择执行相应的case。如果所有case都不符合，则执行最后的default部分。需要注意的是，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。

var x = 1;

switch (x) {

case 1:

console.log('x 等于1');

case 2:

console.log('x 等于2');

default:

console.log('x 等于其他值');

}// x等于1// x等于2// x等于其他值

上面代码中，case代码块之中没有break语句，导致不会跳出switch结构，而会一直执行下去。正确的写法是像下面这样。

switch (x) {

case 1:

console.log('x 等于1');

break;

case 2:

console.log('x 等于2');

break;

default:

console.log('x 等于其他值');

}

switch语句部分和case语句部分，都可以使用表达式。

switch (1 + 3) {

case 2 + 2:

f();

break;

default:

neverHappens();

}

上面代码的default部分，是永远不会执行到的。

需要注意的是，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，采用的是严格相等运算符（===），而不是相等运算符（==），这意味着比较时不会发生类型转换。

var x = 1;

switch (x) {

case true:

console.log('x 发生类型转换');

break;

default:

console.log('x 没有发生类型转换');

}// x 没有发生类型转换

上面代码中，由于变量x没有发生类型转换，所以不会执行case true的情况。这表明，switch语句内部采用的是“严格相等运算符”，详细解释请参考《运算符》一节。

### **6.4 三元运算符 ?:**

JavaScript 还有一个三元运算符（即该运算符需要三个运算子）?:，也可以用于逻辑判断。

(条件) ? 表达式1 : 表达式2

上面代码中，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。

var even = (n % 2 === 0) ? true : false;

上面代码中，如果n可以被2整除，则even等于true，否则等于false。它等同于下面的形式。

var even;

if (n % 2 === 0) {

even = true;

} else {

even = false;

}

这个三元运算符可以被视为if...else...的简写形式，因此可以用于多种场合。

var myVar;

console.log(

myVar ?

'myVar has a value' :

'myVar does not have a value'

)// myVar does not have a value

上面代码利用三元运算符，输出相应的提示。

var msg = '数字' + n + '是' + (n % 2 === 0 ? '偶数' : '奇数');

上面代码利用三元运算符，在字符串之中插入不同的值。

## **循环语句**

循环语句用于重复执行某个操作，它有多种形式。

### **7.1 while 循环**

While语句包括一个循环条件和一段代码块，只要条件为真，就不断循环执行代码块。

while (条件)

语句;

// 或者

while (条件) 语句;

while语句的循环条件是一个表达式，必须放在圆括号中。代码块部分，如果只有一条语句，可以省略大括号，否则就必须加上大括号。

while (条件) {

语句;

}

下面是while语句的一个例子。

var i = 0;

while (i < 100) {

console.log('i 当前为：' + i);

i = i + 1;

}

上面的代码将循环100次，直到i等于100为止。

下面的例子是一个无限循环，因为循环条件总是为真。

while (true) {

console.log('Hello, world');

}

### **7.2 for 循环**

for语句是循环命令的另一种形式，可以指定循环的起点、终点和终止条件。它的格式如下。

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语句后面的括号里面，有三个表达式。

* 初始化表达式（initialize）：确定循环变量的初始值，只在循环开始时执行一次。
* 条件表达式（test）：每轮循环开始时，都要执行这个条件表达式，只有值为真，才继续进行循环。
* 递增表达式（increment）：每轮循环的最后一个操作，通常用来递增循环变量。

下面是一个例子。

var x = 3;

for (var i = 0; i < x; i++) {

console.log(i);

}// 0// 1// 2

上面代码中，初始化表达式是var i = 0，即初始化一个变量i；测试表达式是i < x，即只要i小于x，就会执行循环；递增表达式是i++，即每次循环结束后，i增大1。

所有for循环，都可以改写成while循环。上面的例子改为while循环，代码如下。

var x = 3;

var i = 0;

while (i < x) {

console.log(i);

i++;

}

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。

for ( ; ; ){

console.log('Hello World');

}

上面代码省略了for语句表达式的三个部分，结果就导致了一个无限循环。

### **7.3 do...while 循环**

do...while循环与while循环类似，唯一的区别就是先运行一次循环体，然后判断循环条件。

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。

下面是一个例子。

var x = 3;

var i = 0;

do {

console.log(i);

i++;

} while(i < x);

### **7.4 break 语句和 continue 语句**

break语句和continue语句都具有跳转作用，可以让代码不按既有的顺序执行。

break语句用于跳出代码块或循环。

var i = 0;

while(i < 100) {

console.log('i 当前为：' + i);

i++;

if (i === 10) break;

}

上面代码只会执行10次循环，一旦i等于10，就会跳出循环。

for循环也可以使用break语句跳出循环。

for (var i = 0; i < 5; i++) {

console.log(i);

if (i === 3)

break;

}// 0// 1// 2// 3

上面代码执行到i等于3，就会跳出循环。

continue语句用于立即终止本轮循环，返回循环结构的头部，开始下一轮循环。

var i = 0;

while (i < 100){

i++;

if (i % 2 === 0) continue;

console.log('i 当前为：' + i);

}

上面代码只有在i为奇数时，才会输出i的值。如果i为偶数，则直接进入下一轮循环。

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。