JS快速入门

1

JS快速入门

JS快速入门

JavaScript代码可以直接嵌在网页的任何地方，不过通常我们都把JavaScript代码放到<head>中：由<script>...</script>包含的代码就是JavaScript代码，它将直接被浏览器执行。

```
<html>
<head>
  <script>
    alert('Hello, world');
  </script>
</head>
<body>
  ...
</body>
</html>
```

第二种方法是把JavaScript代码放到一个单独的.js文件，然后在HTML中通过<script src="..."></script>引入这个文件：这样，/static/js/abc.js就会被浏览器执行。

把JavaScript代码放入一个单独的.js文件中更利于维护代码，并且多个页面可以各自引用同一份.js文件。可以在同一个页面中引入多个.js文件，还可以在页面中多次编写<script> js代码... </script>，浏览器按照顺序依次执行。

```
<html>
<head>
  <script src="/static/js/abc.js"></script>
</head>
<body>
  ...
</body>
</html>
```

JS快速入门

如何编写JavaScript

可以用任何文本编辑器来编写JavaScript代码。这里我们推荐以下几种文本编辑器：

- **Visual Studio Code**

微软出的Visual Studio Code，可以看做迷你版Visual Studio，免费！跨平台！内置JavaScript支持，强烈推荐使用！

- **Sublime Text**

Sublime Text是一个好用的文本编辑器，免费，但不注册会不定时弹出提示框。

- **Notepad++**

Notepad++也是免费的文本编辑器，但仅限Windows下使用。

- **WebStorm**

WebStorm 是jetbrains公司旗下一款JavaScript 开发工具。目前已经被广大中国JS开发者誉为“Web前端开发神器”、“最强大的HTML5编辑器”

注意：不可以用Word或写字板来编写JavaScript或HTML，因为带格式的文本保存后不是纯文本文件，无法被浏览器正常读取。也尽量不要用记事本编写，它会自作聪明地在保存UTF-8格式文本时添加BOM头。

如何运行JavaScript

要让浏览器运行JavaScript，必须先有一个HTML页面，在HTML页面中引入JavaScript，然后，让浏览器加载该HTML页面，就可以执行JavaScript代码。

你也许会想，直接在我的硬盘上创建好HTML和JavaScript文件，然后用浏览器打开，不就可以看到效果了吗?

这种方式运行部分JavaScript代码没有问题，但由于浏览器的安全限制，以file://开头的地址无法执行如联网等JavaScript代码，最终，你还是需要架设一个Web服务器，然后以http://开头的地址来正常执行所有JavaScript代码。

调试

俗话说得好，“工欲善其事，必先利其器。”，写JavaScript的时候，如果期望显示ABC，结果却显示XYZ，到底代码哪里出了问题？不要抓狂，也不要泄气，作为小白，要坚信：JavaScript本身没有问题，浏览器执行也没有问题，有问题的一定是我的代码。

如何找出问题代码？这就需要调试。

怎么在浏览器中调试JavaScript代码呢？

首先，你需要安装Google Chrome浏览器，Chrome浏览器对开发者非常友好，可以让你方便地调试JavaScript代码。从这里下载Chrome浏览器。打开网页出问题的童鞋请移步国内镜像。

安装后，随便打开一个网页，然后点击菜单“查看(View)”-“开发者(Developer)”-“开发者工具(Developer Tools)”，浏览器窗口就会一分为二，下方就是开发者工具：

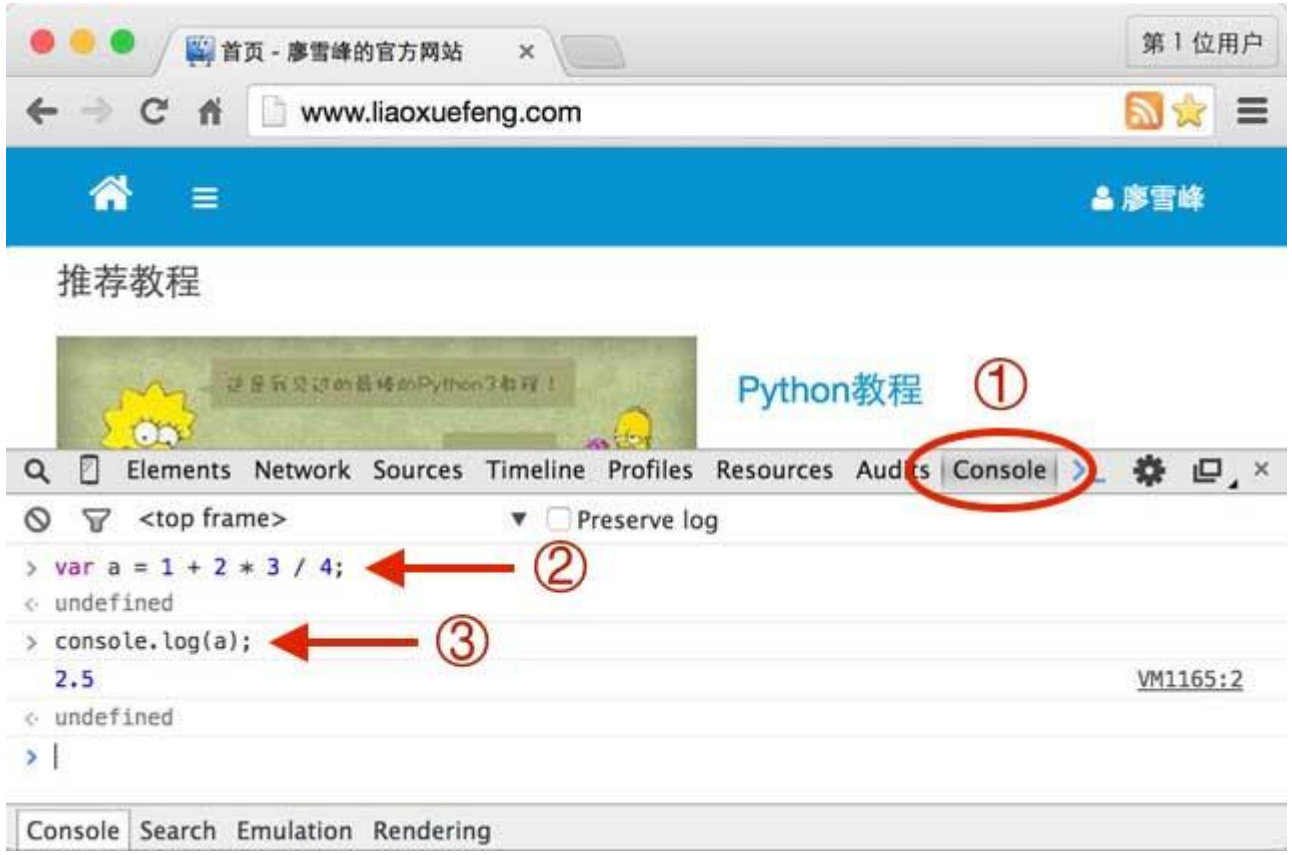
JS快速入门

先点击“控制台(Console)”，在这个面板里可以直接输入JavaScript代码，按回车后执行。

要查看一个变量的内容，在Console中输入console.log(a);，回车后显示的值就是变量的内容。

关闭Console请点击右上角的“×”按钮。请熟练掌握Console的使用方法，在编写JavaScript代码时，经常需要在Console运行测试代码。

如果你对自己还有更高的要求，可以研究开发者工具的“源码(Sources)”，掌握断点、单步执行等高级调试技巧。



2

基本语法

基本语法

语法

JavaScript的语法和Java语言类似，每个语句以;结束，语句块用{...}。但是，JavaScript并不强制要求在每个语句的结尾加;，浏览器中负责执行JavaScript代码的引擎会自动在每个语句的结尾补上;。

下面的一行代码包含两个语句，每个语句用;表示语句结束：

```
var x = 1; var y = 2; // 不建议一行写多个语句!
```

语句块是一组语句的集合，例如，下面的代码先做了一个判断，如果判断成立，将执行{...}中的所有语句：

```
if (2 > 1) {
    x = 1;
    y = 2;
    z = 3;
}
```

注意花括号{...}内的语句具有缩进，通常是4个空格。缩进不是JavaScript语法要求必须的，但缩进有助于我们理解代码的层次，所以编写代码时要遵守缩进规则。很多文本编辑器具有“自动缩进”的功能，可以帮助整理代码。

基本语法

注释

以//开头直到行末的字符被视为行注释，注释是给开发人员看到，JavaScript引擎会自动忽略：

```
// 这是一行注释
alert('hello'); // 这也是注释
```

另一种块注释是用/*...*/把多行字符包裹起来，把一大“块”视为一个注释：

```
/* 从这里开始是块注释
仍然是注释
仍然是注释
注释结束 */
```

3

数据类型和变量

数据类型和变量

计算机顾名思义就是可以做数学计算的机器，因此，计算机程序理所当然地可以处理各种数值。但是，计算机能处理的远不止数值，还可以处理文本、图形、音频、视频、网页等各种各样的数据，不同的数据，需要定义不同的数据类型。在JavaScript中定义了以下几种数据类型：

Number

JavaScript不区分整数和浮点数，统一用Number表示，以下都是合法的Number类型：

```
123; // 整数123
0.456; // 浮点数0.456
1.2345e3; // 科学计数法表示1.2345x1000，等同于1234.5
-99; // 负数
NaN; // NaN表示Not a Number，当无法计算结果时用NaN表示
Infinity; // Infinity表示无限大，当数值超过了JavaScript的Number所能表示的最大值时，就表示为Infinity
```

计算机由于使用二进制，所以，有时候用十六进制表示整数比较方便，十六进制用0x前缀和0-9，a-f表示，例如：0xff00，0xa5b4c3d2，等等，它们和十进制表示的数值完全一样。

数据类型和变量

字符串

字符串是以单引号'或双引号"括起来的任意文本，比如'abc'，"xyz"等等。请注意，''或""本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc'只有a，b，c这3个字符。

布尔值

布尔值和布尔代数的表示完全一致，一个布尔值只有true、false两种值，要么是true，要么是false，可以直接用true、false表示布尔值，也可以通过布尔运算计算出来：

```
true; // 这是一个true值
false; // 这是一个false值
2 > 1; // 这是一个true值
2 >= 3; // 这是一个false值
```

数据类型和变量

&&运算是与运算，只有所有都为true，&&运算结果才是true：

```
true && true; // 这个&&语句计算结果为true
true && false; // 这个&&语句计算结果为false
false && true && false; // 这个&&语句计算结果为false
```

||运算是或运算，只要其中有一个为true，||运算结果就是true：

```
false || false; // 这个||语句计算结果为false
true || false; // 这个||语句计算结果为true
false || true || false; // 这个||语句计算结果为true
```

!运算是非运算，它是一个单目运算符，把true变成false，false变成true：

```
! true; // 结果为false
! false; // 结果为true
! (2 > 5); // 结果为true
```

布尔值经常用在条件判断中，比如：

```
var age = 15;
if (age >= 18) {
    alert('adult');
} else {
    alert('teenager');
}
```

比较运算符

当我们对Number做比较时，可以通过比较运算符得到一个布尔值：

```
2 > 5; // false
5 >= 2; // true
7 == 7; // true
```

实际上，JavaScript允许对任意数据类型做比较：

数据类型和变量

```
false == 0; // true
false === 0; // false
```

要特别注意相等运算符==。JavaScript在设计时，有两种比较运算符：

第一种是==比较，它会自动转换数据类型再比较，很多时候，会得到非常诡异的结果；

第二种是===比较，它不会自动转换数据类型，如果数据类型不一致，返回false，如果一致，再比较。

由于JavaScript这个设计缺陷，不要使用==比较，始终坚持使用===比较。

另一个例外是NaN这个特殊的Number与所有其他值都不相等，包括它自己：

```
NaN === NaN; // false
```

唯一能判断NaN的方法是通过isNaN()函数：

```
isNaN(NaN); // true
```

null和undefined

null表示一个“空”的值，它和0以及空字符串''不同，0是一个数值，''表示长度为0的字符串，而null表示“空”。

在其他语言中，也有类似JavaScript的null的表示，例如Java也用null，Swift用nil，Python用None表示。但是，在JavaScript中，还有一个和null类似的undefined，它表示“未定义”。

JavaScript的设计者希望用null表示一个空的值，而undefined表示值未定义。事实证明，这并没有什么区别，区分两者的意义不大。大多数情况下，我们都应该用null。undefined仅仅在判断函数参数是否传递的情况下有用。

数据类型和变量

数组

数组是一组按顺序排列的集合，集合的每个值称为元素。JavaScript的数组可以包括任意数据类型。例如：

```
[1, 2, 3.14, 'Hello', null, true];
```

上述数组包含6个元素。数组用[]表示，元素之间用,分隔。另一种创建数组的方法是通过Array()函数实现：

```
new Array(1, 2, 3); // 创建了数组[1, 2, 3]
```

然而，出于代码的可读性考虑，强烈建议直接使用[]。

数组的元素可以通过索引来访问。请注意，索引的起始值为0：

```
var arr = [1, 2, 3.14, 'Hello', null, true];
arr[0]; // 返回索引为0的元素，即1
arr[5]; // 返回索引为5的元素，即true
arr[6]; // 索引超出了范围，返回undefined
```

数据类型和变量

对象

JavaScript的对象是一组由键-值组成的无序集合，例如：

```
var person = {
    name: 'Bob',
    age: 20,
    tags: ['js', 'web', 'mobile'],
    city: 'Beijing',
    hasCar: true,
    zipcode: null
};
```

JavaScript对象的键都是字符串类型，值可以是任意数据类型。上述person对象一共定义了6个键值对，其中每个键又称为对象的属性，例如，person的name属性为'Bob'，zipcode属性为null。

要获取一个对象的属性，我们用对象变量.属性名的方式：

```
person.name; // 'Bob'
person.zipcode; // null
```

数据类型和变量

变量

变量的概念基本上和初中代数的方程变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意数据类型。变量在JavaScript中就是用一个变量名表示，变量名是大小写英文、数字、$和_的组合，且不能用数字开头。变量名也不能是JavaScript的关键字，如if、while等。申明一个变量用var语句，比如：

```
var a; // 申明了变量a，此时a的值为undefined
var $b = 1; // 申明了变量$b，同时给$b赋值，此时$b的值为1
var s_007 = '007'; // s_007是一个字符串
var Answer = true; // Answer是一个布尔值true
var t = null; // t的值是null
```

变量名也可以用中文，但是，请不要给自己找麻烦。在JavaScript中，使用等号=对变量进行赋值。可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量，但是要注意只能用var申明一次，例如：

```
var a = 123; // a的值是整数123
a = 'ABC'; // a变为字符串
```

数据类型和变量

这种变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。静态语言在定义变量时必须指定变量类型，如果赋值的时候类型不匹配，就会报错。例如Java是静态语言，赋值语句如下：

```
int a = 123; // a是整数类型变量，类型用int申明
a = "ABC"; // 错误：不能把字符串赋给整型变量
```

和静态语言相比，动态语言更灵活，就是这个原因。

请不要把赋值语句的等号等同于数学的等号。比如下面的代码：

```
var x = 10;
x = x + 2;
```

如果从数学上理解x = x + 2那无论如何是不成立的，在程序中，赋值语句先计算右侧的表达式x + 2，得到结果12，再赋给变量x。由于x之前的值是10，重新赋值后，x的值变成12。

数据类型和变量

strict模式

JavaScript在设计之初，为了方便初学者学习，并不强制要求用var申明变量。这个设计错误带来了严重的后果：如果一个变量没有通过var申明就被使用，那么该变量就自动被申明为全局变量：

```
i = 10; // i现在是全局变量
```

在同一个页面的不同的JavaScript文件中，如果都不用var申明，恰好都使用了变量i，将造成变量i互相影响，产生难以调试的错误结果。

使用var申明的变量则不是全局变量，它的范围被限制在该变量被申明的函数体内（函数的概念将稍后讲解），同名变量在不同的函数体内互不冲突。为了修补JavaScript这一严重设计缺陷，ECMA在后续规范中推出了strict模式，在strict模式下运行的JavaScript代码，强制通过var申明变量，未使用var申明变量就使用的，将导致运行错误。

启用strict模式的方法是在JavaScript代码的第一行写上：

数据类型和变量

```
'use strict';
```

这是一个字符串，不支持strict模式的浏览器会把它当做一个字符串语句执行，支持strict模式的浏览器将开启strict模式运行JavaScript。语法检测更为严格。

4

字符串

数据类型和变量

字符串

JavaScript的字符串就是用''或""括起来的字符表示。

如果'本身也是一个字符，那就可以用""括起来，比如"I'm OK"包含的字符是I，'，m，空格，O，K这6个字符。

如果字符串内部既包含'又包含"怎么办？可以用转义字符\来标识，比如：

```
'I\'m \"OK\"!';
```

表示的字符串内容是：I'm "OK"!

转义字符\可以转义很多字符，比如\n表示换行，\t表示制表符，字符\本身也要转义，所以\\表示的字符就是\。

ASCII字符可以以\x##形式的十六进制表示，例如：

```
'\x41'; // 完全等同于 'A'
```

还可以用\u####表示一个Unicode字符：

```
'\u4e2d\u6587'; // 完全等同于'中文'
```

多行字符串

由于多行字符串用\n写起来比较费事，所以最新的ES6标准新增了一种多行字符串的表示方法，用反引号 ` ... ` 表示：

```
`这是一个
多行
字符串`;
```

数据类型和变量

模板字符串

要把多个字符串连接起来，可以用+号连接：

```
var name = '小明';
var age = 20;
var message = '你好, ' + name + ', 你今年' + age + '岁了!';
alert(message);
```

如果有很多变量需要连接，用+号就比较麻烦。ES6新增了一种模板字符串，表示方法和上面的多行字符串一样，但是它会自动替换字符串中的变量：

```
var name = '小明';
var age = 20;
var message = `你好, ${name}, 你今年${age}岁了!`;
alert(message);
```

数据类型和变量

操作字符串

字符串常见的操作如下：

```
var s = 'Hello, world!';
s.length; // 13
```

要获取字符串某个指定位置的字符，使用类似Array的下标操作，索引号从0开始：

```
var s = 'Hello, world!';

s[0]; // 'H'
s[6]; // ' '
s[7]; // 'w'
s[12]; // '!'
s[13]; // undefined 超出范围的索引不会报错，但一律返回undefined
```

需要特别注意的是，字符串是不可变的，如果对字符串的某个索引赋值，不会有任何错误，但是，也没有任何效果：

数据类型和变量

JavaScript为字符串提供了一些常用方法，注意，调用这些方法本身不会改变原有字符串的内容，而是返回一个新字符串：

toUpperCase

toUpperCase()把一个字符串全部变为大写：

```
var s = 'Hello';
s.toUpperCase(); // 返回'HELLO'
```

toLowerCase

toLowerCase()把一个字符串全部变为小写：

```
var s = 'Hello';
var lower = s.toLowerCase(); // 返回'hello'并赋值给变量lower
lower; // 'hello'
```

数据类型和变量

indexOf

indexOf()会搜索指定字符串出现的位置：

```
var s = 'hello, world';
s.indexOf('world'); // 返回7
s.indexOf('World'); // 没有找到指定的子串，返回-1
```

substring

substring()返回指定索引区间的子串：

```
var s = 'hello, world'
s.substring(0, 5); // 从索引0开始到5（不包括5），返回'hello'
s.substring(7); // 从索引7开始到结束，返回'world'
```

5

对象

数据类型和变量

JavaScript的对象是一种无序的集合数据类型，它由若干键值对组成。

JavaScript的对象用于描述现实世界中的某个对象。例如，为了描述“小明”这个淘气的小朋友，我们可以用若干键值对来描述他：

```
var xiaoming = {
    name: '小明',
    birth: 1990,
    school: 'No.1 Middle School',
    height: 1.70,
    weight: 65,
    score: null
};
```

JavaScript用一个{...}表示一个对象，键值对以xxx：xxx形式申明，用,隔开。注意，最后一个键值对不需要在末尾加,，如果加了，有的浏览器（如低版本的IE）将报错。

上述对象申明了一个name属性，值是'小明'，birth属性，值是1990，以及其他一些属性。最后，把这个对象赋值给变量xiaoming后，就可以通过变量xiaoming来获取小明的属性了：

数据类型和变量

```
xiaoming.name; // '小明'
xiaoming.birth; // 1990
```

访问属性是通过.操作符完成的，但这要求属性名必须是一个有效的变量名。如果属性名包含特殊字符，就必须用''括起来：

```
var xiaohong = {
    name: '小红',
    'middle-school': 'No.1 Middle School'
};
```

xiaohong的属性名middle-school不是一个有效的变量，就需要用''括起来。访问这个属性也无法使用.操作符，必须用['xxx']来访问：

```
xiaohong['middle-school']; // 'No.1 Middle School'
xiaohong['name']; // '小红'
xiaohong.name; // '小红'
```

数据类型和变量

也可以用**xiaohong['name']**来访问xiaohong的name属性，不过**xiaohong.name**的写法更简洁。我们在编写JavaScript代码的时候，属性名尽量使用标准的变量名，这样就可以直接通过object.prop的形式访问一个属性了。

实际上JavaScript对象的所有属性都是字符串，不过属性对应的值可以是任意数据类型。

如果访问一个不存在的属性会返回什么呢?

JavaScript规定，访问不存在的属性**不报错，而是返回undefined**

6

条件判断

条件判断

JavaScript使用if () { ... } else { ... }来进行条件判断。例如，根据年龄显示不同内容，可以用if语句实现如

```
var age = 20;
if (age >= 18) { // 如果age >= 18为true，则执行if语句块
    alert('adult');
} else { // 否则执行else语句块
    alert('teenager');
}
```

其中else语句是可选的。如果语句块只包含一条语句，那么可以省略{}：

```
var age = 20;
if (age >= 18)
    alert('adult');
else
    alert('teenager');
```

省略{}的危险之处在于，如果后来想添加一些语句，却忘了写{}，就改变了if...else...的语义，例如：

```
var age = 20;
if (age >= 18)
    alert('adult');
else
    console.log('age < 18'); // 添加一行日志
    alert('teenager'); // <- 这行语句已经不在else的控制范围了
```

条件判断

上述代码的else子句实际上只负责执行console.log('age < 18');，原有的alert('teenager');已经不属于if...else...的控制范围了，它每次都会执行。

相反地，有{}的语句就不会出错：

```
var age = 20;
if (age >= 18) {
    alert('adult');
} else {
    console.log('age < 18');
    alert('teenager');
}
```

这就是为什么我们建议永远都要写上{}。

多行条件判断

如果还要更细致地判断条件，可以使用多个if...else...的组合：

```
var age = 3;
if (age >= 18) {
    alert('adult');
} else if (age >= 6) {
    alert('teenager');
} else {
    alert('kid');
}
```

上述多个if...else...的组合实际上相当于两层if...else...：

```
var age = 3;
if (age >= 18) {
    alert('adult');
} else {
    if (age >= 6) {
        alert('teenager');
    } else {
        alert('kid');
    }
}
```

但是我们通常把else if连写在一起，来增加可读性。这里的else略掉了{}是没有问题的，因为它只包含一个if语句。注意最后一个单独的else不要略掉{}。

请注意，if...else...语句的执行特点是二选一，在多个if...else...语句中，如果某个条件成立，则后续就不再继续判断了。

7

循环

条件判断

要计算1+2+3，我们可以直接写表达式：

```
1 + 2 + 3; // 6
```

要计算1+2+3+...+10，勉强也能写出来。

但是，要计算1+2+3+...+10000，直接写表达式就不可能了。

为了让计算机能计算成千上万次的重复运算，我们就需要循环语句。

JavaScript的循环有两种，一种是for循环，通过初始条件、结束条件和递增条件来循环执行语句块：

```
var x = 0;
var i;
for (i=1; i<=10000; i++) {
    x = x + i;
}
x; // 50005000
```

让我们来分析一下for循环的控制条件：

i=1 这是初始条件，将变量i置为1；

条件判断

i<=10000 这是判断条件，满足时就继续循环，不满足就退出循环；

i++ 这是每次循环后的递增条件，由于每次循环后变量i都会加1，因此它终将在若干次循环后不满足判断条件i<=10000而退出循环。

for循环最常用的地方是利用索引来遍历数组：

```
var arr = ['Apple', 'Google', 'Microsoft'];
var i, x;
for (i=0; i<arr.length; i++) {
    x = arr[i];
    console.log(x);
}
```

for循环的3个条件都是可以省略的，如果没有退出循环的判断条件，就必须使用break语句退出循环，否则就是死循环：

```
var x = 0;
for (;;) { // 将无限循环下去
    if (x > 100) {
        break; // 通过if判断来退出循环
    }
    x ++;
}
```

条件判断

for ... in

for循环的一个变体是for ... in循环，它可以把一个对象的所有属性依次循环出来：

```
var o = {
    name: 'Jack',
    age: 20,
    city: 'Beijing'
};
for (var key in o) {
    console.log(key); // 'name', 'age', 'city'
}
```

要过滤掉对象继承的属性，用hasOwnProperty()来实现：

```
var o = {
    name: 'Jack',
    age: 20,
    city: 'Beijing'
};
for (var key in o) {
    if (o.hasOwnProperty(key)) {
        console.log(key); // 'name', 'age', 'city'
    }
}
```

请注意，for ... in对Array的循环得到的是String而不是Number。

条件判断

while

for循环在已知循环的初始和结束条件时非常有用。而上述忽略了条件的for循环容易让人看不清循环的逻辑，此时用while循环更佳。

while循环只有一个判断条件，条件满足，就不断循环，条件不满足时则退出循环。比如我们要计算100以内所有奇数之和，可以用while循环实现：

```
var x = 0;
var n = 99;
while (n > 0) {
    x = x + n;
    n = n - 2;
}
x; // 2500
```

在循环内部变量n不断自减，直到变为-1时，不再满足while条件，循环退出。

条件判断

do ... while

最后一种循环是do { ... } while()循环，它和while循环的唯一区别在于，不是在每次循环开始的时候判断条件，而是在每次循环完成的时候判断条件：

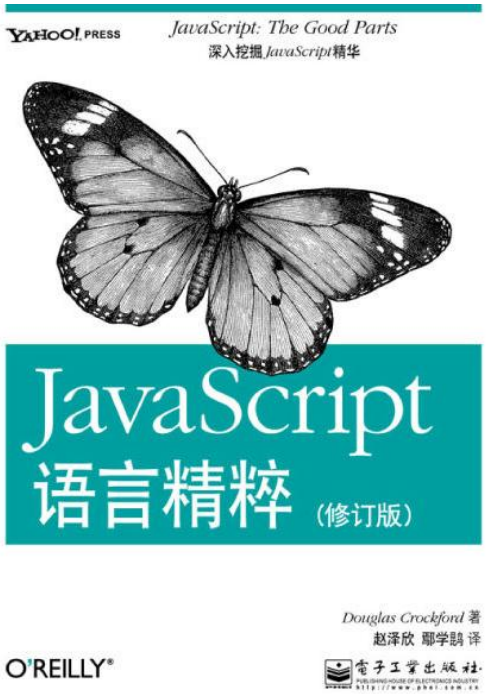
```
var n = 0;
do {
    n = n + 1;
} while (n < 100);
n; // 100
```

用do { ... } while()循环要小心，循环体会至少执行1次，而for和while循环则可能一次都不执行。

循环是让计算机做重复任务的有效的方法，有些时候，如果代码写得有问题，会让程序陷入“死循环”，也就是永远循环下去。JavaScript的死循环会让浏览器无法正常显示或执行当前页面的逻辑，有的浏览器会直接挂掉，有的浏览器会在一段时间后提示你强行终止JavaScript的执行，因此，要特别注意死循环的问题。

在编写循环代码时，务必小心编写初始条件和判断条件，尤其是边界值。特别注意i < 100和i <= 100是不同的判断逻辑。

推荐书籍



1. False

- false
- null
- undefined
- 空字符串 ' '
- 数字 0
- 数字 NaN

2. ||运算符可以用来填充默认值

var status = flight.status || 'unknown'

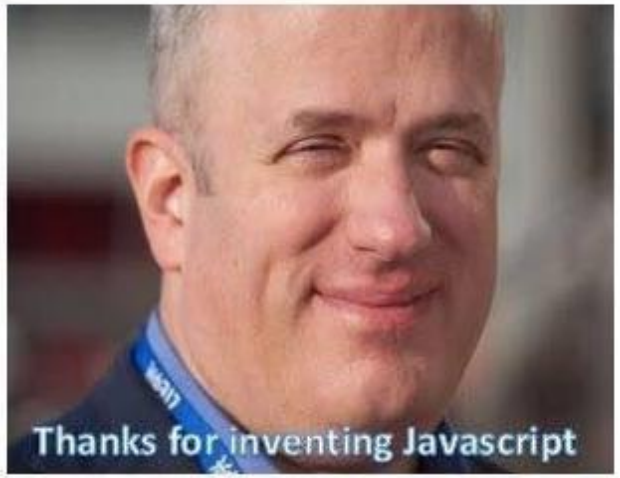
3. 函数调用模式

- this的使用

JS迷之微笑

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
```

```
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```



Q&A