# 标识符与关键字

## 标识符

1. JavaScript中的标识符由字母、数字、下划线和美元符号组成，不能以数字开头，也不能为关键字。
2. 标识符严格区分大小写。
3. 尽量做到见名知意。
4. 当标识符需要多个单词进行表示时，常见的表示方式有下划线法（如user\_name）、驼峰法（如userName）和帕斯卡法（如UserName）；可根据开发需求统一命名规范，如下换线通常应用于变量的命名，驼峰法通常应用于函数的命名等。

## 关键字

关键字是指被事先定义好并赋予特殊含义的单词，关键字不能作为标识符使用，否则会出现语法错误。JavaScript关键字如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| breka | case | catch | calss | const | continue |
| debugger | default | delete | do | else | export |
| extends | false | finally | for | function | if |
| import | in | instanceof | new | null | return |
| super | switch | this | throw | try | true |
| typeof | var | void | while | with | yield |

# 变量

变量用于存储数据。

## 声明变量

在JavaScript中，使用var关键字声明变量。其语法格式为：

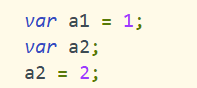
**var 变量名=变量值 ;**

1）未赋初始值的变量，默认值为undefined

2）如果需要一次声明多个表里，则变量之间使用“，”分隔

## 变量赋值

声明变量时可以直接够变量赋值，也可以在声明变量后再给变量赋值。如下：



## 使用常量

在JavaScript中，使用const关键字声明常量。其语法格式为：

**const 常量名=常量值 ;**

习惯上常量名称总是使用大写字母表示，且在声明常量时必须赋值，常量一旦声明便不可被修改。

# 数据类型

JavaScript中的数据类型分为基本数据类型和引用数据类型。

1. 基本数据类型：Boolean型、String型、Number型、Null型、Undefined型
2. 引用数据类型：Object型

## 基本数据类型

1. 布尔型

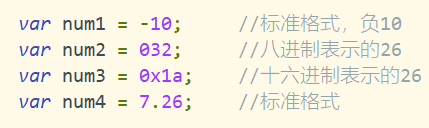
布尔型（Boolean）是JavaScript中常用的数据类型之一，通常用于逻辑判断，它只有true和false两个值，表示事物的“真”和“假”。如：



JavaScript严格遵循字母大小写，true和false必须全部为小写才表示布尔型。

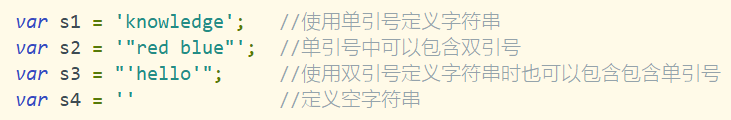
1. 数值型

数值型（Number）是最基本的数据类型，与其他程序语言不同，JavaScript中的数值型不区分整数和浮点数，所有数字都是数值型。在使用时它还可以添加“-”符号表示负数，添加“+”表示正数（通常忽略），或是设置成NaN表示非数值、以及其他进制的格式。如：



1. 字符型

字符型（String）是JavaScript用来表示文本的数据类型，JavaScript中的字符串使用单引号或双引号声明。如：



使用单引号定界的字符串中可以包含双引号，使用双引号定界的字符串可以包含单引号；但是如果要在单引号中使用单引号，或在双引号中使用双引号，则需要使用转义字符“\”对其进行转义。如：



除此之外，在字符串中使用换行、Tab等特殊符号时，也需要使用字符“\”进行转义。常用的需要转义的特殊字符如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 特殊字符 | 含义 | 特殊字符 | 含义 |
| \’ | 单引号 | \” | 双引号 |
| \n | 回车换行 | \v | 跳格（Tab、水平） |
| \t | Tab符号 | \r | 换行 |
| \f | 换页 | \\ | 反斜杠（\） |
| \b | 退格 | \0 | Null字节 |

1. 空型

空型（Null）只有一个特殊的null值，用于表示一个不存在的或无效的对象或地址。

并且只有小写的null才表示空型（Null）。

1. 未定义型

未定义型（Undefined）只有一个特殊的undefined值。声明的变量还未被初始化时

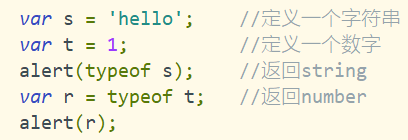
，变量的默认值为undefined。与null不同的是undefined表示没有为变量设置值，而null则表示变量（对象或地址）不存在或无效。需要注意的是，null和undefined与空字符串（’’）和数字0都不相等。

## 数据类型检测

在JavaScript中，可以使用使用typeof关键字检测一个变量属于什么数据类型。其语法格式为：

**typeof 变量 ;**

结果以字符串的形式返回该变量的数据类型。如：

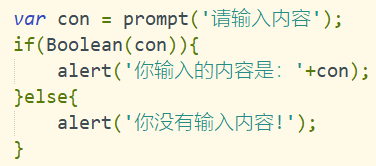


## 数据类型转换

对两个数据进行操作时，若其数据类型不相同，则需要对齐进行数据类型转换。除了可以利用JavaScript的自动转换外，还可以根据程序的需要具体制定数据的转换类型。

1. **转布尔型**

Boolean( )函数会将任何非空字符串和非零的数值转换为true，将空字符串、0、NaN、undefined和null转换为false。如：



当输入框弹出后，如果用户输入了内容，点击确定后，则转为true；如果用户没有输入内容，则转为false。因为输入了内容将返回输入的内容，此时Boolean( )函数会将非空字符串转为true；而如果没有输入内容则返回null，此时Boolean( )函数会将其转为false。

1. **转数值型**

在开发中经常需要将包含数字的字符串转换为能够进行算术运算的数字，为此，JavaScript提供了Number( )函数、parseInt( )函数、parseFloat( )函数对参与运算的数据进行转换，保证都是数值型数据。

|  |  |  |  |
| --- | --- | --- | --- |
| 待转数据 | Number( ) | parseInt( ) | parseFloat( ) |
| 纯数字字符串 | 转成相应的数字 | 转成相应的数字 | 转成相应的数字 |
| 空字符串 | 0 | NaN | NaN |
| 数字开头的字符串 | NaN | 转成开头的数字 | 转成开头的数字 |
| 非数字开头的字符串 | NaN | NaN | NaN |
| null | 0 | NaN | NaN |
| undefined | NaN | NaN | NaN |
| false | 0 | NaN | NaN |
| true | 1 | NaN | NaN |

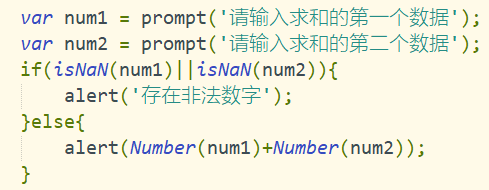
1）所有函数在转换纯数字时会忽略前导零，如字符串“0123”会被转换为123；

2）parseFloat( )函数会将数据转换为浮点数（可以理解为小数）；

3）parseInt( )函数会直接省略小数部分，返回数据的整数部分；

4）在实际开发中一般还需要对转换后的结果是否是NaN进行判断，只有不是NaN

时，才能够进行运算。可以利用JavaScript提供的isNaN( )函数来确定，当给定值是undefined、NaN或{}（对象）时返回true，否则返回false。若给定的数据不是数值类型时，isNaN函数首先会尝试将其转换为数值类型，再进行判断。



1. **转字符型**

在开发中，需要将数据转换成字符型时，可以利用JavaScript提供的String( )函数或toString( )函数进行转换，它们的区别是前者可以将任意类型转换为字符型；而后者除了null和undefined没有toString( )函数外，其他数据类型都可以完成字符的转换。toString( )方法还可以完成进制的转换，如下：



# 表达式

表达式可以是各种类型的数据、变量和运算符的集合。

# 运算符

## 算术运算符

算术运算符用于对数值类型的变量及常量进行算术运算。

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **运算** | **示例** | **结果** |
| + | 加 | 5+5 | 10 |
| - | 减 | 6-2 | 4 |
| \* | 乘 | 3\*4 | 12 |
| / | 除 | 3/2 | 1.5 |
| % | 求余 | 5%7 | 5 |
| \*\* | 幂运算（ES7新特性） | 2\*\*2 | 4 |
| ++ | 自增（前置） | a=2,b=++a; | a=3;b=3 |
| ++ | 自增（后置） | a=2,b=a--; | a=1;b=2; |
| -- | 自减（前置） | a=2,b=--a; | a=1;b=1; |
| -- | 自减（后置） | a=2,b=a--; | a=1;b=2; |

在JavaScript运算中，进行算术运算需要注意一下几点：

1. 进行四则运算时，运算顺序要遵循数学中“先乘除后加减”的原则；
2. 在进行取模运算时，运算结果的正负取决于被模数（%左边的数）的符号，与模数（%右边的数）的符号无关。例如，（-8）%7=-1，而8%（-7）=1；
3. 在开发中尽量避免利用小数进行运算，有时可能因JavaScript的精度导致结果的偏差。例如，1.66+1.77，我们理想中的值应该是3.43，但是JavaScript的计算结果却是3.429999999999997；此时，可以将参与运算的数据转换为整数，计算后再转换为小数即可。例如，将1.66和1.77分别乘以100，相加后再除以100，即可得到3.43；
4. “+”和“-”在算术运算时还可以表示正数和负数，例如：（+2.1）+（-1.2）的运算结果为1；
5. 递增和递减运算符仅对数值型和布尔型数据操作，操作时会将布尔值true当作为1，false当作0。

## 字符串运算符

1. JavaScript中，“+”操作的两个数据中只要有一个是字符串，则“+”就表示字符串运算符，用于返回两个数据拼接够的字符串。
2. 利用字符串运算符“+”的特性，可以将布尔型、数值型或为null的数据，与空字符串进行拼接，就会完成到字符串型的自动转换。

## 赋值运算符

赋值运算符用于将运算符右边的值赋给左边的变量。其中，“=”是最基本的赋值运算符，而非数学意义上的相等关系。常用的赋值运算符如下（包括了复合赋值运算符）：

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算 | 示例 | 结果 |
| = | 赋值 | a=3,b=2; | a=3;b=2; |
| += | 加并赋值 | a=3,b=2;a+=b; | a=5;b=2; |
| -+ | 减并赋值 | a=3,b=2;a-=b; | a=1;b=2; |
| \*= | 乘并赋值 | a=3,b=2;a\*=b; | a=6;b=2; |
| /= | 除并赋值 | a=3,b=2;a/=b; | a=1.5;b=2; |
| %= | 模并赋值 | a=3,b=2;a%=b; | a=1;b=2; |
| += | 连接并赋值 | a=’abc’;a+=’def’; | a=’abcdef’; |
| \*\*= | 幂运算并赋值 | a=2;a\*\*=5; | a=32; |
| <<= | 左移位赋值 | a=9,b=2;a<<=2; | a=36;b=2; |
| >>= | 右移位赋值 | a=-9,b=2;a>>=b; | a=-3;b=2; |
| >>>= | 无符号右移位赋值 | a=-9,b=2;a>>>=b; | a=1073741821;b=2 |
| &= | 按位与赋值 | a=3,b=9;a&=b | a=1;b=9; |
| ^= | 按位异或赋值 | a=3,b=9;a^=b; | a=10;b=9; |
| |= | 按位与赋值 | a=3,b=9;a|=b; | a=11;b=9; |

1. 表中关于位赋值运算符此处了解即可，位运算符具体如何运算会在后面进行详细讲解。
2. 使用“=”赋值运算符不仅可以为指定变量赋值，还可以利用一条赋值语句同时对多个变量进行赋值，具体示例如下：



## 比较运算符

比较运算符用于对两个数值或变量进行比较，其结果返回一个布尔值，即true或false。

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算 | 示例（x=5） | 结果 |
| > | 大于 | x>5 | false |
| >= | 大于或等于 | x>=5 | true |
| < | 小于 | x<5 | false |
| <= | 小于或等于 | x<=5 | true |
| == | 等于 | x==4 | false |
| != | 不等于 | x!=4 | true |
| === | 全等 | x===5 | true |
| !== | 不全等 | x!==’5’ | true |

1. 不相同类型的数据进行比较时，首先会自动将其转换成相同类型的数据后再进行比较，例如，字符‘123’与123比较时，首先会将字符串‘123’转换成数值型，然后再与123进行比较。
2. 运算符“==”和“！=”与运算符“===”和“！==”在进行比较时，前两个运算符值比较数据的值是否相等，而后两个运算符不仅要比较值是否相等，还要比较数据的类型是否相等。

## 逻辑运算符

逻辑运算符常用于布尔型的数据进行操作。

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算 | 示例 | 结果 |
| && | 与 | a&&b | a和b都为true，结果为true，否则为false |
| || | 或 | a||b | a和b至少有一个为true，则结果为true，否则为false |
| ! | 非 | !a | 若a为false，结尾为true，否则相反 |

逻辑运算符在使用时，是从左到右的顺序进行求值，因此运算时需要注意，可能会出现“短路”的情况，具体如下所示：

1. 当使用“&&”连接两个表达式时，如果左边表达式的值为false，则右边的表达式不会执行，最终结果为false。
2. 当使用“||”连接两个表达式时，如果左边表达式的值为true，则右边的表达式不会执行，最终结果为true。

## 三元运算符

三元运算符时一种需要三个操作数的运算符，运算的结果根据给定条件决定。具体语法如下：

**条件表达式?表达式1:表达式2;**

先求条件表达式的值，如果为true，则返回表达式1的执行结果；如果条件表达式的值为false，则返回表达式2的执行结果。

## 位运算符

JavaScript中将参与位运算符的数据转换为二进制进行运算。

|  |  |
| --- | --- |
| 运算 | 名称 |
| & | 按位与 |
| | | 按位或 |
| ~ | 按位非 |
| ^ | 按位异或 |
| << | 左移 |
| >> | 右移 |
| >>> | 无符号右移 |

需要注意的是，JavaScript中位运算符仅能对数值型的数据进行运算。在对数字进行位运算之前，程序会将所有的操作数转换成二进制数，然后再逐位运算。

## 运算符的优先级

对一些比较复杂的表达式进行运算时，首先要明确表达式中所有运算符参与运算的先后顺序，我们把这种顺序称作运算符的优先级。下列表中的优先级由上至下递减：

|  |  |
| --- | --- |
| 结合方向 | 运算符 |
| 无 | （） |
| 左 | . [] new（有参数，无结合性） |
| 右 | new（无参数） |
| 无 | ++（后置） --（后置） |
| 右 | ! ~ -（负数） +（正数） ++（前置）  --（前置） typeof void delete |
| 右 | \*\* |
| 左 | \* / % |
| 左 | + - |
| 左 | << >> >>> |
| 左 | < <= > >= in instanceof |
| 左 | == != === !== |
| 左 | & |
| 左 | ^ |
| 左 | | |
| 左 | && |
| 左 | || |
| 右 | ?: |
| 右 | = += = \*= /= %= <<= >>= >>>= &= ^= |= |
| 左 | , |

1. 左结合方向表示同级运算符的执行顺序为从左向右，右结合方向则表示执行顺序从右向左。
2. 优先级最高的运算符时（），当表达式中有多个圆括号时，最内层圆括号中的表达式优先级最高。
3. 为复杂的表达式适当的添加圆括号，可避免复杂的运算符优先级规则，让代码更为清楚，并且可以避免错误的发生。

# 流程控制

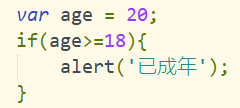
JavaScript中有三大流程控制语句：顺序结构、选择结构和循环结构。

## 选择结构

选择结构语句需要根据给出的条件进行判断来决定执行对应的代码。常用的选择结构有单分支（if）、双分支（if…else）、和多分支（if…else if…else和switch）语句三种。

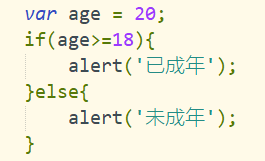
1. **单分支语句**

if条件判断语句也被称为单分支语句，当满足某种条件时，就进行某种处理。例如：



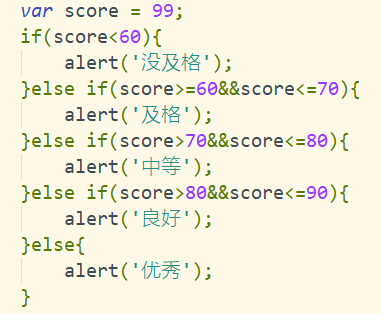
1. **双分支语句**

if…else语句也称为双分支语句，当满足某种条件时，就进行某种处理，否则进行另一种处理。例如：



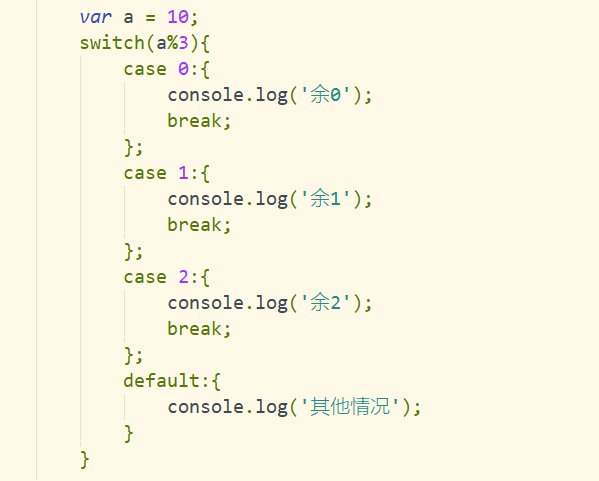
1. **多分支语句**

·if…else if…else：也称多分支语句，可针对不同情况进行不同处理。例如：



·switch：也是多分支语句，功能与if多分支语句相同，不同的是它只能针对某个

表达式的值做出判断，从而决定执行哪一段代码。例如：



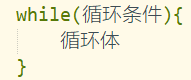
首先计算表达式的值，然后将获得的值与case中的值依次比较，若相等，则执行case后的对应代码段；当遇到break语句时，跳出switch语句。若没有匹配到case，则执行default中的代码段，default代码段式可选的，应根据实际情况选择是否设置default。

## 循环结构

循环结构就是可以实现一段代码的重复执行。JavaScript提供的循环语句有：while、do…while和for循环语句3种。

1. **while循环**

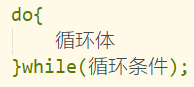
while循环是根据循环条件来判断是否重复执行一段代码。语法如下：



当循环条件为true时，执行循环体，执行后，又判断循环条件是否为真，直到为假时才不执行循环体，转而执行while循环后面的语句。

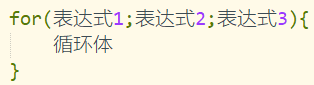
1. **do…while循环**

do…循环的功能与while语句类似，唯一的区别在于，while是先判断后执行循环体，而do…while会无条件执行一次循环体后再判断循环条件是否继续执行循环体。语法如下：

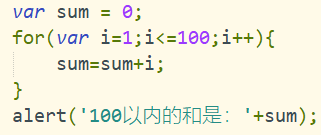


1. **for循环**

for循环是最常用的循环语句，它适合循环次数已知的情况。语法如下：



例如使用for循环求1~100的和，代码如下：



值得一提的是，for循环语句括号内的每个表达式都可以为空，但是必须保留分号分割符。当每个表达式都为空时，表示该for循环语句的循环条件永远满足，会进无限循环的状态，如果此时需要结束无限循环，可在for语句循环体中使用跳转语句进行控制。

## 跳转语句

跳转语句用于实现程序执行过程中的流程跳转。常用的跳转语句又break和continue语句。

1. **break**

break语句可应用在switch语句和循环语句中，其作用是终止当前语句的执行，跳出switch选择结构或循环结构，执行后面的代码。

1. **continue**

continue语句用于循环语句中，表示结束本次循环，开始下一轮循环。