# 第三章 类型、值和变量

数据类型：能表示并操作的值的类型称作数据类型。

变量：是一个值的符号名称，可以通过名称来获得对值的引用。

数据类型的种类：原始类型和对象类型、拥有方法和不拥有方法、可变或不可变

原始类型：数字、字符串、布尔值

特殊原始值：null（空）、undefined（未定义），代表各自特殊类型的唯一成员，无任何方法。

对象的定义：对象是属性的集合，每个属性由“名/值对”组成。普通对象是无序集合，数组是带编号值的有序集合。

函数：是具有与它相关联的可执行代码的对象，通过调用函数来运行可执行代码，并返回结果。

不可变类型：数字、布尔、null、undefined、字符串。

Javacript采用词法作用域，使用var来声明变量，不再任何函数内声明的变量称为全局变量。

补充：词法作用域也叫静态作用域，从敲代码一刻（词法分析）开始，就已经定好了。反之是动态作用域，作用域链基于调用堆栈。

## 3.1数字

Javascript不区分浮点数和整数，统一用浮点数表示

Javascript采用IEEE 754标准定义64位浮点格式表示数字。

能表示的最大值：1.79769\* 10^308

能表示的最小值: 正负 5 \* 10^-324

整数范围：-2^53~2^53

Javascript的实际操作数使用32位整数，例如：数组索引，以及位操作符

数字直接量：一个数字直接出现在代码中。

补充：

IEEE 754：定义了计算机程序设计环境中的二进制和十进制的浮点数的交换、算数格式以及方法。

### 补充---原码、反码、补码、移码：

#### 原码

机器数，首位是符号位：0为正、1为负。计算机存储、处理、运算数据通常以8位、16位、32位或64位来进行。以8位为例，原码是有缺陷的，0000 0000（+0）和1000 0000（-0）是同一个数，所以只有255种状态，补码中 1000 0000 表示为128 解决了这个问题。

现代计算机中不用原码表示正数、只用其来表示浮点数的尾数部分。

**优点：**

1. **实现乘法、除法比较简单**
2. **与真值对应关系直观方便**

**缺点：**

**1、0的表示不唯一，有+0和-0；**

**2、加减运算复杂，需要判断正负号**

#### 反码

用最大数1111 1111减去一个数，就可以得到它的反。原码求反，正数不变，负数保留符号位其余求反。

反码在计算机中很少使用，一般用作中间变化。

缺点：

1、0的表示不唯一

2、运算时，必须考虑循环进位。

优点：

#### **补码**

正数不变，负数保留符号位，先求反，然后加一。

#### 移码

补码的符号位取反。

任何计量系统都必须有：容量和精度

模：代表有多少状态数。8位计算机的模：2^8 = 256

10

原码：0000 1010

反码：0000 1010

补码：0000 1010

-10

原码：1000 1010

反码：1111 0101

补码：1111 0110

### 3.1.1 整型直接量

可识别十六进制、十进制、八进制（严格模式下不可以）

十六进制：0x OR 0X,后跟十六进制数串的直接量，不区分大小写。

八进制：以0开始，后跟0~7的数字组成的序列，所以不建议在数字前加0，小心出错。

### 3.1.2 浮点型直接量

Digits：数字、位数。

[d][.d][(E|e)[(+|-)]d]

3.14

0.314

.314

3.14e23

3.14e+23

3.14e-23

### 3.1.3 JavaScript 中的算数运算

1、通过算术运算符来进行数字运算

2、+-\*/%

3、Math.pow(2, 4); //2 的 4次幂，pow—幂、乘方

4、ceil\floor\round\abs\max\min\random()

5、PI\E\random()

6、pow\sqrt

7、sin\cos\

Math.sqrt(-3) = NaN;

0/0 = NaN;

Infinity/Infinity = NaN;

被零整除，只会等于Infinity或者-Infinity

被Infinity整除，只会等于+0或者-0（作为除数作用不相同）

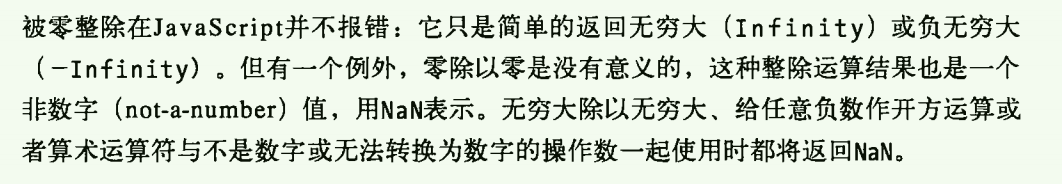
上溢：Infinity、-Infinity

下溢：+0、-0（两者几乎一样，作为除数除外）

isNaN()

false： 123、 “123”、null、Infinity

true：NaN、｛｝、undefined



### 3.1.4 二进制浮点数和四舍五入错误

1、使用实数的时候，常常只是真实值的一个近似表示。

### 3.1.5 日期和时间

略—后续写类。

## 3.2文本

1、采用UTF-16字符集

2、字符串由无符号的16位值组成的序列

3、字符串操作方法作用于16位值，而非字符。

### 3.2.1 字符串直接量

1、字符串直接量：由单引号或双引号括起来的字符序列

2、单引号定界的字符串可以包含双引号，同理。

3、ES3字符串必须一行，ES5可以“\”来分行（注意空格）

5、一般：HTML事件处理代码使用双引号，JS字符串使用单引号。

### 3.2.2 转义字符

空： \u0000 \0 null是真实的空，什么也不显示，ascall 0X00

空格：\u0020 空格，ascall 0X0020

回车：\r 输出回到行首，一般会删掉行前的字符串，

换行：\n

Linux 每行结尾是”\n”, window----\n\r mac-----\r

水平制表符：\t 和tab一个属性，空格长度不确定（2）

垂直制表符：\v

换页符：\f linux和clear一个效果

退格符：\b

双引号、单引号、反斜线、\xXX、\UXXXX

除这十二种外，单独出现\，会忽略。

### 3.2.3 字符串的使用

1、“+”号运算符作用于字符串，表示字符串连接。

2、函数

返回制定位置的字符：charAt(len); （返回空字符串）

substring(idx1, idx2);

返回两个制定下标之间的字符：

1. 选取小数作为start，大的作为end；
2. 返回一个包含start，不包含end的字符串

3、start大于等于0

Substr(start,length);（es未优化，不建议使用）

1. start若为负数，则从后面开始。
2. IE4中start设置为0，有BUG；

Slice（start,end）；

从数组中返回制定元素。

1. 若start 小于零，则，end必须省略， start=len-abs(start)-1;end = len-1;
2. 若start 大于等于零，end 小于零，则end = length – abs（end）；

IndexOf(searchvalue, fromindex);

指定字符串第一次出现的位置

LastIndexof(searchvalue, fromindex);

制定字符串最后一次出现的位置；

Split(separator/regexp, howmany);

 方法用于把一个字符串分割成字符串数组。

1\””为参数

1. Array.join
2. Separator为正则

Replace(regexp/substr, replacement);

Replacement可以为函数

toUpperCase()\toLowerCase()

toLocalUppercase();\toLocalLowerCase()

与 toUpperCase() 不同的是，toLocaleUpperCase() 方法按照本地方式把字符串转换为大写。只有几种语言（如土耳其语）具有地方特有的大小写映射，所有该方法的返回值通常与 toUpperCase() 一样。

字符串可以使用方括号或者charAt来访问单个字符的值。

### 3.2.4模式匹配

1、正则：文本匹配模式对象。

2、Javascript采用Perl中的正则表达式语法。

3、两条斜线之间的文本构成了一个正则表达式直接量

4、第二条斜线后可以跟随一个或多个字母，来修饰匹配模式的含义。（i忽略大小写）

示例：

var test = “testing: 1, 2, 3”;

1. 匹配所有包含一个或多个数字的实例： pattern = /\d+/g;
2. 匹配成功 pattern.test(test) === true;
3. 首次匹配成功的位置： test.search(parttern) === 9
4. 所有匹配成功的数组： test.march(parttern) === [“1”, “2,”, “3”]
5. 替换： test.replace(pattern, “#”) === “testing: #, #, #”
6. 用非数字截取字符串：test.split(/\D+/) === [“”, “1”, “2”, “3”]

### 3.3布尔值

1、javacript 比较语句的结果通常是布尔值

2、七个值为false： undefined\null\0\false\””\-0\NaN

3、其他值都是true

4、布尔值包含toString（）方法

5、true、false是保留字

6、六假值：undefined、null、0、-0、NaN、“”

### 3.4 null和Undefined

Null

1、null是关键字，它表示一个特殊值，描述为“空值”

2、typeof null === object， 可以理解null 为空对象

3、null 是它自有类型的唯一一个成员，且不包含任何方法

Undefined

1. 表示变量没有初始化---值不存在
2. Undefined是它自有类型中的唯一成员，且不包含任何方法

Var a ;

Console.log(a); //undefined

Console.log(a.b); //type error:property is undefined

Console.log(b); //reference error: not defined

### 3.5全局对象

1、javascript启动的时候，创建一个新的全局对象，并初始化它。

### 3.6 包装对象

1、javascript对象通过“.”操作符来引用属性

2、如果属性为函数，称其为方法。通过0.m()来调用方法

字符串为什么具有属性和方法：

只要引用了字符串的属性，javascript则会调用new String来创建新的字符串对象，来处理引用属性，引用结束，销毁对象。而null和undefined没有包装对象，访问会返回类型错误。

Var s = “test”

s.len = 10;

console.log(s.len); //undefined

“test” == new String(“test”); //true

“test” === new String(“test”); //false

### 3.7 不可变的原始值和可变的对象引用

1、不可变的原始值：undefined\null\bool\string\number

2、原始值的比较：值的比较

3、对象的比较：引用的比较，当且进档引用同一个基对象时候，才相同。

### 3.8类型转换



#### 3.8.1 转换和相等性

“==” 从不试图讲操作数变为布尔类型

#### 3.8.2 显式类型转换

1、null和undefined没有toString函数

2、toString和String返回结果一致

3、x + “” \ +x \ x – 0 \ !!x

4、obj.toString(2)： 返回反映这个对象的字符串

5、toFixed():保留小数点后指定位数

6、toExponential(): 指数计数法、保留小数点后指定位数

7、toPrecision(): 指定有效数字、若整数长度大于指定值，则使用指数形式。

8、三者都使用四舍五入、填充0

9、Number（value），只能基于十进制，且不能出现非法尾随字符。

10、parseInt、parseFloat： 全局函数、跳过空格且尽可能解析更多字符并忽略后面内容、第一个非空格字符是非法数字直接量则返回NaN、能识别‘0x’‘0X’十六进制

11、parseInt（value， radix）: 当前数字直接量的基数类型：2~36

#### 3.8.3 对象转化为原始值

1、各个版本的toString

Object //[Object object]

Array //元素转化为字符串，元素与元素之间用逗号分割开

Function //返回这个函数的实现定义的表示方式

RegExp //返回正则表达式直接量的字符串

Date //返回一个可读的日期时间字符串

2、各个版本的ValueOf

Number\bool\string //返回本身---原始值

Object\Function\Array\RegExp //返回本身---对象

Date //返回一个number，getTime();

1. 对象到字符串转化步骤：

a.toString存在，且toString返回原始值，原始值转化为字符串，结束。

b.ToString没有返回原始值，且valueOf存在，且valueOf返回原始值，同上。

c.若toString和valueof都无法获得一个原始值，则type error；

1. 对象到数字转化步骤：

同上

5、+、==、 !=、 > 、<、>= 、<=

条件：其中一个操作符为对象，转换遵寻对象到原始值的转换方式

非日期类型：对象到原始值，先调用valueof，返回原始值直接使用不再转换

日期类型：使用对象到字符串的转换模式

*//+\==\!=\<... 日期类型  
// var date = new Date();  
// console.log(typeof +date); //number  
// console.log(typeof (date + 1)); //string  
// console.log(date == date.toString()); //true  
// console.log(date != date.toString()); //false  
// console.log(date == date.valueOf()); //false  
// console.log(date != date.valueOf()); //true  
  
//+\==\!= \<... 字符串对象类型  
// var str = new String("123");  
// console.log(typeof str); //object  
// console.log(typeof (str.toString())); //string  
// console.log(typeof (str.valueOf())); //string  
// console.log(str === str.valueOf()); //false  
// console.log(str.valueOf() === str.toString());//true  
// console.log(typeof (+str)); //number  
// console.log(typeof (str + 1));//string.  
// console.log(str == str.toString());//true  
// console.log(str == str.valueOf());//true  
// console.log(str != str.toString());//false  
// console.log(str != str.valueOf());//false  
  
//+\==\!=\<... 对象类型  
// var obj = {};  
// console.log(obj.toString()); //[object Object]  
// console.log(obj.valueOf()); //{}---is a function  
// console.log(obj.valueOf() === obj); //true  
// console.log(obj.toString() + "..."); //[object Object]...  
// console.log((obj.valueOf()) + "..."); //[object Object]...  
// console.log(typeof obj); //object  
// console.log(typeof (+obj));//number  
// console.log(+obj);//NaN  
// console.log(typeof (obj.toString())); //string  
// console.log(typeof (obj.valueOf())); //object  
// console.log(obj == obj.toString()); //true  
// console.log(obj == obj.valueOf()); //true  
// console.log(obj.toString() == obj.valueOf()); //true*

*//+\==\!=\<... 数字对象类型  
// var number = new Number(123);  
// console.log(typeof number); //object  
// console.log(typeof (number.toString())); //string  
// console.log(typeof (number.valueOf())); //number  
// console.log(typeof (number + 1)); //number  
// console.log("123" == 123); //true  
// console.log(number == number.toString()); //true  
// console.log(number == number.valueOf()); //true  
// console.log(number === number.valueOf()); //false  
// console.log(number.toString() === number.valueOf()); //false*

### 3.9 变量声明

1、使用一个变量前，需要先声明。

2、使用var关键字来声明，系统默认赋值undefined

3、可以同时声明多个变量、可同时赋值

4、重复声明合法无害

5、不声明就引用变量，报错。

6、不声明就引用变量，会当成全局变量（几乎一致）或者报错。（非严格模式和严格模式）

### 3.10 变量作用域

略

#### 3.10.1 函数作用域和声明提前

1、函数作用域VS块级作用域

变量在声明它们的函数体以及这个函数体潜逃的任意函数内部都是有定义的。

花括号内每一段代码都具有各自的作用域，而且变量在声明他们的代码段之外是不可见的。

2、声明提前

函数作用域在函数内声明的所有变量在函数体内都是可见的。

预编译的时候：把声明（不涉及赋值）被“提前”至函数体的顶部

#### 3.10.2 作为属性的变量

1、javascript全局变量是全局对象的一个属性。

2、var 声明的变量，不可配置（删除）

3、全局变量是全局对象的可配置属性（可删除）

4、es3 调用对象 es5 声明上下文

#### 3.10.3作用域链

Javascript是基于词法作用域的语言，在词法分析阶段就能明白变量的作用域。全局变量在程序中都是有定义的。局部变量在函数体内和嵌套函数内都是有定义的。

将一个局部变量看作一个自定义实现的属性，换个角度来解读变量作用域。每一段javscript代码（全局代码或者函数）都有一个与之关联的作用域链（scope chain）。这个作用域链是一个对象列表或者一个链表，这组对象定义了这段代码“作用域”中的变量。当javascript需要查找变量x的值的时候，如果这个对象有一个x的属性，则会使用这个属性的值，如果第一个对象中不存在x的属性，javascript会继续查找链上的下一个对象。如果第二个对象依然没有名为X的属性，则会继续查找下一个对象，以此类推。如果作用域链上没有任何一个对象有含有属性X，那么就认为这段代码的作用域链上不存在X，并最终抛出一个引用错误异常。

在javascript最顶层代码中（也就是不包含任何函数定义的代码），作用域链由一个全局对象组成。在不包含嵌套的函数体内，作用域链上有两个对象，第一个是定义函数参数，和局部变量的对象，第二个是全局对象。在一个嵌套的函数体内，作用域链上至少有三个对象。理解对象链的创建规则是非常重要的。当定义一个函数时候，它实际上保存一个作用域链。当调用这个函数时，它创建一个新的对象来存储它的局部变量，并将这个对象添加至保存的那个作用域链上，同时创建一个更长的表示函数调用的作用域的“链”。对于嵌套函数来说，事情更有意思，没底调用外部函数的时，内部函数又会重新定义一遍。因为每次调用外部函数 时候，作用域链都是不同的。内部函数在每次定义的时候都有微妙的差别---在每次调用外部 函数时，内部函数的代码是相同的，而且关联这段代码的作用域链也不相同。