## 3.4 数据类型

ECMAscriprt五种（简单）基本数据类型

Undefined、Null、String、Number、Boolean

一种复杂数据类型:Object

### 3.4.typeof操作符

### 3.4.2Undefined类型

只有一个值undefined的数据类型

Var message

alert(message); //“undefined”

alert(age); // 错误

### 3.4.3 Null类型

第二个只有一个值null的数据类型

实际上，undefined值是派生自null值得，因此规定

alert(null == undefined);

但他们的用途完全不同

### 3.4.4Boolean类型

Boolean()

### 3.4.5 Number类型

#### 1、浮点数值

Var floatNum = .1; // 有效，但不推荐

Var floatNuim1 = 1. ; //解析为1

Var floatNum2 = 10.0; //解析为10

Var floatNum = 3.125e7; //等价于31250000

Var floatNum = 3e-7，//等价于0.0000003

{

Var float num1 = 0.1;

Var float num2 = 0.2;

Alert((num1+num2) == 0.3) //由于内存原因 0.1+0.2！=0.3

}

#### 2、数值范围

Infinity（正无穷）-Infinity(负无穷)

Infinity无法参与运算，isFinite()用来判断是否在最大最小之间

#### 3、NaN（非数值）

NaN与任何值的都不相等，包括NaN本身

Alert(NaN == NaN)；//false

任何不能被转换为数值的值都会返回true

Alert (isNaN(NaN)); //true

Alert(isNaN(10)); //false

Alert(isNaN(“10”))； //false

Alert(isNaN(“blue”)); //true

Alert(isNaN(true)); //true

#### 4、数值转换

可以把非数值转换为数值的函数：Number(),parseInt(),parseFloat()

Number（）用于任何数据类型

parseInt(),parseFloat()用于把字符串转换成数值

var num1 = Number（“Hello world!”）; //NaN

var num2 = Number(“ ”); //0

var num3 = Number(“000011” ); //11

var num4 = Number(true); //1

var num1 = parseInt(“1234blue”); //1234

var num2 = parseInt(“ ”); //NaN

var num3 = parseInt(“0xA”) //10

var num4 = parseInt(“22.5”) //22

var num5 = parseInt(“070”) //56

var num6 = parseInt(“70”) //70

var num7 = parseInt(“0xf”) //15

parseInt 指定基数

var num = parseInt(“10”,8) ;//8 按8进制解析

parseFloat

parseFloat与parseInt类似，区别在于

parseFloat第一个小数点有效，第二个无效

parseFloat始终忽略前导的零，只解析十进制

var num1 = parseFloat(“1234blue”); //1234(整数)

var num2 = parseFloat(“0xA”); //0

var num3 = parseFloat(“22.5”); //22.5

var num4 = parseFloat(“22.34.5”); //22.34

var num5 = parseFloat(“0908.5”); //908.5

var num6 = parseFloat(“3.125e7”); //31250000

### 3.4.6 String类型

Var text = “This is the letter sigma:\u03a3.”;

alert (text.length); //输出28

toString()

null和undefined没有toString()方法

String（）

如果有toStirng()方法，调用toString()方法

若值为null,返回‘null’,

若值为undefined,返回‘undefined’。

### 3.4.7 Object类型

Object的每个实例都具有下列属性和方法

constructor:保存着用于创建当前对象的函数。

hasOwnProperty(propertyName):用于检查给定的属性在当前对象实例中是否存在

isPrototypeOf(object):用于检查传入的对象是否是传入对象的原型

propertyIsEnumerable(propertyName):用于检查给定的属性是否能够使用for-in语句枚举。

toLocaleString(): 返回对象的字符串表示，该字符串与执行环境的地区对应。

toString():返回对象的字符串表示

valueOf();返回对象的字符串、数值活布尔值表示，通常与toString()方法的返回值相同。

## 3.5操作符

### 3.5.1 一元操作符

1)递增和递减操作符

var num1=2；

var num2 = 20;

var num3=num1--+num2; //等于22

var num4 = --num1+num2; //等于21

var s1 = “2”;

var s2 = “z”;

var b= false;

var f =1.1;

var o={

valueof: function(){

return -1;

}

}

s1++; //值变成数值3

s2++; //值变成NaN;

b++; //值变成数值1；

f--; //值变成0.100000000000000009（由于浮点舍入错误导致）；

o--; //值变成数值-2；

1. 一元加和减操作符

var num=25;

]num = +num; //仍然是25

var s1 = “01”;

var s2 = “1.1”;

var s3 = “z”;

var b = “false”;

var f = 1.1;

var o = {

valueOf:function(){

return -1;

}

};

s1 = +s1;//值变成1

s2= +s2;//值变成2.1

s3= +s3;//值变成NaN

b = +b;//值变成0

f = +f;//值未变，任然是1.1；

o =+o;//值变成数值-1；

一元减操作符主要用于表示负数，如将1转成-1，用于数值时，该值变成负数。应用于非数值时，一元减操作符遵循与一元加操作符相同的规则，最后将得到的数值转换为负数。

### 3.5.2位操作符

位操作符并不直接操作64位的值。而是先将64位的值转换成32位的整数，然后进行操作，最后再将结果返回64位。

对于有符号的整数。32位中的前31位用于表示整数的值。第32位表示数值的符号，0表示正数，1标书负数。这个表示符号的位叫做符号位。正数以纯二进制格式存储。

负数同样以二进制码存储，但使用的格式是二进制补码。计算数值的二进制补码，需要经过三个步骤。

1. 求这个数值绝对值的二进制码（例如，要求-18的二进制补码，先求18的二进制码）；
2. 求二进制反码，即将0替换为1，将1替换为0；
3. 得到的二进制反码加1。
4. 按位非（NOT）~

var num1 = 25; //二进制00000000000000000000000000011001

var num2 = ~ num1; //二进制11111111111111111111111111100110

alert(num2); //-26

对25执行按位非的操作，得到了-26，验证了按位非的本质，即操作数的负值减1，因此，下面的代码也能得到相同的结果。

var num1 = 25;

var num2 = -num1-1;

alert(num2); //-26

由于按位非是在数值表示的最底层操作，速度更快。

1. 按位与（AND）&

按位与操作只在两个数值对应位都是1时才返回1，任何一位是0，结果都是0.

var result = 25 & 3；

alert(result); //1

25 = 0000 0000 0000 0000 0000 0000 0001 1001

3 = 0000 0000 0000 0000 0000 0000 0000 0011

AND 0000 0000 0000 0000 0000 0000 0000 0001

3)按位或（OR）|

有一个位是1的情况下就返回1，只有两个位都是0的情况下才返回0；

var result = 25 | 3;

alert(result); //27

4)按位异或（XOR）^

两个数值对应上只有一个1时才返回1，对应的两位都是1或0，返回0。

var result = 25 ^ 3;

alert(result) ; //26

5）左移<<

将数值的所有位向左移动指定的位数。

var oldValue = 2; //等于二进制的10；

var newValue = oldValue <<5; //等于二进制的1000000，十进制的64

左移不会影响操作数的符号位，换句话说，如果将-2向左移动5位，结果将是-64，而非64。

6）有符号的右移 >>

有符号的右移操作与左移操作恰好相反，即64向右移动5位，结果将变回2

var oldValue=64; //等于二进制的1000000

var newValue = oldValue>>5; //等于二进制的10，即十进制的2

7）无符号右移 >>>  
 对正数来说，无符号右移与有符号右移相同。对负数来说，无符号右移操作符会把负数的二进制码当成正数的二进制码，而且由于负数以其绝对值得二进制补码形式表示，会导致无符号右移后的结果非常大。

var oldValue = -64; //等于二进制的11111111111111111111111111000000；

var newValue = oldValue >>>5; //等于十进制的134217726

### 3.5.3 布尔操作符

1）逻辑非 ！

逻辑非操作符遵循下列规则：

对象：false

空字符串：true

非空字符串：false

数值0：true

非0数值：false

null:true

NaN:true

undefined:true

2)逻辑与 &&

逻辑与可以应用于任何类型的操作数,不仅仅是布尔值,有一个操作数不是布尔值,返回数就不一定是布尔值。

第一个操作数是对象，返回第二个操作数（验证未成功）

第二个操作数是对象，只有第一个操作数求职为true时才返回该对象

若两个操作数均为对象则返回第二个操作数。

有一个操作数是null，返回null

有一个操作数是NaN,返回NaN

有一个操作数是undefined,返回undefined。

逻辑与属于短路操作

let found =false; //为true时，结果会发生错误

let result = (found && someUndefinedVariable); //不会发生错误

alert (result); //会执行（‘false’）

1. 逻辑或 ||

第一个操作数是对象返回第一个操作数

第一个操作数求值结果为false,返回第二个操作数

两个操作数都是null.返回null

两个操作数都是NaN,返回NaN

都是undefined返回undefined。

短路操作符

let found = true;

let result = (found || someUndefinedVariable); //不会发生错误

alert(result); //会执行（‘true’）

### 3.5.4 乘性操作符 ECMAScript定义了三个乘性操作符：乘法、除法和求模。如果参与乘性计算的某操作数不是数值，后台会先使用Number()转型函数将其转换为数值。例如，空字符串被当作0，布尔值true被当作1。

1）乘法 \*

如果操作都是数值，执行常规的乘法计算。若乘积超过了ECMAScript数值的表示范围，则返回Infinity或-Infinity;

一个操作数是NaN，结果为NaN。

Infinity与0相乘，结果是NaN;

Infinity与非0数值相乘，结果是Infinity或-Infinity

Infinity与Infinity相乘，结果是Infinity

如果有一个操作数不是数值，则在后台调用Number()将其转换为数值，再应用上面的规则。

1. 除法 /

与乘法操作符类似

如果操作数都是数值，执行常规的除法计算

一个操作数是NaN,结果为NaN

Infinity被Infinity除，NaN；

零被零除，NaN;

非零的有限数被零除，Infinity或-Infinity

Infinity被任何非零数值除，Infinity或-Infinity。

有一个操作数不是数值，后台调用Number()后应用上述规则

1. 求模 %（余数）

操作数都是数值，常规的除法运算，返回余数

被除数无穷大除数有限大，NaN

被除数有限大，除数是0，NaN

Infinity被Infinity除，NaN

被除数有限大除数无穷大，结果是被除数。

被除数是0，结果是0

操作数不是数值，后台调用Number（）转换为数值后应用上述规则

**3.5.5 加性操作符**

1）加法 +

两个操作数都是数值，执行常规加法计算，根据下列规则返回结果

一个操作数是NaN，返回NaN

Infinity加Infinity，Infinity

-Infinity加-Infinity,-Infinity

Infinity 加-Infinity,NaN

+0加+0，+0；

-0+-0,-0;

+0+-0;+0;

如果两个操作数都是字符串，将第二个操作数与第一个操作数拼接；

如果只有一个操作数是字符串，则将另一个操作数转换为字符串后将两个字符串拼接。

如果有一个操作数是对象、数值或布尔值，调用toString()方法取得相应的字符串值，再应用上述规则，undefined和null调用String()函数取得字符串‘undefined’和‘null’；

2)减法 –

两个操作数都是数值，执行常规算数减法操作

Infinity减Infinity，NaN

-infinity减-infinity，NaN

infinity减-inFinity,Infinity

-Infinity减Infinity,-Infinity

+0减+0，+0；

+0减-0，-0；

-0减-0，-0；

若有操作数是字符串、布尔朱、null或undefined，则在后台调用Number()函数转换为数值，再依据前面的规则计算，若转换的结果为NaN,则结果为NaN.

若有一个操作数是对象，调用对象的valueOf（）方法取得表示该对象的 数值，如果得到NaN,则结果为NaN，如果没有valueOf方法，则调用toString()方法。

5-true //4

NaN-1 //NaN

5-3 //2

5-“” //5

5-‘2 //3

5-null //5

**3.5.6 关系操作符**

(<)(>)(<=)(>=)均返回一个布尔值

两个操作数都是数值，指向数值比较

都是字符串，比较两个字符串对应的字符编码值

一个操作数是数值，则将另一个操作数转换为一个数值，执行数值比较

一个操作数是对象，则调用这个对象的valueOf()方法，没有valueOf()调用toString()方法

一个操作数是布尔，先将其转换为数值，再进行比较

‘Brick’< ‘alphabet’; //true

‘23’<‘3’ //true

‘23’<3 //false

‘a’<3 //false,因为‘a’被转换成了NaN 任何操作数与NaN比较，结果都是false

NaN<3 //false

NaN>=3 //false

**3.5.7相等操作符**

1）相等和不相等 == ！=

如果有一个操作数是布尔值，在比较值钱先将其转换为数值false转换为0，true转换为1；

一个操作数是字符串，另一个是数值，先将字符串转换为数值

一个操作数是对象，另一个不是，调用valueOf（）后进行比较

null和undefined是相等的

比较相等性前，不能将null和undefined转换成其他任何值

有一个操作数是NaN，相等操作false,不等操作true,NaN不等于NaN

若两个都是对象，比较他们是不是同一个对象，如果两个操作都指向同一个对象，则返回true,否则返回false.

null==undefined //true

‘NaN’==NaN //false

5== NaN //false

NaN!=NaN //true

false==0 //true

true==1 //true

true==2 //false

undefined==0 // false

null==0 //false

‘5’==5 //true

2)全等和不全等 === ！==

只在两个操作数未经转换就相等的情况下返回true，

undefined==null //true

undefined === null //false

**3.5.8条件操作符**

variable = Boolean\_expression ? true\_value : false\_value;

返回最大值

var max = (num1 > num2 ) ? num1 : num2

**3.5.9 赋值操作符 =**

乘/赋值（\*=）；

除/赋值（/=）

模/赋值（%=）

加/赋值（+=）

减/赋值（-=）

左移/赋值（<<=）

有符号右移/赋值（>>=）

无符号右移/赋值（>>>=）

**3.5.10 逗号操作符**

var num1=1,num2=2,num3=3;

逗号操作符除声明多个变量外，还可用于赋值，赋值时，逗号操作符总会返回表达式中最后一项。

var num= （3,4,4,3,3,3，0）； //num的值为0；

## 3.6语句

**3.6.1 if语句**