Day 02---变量和数据类型

一、数据类型

> JavaScript的数据类型共分两类：简单数据类型 (也称原始数据类型或基本数据类型)和复杂数据类型(也称引用数据类型或对象类型)。

2.1 简单数据类型

> 简单数据类型共分5种：Number、Boolean、String、Null、Undefined (ECMAScript6新增：Symbol)

数据类型	数据值	说明
Undefined	undefined	当声明一个变量但是并没有赋值时，变量的类型是Undefine类型
Null	null	对象指针为空
Boolean	true　false	true\false两个值都必须小写
Number	10　3.14　3. .5 整数与浮点数	NaN　---- not a number本应该返回数值型数据的函数，如果返回的值不是数值型测返回NaN　isNaN()---不是数字返回true　是数字返回false
String	“Hello” ‘Hello’　“a” ’a’	后续学习
Object	对象	后续学习

2.1.1 Number类型

> 数值类型共分4种值：整数、浮点数、无穷大、NaN。
> 对数值来说，最常见的数值字面量是10进制。比如：20, 30, 5.5 这些值都是用10进制表示的。

1. 整数: 对整数来说，还可以使用8进制和16进制的整数来表示。8进制的范围是(0-7)， 16进制的范围是(0-F)。
   例如：

```
var num = 25;  // 10进制的25
alert(num); //弹出：25
var num1 = 025; // 8进制用0开始。表示一个8进制的25。
alert(num1);   // 弹出：21 注意：弹出显示的时候，总是用10进制的形式。
var num2 = 0x25; // 0x开始表示这个数是16进制。这是个16进制中的25。 注意：这里的x可以大写X也可以小写x
alert(num2);    //弹出：37
var num3 = 0XAF;
alert(num3);  //弹出：175

注意：如果8进制数字超出了进制范围(比如出现8、9)，则自动忽略前导0，把这个数字作为十进制的数字类处理。例如：
var num4 = 089;
alert(num4);  //弹出：89。 并不会出现错误
```

2. 浮点数：所谓浮点数值，就是该数值中必须包含一个小数点，并且小数点后面必须至少有一位数字。浮点数直接量一般有两种写法：直接带小数点的和使用科学计数法。

```
var f1 = 3.14;   // 带小数点的直接量写法
var f2 = 3.158899e5; //科学计数法：   表示 3.158899 * 10^5
```

3. 无穷大(Infinity):
   ECMAScript并不能储存所有的数，所以能表示的数有个范围。所能表示的数最大值和最小值都保存在Number这个内置对象中(后面会详细介绍什么是对象。)

```
alert(Number.MAX_VALUE);
alert(Number.MIN_VALUE);
//如果数值超出了最大值和最小值，则用Infinity和-Infinity表示。
alert(1.1 / 0); //弹出：Infinity
alert(-1.1 / 0);    //弹出：-Infinity
```

4. NaN(not a number 不是个数): 表示不是一个数字，当0/0 的时候不是无穷大，而是NaN。或者把一个非数字形式的字符串转换成数字时都会返回NaN。例如：

```
alert(0 / 0);  //弹出：NaN
alert(parseInt("60"));  //弹出：60 说明：parseInt("");可以把字符串形式的数字转换成Number
alert(parseInt("a"));   //弹出：NaN

//注意：NaN是个非常特别的东东。因为即使他自己和自己都不相等。
var v = 0 / 0;
alert(v == v);  //弹出false
alert(v != v);  //弹出true。  所以：可以通过这种方式来判断这个数是否为NaN

/*
  另外：isNaN()函数如果 x 是特殊的非数字值 NaN（或者能被转换为这样的值），返回的值就是 true。如果 x 是其他值，
  则 返回 false。
*/
alert(isNaN(3));//返回false
alert(isNaN("3"));   //返回false
alert(isNaN("blue"));//返回true
alert(isNaN(true));//返回false . true可以转换成数字1
alert(isNaN(null));//返回false    null可以转换为数字0
alert(isNaN(undefined));//返回true
```

2.1.2 Boolean类型

> Boolean类型的值是程序中用的很多的一种类型。它只有两个值：true和false。true表示逻辑上的对/正确，false表示逻辑上的错。

2.1.3 String类型

> String指的是用 ""(双引号) 或 ''(单引号) 括起来的字符序列。字符序列可以由0个或多个字符组成。
> 例如：”123” “a” ‘name’ ‘blue’ ’nihao’ ‘xiaohong‘ ”nihao”。
>
> 说明：
>
> 1、单引号或双引号要匹配，不能一边用双引号，一边用单引号 ，左右符号要匹配。
>
> 2、JavaScript统一每个字符使用Unitcode码来进行编码，每个字符占16位(2个字节)。
>
> 3. 在其他语言中多用单引号表示一个字符，双引号表示字节。但是对JavaScript来说，不存在字符类型的数据。
>
> 4. 有些字符不可见或有特殊意义，比如换行符，制表符, 双引号等，则 JavaScript 提供了相应的转意字符，来表示这些不可见字符。见下表：

代码	输出
\'	单引号
\"	双引号
\&	和号
\\	反斜杠
\n	换行符
\r	回车符
\t	制表符
\b	退格符
\f	换页符

2.1.4 Undefined

> Undefined 类型只有一个值，即特殊的 undefined 。在使用 var 声明变量但未对其加以初始化时，这个变量的值就是 undefined ， 例如：

```
var v;
alert(v); //弹出：undefined。   变量v声明但是没有赋值初始化，则为undefined
alert(m);  //m没有定义，直接使用会发生错误。
```

2.1.5 Null

> 1. Null 类型是第二个只有一个值的数据类型，这个特殊的值是 null (n是小写)。从逻辑角度来看， null 值表
>
> 示一个空对象指针。
>
> 2. 值 undefined 实际上是从值 null 派生来的，因此 ECMAScript 把它们定义为相等的。
>
> 说明：
>
> 尽管这两个值相等，但它们的含义不同。
> undefined 是声明了变量但未对其初始化时赋予该变量的值，null 则用于表示尚未存在的对象。
> 如果函数或方法要返回的是对象，那么找不到该对象时，返回的通常是null。

```
alert(null == undefined); // 弹出： true。   待讲完面向对象在来理解会更加清晰。大家目前只需要了解即可。
```

2.2 复杂数据类型

> 复杂数据类型又称之为引用数据类型。主要包括三种：对象(object)、函数(function)和数组(array)。
>
> 不过，从本质来看，这三种都属于对象。
>
> 关于复杂数据类型，我们在面向对象部分重点详细介绍。

二、数据类型转换

> 数据类型转换有两种转换：自动转换和强制转换。

1. 自动类型转换：

> 在解释执行的过程中，会根据需要进行相应的自动类型转换。比如：需要一个Boolean类型的值，而现在是一个字符串，则会根据相应的规则把字符串自动转换为Boolean值。

```
1、转换成布尔类型
undefined -> false
null -> false
数值        0,0.0,NaN -> false    其他数值-> true
字符串     ""->false     "etef" -> true
其他对象-> true
总结：存在的东东，转换成true，  不存在的东东转换成flase

2、转换成数值类型
undefined -> NaN
null -> 0
字符串—>数值->      "34234"->34234,  "324sdfs3423" -> NaN,  "" -> 0
布尔类型—>数值     true->1    false->0
总结：能转成数字的就转成对应的数字。不能转成数字的要么0或1要么NaN

3、转换成字符串类型
undefined ->"undefined"
null ->"null"
布尔类型      true ->”true”      false ->"false"
数值类型转成字符串      12321 ->"12321"    NaN ->"NaN"
总结：字面值是什么就转成什么
```

2. 强制类型转换。

```
//1、其他类型转成成字符串。
var a = 10;
alert(a + "");  // 直接和一个长度为0的空字符链接
alert(a.toString()); //调用这个变量的toString()方法
alert(String(a));  //使用String转型函数
//2、字符串转数字。
var s = "123";
alert(parseInt(s));
alert(parseFloat(s));
/*
    注意：使用parseXxx转换
    1、转换为数字的时候，会忽略前面的空格
    2、如果是数组开头然后是字母，则会只转前面的数字
    3、如果是是字母开头则返回NaN
    4、在使用parseInt的时候，可以传入第二个参数，表示这个数的进制，然后就转换成对应的10进制数返回。
*/
alert(parseInt("  12"));  //忽略前面的空白字符
alert(ParseInt("12ab3");  //弹出：12.  从字母后面的统统忽略掉
alert(parseInt("a123"));  // 弹出：NaN

//3、使用转型函数互转(暂时先了解)  var为一个变量或常量都可以
String(var);//其他类型转字符串
Number(var);  //其他类型转数字
Boolean(var);  // 其他类型转布尔
```

三、表达式和运算符

3.1 表达式

> 表达式是用于JavaScript脚本运行时进行计算的式子，可以包含常量、变量、运算符。表达式可以任意复杂。表达式的值就是经过一系列运算之后的最终值。

```
var n = 10;
var m = 30;
alert(n + m);  // n+m就是一个表达式。
```

3.2 运算符

> 运算符也叫操作符。作用是去操作数据值，然后进行各种计算。ECMAScript 操作符的与众不同之处在于，它们能够适用于很多值，例如字符串、数字值、布尔值，甚至对象。(不过，在应用于对象时，相应的操作符通常都会调用对象的 valueOf()和（或） toString() 方法，以便取得可以操作的值。 暂时了解)

3.2.1 算术运算符

> +、-、*、/、%。
>
> 这几个运算符都需要两个数据参与运算。

1. + 运算符：运算规则

- 如果两个都是Number则，则就按照普通的数学加法运算。
- 如果有一个是字符串，就按照字符串的串联的方式连接。(如果另外一个不是字符串，则先转换成字符串再连)。
- 如果有一个是NaN，则结果就是NaN。
- 如果同时是Infinity或-Infinity，则结果就是Infinity或-Infinity。
- 如果一个是Infinity另外一个是-Infinity，则结果为NaN。

2. - 运算符：运算符规则

- 如果两个都是Number则，则就按照普通的数学减法运算。
- 如果有一个操作数是字符串、布尔值、null 或undefined，则先在后台调用Number()转型函数将其转换为数值，然后再根据正常减法计算。如果转换的结果有一个是NaN,则减法的结果就是NaN
- 如果有一个操作数是对象，则调用对象的valueOf()方法以取得表示该对象的数值。如果得到的值是NaN，则减法的结果就是NaN。如果对象没有valueOf()方法，则调用其toString()方法并将得到的字符串转换为数值(了解)

3. * 运算符：运算符规则。

- 运算规则同减法。

4. / 运算符：运算规则

- 就是普通的除法运算，运算规则同 *

5. %运算符：运算规则

- 求余(求模)运算。

```
var v = 10 % 2;   // 余0
alert(10 % 3); // 结果 :  1
alert(-10 % 3); // 结果:  -1   说明：求余的时候，符号与被除数保持一致。
alert(10 % -3); // 结果:  1
alert(1 % 0.3);  // 结果：0.1
```

3.2.2 一元运算符

1. ++自增操作: 对要操作的变量进行+1的操作，并把+1之后的结果重写赋值给这个变量。可以在变量前也可以在变量后。任何变量都可以使用，但是对非Number类型的变量操作的时候，需要先按照前面的规则转换成Number然后再自增1.

```
var a = 10;
var b = a++;  //把a的值自增1，a的值变为11.  注意：a++表达式的值，仍然是a自增之前的值，所以b的值是是10

var c = 10;
var d = ++a;  //把a的值自增1，a的值变为11.  注意：++a表达式的值，是a自增之后的值，所以d的值是11

var m = "123";
m++;  //先把m使new Number(m)之后再转换进行自增的操作。
```

2. --自减操作: 对要操作的变量进行减1的操作，并把-1之后的结果重新赋值给这个变量。 计算方法同++。

```
var a = 10;
var b = a--;  //把a的值自减1，a的值变为9.  注意：a--表达式的值，仍然是a自减之前的值，所以b的值是是10

var c = 10;
var d = --a;  //把a的值自减1，a的值变为9.  注意：--a表达式的值，是a自减之后的值，所以d的值是9

var m = "123";
m--;  //先把m使用转型函数Number(m)之后再转换进行自减的操作。
```

3.2.3 赋值运算符和复合赋值运算符

> =、+=、-=、*=、/=、%=
>
> 使用赋值操作符的注意目的是简化表达式的书写，并不会带来任何的性能的提示。

```
var n = 10;
n += 4;  //等价于 n = n + 4;  运算结束后n的值为14

var m = "6";
m -= 5;   // 会先把m转换成数字之后(m = new Number(m))，再执行  m = m - 5;
```

3.2.4 比较运算符

> >、>=、<、<=、==、===、 !=、!==
>
> 比较运算符是对运算符的坐标和右边的两个数据进行比较。

- 如果两个都是数字，就是普通的数字比较。
- 如果有一个是数字，就把另外一个不是数字的使用new Number(var)的方式转换为数字再比较
- 如果两个都是字符串，则比较的是字符串中的字符在字母表中的顺序。(或者说比较是他们的编码值)
- 如果有一个是NaN，则结果一定是false。但是有一种情况，如果比较特殊NaN != NaN则是true。

> == 和 != 是判断两个操作数是否相等。规则：

- 如果两个都是简单类型中的Number类型，直接判断数值是否相等。

```
alert(10 == 10.0);  //  true
```

- 如果有一个NaN参与比较，则总是不等的。
- null和undefined之间是相等
- 如果有一个操作数是布尔值，则在比较相等性之前先将其转换为数值——false 转换为 0，而 true 转换为 1；
- 如果一个字符串与数值比较，则把字符串转换成数值之后再比较
- 如果一个对象和一个基本类型比较，则调用对象的valueOf方法，转变成基本类型之后再比较(暂时先了解)
- 如果连个对象比较，则比较两个对象是否为同一个对象。是就相同，不是就不同。(暂时先了解)

```
alert(NaN == false);  // fasle
alert(NaN == true);  // false
alert(NaN == 1); //false;
alert(NaN == NaN); // fasle
alert(NaN != NaN); // true

alert(null == null); // true
alert(null == undefined); // true
alert(undefined == undefined); // true

alert("1" == 1); //true
alert("1a" == 1);  //false  “1a"会被转换为NaN

alert(1 == true); //true  会把true转成1之后再比较
alert(0 == null);  //false  null与任何的非null的都不相等
```

> === !== 全等判断
>
> 全等在转换前不做任何的转换，如果两个一样就相等，否则就不等。

```
alert("1" === 1);  //false 一个string类型，一个number类型，所以不等
alert(null === undefined);  // false
alert("abc" === "abc");  // true
```

3.2.5 逻辑运算符

> 只有三种逻辑运算符：&&、||、！。而且 && 和|| 都是短路的形式。 注意：不像其他语言有非短路的&、|, 也没有^(异或)

1. 逻辑非 ！ 。注意：这个是运算符只需要一个数据参与运算。不管参与运算的 是什么数据，最终结果一定是Boolean值.逻辑非的运算逻辑是，不管什么类型，先转换成Boolean值，再取反得到最终的结果。 取反的意思就是true变false，false变true

```
alert(!Null);  //true。  因为null会转为fasle，所以取反之后变为true
alert(!0);  // true
```

2. 逻辑与 && 。 只要有一个是false，则返回false。(只有两个都是true的情况下才返回true)。 逻辑与&&的结果不一定是Boolean值 以下为特殊情况：

- 有短路的特点：如果 第一个 是false或者可以转换成false，则 结果就是第一个 。
- 如果 第一个 是true或者可以转换成true，则 结果就是第二个 。

```
var a = 10;
alert(false && a++); //第一个操作数是false，所以最终结果是false。由于短路的关系，所以不会再去计算第二个表达式。
alert(a); //  结果：10

var b = 20;
alert(null && b++);  // 第一个操作数是null，结果为null，且不会再去判断第二个。
alert(b);  // 结果：20

var c = 30;
alert(new Number(5) && c++);  //结果 30.第一个操作数是对象，所以直接返回第二个操作数。
alert(c); //结果：31
```

3. 逻辑或 || 。 只要有一个是true，结果就是true。 与逻辑与一样，结果也不一定是Boolean值。特殊情况如下：

- 也有短路的特点：如果 第一个 是true或者能转换成true，则直接返回第一个,不再去判断第二个 (判断优先级高)
- 如果 第一个 是false或可以转换成false，则直接返回第二个。(判断优先级次之)

```
var a = 10;
alert(true || a++);  // 结果true，且不会计算第二个表达式

var b = 20;
alert(new Number(5) || b++); //结果为第一个(5)，且不会计算第二个。

var c = 30;
alert(null || c++); //结果为null，且不会计算第二个。
```

3.2.6 三元运算符(条件运算符)

> 三元运算符的意思是指，需要三个操作数参与运算。
>
> 语法：逻辑值(表达式)？ 表达式1： 表达式2
>
> 结果：如果逻辑值是true，则最终返回表达式1的值，如果逻辑值是false，则最终结果返回表达式2的值。
>
> 注意：如果逻辑值不是Boolean值，则先按照转型函数Boolean()的方式转变成Boolean，在进行计算

```
var a = 0 ? 1 : 2;  //结果是2
var b = null ? 1 : 2; //结果是2
var c = "a" ? 1 : 2; //结果是1
```

3.2.7 typeof操作符

> typeof 操作符用来检测给定的变量的类型。
>
> 语法： typeof 变量;
>
> 注意：typeof是一个操作符，而不是方法，所以typeof后面的操作数完全不需要添加括号。

```
var a = "abc";
alert(typeof a);  //弹出：string  注意：s是小写
```

> typeof操作符的计算结果只有一下6种：

1. "undefined" ——如果这个值未声明，或者声明了但没有赋值
2. "boolean" ——如果这个值是布尔值；
3. "string" ——如果这个值是字符串；
4. "number" ——如果这个值是数值；
5. "object" ——如果这个值是对象或 null ；
6. "function" ——如果这个值是函数。

```
var a;
alert(typeof a);  // undefined
alert(typeof b);  // undefined
alert(null);  //object
alert("abc"); //string
alert(new Object()); //object
alert(new String()); // object
```

3.2.8 逗号运算符

> - 使用逗号操作符可以在一条语句中执行多个操作。
> - 逗号操作符还可以用于赋值。在用于赋值时，逗号操作符总会返回表达式中的最后一项

```
//var a, b, c, d;   //使用逗号操作符同时声明了4个变量。
var a = (1, 2, 3, 4);   // a= 4.  逗号操作符返回最后一个值。
alert(a);  //弹出：  4
```

3.3 运算符的优先级

> 运算符的优先级见下表。从上向下，优先级越来越低。
>
> 注意：优先级不用刻意去记，当把我不准优先级的时候，就添加()就可以了。理论上，在一个表达式中括号是没有任何的限制的。

运算符	描述
. [] ()	字段访问、数组下标、函数调用以及表达式分组
++ -- - ~ ! delete new typeof void	一元运算符、返回数据类型、对象创建、未定义值
* / %	乘法、除法、取模
+ - +	加法、减法、字符串连接
<< >> >>>	移位
< <= > >= instanceof	小于、小于等于、大于、大于等于、instanceof
== != === !==	等于、不等于、严格相等、非严格相等
&	按位与
^	按位异或
|	按位或
&&	逻辑与
||	逻辑或
?:	条件
= oP=	赋值、运算赋值
,	多重求值