js中所有数据都是以64位存取，

8 Byte = 64bit。8个字节，64位

基础数据类型:

在 JavaScript 中有 5 种不同的数据类型：

string 、number 、boolean 、object 、function

3 种对象类型：

Object 、Date 、Array

2 个不包含任何值的数据类型：

null 、undefined

一.数值转换：

**number（） paresInt( ) parsefloat( )**

**number（）**

**1.可以将任何非数值数，转化为数值**

**2.如果是 null ,则返回 0**

**3.如果是 undefined ，则返回 NaN**

**4.如果是字符串：**

**字符串是空的，则转换为 0**

**字符串是只包含数字，十六进制的数字，都会将其转换为十进制数字**

**字符串包含有效的浮点格式，将其转化为相应的有效的浮点数**

**除此之外的任何形式，都是 NaN**

**Var num= Number("") // 0**

**Var num= Number("undefined") // NaN**

**Var num= Number( null ) // 0**

**Var num= Number("123 hello") // NaN**

**Var num= Number("0001") // 1**

**paresInt( )**

**1.在转换字符串时，更多的是看其是否符合数值模式。会忽略前面的空格，直至找到第一个非空格的字符，如果第一个字符不是数字字符或是负号，就返回NaN**

**2.如果是以0x开头的会当成16进制解析**

**3.如果是以0开头**

**现在是ES3会当作8进制解析，ES6会当作10进制解析**

**因此提供了第二个参数，指定转换出来的进制**

**Var num=parseInt( "123heoo" ) ///123**

**Var num=parseInt( "heoo123" ) ///nan**

**Var num=parseInt( "null " ) //nan**

**Var num=parseInt( "0" ) //0**

**Var num=parseInt( "" )//其他的都会被解析为NaN**

**Var num=parseInt( " ")其他的都会被解析为NaN**

**parsefloat( )**

**1.也是从第一个字符开始解析，直到遇到一个无效的浮点数数字为止**

**也就是说，第一个小数点是有效的，而第二个小数点就无效了，后面的字符串会被忽略**

**2.都会直接忽略前导的0**

**3.可以识别数字加字符的形式，直接识别数字**

**4.十六进制的数，会被直接转化为0**

**5.如果字符串包含的是一个可以解析为整数的数，就直接解析为整数**

**小数点后面几位都为0 没有小数点**

**6.没有多加参数，指定进制的操作**

在这个运算中e代表10，+或者后面直接跟着e都代表进2位-代表退位

**document**.write(parseFloat(**"3.14"**)+**"Float"**) //3.14

**document**.write(parseFloat(**"314e-2"**)+**"Float"**)//3.14

**document**.write(parseFloat(**"0.0314E+2"**)+**"Float"**)//3.14

**document**.write(parseFloat(**"3.14morenon-digitcharacters"**)+**"Float"**)//3.14

**document**.write(parseFloat(**"314e5"**)+**"Float"**)//31400000

**var*num1***=parseFloat(**"123heoo"**);*///123*

**var*num1***=parseFloat(**"heoo123"**);*//NaN*

*varnum2=parseFloat("null");//NaN*

**var*num3***=parseFloat(**"02"**);*//*

**var*num4***=parseFloat(**"0x5"**);*//*

**var*num5***=parseFloat(**"3.125e7"**);*//31250000*

*/\*varnum5=parseFloat("125e7e");//1250000000\*/*

**var*num6***=parseFloat(**"12345.00"**);*//*

**var*num7***=parseFloat(**"NaN"**);*//NaN*

**var*num8***=parseFloat(**""**);*//NaN*

**var*num9***=parseFloat(**""**);*//NaN*

**var*num10***=parseFloat(**"undefined"**);*//NaN*

**NaN:表示一个本来要返回数值的操作数未返回数值的情况**

**①：\*/NaN=NaN**

**②：NaN ！= \***

**③：任何涉及 NaN 的操作都会返回 NaN**

**④：isNaN() 在接受到一个值以后，会尝试将这个值转换为数值。某些不是数值的值**

**会直接转为数值，而任何不能被转换为数值的值都会导致这个函数返回 true**

**⑤：也可以被用于对象**

**字符串：**

**toString( ):**

**1.几乎每个值都有，toStirng方法，只有 null ，undefined值 没有这个方法**

**2.将数字转换成字符串时，在其中传参数可以将数字转化为指定的进制的字符串**

**String( )：**

**1.所有的值都可以使用这个方法**

**object类型：**

**1.hasOwnProperty("propertyName"):用于检查给定的属性在当前对象这是否存在，参数的属性名必须以字符串的形式指定**

**2.isPrototypeof( object ):用于检查传入的对象是否的当前对象的原型**

**3.propertyIsEnumerable( "propertyName" ):用于检查给定的属性是否能使用 for-in 语句来枚举，**

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

**5.toSting():返回对象的字符串表示**

**6.valueof（）：返回对象的字符串，数值或布尔值表示**

**7.construector():保存着用于创建当前对象的函数。**

**一元操作符：**

**易错：**

**一.递增递减操作:**

**1.Var a=0;**

**++a; 先加后等于 相当于 a=a+1;**

**a++；先等于后加**

**2.对于递增递减 操作符**

**①在应用一个包含有效字数字字符的字符串时，先将其转化为数字值，再执行加减1的操作**

**②在应用一个不包含有效数字字符的字符串时，将变量设置为NaN，**

**③在应用布尔值时，先将其转化，在进行增一减一的操作**

**④在应用于对象时，先调用对象的valueof,获得一个可以操作的值，，然后按照前面的规则进行操作，如果结果是NaN**

**,则调用toStirng()方法后再应用前述规则，对象变量变成数值变量。**

**二.一元加减的操作**

**一元加的操作**

**1.对于数值不会产生任何影响**

**2.**

**①对于非数值应用这个操作时，操作符会像number()转型函数一样，对这个值进行转换**

**②对于字符应用这个操作时，对象先调用他们的valueof()和toString()方法，再转换得到的值**

**一元减的操作主要是用于表示负数**

**位操作符：**

**1.按位非 ~**

**布尔操作**

**0**

**空字符串**

**NaN**

**Undefined**

**Null**

**2.按位与 & 对应位上的数都是1，结果才是1**

**3.按位或 | 对应位上的数有一位为1，结果就为1**

**①都是先转化为32位的二进制数**

**布尔操作符**

**注意：逻辑 与和或 操作是短路操作，如果第一个操作数可以决定结果，就不会对第二个操作数求值**

**1. ||**

**①.如果第一个操作数是对象，则返回第一个操作数**

**②.如果第一个操作数的结果是false，则返回第二个操作数**

**③.如果两个操作数都是对象，则返回第一个操作数**

**④.如果两个操作数是 NaN，则返回 NaN**

**⑤.如果两个操作数是 undefined ,则返回 undefined**

**⑥.如果两个操作数都是 null , 则返回 null**

**2. &&**

**①.如果第一个操作数是对象，则返回第二个操作数**

**②.如果第二个操作数是对象，则只有在第一个操作数的求值结果为true的情况下才会返回给对象**

**③.如果两个操作数都是对象，则返回第二个操作对象**

**④.如果第一个操作数是NaN，则返回NaN**

**⑤.如果第一个操作数是undefined,则返回undefined**

**4.按位异或 ^**

同为1或同为0取0，不同取1

**先将操作数转为布尔值，再对其求反**

**5.左移 << 0从右边填充，数字变大**

**6.右移**

**① 有符号的右移 >> 0从左边填充（符号位不动），原本是正数的数字，变化后会变小**

**② 无符号的右移 >>> 0从左边填充，不会在意符号，如果原本是负数，则变化会特别大**

**布尔操作**

**0**

**空字符串**

**NaN**

**Undefined**

**Null**

**乘性操作符：**

**1.乘法：**

**①：如果只有一个操作数有符号，那么结果就是负数。如果乘积超过了ES的表示范围，则返回Infinity或者-Infinity**

**②：NaN \* X = NaN**

**③：Infinity \* 0 = 0 NaN**

**④：Infinity \* 非0数值 = Infinity / -Infinity**

**⑤：Infinity \*Infinity =Infinity**

**⑥：如果有一个操作数不是数值，则在后台调用Number()将其转化为数值，然后再应用上面的规则**

**2.除法**

**加性操作符：**

**①：Infinity/Infinity=NaN**

**②：0 / 0 = NaN**

**③：如果其中一个操作数是 NaN ，则结果是 NaN**

**④：非0的有限数 / 0 = Infinity / -Infinity**

**⑤：Infinity /\* =Infinity**

**1.加法：**

**①：+Infinity + -Infinity=NaN**

**②：Infinity + Infinity=Infinity**

**③：-Infinity + -Infinity=Infinity**

**④：+0 + +0=+0**

**⑤：-0 + -0=-0**

**⑥：+0 + -0=+0**

**⑦：字符串+字符串 =拼接为一个字符串**

**⑧：字符串+ 数值 = 字符串 + 将数值转化为字符串 再拼接**

**2.减法：**

**①：+Infinity + -Infinity=NaN**

**②：Infinity - Infinity=NaN**

**③：-0 + +0=-0**

**④：Infinity + Infinity=Infinity**

**⑤：-Infinity + -Infinity=Infinity**

**四.关系操作符：**

**= = 先转换成 数值 再比较，**

**Null = =undefined 1**

**Nan ==nan 0**

**"nan" = =nan 0**

**Undefined = =0 0**

**Null = =0 0**

**全等： 不经过类型转换操作，直接进行比较**

**var*x***=**"xnjc"**

**var*y***=Number(***x***)

**document**.write(***y***+2); //结果是NaN

一.js中”5”+4=?

答案：”54”

考点：

js中数据类型的隐式转换，不同类型的数据在计算过程中自动进行转换。

数字+字符串：数字转换成字符串。

2.js中NaN\*4=?

答案：NaN

考点：NaN转化成数字后还是NaN。

3.js中null\*4.5=?

答案：0

考点：null转化为数字后为0。

4.js中alert(5\*1.015===5.075)?

答案：false

考点：js中的计算问题，5\*1.015=5.074999999999999

二. null和undefined的区别？

null是一个表示"无"的对象，转为数值时为0；undefined是一个表示"无"的原始值，转为数值时为NaN。

当声明的变量还未被初始化时，变量的默认值为undefined。 null用来表示尚未存在的对象

undefined表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法是：

（1）变量被声明了，但没有赋值时，就等于undefined。

（2）调用函数时，应该提供的参数没有提供，该参数等于undefined。

（3）对象没有赋值的属性，该属性的值为undefined。

（4）函数没有返回值时，默认返回undefined。

null表示"没有对象"，即该处不应该有值。典型用法是：

（1） 作为函数的参数，表示该函数的参数不是对象。

（2） 作为对象原型链的终点。