# 基本数据类型

在JS中一共有6种数据类型

String 字符串

Number 数值

Boolean 布尔值

Null 空值

Undefined 未定义

Object 对象

其中 String Number Boolean Null Undefined属于基本数据类型

而Object属于引用数据类型

## String 字符串

- 在JS中字符串需要使用引号引起来，双引号或单引号都行，不能一单一双

- str变量 "str"字符串

- 引号是不能嵌套的

当需要打印一些特殊字符时，可以使用\来作为转义字符

\" 表示 "

\' 表示 '

\\ 表示 \

\t 表示制表符table

\n 表示换行

\uxxxx 表示一个Unicode编码

typeof

- 通过typeof可以来检查一个变量的类型

- 语法：typeof 变量

- 它会将检查的结果返回

- 使用typeof检查一个字符串时，会返回string

## Number 数值

- 在JS中所有的数字都是Number类型， 包括整数和浮点数

使用typeof检查一个数值时，会返回number

JS中可以表示的最大的数字

Number.MAX\_VALUE 来获取最大值

1.7976931348623157e+308

Number.MIN\_VALUE 0以上的最小值

5e-324

如果我们使用的数字超过了最大值，则会返回Infinity，表示无穷

±Infinity 正负无穷,它的类型也是Number

为什么在js中Number.MAX\_VALUE + 1不是Infinity？

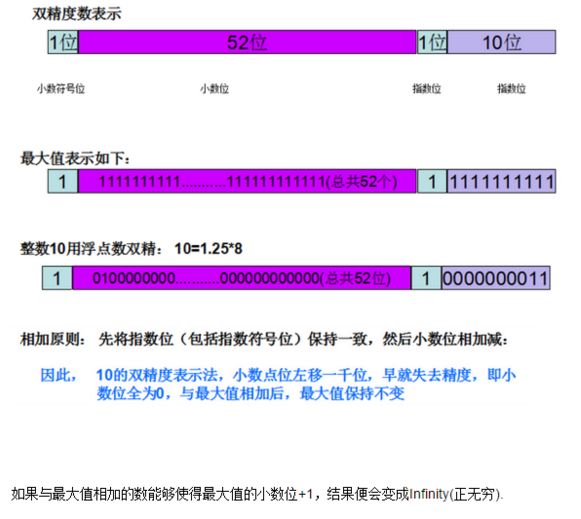
下图中说得左移一千位是指：

最大值的二进制科学计数法表示：1.111111...1111 \* 2^1024

整数10的二进制科学计数法表示：1.010 \* 2^3

计算时要将指数转换成一致，所以整数10小数点就得左移一千多位，就变成了这样：0.00000...1010 \* 2^1024

但是最大值尾数位总共为52位，转换后的10尾数位变成了一千多位，两者相加根本不会影响到最大值，因为计算机最多保存52位尾数位，保存不了1000多位的，所以舍去了，所以结论就是两者相加不是infinity



特殊的数字 NaN (Not A Number)

- 用来表示一个非法的数字

- 使用typeof检查一个NaN时，也会返回number

需要表示一些其他进制的数字

- 十六进制数字： 0x数字

- 八进制数字：0数字

- 二进制的数字：0b数字 IE不支持 Hbuilder也不支持

在JS中大部分的整数的运算都可以确保结果正确，但是如果对浮点数进行运算时，可能出现不可预期的结果，所以在JS中尽量不要做对精度要求高的运算，比如 钱相关的

## Boolean 布尔值

- 我们可以使用布尔值来进行逻辑判断

- 布尔值只有两个true和false

- true表示逻辑的真

- false表示逻辑的假

- 使用typeof检查一个布尔值时，会返回boolean

## Undefined 未定义

- Undefined表示一个声明但是没有赋值的变量

- Undefined的类型只有一个值 undefined

- 使用typeof检查一个Undefined类型的数据时，会返回undefined

## Null 空值

- Null类型只有一个值，就是null

- null值专门用来表示为空的对象

- 使用typeof检查一个Null类型的值时，会返回object

# 类型转换

所谓的类型转换就是指将其他的数据类型转换为String Number 或 Boolean

- 强制类型转换

- 转换为String

- 转换为Number

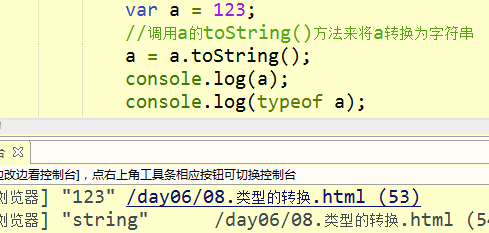
- 转换为Boolean

## 其他数据类型转换为String

### 调用被转换数据类型的toString()方法

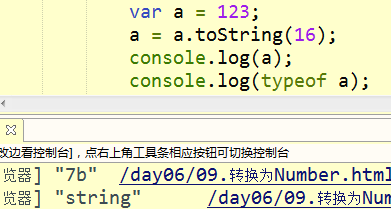
- toString()方法不会改变原变量的值，而是将值转换为字符串并返回

- 由于null和undefined中没有toString()方法，所以不能调用toString()将他们转换为字符串，调用会报错



对于数字来说，toString()方法，除了可以将一个数字转换为字符串以外，它还可以将一个数字转换为指定进制的字符串

在数值的toString()方法中，可以传递一个进制的整数作为参数，这样数值将会被转换为指定进制的数字的字符串



### 调用String()函数

- 将被转换的数据做为参数传递给String()

- 原理：

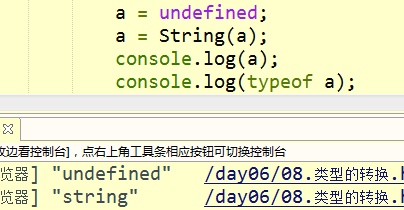
- 对于具有toString()方法的类型来说，

String()会调用它们的toString()方法来将其转换为字符串

- 对于null和undefined来说，

会将null 直接转换为 "null"

将undefined 直接转换为 "undefined"



调用 XXX的YYY方法就是 XXX.YYY()

document.write();

console.log();

调用XXX函数就是XXX()

alert()

## 其他数据类型转换为Number

### 调用Number()函数

- 转换的情况：

1.对于字符串来说

- 如果字符串是一个合法的数字，则将其转换为对应的数字

- 如果字符串不是一个合法的数字，则转换为NaN

- 如果字符串是一个空串或纯空格的字符串，则转换为0

2.对于布尔值来说

- true转换为1

- false转换为0

3.对于Null

- null转换为0

4.对于Undefined

- undefined转换为NaN

除了使用Number()函数，JS中还为我们提供了两个函数，,专门用来将一个字符串转换为数字的

### parseInt()

- 将一个字符串转换为整数

- 使用该函数解析字符串时，它会从字符串的左侧开始解析，直到解析到最后一个有效的整数位

var c = "456px"; //456

c = "789a678b456"; //789

c = "123.456"; //123

c = "hello123.456"; //NaN

像"070"这样的字符串，火狐 chrome等浏览器会当成10进制数字进行解析，转换为70，但是IE8，会将其当成8进制的数字解析，解析为56，在使用parseInt时，可以传递一个进制作为参数，来说明数字的进制

c = "070";

//调用parseInt将其转换为Number

c = parseInt(c,10); //70

c = parseInt(c,8); //56

如果在parseInt()中传递一个非字符串作为参数，它会将其转换为字符串，然后再解析 var a = true; a = parseInt(a); //NaN

可以使用parseInt()对一个小数进行取整，但是使用这个函数进行取整性能较差，因为对小数（Number）取整，需先将其转换为字符串，然后再取整

### parseFloat()

- 将一个字符串转换为小数

- parseFloat()和parseInt()类似，不同的是它可以解析到有效的小数位

var d = "123"; //123

d = "123px"; //123

d = "123.45px"; //123.45

d = "123.45.67"; //123.45

d = "hello123"; //NaN

## 其他数据类型转换为Boolean

### 调用Boolean()函数

- 转换的情况：

- 对于数字

- 除了0和NaN剩下都是true

- 对于字符串

- 除了空串剩下的都是true

- null和undefined都是false

- 对象默认都是true

# 运算符

运算符（操作符）

- 通过运算符可以对一个或多个值进行运算，并返回结果

## typeof运算符

- 用来检查一个变量的数据类型，并将类型返回

- 语法：typeof 变量

- typeof运算符返回的结果是个字符串，这个字符串用来描述数据类型

## 算数运算符（二元运算符）：

+

+可以对符号两侧的值进行加法运算，并将运算结果返回

- 当对非数字(除了字符串)进行加法运算时，它会将其先转换为数字，然后再运算

- 如果对两个字符串进行相加，则会进行拼串的操作，将两个字符串拼成一个字符串

- 任何值和字符串进行加法运算时，都会先将其转换为字符串，然后再和字符串进行拼串的操作，我们经常通过为一个任意的值加 空串 的形式来将其转换为字符串，这种形式我们称为隐式的类型转换，原理实际上也是通过String()函数转换的，但是这种方式更加简单。

var a = 10 + true; //11

a = true + false; //1

a = false + null; //0

任何值和NaN做任何运算都是NaN，但和字符串做加法运算时，会将NaN转换为字符串”NaN”，会进行拼串操作

a = NaN + 10; //NaN

a = undefined + 100; //NaN

a = NaN + "hello"; //NaNhello

a = "hello" + "world"; //"helloworld"

a = 11 + "abc"; //"11abc"

var result = 1 + "2"; //12

result = 1 + 2 + "3"; //33

result = "1" + 2 + 3; //123

result = "1" + (2 + 3); //15

还可以利用+来往页面中添加代码

var str = "<table>"+

"<tr>"+

"<td></td>"+

"</tr>"+

"<tr>"+

"<td></td>"+

"</tr>"+

"<tr>"+

"<td></td>"+

"</tr>"+

"</table>";

-

- 可以对符号两侧的值进行减法运算

\*

\* 可以对符号两侧的值进行乘法运算

/

/ 可以对符号两侧的值进行除法运算

这些符号的运算规则和数学中是一摸一样的，不同的是在对非数字类型进行运算时，会先将其转换为数字然后再运算，我们还可以通过为一个任意的数据类型 -0 \*1 /1 来将其转换为Number类型，原理和Number()函数一样

%

% 可以对符号两侧的值进行取模（取余）运算

- 同样对非数字进行取模的运算，也会转换为数字然后再运算

## 一元运算符

一元运算符只需要一个操作数

### 一元的+

- 一元+就相当于数学运算中的 正号

- 一元+不会对一个数值产生任何变化

### 一元的-

- 一元-相当于数学运算中的 负号

- 负号可以对一个数值进行符号位取反

- 如果对非数值进行一元的+或-运算，它会向将其转换为number，然后再运算，我们可以使用一元的+将其他数据类型转换为Number，原理和Number()函数一样

### 自增和自减

自增：

- 所谓的自增就是指在变量原值的基础上增加一个1，会对原变量产生影响

- 我们使用++来进行自增运算

++可以写在变量前或者变量后，我们称为前++(++a) 和 后++(a++)

无论是a++还是++a都会立即使原变量自增1，调用几次就自增几次

不同的是a++和++a的值不同：

a++的值是变量自增前的值（变量原值）

++a的值是变量自增后的值（变量新值）

自减

- 自减指在原值的基础上 -1，会对原变量产生影响

- 使用 -- 进行自减运算

--分为前--（--a） 和 后--（a--）

- 无论是--a 还是 a-- 都会使原变量立即自减1

- 不同的是 a-- 的值变量自减前的值（变量的原值）

而--a 的值是变量自减后的值（变量的新值;）

## 逻辑运算符

### ! 逻辑非

- 非可以对一个布尔值进行取反操作

- true变成false，false变true

- 如果为一个值取两次反和没取一样

- 如果对一个非布尔值进行取反操作，它会将其转换为布尔值然后再运算，利用这个特点，我们可以为任意的数据类型取两次反，来将其转换为布尔值，原理和Boolean()函数一样

### &&逻辑与

- &&可以对符号两侧的值进行与运算

- 只有两端都是true时，才会返回true。只要有一个false就会返回false。

JS中与运算是短路的与，如果第一个值是false，则不看第二个值

true && alert("看我出来不出来"); // 第一个值为true，alert()会执行

false && alert("看我出来不出来"); //第一个值为false，alert()不会执行

### ||逻辑或

- ||可以对符号两侧的值进行或运算

- 只要有一个true就会返回ture，如果两端都是false才会返回false

JS中的或，是短路的或，如果第一个值为true，则不看第二个值

false || alert("看我出来不出来"); //第一个值为false，会看第二个

true || alert("看我出来不出来"); //第一个值为true，不看第二个

### 非布尔值的逻辑运算

对于非布尔值，做与或运算时，它会先将其转换为布尔值，然后再运算，最终返回变量的原值！

非布尔值的与运算：

如果第一个值为false，则返回第一个；如果第一个值为true，则返回第二个

非布尔值的或运算

如果第一个值为true，则返回第一个；如果第一个值为false，则返回第二个

## 赋值运算符

赋值运算符可以将一个值赋值给一个变量

=

=可以将符号右侧的值，赋值给符号左侧的变量

+=

a += x 等价于 a = a + x

-=

a -= x 等价于 a = a - x

\*=

a \*= x 等价于 a = a \* x

/=

a /= x 等价于 a = a / x

%=

a %= x 等价于 a = a % x

## 关系运算符

关系运算符可以比较两个值之间的大小关系

> 大于号，比较符号左侧的值是否大于右侧的值

如果关系成立，则返回true，否则返回false

>= 大于等于，比较左侧的值是否大于或等于右侧的值

如果关系成立，则返回true，否则返回false

< 小于号，比较符号左侧的值是否小于右侧的值

如果关系成立，则返回true，否则返回false

<= 小于等于，比较左侧的值是否小于或等于右侧的值

如果关系成立，则返回true，否则返回false

NaN和任何值比较都是false

如果对非数值类型进行关系运算，则会将他们转换为Number然后再比较

当关系运算符的两侧都是字符串时，它不会将字符串转换为Number，而是会比较字符串的Unicode编码

\* 当字符串中含有多个字符时，会逐位进行比较

\* 第一位和第一位比，当第一位比出结果后，不再比较以后的，否则第二位和第二位比 以此类推

\* 通过这种方式，可以将文本按照字母顺序进行排序，但是对于中文来说意义不大

\* 尤其在比较两个数字类型的字符串时，一定要注意，很有可能得到错误的结果

## 等性运算符

### 相等运算符

==

\* - 相等运算符，它可以用来判断两个值是否相等，如果相等，则返回true，否则返回false

\* - 如果对两个类型不同的变量进行相等比较，会先将其转换为相同的类型，然后再比较（至于具体转换为什么类型，不定，但是大部分情况是转换Number）

字符串和布尔值比时，都转换为Number然后再比：

console.log("1" == true); //true

不一定都转换为Number：

console.log(null == 0); //false

由于undefined衍生自null，所以这两个类型的值做相等比较时会返回true

console.log(null == undefined); //true

NaN不和任何值相等，包括它自己

console.log(NaN == NaN); //false

由于NaN不和任何值相等，无法判断一个变量是否是NaN，在JS中为我们提供了一个isNaN()函数，这个函数专门可以判断一个值是否是NaN，如果是则返回true，否则返回false

var a = NaN;

console.log(isNaN(a)); //true

### 全等运算符

===

全等运算符，可以判断两个值是否全等，如果全等则返回true，否则返回false

\* 全等和相等的区别就是相等会做自动的类型转换，而全等不会

\* 使用全等时，如果两个值的类型不同，直接返回false

null和undefined相等，但是不全等：

console.log(null === undefined); //false

### 不等运算符

！=

不等运算符，和相等相反，如果两个值不等则返回true，否则返回false

不等会对值进行自动的类型转换，如果转换后相等，它也认为相等

### 不全等运算符

!==

不全等运算符，检查两个值是否不全等，如果成立则返回true，否则返回false

不全等不会做自动的类型转换，如果两个值的类型不同，直接返回true

## 三元运算符（三目运算符）

三元运算符需要三个操作数

?:

-语法：

条件表达式?语句1:语句2

- 执行流程：

- 三元运算符在执行时，会先对条件表达式进行求值判断

如果判断结果为true，则执行语句1，并将执行结果返回

如果判断结果为false，则执行语句2，并将执行结果返回

## 逗号

使用逗号可以在一条语句中执行多次操作。

比如：var num1=1, num2=2, num3=3;

使用逗号运算符分隔的语句会从左到右顺序依次执行

## 运算符的优先级

运算符的优先级和数学中的类似，比如：先算乘除后算加减，也可以通过括号来改变运算的顺序

遇到优先级不清楚，可以查询优先级的表，在表中，位置越靠上，优先级越高，越先运算，同一行的运算符优先级一样，优先级一样从左往右计算



但是这个优先级的表，不需要记忆，如果遇到拿不准，则使用()来改变优先级