javaScrip语言基础

> 本文将介绍javaScript最基础的知识点，包括但不限于JavaScript的简单介绍、历史、引入方式、变量、标识符命名规范、数据类型以及操作符等内容。

JavaScript简介

概述 JavaScript是一门动态、 弱类型 的解释型高级编程语言，它基于 原型 ， 支持面向对象 和函数式编程等多种编程范式，通常简称为js。在世界上的绝大多数网站中都能看到JavaScript的身影，世界上所有的主流浏览器(Chrome、IE、Firefox、Safari、Opera)都支持它。

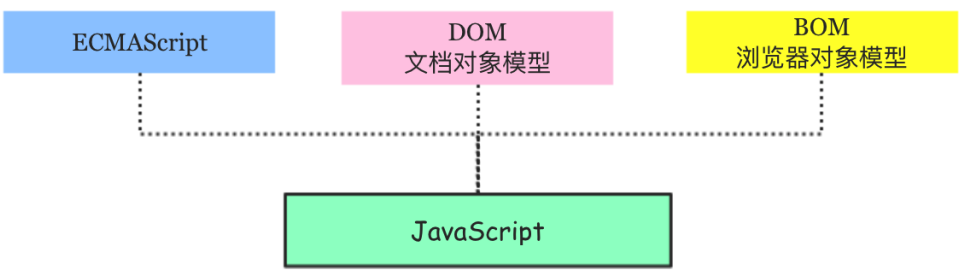
作者 Brendan Eich

背景 JavaScript诞生于1995年，其诞生的初衷是为了减轻服务器端的压力而在客户端提供一种表单验证的功能。最初命名为Mocha，1995年9月在Netscape Navigator 2.0的Beta版中改名为 `LiveScript` ，同年12月，Netscape Navigator 2.0 Beta 3中部署时被重命名为JavaScript，当时网景公司与昇阳电脑公司(Sun)组成的开发联盟为了让这门语言搭上Java这个编程语言“热词”，将其临时改名为JavaScript(其实就像现在某些网红蹭热度一样)。

标准 1996年11月，网景正式向 ECMA（欧洲计算机制造商协会） 提交语言标准。1997年6月，ECMA以JavaScript语言为基础制定了ECMAScript标准规范ECMA-262。JavaScript成为了ECMAScript最著名的实现之一。实现ECMAScript规范的语言还有Adobe的ActionScript和微软的JScript。

范围 虽然在大多数情况下，我们都认为ECMAScript和JavaScript表达的是相同的含义，但实际上JavaScript所表达的却比ECMAScript要广泛的多。完整的JavaScript应该由以下三部分组成：

- ECMAScript 由ECMA-262定义，提供核心语法功能。
- DOM 全称 `Document Object Model` 文档对象模型，提供访问和操作网页的API。
- BOM 全称 `Browser Object Model` 浏览器对象模型，提供与浏览器交互的方法和接口。



历史 ECMAScript迄今已经历多个版本的迭代，下面给出主要的版本历史。

版本	发布日期	版本说明
ES 1	1997年06月	首版
ES 2	1998年06月	修正格式，以使得其形式与ISO/IEC16262国际标准一致
ES 3	1999年12月	强大的正则表达式，更好的词法作用域链处理，新的控制指令，异常处理，错误定义更加明确，数据输出的格式化及其它改变
ES 4	废弃	由于关于语言的复杂性出现分歧，第4版本被放弃，其中的部分成为了第5版本及Harmony的基础
ES 5	2009年12月	新增“严格模式（strict mode），一个子集用作提供更彻底的错误检查,以避免结构出错。澄清了许多第3版本的模糊规范，并适应了与规范不一致的真实世界实现的行为。增加了部分新功能，如getters及setters，支持JSON以及在对象属性上更完整的反射。
ES 6	2015年06月	ECMAScript 2015（ES2015），第 6 版，最早被称作ES6，添加了类和模块的语法，其他特性包括迭代器，生成器和生成器表达式，箭头函数，Buffer数据，静态类型数组，集合(maps，sets 和 weak maps) promise、reflection 和 proxies。
ES 7	2016年06月	ECMAScript 2016（ES2016），第 7 版，多个新的概念和语言特性
ES 8	2017年06月	ECMAScript 2017（ES2017），第 8 版，多个新的概念和语言特性
ES 9	2018年06月	ECMAScript 2018 （ES2018），第 9 版，包含了异步循环，生成器，新的正则表达式特性和 rest/spread 语法。

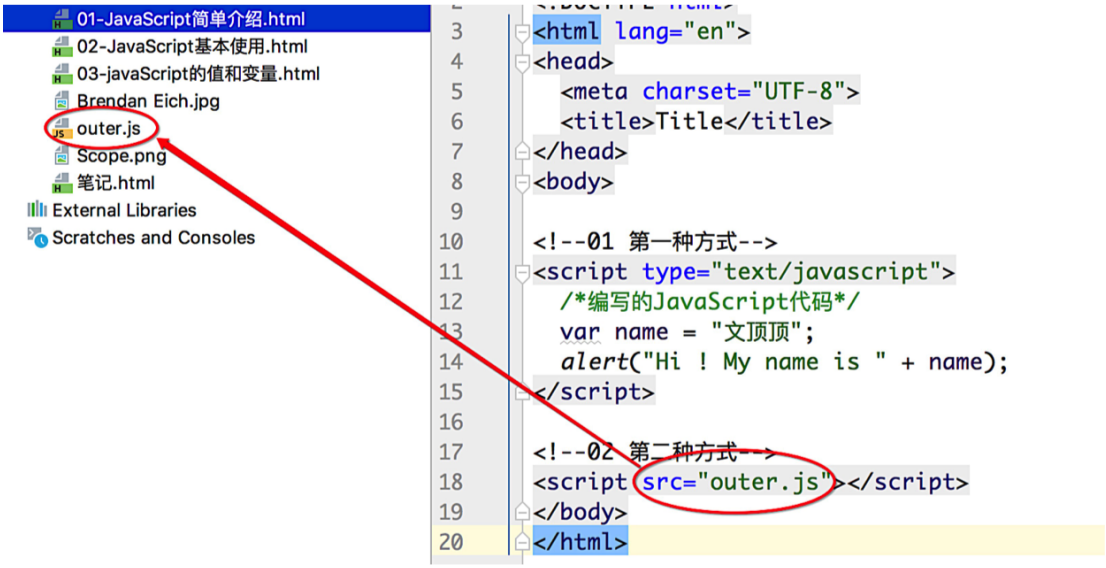
关系 ECMAScript是标准(规范)，JavaScript是实现。H5是一种新的技术，JS用于实现H5新标签深层的扩展功能。HTML表示网页的核心内容和结构，CSS用于设置网页的样式，JavaScript控制网页的行为。

应用 JavaScript主要用于浏览器Web、物联网、游戏、桌面和移动应用开发和以及服务器端的开发。

参考 ECMA官网 ECMA-262规范PDF文件 布兰登·艾克博客

JavaScript初体验

在html页面中使用JavaScript



在html页面中编写JavaScript代码需要借助script标签，具体的使用方式有两种。 **① 在页面中嵌入JavaScript代码** 。在html页面中创建script标签，设置script标签的type属性为text/javascript，并在标签中直接编写JavaScript代码即可。 **② 在页面中引入外部的js文件** 。在html页面中创建script标签，把javaScript代码单独保存在.js后缀的文件中，然后通过设置script标签的src属性来引入js文件。

script 标签的 type 属性可以省略，默认值即为 `text/javascript` 。
script 标签的 属性(节点)主要有： `type` (类型)、 `src` (资源地址)、 `async` (异步加载)、 `defer` (延迟执行)、 `charset` (字符集)等。

注释

JavaScript遵循C语言的注释风格，支持单行和多行注释。

```
单行注释 //
多行注释 /*...*/
```

语句

概念 在JavaScript中，可以简单认为一行完整的代码就是一条语句(`statement`)。
分类 整体来讲，JavaScript的语句大致可以区分为 `声明赋值语句` 和 `控制语句` 两种。
分号 JavaScript中使用 `分号(;)` 来间隔多条语句，若各语句独占一行那么大部分情况下可省略分号。

控制输出

JavaScript需要具体的JavaScript引擎(解析器)来解析，该引擎通常由浏览器提供，即JavaScript代码需要运行在浏览器中。JavaScript代码中常见的控制输出方式有以下三种：

- 输出到页面 `document.write()`
- 弹出框显示 `alert()`
- 控制台输出 `console.log()`

```
/*01 弹出对话框提示*/
alert("喜欢我就点我吧！");
alert("小姐姐你好，我是阿狸~");
alert("拜拜 >.< ");
```

```
/*02 直接向在网页中输出内容*/
document.write("蝉鸣的夏季，刚好遇见你。");
document.write("<h1>遇见</h1>");

/*03 控制台打印输出*/
console.log(123);
console.log("文顶顶");
```

变量

直接量

说明 直接量(literal)指的是程序中直接使用的数据值。

当一个值(数字、字符串等)直接出现在JavaScript程序中时，我们称之为 直接量 。

JavaScript语言中直接量(字面量)有很多，包括数字直接量、字符串直接量、数组直接量、对象直接量以及正则表达式直接量等等，下面简单给出一些直接量的示例。

HTML

```
null;            //空
18;              //数字
19.3;            //小数
"wendingding";   //字符串文本
true;            //布尔值
false;           //布尔值
/abc/gi;         //正则表达式直接量
```

变量基础

定义 变量是编程语言中能够存储计算结果或表示值的抽象概念。
使用 在JavaScript语言中 变量需要先声明再使用。
声明 使用 var 关键字来声明变量，如果省略var关键字那么该变量默认成为全局变量。
作用 记录特定的内容，并通过变量名来访问它们。
备注 JavaScript变量是无类型的(untype)，任何变量都可以被赋予任何类型的值。
原理 当使用var关键字声明变量时，计算机会从内存中分配储存空间来存放不同类型的内容。

```
/*01 先声明两个变量，然后再输出变量的值到控制台*/
var age = 18;
var name = "wendingding";
console.log(age);
console.log(name);
age = 20;
console.log(age);                    //变量的值可以被修改

/*02 可以一次性声明多个变量*/
var address = "北京市",className = "H5";
console.log(address,className);   //北京市 H5
```

语法说明 var name_1 [= value1][,...,name_n [= value_n]]]

```
/*声明示例*/
var i;
var a = '<strong></strong>';
var p,q;
var x = 2,y = 3,z;

/*示例说明
* JavaScript中的=和数学中的=不一样，在JavaScript中=是赋值运算符。
* var a = '<strong></strong>';  这句话分成两个部分。
* 左值 ：在等号左侧，是变量名（同时被赋值）
* 右值 ：在等号右侧，是存放进变量中的东西（给变量赋值）
* 备注 ：使用var关键字多次声明同一个变量是无所谓的(正确、无意义也不报错)。
* */
```

命名规范

标识符 标识符(`identifier`)指的是JavaScript代码中变量、函数、属性的名字，或者函数的参数。标识符(变量)在命名的时候并不能随心所欲，也有对应的规则和要求。下面列出具体的命名规范：

> ① 标识符可以使用下划线、字母、数字和$符号。
> ② 标识符不能以数字开头。
> ③ 标识符区分大小写(区别于HTML)。
> ④ 标识符不能使用JavaScript的关键字和保留字。

```
/*01 常见(合法)的标识符命名方式*/
/*纯字母*/
var age  = 18;
var name = "wendingding";
var stuAge  = 21;
var stuName = "宁夏";

/*字母、数字、下划线、$的组合*/
var num1 = 1.5;
var num2 = 200;
var num$ = num1 + num2;
var stu_Score = 99;

/*02 错误的命名演示*/
var if = 123;               //错误：使用关键字
var super =  "哈哈哈"        //错误：使用保留字
var 2age = 123;             //错误：数字开头
var stu-address = "北京市";  //错误：使用了非法的-
```

命名风格

JavaScript语言常用的标识符命名风格是 `驼峰标识法(camel-case)` ，即标识符的名称由多个单词组合的时候，每个单词的首字母大写以区分。驼峰标识又可以分成大驼峰标识和小驼峰标识，它们的区别在于整个标识符的首字母需要大写。

```
小驼峰标识 var wenDingDing  = "神秘人";
```

```
大驼峰标识 var WenDingDing  = "神秘人";
```

当然，在写代码的时候给标识符命名并非一定要使用驼峰标识，这只是一种建议的风格，譬如有的开发者就喜欢用下划线来连接单词，类似于wen_ding_ding、stu_Name这样。类似的还有匈牙利命名法等，但在JavaScript编程中不建议使用。

JavaScript规定的关键字 关键字通常用于执行特定的操作。

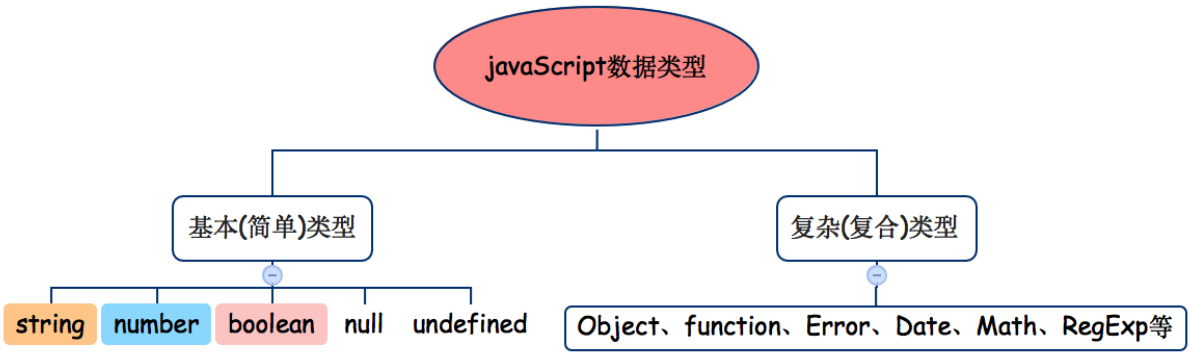
break	do	instanceof	typeof
case	else	new	var
catch	finally	return	void
continue	for	switch	while
debugger*	function	this	with
default	if	throw	delete
in	try		

JavaScript规定的保留字 保留字是给语言未来发展而预留的。

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

数据类型

> 在编程语言中，能够表示并操作的值的类型被称为数据类型(`type`)，能够支持多种数据类型是每一门编程语言的基本特征。在编写程序的时候，如果我们需要将某个(些)值保存起来以备将来使用时，就会将该 **值** 赋值 给一个 变量 (将值保存到变量中)。



JavaScript语言的数据类型可以简单的分成 基本(简单)类型 和 复杂(复合)类型 。

基本类型主要包括： 字符串(string) 、 数值(number) 、 布尔值(boolean) 、 Null 和 undefined 五种。其中Null类型有一个值，即null表示为空，而undefined类型也只有一个对应值undefined，表示变量未定义(即声明变量后未给变量赋值)。

复杂类型主要是对象类型，包括Object对象、Function函数、RegExp正则等，这里不做具体的展开。

typeof关键字

如果我们需要判断变量的类型，那么可以使用 typeof 关键字(操作符)。

语法 typeof 变量 | typeof(变量)
结果 typeof 关键字执行后的结果总是为一个string类型的字符串。

```
/*多种类型的变量*/
var age    = 18;                        //数值类型
var name   = "宁夏";                     //字符串类型
var isFun = true;                       //布尔类型值
var a;                                  //未定义
var obj    = {id:1,desc:"描述信息"};     //Object类型
function fn() {
  console.log("我是fn函数");
}

console.log(typeof age);              //number
console.log(typeof name);             //string
console.log(typeof isFun);            //boolean
console.log(typeof a);                //undefined
console.log(typeof obj);              //object
```

```
/*typeof的两种使用方式*/
console.log(typeof fn);             //function
console.log(typeof(fn));            //function

obj = null;
console.log(typeof obj);            //object
```

注意 对null执行typeof计算的结果为object,其实这被认为是JavaScript这门语言的一个设计错误。

字符串类型

定义 由〇个或多个16位Unicode字符组成的字符序列。

表示 字符串可以由双引号或单引号表示。

操作 可以通过length属性来获取字符串的长度，且多个字符串之间可以通过 + 来进行拼接。

```
var str1 = "Hi ~";
var str2 = "Wendingding!";
var str3 = str1 + " " +str2;    /*字符串的拼接 */
console.log(str3);              /*输出结果：Hi ~ Wendingding!*/
console.log(str3.length);       /*输出结果：17 */
```

注意 JavaScript中的字符串是不可变的，这也就意味着要改变某个变量保存的字符串，那么需要先销毁原来的字符串然后再用另外一个包含新值的字符串来填充该变量。

```
var test = 'Hi ! ';
test = test + 'Nice to meet u ~';
console.log(test);  //Hi ! Nice to meet u ~

/*描述上述代码的内部执行细节
* 说明：上述示例代码中 test变量的值最开始时为Hi ！，而后变成了Hi ! Nice to meet u ~
*       但这并不意味着字符串是可变的，要想理解这一点需要把变量和字符串区分开来。
* 上述第二行代码的实现过程为(整个过程在浏览器后台处理)：

* [1] 先创建一个能容纳21位字符的新字符串。
* [2] 在新创建的字符串中填充Hi ! 和 Nice to meet u ~内容。
* [3] 销毁原先的Hi ! 和 Nice to meet u ~ 字符串，因为它们没用了。
*/
```

布尔类型

说明 布尔类型用来表示正确和错误两种状态(同灯泡有开和关两种状态一样)。

取值 布尔类型(boolean)只有两个值，分别是true和false。

注意 布尔类型常用于条件表达式，布尔类型的值和字符串以及数值等可以相互转换。

undefined类型

undefined类型的值也只有一个，那就是 `undefined` 。我们在使用var来声明变量，但是没有对该变量进行初始化的时候，变量的值就为undefined，表示未定义。

Null类型

Null类型的值只有一个，那就是 null(关键字) ，通常表示空对象指针。
注意 ① typeof null 的结果为 object 而非 null。
注意 ② 实际上，undefined的值派生自null，因此ECMA-262规定它们的相等性测试需要返回true。

```
/*01 布尔类型值*/
var boolA = true;
var boolB = false;

/*02 声明变量但未赋值(未定义)*/
var test;

/*03 设置变量的值为null*/
var boolA = null;

console.log(boolA);                //null 表示空对象
console.log(test);                 //undefined 未定义

/*04 测试undefined和null*/
console.log(undefined == null);    //true
```

数值类型

定义 数值简单说就是数字，在JavaScript语言中数值类型包含整数和浮点数(小数)。
小数 浮点数就是小数，数值中必须包含一个小数点，小数点后面必须至少有一位数字。
备注 实际上JavaScript内部并不直接区分整数值和浮点数值，其所有数字均用浮点数值表示。

```
/*01 数值的两种类型*/
var num1 = 123;        /*整型-数据1*/
var num2 = 4.0075e7;   /*整型-数据2  科学计数发*/

/*浮点数特点：数值中必须包含一个小数点，小数点后面必须至少有一位数字。*/

var floatNum1 = 8.26;  /*浮点型-数据1*/
var floatNum2 = 1.1;   /*浮点型-数据2*/
var floatNum3 = 0.5;   /*浮点型-数据3*/
var floatNum4 = .8;    /*浮点型-数据4-不建议*/
var floatNum5 = 3.2e-4;/*浮点型-数据4 科学计数法*/

/*02 整数和浮点数的默认转换*/
var intNum1 = 10.0;  /*整数，解析为10*/
var intNum2 = 2.;    /*整数，解析为2*/

/* 03  浮点数注意点
 * 001 默认，当小数点后面超过6个0，则以科学计数法的方式来表示。
 * 002 浮点数值的最高精度为17位小数，但算术运算时其精度不如整数。
 * 003 JavaScript使用基于IEEE754数值的浮点格式表示，计算因此存在舍入误差问题。
 */
```

进制 JavaScript中的数值类型支持多种进制，包括二进制、八进制、十进制和十六进制等。
说明 在进行算术运算时，所有八进制、十六进制的数据最终都会转换为十进制的数据。
特点 八进制的特点是数字以 0 开头，十六进制则以 0x 或 0X 开头。
补充 实际上ECMAScript并不支持八进制直接量，且严格模式下八进制直接量被禁止，因此不建议用。十六

进制值是 0 ~ 9之间的数字和a(A) ~ f(F)之间的字母 构成，字母对应的数字为10~15。此外，我们还可以通过调用 `toString` 方法传递参数的方式来实现进制的转换。

```
/*进制的转换  通过toString方法*/
var num1 = 17;
console.log(num1);               //默认以十进制的方式打印(数字)  17
console.log(num1.toString());    //默认以十进制的方式打印(字符串)
console.log(num1.toString(2));   //设置以二进制的方式打印(字符串) 10001
console.log(num1.toString(8));   //设置以八进制的方式打印(字符串) 21
console.log(num1.toString(16)); //设置是十六进制的方式打印(字符串)11

/*二进制、八进制、16进制的数据*/
console.log(070);   //8进制的数据    对应的十进制值为56
console.log(0x11);  //16进制的数据   对应的十进制数值为17
```

`NaN` 全称 `Not a Number` (非数值)，NaN用于表示本来要返回数值的操作数而实际未返回的情况。

> ① 任何涉及NaN的操作都会返回NaN。
> ② NaN与任何值都不相等，包括NaN自身。

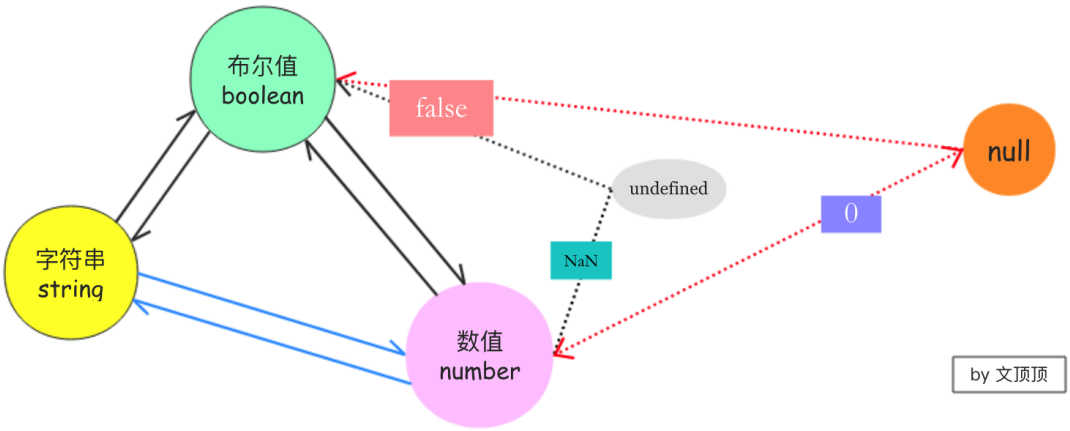
说明 上面列出了NaN的两个特点，针对NaN的这两个特点，ECMAScript提供了isNaN() 函数。isNaN()函数接收一个参数，该参数可以是任何类型的，该函数在执行的时候会尝试把参数转换为数值，如果参数不能被转换为数值(转换失败)，那么返回true，否则返回false。

```
console.log(isNaN(NaN));      //true
console.log(isNaN(10));       //false
console.log(isNaN("20.3"));   //false
console.log(isNaN("5red"));   //true   无法转换为数值
console.log(isNaN(true));     //false
console.log(isNaN("red"));    //true   无法转换为数值
```

> **二进制浮点数的误差问题** JavaScript在使用数字(实数)的时候，常常只是真实值的一个近似表示。原因就在于JavaScript采用的是IEEE-754浮点数表示法(这是一种二进制浮点数表示法),这种表示法可以精确地表示分数，比如1/2、1/8和1/1024等，而我们开发中常用的反而都是十进制分数，比如1/10、1/100等，神奇的地方就在于这种表示法无法精确的表示类似于0.1 、0.2和0.3这种简单的数字。正是因为上面的原因，所以JavaScript语言中才会存在奇葩的 0.1 + 0.2 == 0.3 不成立的问题。

类型的转换

在JavaScript的基本数据类型中，字符串、数值以及其他类型之间是可以相互转换的，而这种转换大概又可以细分成两种，其一是在进行算术运算时默认会执行的 `自动转换` ，其二就是 `强制转换` 了。



类型间的强制转换

强制类型转换需要用到一些特定的函数，这些函数可以是 `Number()` 、 `Bumber()` 、 `String()` 也可以是 `parseInt()` 、 `parseFloat()` 、 `toString()` 等，下面将通过代码来演示它们的具体使用。

```
/*01 Number(构造)函数把其它类型转换为数值*/
console.log(Number(null));          //0
console.log(Number(undefined));    //NaN
console.log(Number("miaoXia"));    //NaN
console.log(Number("18blue"));     //NaN
console.log(Number("18"));         //18
console.log(Number(true));         //1
console.log(Number(false));        //0

/*02-1 String函数用于把其它类型转换为字符串*/
console.log(String(null));        //"null"
console.log(String(undefined));   //"undefined"
console.log(String(123));         //"123"
console.log(String(21.5));        //"21.5"
console.log(String(-0));          //"0"
console.log(String(true));        //"true"
console.log(String(false));       //"false"

/*02-2 toString函数
* a、其实其它类型的值直接调用toString方法也能强转为字符串
* b、toString方法可以接收一个参数，该参数用于表示转换时的进制数
* c、如果toString方法的参数缺省，那么默认采用的十进制
* d、null和undefined值无法调用toString方法
* */
console.log((123).toString());   //"123"
console.log(true.toString());    //"true"
console.log(false.toString());   //"false"
console.log(NaN.toString());     //"NaN"

/*03 Boolean函数用于将其它类型转换为字符串*/
console.log(Boolean(null));      //false
console.log(Boolean(undefined));//false
console.log(Boolean("Nice"));    //true
console.log(Boolean(""));        //false
console.log(Boolean(" "));       //true
console.log(Boolean(123));       //true
console.log(Boolean(0));         //fasle
console.log(Boolean(NaN));       //fasle
```

在上面的代码示例中Number函数用于将其他类型的数据转换成数字，而 `parseInt()` 和 `parseFloat()` 函数相对于Number()函数而言更加灵活。

`parseInt()` 函数用于解析整数，如果字符串前缀是0x或0X，则将会被解析为十六进制数。解析规则为：跳过任意数量的前导空格，尽可能解析更多数值字符，并忽略数字后面的内容，如果第一个非空格字符是非法的数字直接量，将最终返回NaN。

```
/*parseInt()基本使用*/
 console.log(parseInt(" 123"));         //123 忽略前面的N个空格
 console.log(parseInt("123"));          //123
 console.log(parseInt("123.59"));       //123 仅解析为整数
 console.log(parseInt("-13.14"));       //-13
 console.log(parseInt("826 Birthday"));//826
 console.log(parseInt("Birthday826"))  //NaN
 console.log(parseInt("0.1"))           //0
 console.log(parseInt(".1"))            //NaN 整数不能以.开头
 console.log(parseInt("0xff"))          //255 以十六进制来解析
 console.log(parseInt("071"));          //71  十进制处理(非八进制)
```

> `parseInt()` 被定义为 `declare function parseInt(s: string, radix?: number): number;` 该函数的第一个参数为字符串，它还可以接收第二个参数用于指定数字转换的进制基数，合法的取值范围是2~36。

```
console.log(parseInt("111",2));       // 7  = 1 * 2 * 2 + 1 * 2 + 1
console.log(parseInt("aa",16));       //170 = 10 * 16 + 10
console.log(parseInt("076",8));       //62  = 7 * 8 + 6
console.log(parseInt("077",10));      //77
```

`parseFloat()` 函数用于解析浮点数。解析规则为：跳过任意数量的前导空格，检索纯数字字符串后面第一个.后的不为数字的字符，并对之前所有的结果进行返回，如果第一个非空格字符是非法的数字直接量，将最终返回NaN，如果没有.则以整数解析的方式处理。

```
/*parseFloat()基本使用*/
console.log(parseFloat(" 123"));            //123
console.log(parseFloat(" 123.55"));         //123.55
console.log(parseFloat(" 8.26 Birthday"));  //8.26
console.log(parseFloat(" Birthday 8.26"));  //NaN
console.log(parseFloat("0.1"));             //0.1
console.log(parseFloat(".1"));              //0.1
console.log(parseFloat("0xff"));            //0
console.log(parseFloat("abc 12.5"));        //NaN
console.log(parseFloat("$12.5"));           //NaN
```

值	数值	字符串	布尔值
undefined	NaN	“undefined"	false
null	0	“null"	false
true	1	“true"	
false	0	“false"	
“”	0		false
“18”	18		true
“18blue”	NaN		true
“blue18"	NaN		true
“abc"	NaN		true
0		“0"	false
-0		“0"	false
NaN		“NaN"	false
Infinity		“Infinity"	true
123		“123"	true

- Posted by 博客园·文顶顶 | 花田半亩
- 联系作者 简书·文顶顶 新浪微博·Coder_文顶顶
-