Types, 原始类型 vs 引用类型

number

NaN：一个不是数字的数字,它跟任何数作比较都不相等，跟自己作比较也不相等

string

null

null:用于给指定的变量进行内存释放

undefined

boolean

symbol (ES6):

创建 symbol 最简单的方式是调用 **Symbol()** 方法.

任何两个 sybmols 都不相等.

const symbol1 = Symbol();
const symbol2 = Symbol();
symbol1 === symbol2; // false

调用Symbol()来创建一个新的Symbol值，它不会等同于其他值。

与字符串和数字一样，你可以使用symbol来作为属性值。因为它不等同于其它任何字符串，这个symbol-keyed属性可以确保不会与其它任何属性冲突。

var mySymbol = Symbol();
obj[mySymbol] = "ok!"; // guaranteed not to collide
console.log(obj[mySymbol]); // ok!

基本运用：

let name = Symbol('xiaohesong')
typeof name // 'symbol'
let obj = {}
obj[name] = 'xhs'
console.log(obj[name]) //xhs

**symbol for**

这个东西是可共享，在创建的时候会检查全局是否寻在这个key的symbol.如果存在就直接返回这个symbol,如果不存在就会**创建**，并且在全局注册。

let uid = Symbol.for("uid");
let object = {
[uid]: "12345"
};
console.log(object[uid]); // "12345"
console.log(uid); // "Symbol(uid)"
let uid2 = Symbol.for("uid");
console.log(uid === uid2); // true
console.log(object[uid2]); // "12345"
console.log(uid2); // "Symbol(uid)"

Symbol不是一个构造函数，如果用new Symbol会报错（Symbol是一个原始类型的值，不是对象。

var symbol = new Symbol(); // 报错：TypeError

有三种方法来获取symbol:

**Call Symbol()**。我们已经讨论过这种方法了，每一次调用它都将返回一个唯一的symbol。

**Call Symbol.for(string)**。这种方法访问一组已经存在的symbol注册表。与通过Symbol()来定一个唯一值不同的是，symbol注册表中的symbols是共享的。如果你调用Symbol.for(“cat”)三十次，每一次返回都将是同一个symbol。在多页面或者单页面的多模块需要共享symbol时，这是很有效的方法。

**使用标准中定义的Symbol.iterator**。标准委员会自己定义了几种symbols。每一种都有它的特殊意义。

+引用类型：Object

type of 运算符：主要是用来判断原始类型的类型名称

instance of 运算符：主要是用来判断引用类型的类型名称

严格模式 vs 非严格模式

严格模式的使用很简单，只有在代码首部加入字符串  "use strict"。必须在首部即首部指其前面没有任何有效js代码除注释，否则无效。

'==' vs '==='

==在进行比较时，如果比较的双方数据类型不同，通常会先转换成相同的类型再进行比较。如果比较的双方类型相同，这时与[===](http://ks-test.hz.netease.com/blog?id=849#strictEqual)相同：如果比较的双方是对象类型，只有指向同一个对象，才能相等;如果是其他类型，比较值是否相等。

// null和undefined相等
null == undefined;
// null/undefined比较时不会做隐式类型转换
0 != null;

**优先转换成Number型**原则, false转换成Number型，等于 Number(false)，等于0, "0"转换成Number型，等于Number("0"), 等于0, 值相等。' \n\n\n' == false类似，Number(' \n\n\n')等于0, Number(false)等于0, 值相等。

**==和===的异同点：**

比较双方都是对象时，只有指向同一个对象才会相等(包含==/===)。

===要求比较双方类型相同并且值相等。

==在比较双方类型不同的时候通常会进行隐式类型转换。

**易于记忆的==比较的原则：**

null==undefined, null/undefined不进行隐式类型转换。

进行隐式类型或转换时，优先转换成Number型。

'var' vs 'let' vs 'const'

当以var来定义变量，作用域是函数。

当以let来定义变量，作用域是定义它的代码块。

const用于定义常量，作用域也是块，即只在定义它的块中可访问。

variable hoisting

function exampleFunc() {
console.log( expFunc ); // function expFunc() {alert( 'second' )}
function expFunc() {
alert( 'first' );
}
function expFunc() {
alert( 'second' );
};
console.log( expFunc ); // function expFunc() {alert( 'second' )}
var expFunc = 'str';
console.log( expFunc ); // str
}
exampleFunc();

遍历函数体，寻找声明的变量，并按一定的规则生成这些变量放在作用域中（注意这里仅仅是生成变量，忽略等号后面赋值语句，即使声明和赋值写在一起）。

先搜寻函数声明，并以函数名创建变量，如果有多个同名函数声明，取最后一个为准。

再搜寻 var 声明，如果查询到 var 声明的变量已经在上一步函数证明过程中占用了，则不做任何操作。如果没有占用再创建一个变量。

对第一步中的函数（名）变量进行函数初始化。

Array

**数组尾部操作 ：**

push() 向数组的尾部添加一个或多个数据 arr.push(1,78,…) 返回数组新的长度值

pop() 将数组尾部的数据删除 返回删除的元素

**数组头部操作 ：**

unshift() 向数组的头部添加一个或多个元素 arr.unshift(2,34,…) 返回数组新长度值

shift() 删除数组头部元素 arr.shift() 返回删除的元素

**数组中间数据操作（也适用于头尾） ：** 可以 实现 添加 、 删除、 替换 操作

splice( start , len )删除，表示从start位置处删除指定len 长度的数据

splice( start , len , num1,num2,… ) 第三个参数的个数不固定

如果len的值是0 实现在start位置处添加一个或多个数据

如果len的值不为0 ，实现替换操作（替换过程 ： 先删除 后添加 ）

上面的五个方法 都会改变原数组

slice(start,end) 截取从start位置到end-1 位置处的数据，返回截取后的数据

String template

`${a}`

Function, Arrow function vs regular functions

1、普通函数中，this指向其函数的直接调用者；

2、箭头函数中，this指向其定义环境，任何方法都改变不了其指向，如call（）、bind（）等；

3、构造函数中，如果不使用new,则this指向window，如果使用new创建了一个实例，则this指向该实例。

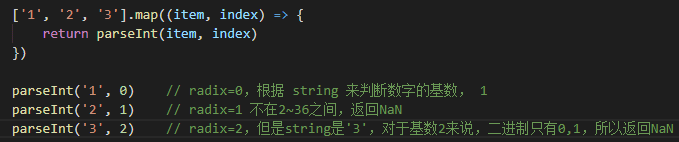
result of ["1", "2", "3"].map(parseInt), why

[1, NaN, NaN]

parseInt(‘1’, 0); // radix为0时，使用默认的10进制。

parseInt(‘2’, 1); // radix值在2-36，无法解析，返回NaN

parseInt(‘3’, 2); // 基数为2，2进制数表示的数中，最大值小于3，无法解析，返回NaN



Map

Map是一组键值对的结构，具有极快的查找速度。

由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉.

var m = new Map(); // 空Map
m.set('Adam', 67); // 添加新的key-value
m.set('Bob', 59);
m.has('Adam'); // 是否存在key 'Adam': true
m.get('Adam'); // 67
m.delete('Adam'); // 删除key 'Adam'
m.get('Adam'); // undefined
var m = new Map();
m.set('Adam', 67);
m.set('Adam', 88);
m.get('Adam'); // 88

Set

bit operations, | ^ ~ &, etc

位运算 XOR 由符号（^）表示，当然，也是直接对二进制形式进行运算。XOR 不同于 OR，当只有一个数位存放的是 1 时，它才返回 1。

var iResult = 25 ^ 3;
alert(iResult); //输出 "26"

25 和 3 进行 XOR 运算的结果是 26：

25 = 0000 0000 0000 0000 0000 0000 0001 1001
3 = 0000 0000 0000 0000 0000 0000 0000 0011
---------------------------------------------
XOR = 0000 0000 0000 0000 0000 0000 0001 1010

可以看出，在两个数字中，共有 4 个数位存放的是 1，这些数位被传递给结果。二进制代码 11010 等于 26。

对左移而言，移动正数和负数规则是相同的；对于右移而言，则有些差别，正数补0，负数补1。