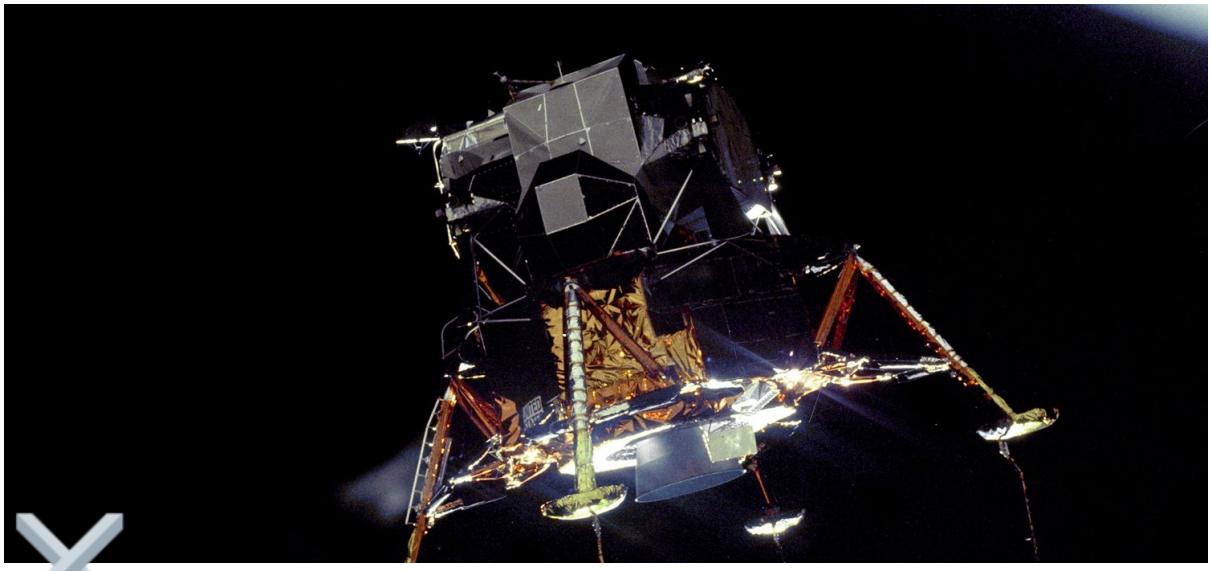


1. 基本数据类型

数据类型

1. **复杂数据类型(引用类型)**, 由于存储数据类型可能复杂,存储后的内容通过地址(16进制)引用进行修改,额外开辟新的空间(堆内存),而不会存储在当前作用域.

   引用类型: 对象 `Object` (数字, 日期....), 函数 `Function`, (在ES6中加入了唯一值 Symbol)

   堆内存: 用于存放引用数据类型,如: 对象内存放键值对,函数体内存放代码字符串

基本数据类型

基本数据类型	Property	1	2	3
数字 `Number`	整数	浮点数	Infinity	-Infinity
数字 `Bigint`	大于 (2^53-1)	小于-(2^53-1)	0	0
字符串 `String`	Untitled		0	0
布尔 `Boolean`	Untitled		0	0
`null`	自成类型		0	0
`undefined`	自成类型		0	0
`symbol`	唯一值		0	0
引用数据类型	Untitled			
	Untitled			

> Tip: 通过tyepof 去判断null类型会得到Object,但这是Javascript本身的问题,去输出函数如alert会得到function类型,但是在Javascript并不存在function类型

数字类型

注意点:

1. 小数计算,应该避免相等性检查.这与计算机存储数字的方式有关,即二进制无法精确的存储0.1这种小数.
2. 使用小数时,要记住小数会有精度问题.
3. `isFinite()` 用于判断一个变量是否为有效数字(非infinity,-infinity,NaN)
4. NaN跟谁都不相等(除了有效数字以外的任何东西, 泛指所以跟谁都不相等)
5. [number value].toFixed([n]) 保留多少位小数

检测:

isNaN(): 检测一个值是否为“非有效数字” **Number()**:

1. 把其他数据类型转换为数字类型,出现非有效数字则为NaN

`String`: 数字直接转换, 空字符串转为0, 其他非有效数字NaN

`Boolean`: false⇒0, true⇒ 1

`null`: 0

`undefined`: NaN

`BigInt`: 正常转换

`Symbol`: 报错

`Object`: 空数组

`[]`转换为0，其他对象遵循**隐式转换**规则

parseInt(string, radix):

语法

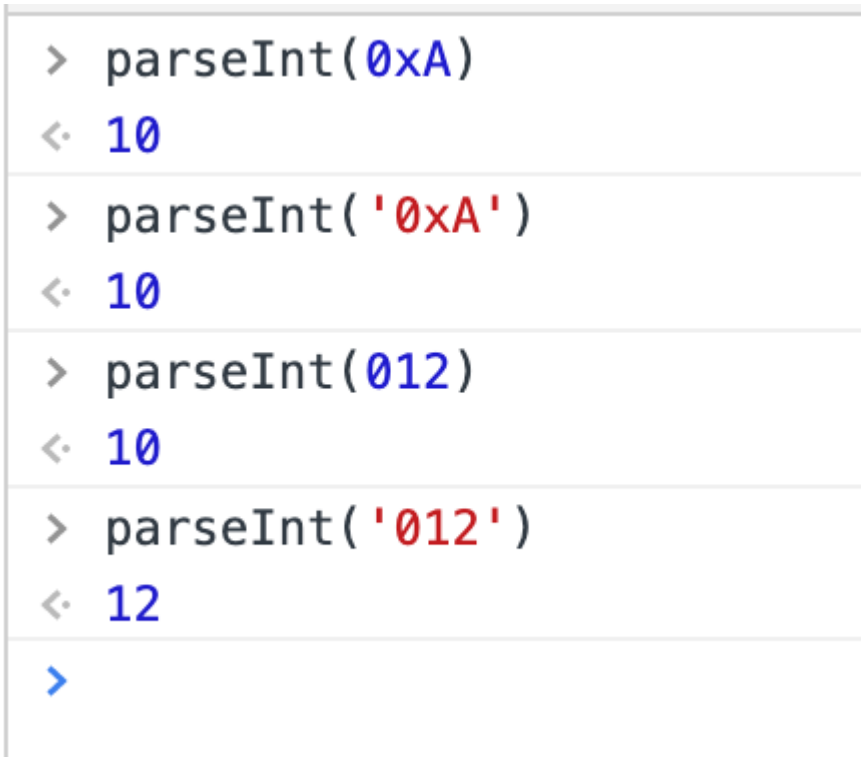
parseInt(*string*, *radix*)

参数	描述
string	必需。要被解析的字符串。
radix	可选。表示要解析的数字的基数。该值介于 2 ~ 36 之间。 如果省略该参数或其值为 0，则数字将以 10 为基础来解析。如果它以 “0x” 或 “0X” 开头，将以 16 为基数。 如果该参数小于 2 或者大于 36，则 parseInt() 将返回 NaN。

1. 将其他类型转换为数字类型
2. 必须保证[value]是一个字符串,如不是则调用toString()
3. 将从[value]第一位开始查找直到遇到第一个非有效数字
4. 第二参数指定的是数字的基数(采用什么进制), 安全值2~36, 该值为0则按照十进制进行解析, 其他的情况直接返回NaN
5. parseInt(0...) 以0开头,在浏览器输出的时候.会被判定为八进制,此时是浏览器做了转换, 而非方法.
6. parseInt(0x..) 以0x开头,则会默认为十六进制

```
parseInt(012) // 符合八进制原则, 输出 10 但此时是浏览器做了转换,而非函数 如下图

parseInt(0xA) // 符合十六进制, 输出10 根据参数函数默认输出16进制
```



parseFloat() 会保留第一个小数点

隐式转换:

引用类型转数字:

1. 获取对象的[Symbol.toPrimitive]属性值
2. 获取对象的的原始值 valueOf()
3. 将对象转换为字符串 toString()
4. 将对象转换为数字 Number()

字符串类型

1. 字符串可通过 `[]` 和 `charAt()` 获取字符串的固定字符,区别在于如果 `[]` 获取不到字符,返回undefined,而 `charAt()` 返回一个空字符串.
2. `for..of` 可以遍历字符串.
3. `slice(start, [end])` 用于字符串的截取.
4. `localeCompare()` 根据当前的语言规则来判断两个字符串的大小.

```
str.substr(n, m) // 从n开始截取 m个字符
```

```
str.substring(n, m) // 从n开始截取到m处, 不包含第m个字符
```

字符串拼接

> 原理同引用类型转数字前三个步骤

如下:

```
{} + 10 => 10 // {} 会被当做一个代码块
({} + 10) => "[object Object]10"
10 + {} => "10[object Object]" // new Number(10) 也是一个对象, 但是他的原始值是10
new Number(10) + 10 => 20
new Number(10)[Symbol.toPrimitive] => undefined
new Number(10).valueOf() => 10
```

Boolean

1. 0, null, NaN, undefined, 空字符串(`''`) 会被转换为false 其余全是true
2. ！[value] 取反
3. !！[value] 相当于 Boolean([value])

Symbol

作用:

1. Symbol类型是唯一值,字面量中使用是给键添加一个`[]`
2. Symbol 能够起到唯一标识的作用且创建的属性会被隐藏,即在循环时,该属性将不会被访问到,或者该库内的脚本,被第三方应用的时候,该属性也不会被访问到,不会引起冲突.
3. 宏观管理标识：保证标志唯一性（vuex/redux）

```
let key = Symbol('aa')
let obj = {    [key] : 'Symbol'    }
let arr = Object.getOwnPropertySymbols(obj)
arr.forEach(item=> {
    console.log(item) // => Symbol(aa)
})
```

底层原理

> 待补充

通过 `Symbol.for()` 进行查询,如果变量不存在,则创建.

```
let id = Symbol.for("id"); // 查询or创建一个description为 id 的Symbol
let id2 = Symbol.for("id"); // id是存在,即此行为查询.
alert( id === id2 )  // true
```

通过 `Symbol.keyFor()` 可以通过变量名查找Symbol.

```
let link = Symbol.for("name"); // 查询or创建一个description为 id 的Symbol
alert(Symbo.keyFor(link)) // name
```

> 注意: Symbol(“id”)与Symbol.for(“id”)是有区别的,前者是局部上的一个唯一值,而后者是全局范围内的.

系统 Symbol

> 这个内容尚未接触,待补充.

基本数据类型操作中要注意的地方

1. 单元运算中,+ 类似于 `Number()` 方法,可以将参数转换为数字

```
+=''  // 0
+= undefined // NaN
+= null // 0
+= " \t \n" // 0
// ------- 同理
+''  // 0
+undefined // NaN
+null // 0
+" \t \n" // 0
```

1. 多元运算中, 出现了字符串或者部分对象时会出现字符串拼接, 其余情况则是转换为Number类型进行相加
2. `i++` 与 `++i`: i++: 变量会先参与运算后自增 ++i: 变量会先自增后参与运算

可选链 “?.”

1. 可选链 “?.” 用于防止访问的值不存在的时候报错,但是实际上用了**可选链**,返回的值仍是undefined,对用户来说还是不友好,但可以自己处理错误.
2. 可选链的结构:

```
let user = {     name = "Link", } // ↓↓这里 alert( user?.name )
```

可见**可选链**是为问号所在节点提供可选,但对 `name` 是不起作用的,也就是说**如果name属性不存在,代码仍会报错**

3. 可选链使用规则 > 注意一下 这里纯属个人推测理解 1) 可选链要求被判断属性的对象必须被声明,但可以为空

```
let user = {};
alert(user?.address) // 代码不会报错

//------------------------------

let user = {};
alert(user.address?.street)  // 同样不会报错,但是address属性,并没有被定义

//------------------------------

alert(user?.address) // 代码报错
```

```
// 记一个复杂度较低的算法, 带到日后刷起 leetcode, 我相信这是我对算法兴趣的开端
function getMaxSubSum(arr) {
  let maxSum = 0
  let partialSum = 0
  for (let item of arr) {
    partialSum += item
    maxSum = Math.max(maxSum, partialSum)
    if (partialSum < 0) partialSum = 0
  }
  return maxSum
}
alert(getMaxSubSum([-1, 2, 3, -9])) // 5
```