JavaScript基础知识

变量

> 可变的量，在JS中我们使用 `var` 关键词来定义一个变量，而变量存储的值是可以改变的
>
> 在JS中变量本身没有什么意义，仅仅是一个名字而已，我们操作变量其实想要操作的都是它存储的那个值

```
1. //->创建了一个叫做aa的变量,并且给变量赋值为1
2. var aa = 1;
3. console.log(aa);
4.
5. aa = 2;
6. console.log(aa);
```

常量

> 相对于变量来说，常量是不会改变的，我们可以把JS中的数据值理解为常量，例如：1就是数字1，不可能变为其它的，所以它就是常量 `每一个具体的数据类型值都是常量`
>
> 如果和变量对比，我们的常量应该是：定义一个常量名字，给它存储一个值，这个值是不能修改的，在新版本ECMAScript中(ES6/ES7)，我们可以使用 `const` 来定义一个常量

```
const bb = 1;
console.log(bb);

bb = 2;//=>Uncaught TypeError: Assignment to constant variable.
console.log(bb);
```

JS中的命名规范

> 从现在开始做一名有职业操守的IT编程者：养成规范的命名习惯

1、在JS中是严格区分大小写的

```
var test = 12;
var Test = 13;
console.log(test);//=>12 两个变量是不同的,JS严格区分大小写
```

2、命名的时候遵循 驼峰命名法

> 一个名字如果是由多个有意义单词组成的，那么第一个单词首字母小写，其余每一个有意义单词的首字母都要大写

```
//=>设置的名字一定要有意义，让别人看到名字大概就了解到变量所代表的含义了
//=>[正确]
var studentInformation;
var studentInfo; //=>Info代表的就是Information

//=>[错误]
var xueshengInfo;
var xsxx;
```

行业中常用的一些短词组：

- info：information 信息
- imp：important 重要的
- init：initially 初始化、最初的
- del：delete 删除

- rm：remove 移除
- add：增加
- insert：插入
- create：创建
- fn：function 函数
- update：修改
- select：查询选择
- query：获取
- get：获取
- con：content 内容 、容器
- …

3、可以使用数字、字母、下划线、$来命名，但是数字不能作为名字的开始

```
var studentInfo;
var studentInfo2;
var student_info;
//var 3studentInfo; //=>错误的，数字不能开头

//--------
var _student; //=>在真实项目中有这样一个约定俗成的规范：以_开头的变量是公共变量(全局变量)，在很多地方都可以获取使用

var $student; //=>真实项目中如果是通过JQ获取的值，那么存储值的变量我们会在前面以$开始
```

4、不能使用关键字和保留字命名

> 关键字：在JS中有特殊含义的，例如：`var`、`for`、`break`、`continue`...
>
> 保留字：未来可能会成为关键字的，例如：`class`

ECMA-262 描述了一组具有特定用途的关键字。这些关键字可用于表示控制语句的开始或结束，或者用于执行特定操作等。按照规则，关键字也是语言保留的，不能用作标识符。以下就是ECMAScript的全部关键字（带*号上标的是第5 版新增的关键字）：

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		

ECMA-262 还描述了另外一组不能用作标识符的保留字。尽管保留字在这门语言中还没有任何特定的用途。但它们有可能在将来被用作关键字。以下是ECMA-262 第3 版定义的全部保留字：

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	

第5 版把在非严格模式下运行时的保留字缩减为下列这些：

class	enum	extends	super
const	export	import	

在严格模式下，第5 版还对以下保留字施加了限制：

implements	package	public	interface
private	static	let	protected
yield			

JS中的数据类型

- 基本数据类型(值类型)

    - number 数字
    - string 字符串
    - boolean 布尔
    - null
    - undefined

- 引用数据类型
    - object 对象数据类型
        - {} 对象
        - [] 数组
        - /^$/ 正则
        - Math 数学函数
        - Date的实例
        - …

    - function 函数数据类型

```
12 -12 -12.5 0 =>number类型的
'' "" =>单双引号包起来的都是字符串
true false =>boolean布尔类型
null 空(没有)
undefined 未定义(没有)

{name:'zxt',age:28} =>对象
[12,23,34] =>数组
/^-?(\d|([0-9]\d+))(\.\d+)?$/ =>正则

function fn(){
    //=>函数
}
```

number数字类型

> 在JS中除了传统的数字，NaN也是number类型的值
>
> NaN：not a number 不是一个数，但是属于number类型的
>
> isNaN：这个方法是用来检测当前的值是否不是一个有效数字的，如果检测的值不是有效数字返回TRUE，是有效数字返回FALSE

```
1. console.log(isNaN(NaN));//=>TRUE
2. console.log(isNaN(1));//=>FALSE
3. console.log(isNaN('1'));//=>FALSE 它是有效数字:当浏览器发现我们检测的值不是NUMBER类型的时候,首先会默认的把值转换为NUMBER类型,然后再验证是否是有效的数字  '1'->1  isNaN(1)->false
4. console.log(isNaN(true));//=>首先把布尔类型转换为数字 TRUE->1 FALSE->0  最后的结果是isNaN(1)->false
5. console.log(isNaN(false));//=>FALSE
6. console.log(isNaN(null));//=>NULL转换为数字0 =>FALSE
7. console.log(isNaN(undefined));//=>UNDEFINED转换为数字的NaN =>TRUE
```

`Number()`

把其它数据类型的值转换为number类型；
isNaN在检测的时候，浏览器默认把其它类型转换为number类型，使用的就是这个方法

```
Number(true) =>1
Number(false) =>0
Number(null) =>0
Number(undefined) =>NaN

Number('') =>0
Number('12') =>12
Number('12.5') =>12.5
Number('true') =>NaN
Number('12px') =>NaN
=>使用Number把字符串转换为数字的时候，空字符串是零，其它字符串中如果出现的字符代表纯数字可以转为正常的数字，如果出现了任何一个非有效数字的字符，最后的结果都是NaN

Number({name:'zxt'}) =>NaN
Number({}) =>NaN
Number([12,23]) =>NaN
Number([12]) =>12
Number(['aa']) =>NaN
Number([]) =>0
Number(/^$/) =>NaN
Number(function(){}) =>NaN
=>使用Number把引用数据类型转为数字类型的时候，先把引用类型转换为字符串(toString)，然后再把字符串转为数字

({name:'zxt'}).toString() ="[object Object]"
({}).toString() ="[object Object]"
[12,23] => "12,23"
```

```
[12] => "12"
[] => ""
['aa'] =>"aa"
```

`parseInt()`

> 也是把其它数据类型转换为数字，整体情况和Number用法一样，区别在于：在转换字符串的时候，Number是只要出现一个非有效数字字符结果就是NaN，parseInt没有这么霸道，它能把有效的部分识别出来转为数字，非有效的部分直接忽略掉

```
Number('12px') =>NaN
parseInt('12px') =>12
parseInt('12px13') =>12   在查找转换的时候，按照从左到右的顺序依次查找，一直到遇到一个非有效数字字符结束（不管后面是否还有有效数字字符，都不在继续查找），把找到的转换为数字
parseInt('px13') =>NaN
parseInt([12,13]) =>12
```

`parseFloat()`

> 用法和parseInt一样，区别在于，parseFloat可以识别小数点

```
1. parseInt('12.5px') =>12
2. parseFloat('12.5px') =>12.5
3. parseFloat('12.5.8px') =>12.5
4. parseFloat('px12.5') =>NaN
```

JS这门语言是松散类型的

> 在JS中创建变量直接使用`var/const/let`定义即可，可以存储任何数据类型的值
>
> 后台语言(JAVA)，创建变量，会根据存储值的不同使用不同的关键字来创建
> int ：整数
> float：短浮点
> double：长浮点
> Array：创建数组
> Object：创建对象
> …

`toFixed()`

> 控制数字保留小数点后面几位

```
12.5.toFixed() =>不写参数,相当于不留小数点，会把数字四舍五入到整数上 =>'13'
12.4.toFixed(0) =>'12'
12.4.toFixed(2) =>'12.40'
Math.PI.toFixed(2) =>'3.14'

Math.PI.toFixed(-2) =>Uncaught RangeError: toFixed() digits argument must be between 0 and 20
```

思考题：parseInt / parseFloat 都支持第二个参数`parseInt('12px',10)`获取后自己查找第二个参数的作用！

boolean布尔类型

> 只有两个值：true真/false假

`Boolean()`

> 把其它数据类型转化为布尔类型
>
> 只有`0、NaN、空字符串、null、undefined`五个会转换为false，其余的都会转换为true

```
1. Boolean(1) =>true
2. Boolean(0) =>false
3. Boolean(-1) =>true
4. Boolean('') =>false
5. Boolean('xxx') =>true
6. Boolean(null) =>false
7. Boolean(undefined) =>false
8. Boolean({}) =>true
9. Boolean([]) =>true 除了那五个其余都是TRUE
```

`!` 或者 `!!`

> 取反，把其它数据类型先转换为布尔类型，然后再取反
> ！：取一次反
> !!：取两次反(相当于没有取反，只剩把其它类型的值转换为布尔类型，和Boolean是相同的效果)

```
1. !null => true
2. !!undefined =>false
3. ![] =>false
4. !![] =>true
```

null和undefined

> null：空对象指针，但它不是对象类型的，而是基本类型的，表示为空或者没有
>
> undefined：未定义，也代表没有

`0或者空字符串` 和 `null或者undefined` 的区别

> 0或者空字符串：挖了坑没种树
> null或者undefined：连坑都没有挖
>
> 在JS中null属于没有开辟内存，而空字符串是开辟了内存，里面没有存内容而已，null消耗的性能更低

`null` 和 `undefined` 的区别

> null：意料之中的没有，一般都是当前暂时没有，后期基本上会有
> undefined：意料之外的没有，一般都是当前没有，以后可能有可能没有，但是规划中是不计后面有没有的
>
> 唐元帅（男）
> 他的女朋友是null
> 他的男朋友是undefined

object对象数据类型

```
var obj={
    name:'zxt',
    age:28,
    sex:'man',
    friend:['tom','jerry','li lei','han mei mei']
};
```

每一个对象数据类型值，都是由零到多组 `属性名` 和 `属性值` 组成的
属性名：描述当前对象具备这些特征（数字或者字符串格式）
属性值：描述某个特征具体的样子（任何数据类型都可以）

对象是由零到多组`键(key:属性名)值(value:属性值)对`组成的，每一组之间用逗号分隔

创建对象

字面量创建方式：var obj={}
实例创建方式：var obj=new Object();

```
var obj={name:'zxt'};//=>不仅可以创建空对象，还可以在创建的时候就增加一些键值对

var obj2=new Object(); //=>空对象
```

对象键值对的操作：增、删、改、查

```
var obj = {};
obj.name = 'zxt';//=>增加一个叫做NAME的属性,属性值是:'zxt'
obj['age'] = 28;//=>增加一个叫做AGE的属性,属性值是:28

obj['age'] = 29;//=>修改AGE对应的属性值：一个对象的属性名是不能重复的(唯一性)，之前没有这个属性，我们的操作是增加操作，之前有这个属性，当前操作就是在修改现有属性名的属性值
obj.age = 30;

obj.age = null; //=>假删除：把属性值设置为空，但是属性名是存在的 <=> obj['age'] = null =>获取age的属性值结果是null
delete obj.age;//=>真删除：把属性名和属性值彻底从对象中移除掉 =>获取age的属性值结果是undefined
//=>获取一个对象某一个属性名对应的属性值，如果当前这个属性在对象中并不存在，获取的结果是undefined

console.log(obj.name);//=>获取NAME属性的值
console.log(obj['name']);
```

总结1：

操作一个对象的属性有两种：

`对象.属性名`：obj.name

`对象[属性名]`：obj[‘name’] 属性名只能是数字或者字符串，如果是字符串的话，需要加单双引号

特殊情况：属性名是数字

var obj={0:’zhufeng’};

obj.0 =>Uncaught SyntaxError: Unexpected number 数字属性名不能使用点的方式处理

obj[0] / obj[‘0’] =>使用这种方式是没有问题的，属性名为数字，也就没有必要在加单双引号了

思考：

obj[age] 和 obj[‘age’] 的区别?

```
//-> age: 变量名，代表的是它存储的值
//-> 'age': 常量，字符串的具体值
var age = 'name';
var obj = {
    name:'zhufeng',
    age:8
};
console.log(obj.age); =>8
console.log(obj['age']); =>8
console.log(obj[age]); => obj[age变量]
=>obj['name'] =>获取name属性名的属性值 =>'z
hufeng'
```

总结2：

> 对象的属性名是唯一的，一个对象的属性名不能重复；获取某个属性名对应属性值的时候，如果属性存在，获取值即可，如果属性不存在，获取的属性值是 undefined；

数据类型检测

- typeof ：用来检测数据类型的运算符
- instanceof ：用来检测当前某一个实例是否属于这个类的运算符
- constructor ：检测当前实例所属类的构造器的属性
- Object.prototype.toString.call() ：检测数据值所属类的方法

`typeof`

> typeof [value] ：返回的是当前[value]的数据类型（`这个类型是一个字符串格式的`），例如：”number”、”boolean”、”string”、”object”…

```
typeof 12 ="number"
typeof NaN ="number"
typeof true ="boolean"
typeof 'zhufeng' ="string"
typeof null ="object"
typeof undefined ="undefined"

typeof {} ="object"
typeof [] ="object"
typeof /^$/ ="object"
typeof function(){} ="function"
```

局限性：

- typeof null =>”object” 检测null的时候返回的是”object”，但是null不是对象数据类型的
- typeof 不能具体细分是大括号普通对象还是数组或者正则，因为检测这些值返回的结果都是”object”

腾讯面试题：

```
console.log(typeof typeof typeof []);
//typeof [] ->'object'
//typeof 'object' ->'string'
//typeof 'string' ->'string'
=>'string'
```

基本数据类型(值类型)和引用数据类型的本质区

别 (很很很重要)

> 当我们把JS代码放在浏览器中运行的时候，浏览器会提供给JS一个赖以生存的环境(执行代码的环境)，我们这个环境叫做`全局作用域(window[前端]/global[后台])`
>
> JS代码会在全局作用域下自上而下执行

```
var a=12;
var b=a; //=>把变量A存储的值赋值给变量B
b=13;
console.log(a);//=>12
```

```
var a={name:'tom'};
var b=a;
b.name='lucy';
console.log(a.name);//=>'lucy'
```

基本数据类型之所以称之为值类型是因为：基本数据类型的值在进行赋值操作的时候，`是直接按照值来操作的`，例如：var a=12; 它是把12这个值直接的赋值给变量a

引用数据类型`是按照引用地址操作的`，不是按照值操作的

var obj = {name:'zhufeng'};
1：创建一个变量叫做obj

2：由于引用数据类型要存储的内容可能有很多，所以浏览器遇到{}或者[]等

- 首先会开辟一个新的存储空间（为了方便后期找到这个空间，给空间设置了一个16进制的地址）
- 对于对象数据类型来说，会把对象中的键值对依次存储到新开辟的空间中

3：最后在把新开辟空间的地址赋值给当前创建的变量，所以：变量存储的不是对象具体的值，而是对象开辟的那个新空间的引用地址