Nodejs

Nodejs是服务器端运行JavaScript的开源、跨平台运行环境。

Nodejs原始作者瑞安·达尔(Ryan Dahl),于2009年发布,使用了V8引擎,并采用事件驱动、非阻塞、异步IO模型。

2010年,npm软件包管理器诞生,通过它,可以方便的发布、分享Nodejs的库和源代码。

Nodejs 4.0引入了ES6语言特性。

我们学习JS,就让它跑在最新版的Nodejs上,为了调试方便,也为了使用最新的ES2017特性。

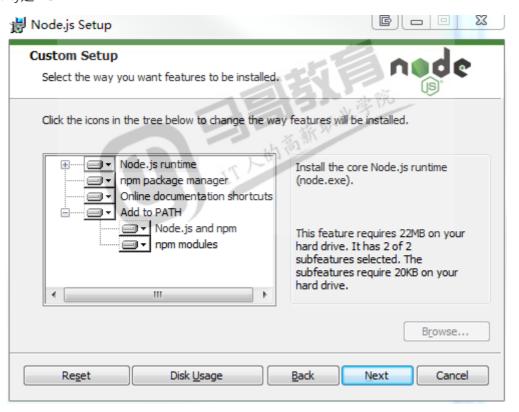
安装

国内可以去阿里云镜像站

https://npm.taobao.org/mirrors/node

Linux解压即可运行。Windows版安装即可。

目前14.x.y是LTS



msi安装会增加path路径

全局安装目录 C:\Program Files\nodejs\

本用户目录 C:\Users\Administrator\AppData\Roaming\npm

\$ node -v 查看版本

开发

文档

搜索MDN,Mozilla Developer Network,提供非常完善HTML、CSS、JS等的技术资料。 https://developer.mozilla.org/zh-CN/docs/Web/JavaScript

指南 https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide 非常好的JS文档

使用任何一种文本编辑器,都可以开发JS,此次使用微软的Visual Studio Code开发。

Visual Studio Code

下载 https://code.visualstudio.com/Download

支持windows、mac、Linux平台。

新版VS Code Windows版分为System 和 User两个版本,当前用户使用安装User版即可。

有代码自动完成功能,还可以安装Node exec插件,将写的js跑在nodejs上。

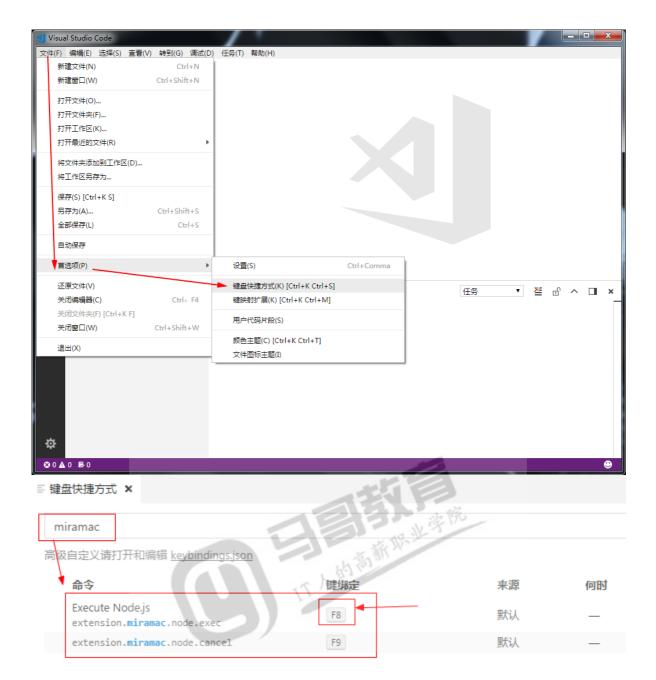


Node Exec插件快捷键: F8运行js脚本, F9停止

前端开发中,JS脚本一般来说是为了控制浏览器的网页的,这里使用了VSCode,只是为了开发调试方便

解决快捷键冲突

F8和某些软件冲突,无法使用,例如某些词典软件。 可以通过调整VSCode的快捷键设置。当然可以修改其他软件的快捷键。



注释

和C、Java一样 // 单行注释

/* 注释 */ 多行注释, 也可以用在语句中

```
1 | str = 'hello' + /*comment*/' magedu'
2 | console.log(str)
```

常量和变量

标识符

标识符必须是字母、下划线、美元符号\$和数字,但必须是字母、下划线、美元符号开头,依然是不能数字开头就行。

标识符区分大小写。

声明

var 声明一个变量 let 声明一个块作用域中的局部变量 const 声明一个常量

```
1 var y //只是声明, y值为undefined
2 var x = 5 // 规范的声明并初始化, 声明全局或局部变量。
3 z = 6 // 不规范的初始化, 不推荐。在严格模式下会产生异常。在赋值之前不能引用, 因为它没有声明。一旦这样赋值就是全局作用域。
```

```
1 function hello()
2 {
3     var a // 只是声明, a为undefined, 作用域在函数中
4     a = 100
5 }
6
7 console.log(a) // 未声明变量a, 异常
9 //a = 200 // 不能声明提升
10 //let a = 200 // 不能声明提升
11 //var a = 200; hello(); // var声明提升hoisting
```

var会把变量提升到全局或当前函数作用域。

常量和变量的选择:如果明确知道一个标识符定义后不再修改,应该尽量声明成const常量,减少被修改的风险,减少Bug。

数据类型

序号	名称	说明
1	number	数值型,包括整型和浮点型
2	boolean	布尔型,true和false
3	string	字符串
4	null	只有一个值null
5	undefined	变量声明未赋值的; 对象未定义的属性
6	symbol	ES6 新引入类型
7	object类型	是以上基本类型的复合类型,是容器

ES是动态语言, 弱类型语言。

虽然先声明了变量,但是变量可以重新赋值为任何类型。

```
1 // 类型转换
 2
   // 弱类型
 3 console.log('====string=====')
    console.log(a = 3 + 'magedu', typeof(a))
    console.log(a = null + 'magedu', typeof(a))
    console.log(a = undefined + 'magedu', typeof(a))
 6
 7
    console.log(a = true + 'magedu', typeof(a))
8
9
    // 数字
10
   console.log('=====number=====')
    console.log(a = null + 8, typeof(a))
11
12
    console.log(a = undefined + 8, typeof(a)) //undefined没法转换成一个对应的数字
    console.log(a = true + 8, typeof(a))
13
14
    console.log(a = false + 8, typeof(a))
15
    // boolean
16
17
    console.log('====bool=====')
18 | console.log(a = null + true, typeof(a))
19
    console.log(a = null + false, typeof(a))
    console.log(a = undefined + true, typeof(a)) //undefined没法转换成一个对应的数字
20
    console.log(a = undefined + false, typeof(a)) // NaN
21
22
    console.log(a = null & true, typeof(a))
23
    console.log(a = undefined & true, typeof(a))
24
   // 短路
25
    console.log(a = null && true, typeof(a)) // 逻辑运算符, null 直接就是false短路
26
27
    console.log(a = false & null, typeof(a)) // 逻辑运算符, false短路返回false
    console.log(a = false && 'magedu', typeof(a)) // boolean
28
29
    console.log(a = true && 'magedu', typeof(a)) // 字符串
30
    console.log(a = true && '' && 'abc', typeof(a)) // 字符串
31
   // null
32
33
    console.log('=====null=====')
    console.log(a = null + undefined, typeof(a))
34
35
36
   // 短路补充
37 console.log(a = [] && 'abc', typeof(a))
38 console.log(a = \{\} && 'abc', typeof(a))
```

弱类型,不需要强制类型转换,会隐式类型转换。

NaN,即Not a Number,转换数字失败。它和任何值都不等,和自己也不等,只能使用Number.isNaN(NaN)判断。

总结:

遇到字符串,加号就是拼接字符串,所有非字符串隐式转换为字符串。

如果没有字符串,加号把其他所有类型都当数字处理,非数字类型隐式转换为数字。undefined特殊,因为它都没有定义值,所以转换数字失败得到一个特殊值NaN。

如果运算符是逻辑运算符,短路符,返回就是短路时的类型。没有隐式转换。

除非你十分明确,否则不要依赖隐式转换。写代码的时候,往往为了程序的健壮,请显式转换。

注意: 以上的原则不要死记, 忘了就实验, 或者显式的类型转换

字符串

将一个值使用'单引号或者"双引号引用起来就是字符串。

ES6提供了反引号定义一个字符串,可以支持多行,还支持插值。

转义字符

名称	说明
\0	Null字节,空字符
\b	退格符
\f	换页符
\n	换行符
\r	换页符 换行符 回车符
\t	Tab (制表符)
\v	垂直制表符
1	单引号
"	双引号
\	反斜杠字符 (\)
\XXX	由从0到377最多三位八进制数XXX表示的 Latin-1 字符。例如,\251是版权符号的 八进制序列
\xXX	由从00和FF的两位十六进制数字XX表示的Latin-1字符。例如,\xA9是版权符号的十六进制序列
\uXXXX	由四位十六进制数字XXXX表示的Unicode字符。例如,\u00A9是版权符号的 Unicode序列。见Unicode escape sequences (Unicode 转义字符)
\u{XXXXX}	Unicode代码点 (code point) 转义字符。例如,\u{2F804} 相当于Unicode转义字符\uD87E\uDC04的简写

字符串操作方法

字符串操作方法很多, 但和Python类似

```
let school = 'magedu'
 2
    console.log(school.length)
 3
    console.log(school.charAt(2)) // g
    console.log(school[2]) // g
 4
    console.log(school.toUpperCase()) // MAGEDU
    console.log(school.concat('.com')) // 连接
 6
 7
    console.log(school.slice(3)) // 切片, 支持负索引
8
    console.log(school.slice(3,5))
9
    console.log(school.slice(-2, -1))
    console.log(school.slice(-2))
10
11
12
    let url = "www.magedu.com"
    console.log(url.split('.'))
13
14
    console.log(url.substr(7,2)) // 返回子串从何处开始,取多长
15
    console.log(url.substring(7,10)) // 返回子串,从何处开始,到什么为止
16
17
    let s = 'magedu.edu'
    console.log(s.indexOf('ed')) // 3
18
19
    console.log(s.indexOf('ed', 4)) // 7
    console.log(s.replace('.edu', '.com'))
20
    s = ' \setminus tmag edu \setminus r \setminus n'
21
                                     人的海斯根业学院
    console.log(s.trim()) // 去除两端的空白字符。trimLeft、trimRight是非标函数,少用
22
23
    console.log('-'.repeat(30))
```

数值型number

在JS中,数据均为双精度浮点型范围只能在-(2^53-1)和 2^53-1之间,整型也不例外。 数字类型还有三种符号值: +Infinity (正无穷) 、-Infinity (负无穷) 和 NaN (not-a-number非数字)。

二进制0b0010、0B110。

八进制0755。注意0855,将被认作十进制,因为8不在八进制中。ES6中最好使用0o前缀表示八进制。 十六进制0xAA、0Xff。

指数表示1E3 (1000), 2e-2 (0.02)

常量属性

```
1 var biggestNum = Number.MAX_VALUE;
  var smallestNum = Number.MIN_VALUE;
  var infiniteNum = Number.POSITIVE_INFINITY;
  var negInfiniteNum = Number.NEGATIVE_INFINITY;
5 var notANum = Number.NaN;
```

数字的方法

方法	描述
Number.parseFloat()	把字符串参数解析成浮点数,和全局方法 parseFloat() 作用一致
Number.parseInt()	把字符串解析成特定基数对应的整型数字,和全局方法 parseInt() 作用 一致
Number.isFinite()	判断传递的值是否为有限数字
Number.isInteger()	判断传递的值是否为整数
Number.isNaN()	判断传递的值是否为 NaN。Nan唯一判断方式

内置数学对象Math

Math提供了绝对值、对数指数运算、三角函数运算、最大值、最小值、随机数、开方等运算函数,提供 了PI值。

```
1 console.log(Math.PI)
console.log(Math.abs(-1))
3 console.log(Math.log2(16))
4 console.log(Math.sqrt(2))
5 console.log(Math.pow(2, 3))
 console.log(Math.random()) // (0, 1)
```

Symbol类型

ES6提供Symbol类型,内建原生类型。

```
人的商新职业学院
1 let sym1 = Symbol()
2 let sym2 = Symbol('key1')
3 let sym3 = Symbol('key1')
 console.log(sym2 == sym3) // false,symbol值是唯一的
```

1、作为对象的属性key

```
1 let s = Symbol()
2
   let t = 'abc'
3
   let a = {
4
       [s]:'xyz', // symbol做key,注意要使用中括号,这个key一定唯一
5
       t:'ttt',
6
       [t]:'ooo' // 中括号内可计算
7
   }
8
9 console.log(a)
10 console.log(a[s])
11 \mid a[s] = 2000
   console.log(a[s])
12
```

2、构建常量

```
1 // 以前用法。其实有变量名就可以知道代表什么意思
2 var COLOR_RED = 'RED';
3 var COLOR_ORANGE = 'ORANGE';
4 var COLOR_YELLOW = 'YELLOW';
  var COLOR_GREEN = 'GREEN';
```

```
6 var COLOR_BLUE = 'BLUE';
7 var COLOR_VIOLET = 'VIOLET';
8
9 // 现在
10 const COLOR_RED = Symbol();
11 const COLOR_ORANGE = Symbol();
12 const COLOR_YELLOW = Symbol();
13 const COLOR_GREEN = Symbol();
14 const COLOR_BLUE = Symbol();
15 const COLOR_VIOLET = Symbol();
```

运算符

算数运算符

+ - * / % 等运算符和Python一样

```
1 console.log(1/2) // 0.5自然除
 2 console.log(1/0)
 3
   console.log(5 % 3)
 4
    console.log(2 ** 3)
 5
 6 console.log(parseInt(0.1), parseInt(0.5), parseInt(0.6), parseInt(1.2),
    parseInt(1.5), parseInt(1.7))
    console.log(parseInt(-0.1), parseInt(-0.5), parseInt(-0.6), parseInt(-1.2),
    parseInt(-1.5), parseInt(-1.7))
    console.log(Math.floor(0.1), Math.floor(0.5), Math.floor(0.6),
    Math.floor(1.2), Math.floor(1.5), Math.floor(1.7))
    console.log(Math.floor(-0.1), Math.floor(-0.5), Math.floor(-0.6),
    Math.floor(-1.2), Math.floor(-1.5), Math.floor(-1.7))
    console.log(Math.ceil(0.1), Math.ceil(0.5), Math.ceil(0.6), Math.ceil(1.2),
    Math.ceil(1.5), Math.ceil(1.7))
    console.log(Math.ceil(-0.1), Math.ceil(-0.5), Math.ceil(-0.6),
11
    Math.ceil(-1.2), Math.ceil(-1.5), Math.ceil(-1.7))
12
    console.log(Math.round(0.1), Math.round(0.5), Math.round(0.6),
13
    Math.round(1.2), Math.round(1.5), Math.round(1.7))
    console.log(Math.round(-0.1), Math.round(-0.5), Math.round(-0.6),
14
    Math.round(-1.2), Math.round(-1.5), Math.round(-1.7))
```

Math.round()

- 6入, 取绝对值更大的值。1.6取2, -1.6取-2
- 4舍, 取绝对值更小的值。1.4取1, -1.4取-1
- 5, 取数轴向右离自己最近的整数。

++ 和 --

单目运算符,代表变量自增、自减 i++ 先用i,用完之后i再自增加1 ++i i先自增,再使用i

```
1 let i = 0
2 let a = i++
3 console.log(a, i) // 打印什么
4 console.log(a, i++) // 打印什么
5 a = -i++
6 console.log(a, ++i) // 打印什么
```

挑战题

```
1 | let i = 0;
2 | let a = ++i+i+++i;
3 | console.log(a); // 答案是几?
```

此题来自C、C++、Java的面试题

- 1、单目运算符优先级高于双目运算符
- 2、加号+是双目运算符,两边的表达式必须先计算好

```
1 | i = 0;
2 | let a = ++i+i+++i++i; // 等价于 (++i) + (i++) + (i++) + i
3 | console.log(a); // 1 + 1 + 2 + 3
```

比较运算符

```
1 | console.log(100 > 200) // false
2 console.log(100 > '200') // 字符、数字? // false
3 console.log(1000 > '200') // 数字? // true
4 | console.log(1000 > '2') // true
    console.log('1000' > 2) // true
5
    console.log('1000' > '2') // false
6
7
    console.log(1000 > '2a') // false
    console.log(3000 > '2a') // false
8
9
10 // 宽松比较
    console.log(300 == '300') // true
11
12
   console.log('200' == '200') // true
13
   // 严格比较 ===
14
15 console.log(300 === '300') // false
16 | console.log('200' === '200') // true
```

从上面的比较中,我们起先以为前几行是隐式转换为字符串的,但是后来发现转换为数字,当3000 > '2a'比较是犯难了。

使用宽松比较的时候,尽可能确保比较的类型相同,否则会引起隐式转换,而且隐式转换的规则很复杂不好把控。

如果不知道类型是否一致,但是就是要求一定要相等,那么请使用 === 和 !== 。 建议比较的时候,一律使用 === 和 !== 。

逻辑运算符

&&、||、! 与、或、非 这些运算符和其他高级语言都一样,支持**短路**。

位运算

& | ^ ~ << >> 位与、位或、异或、取反、左移、右移,和Python意义一样

三元运算符

console.log(('3' > 30)?'真':'假')

逗号操作符

JS运行多个表达式写在一起

其他

名称	说明
instanceof	判断是否属于指定类型
typeof	返回类型字符串
delete	delete操作符, 删除一个对象(an object)或一个对象的属性(an object's property)或者一个数组中某一个键值(an element at a specified index in an array)。
in	如果指定的属性在对象内,则返回true

```
console.log('a' instanceof String) // false
console.log(1 instanceof Number) // false

a = new String('b')
console.log(a instanceof String) // true
console.log(new Number(1) instanceof Number) // true
console.log(a instanceof Object) // true

console.log(typeof('a')) //string
console.log(typeof 'a') //string
console.log(typeof a) //object
```

instanceof 要求必须明确使用类型定义变量,就是对象必须是new关键字声明创建的。它可以用于继承关系的判断。

typeof就是返回对象的类型名称的字符串。

delete 删除对象、属性、数组元素

```
1 | x = 42;
2 | var y = 43;
 3 let z = 60;
 4 myobj = new Number();
 5 myobj.h = 4; // create property h
   console.log(delete x); // returns true (can delete if declared
   implicitly)
   console.log(delete y); // returns false (cannot delete if declared
   with var)
                            // returns false
8 console.log(delete z);
   console.log(delete Math.PI); // returns false (cannot delete predefined
    properties)
   console.log(delete myobj.h); // returns true (can delete user-defined
   console.log(delete myobj); // returns true (can delete if declared
11
   implicitly)
12
   console.log('~~~~~~')
13
var trees = new Array("redwood", "bay", "cedar", "oak", "maple");
15 | for(var i=0;i<trees.length;i++)
       console.log(trees[i])
16
17
   console.log('======')
   delete trees[3]; // 数组中元素被删除,但空着的位置是undefined
18
19 | for(var i=0;i<trees.length;i++)
20
       console.log(trees[i])
```

in 判断属性是否在对象内

```
let trees = new Array("redwood", "bay", "cedar", "oak", "maple");
console.log(0 in trees); // returns true , 0在数组对象的index中
console.log(3 in trees); // returns false, 6不在数组对象的index中
console.log(6 in trees); // returns false, 6不在数组对象的index中
console.log("bay" in trees); // return false, bay不是属性,它是值
console.log("length" in trees); // returns true, length是对象的属性
console.log('~~~~~~~')

delete trees[3];
console.log(3 in trees); // return false
```

```
11 for(var i=0;i<trees.length;i++)</pre>
12
        console.log(trees[i]);
13 console.log('~~~~~')
14
15
    // Custom objects
16
    let mycar = {
        color: "red",
17
18
        year: 1998
19
    };
20 console.log("color" in mycar); // returns true
21 console.log("model" in mycar); // returns false
22 console.log('year' in mycar) // true
```

运算符优先级

运算符由高到低,顺序如下



单目>双目>三目>赋值>逗号

逗号运算符优先级最低, 比赋值语句还低。

记不住,就使用括号。

表达式

基本表达式,和Python差不多

解析式也和Python的相似,但在ES6中非标准不推荐使用 生成器推荐使用生成器函数,ES6开始支持

```
function * inc() {
1
2
       let i = 0, j = 7
       while (true) {
4
           yield i++
5
           if (!j--) return 100
6
       }
7
   }
8
9 let gen = inc()
10 for (let i=0;i<10;i++)
11
        console.log(gen.next())
```

每次调用next() 方法返回一个对象,这个对象包含两个属性: value 和 done, value 属性表示本次 yield 表达式的返回值, done 属性为布尔类型。done是false表示后续还有yield语句执行,如果执行完成或者 return后, done为true。

