# JAVASCRIPT笔记

## 数据类型

### Number

JavaScript不区分整数和浮点数，统一用Number表示，以下都是合法的Number类型：

123; *// 整数123*

0.456; *// 浮点数0.456*

1.2345e3; *// 科学计数法表示1.2345x1000，等同于1234.5*

-99; *// 负数*

NaN; *// NaN表示Not a Number，当无法计算结果时用NaN表示*

Infinity; *// Infinity表示无限大，当数值超过了JavaScript的Number所能表示的最大值时，就表示为Infinity*

### 字符串

ES6新增多行字符串的表示方法 ` 使用反引号标识多行字符串，反引号位于数字1的左面 `

ES6新增模板字符串: var name=’java’; alert(‘你好，我叫${name}’);

字符串也可以通过索引访问字符，类似数组的检索方法

### 布尔值

**false** == 0; // **true**

**false** === 0; // **false**

**NaN与任何值都不相等，包括它自己**

NaN === NaN; *// false*

*判断NaN的唯一函数方法是 isNaN(NaN) //true*

### null和undefined

null表示一个“空”的值，它和0以及空字符串''不同，0是一个数值，''表示长度为0的字符串，而null表示“空”。

在其他语言中，也有类似JavaScript的null的表示，例如Java也用null，Swift用nil，Python用None表示。但是，在JavaScript中，还有一个和null类似的undefined，它表示“未定义”。

JavaScript的设计者希望用null表示一个空的值，而undefined表示值未定义。事实证明，这并没有什么卵用，区分两者的意义不大。大多数情况下，我们都应该用null。undefined仅仅在判断函数参数是否传递的情况下有用。

### 数组

Javascript数组可以包含任意类型的数值

[1, 2, 3.14, 'Hello', null, true];

**数组特性**：对lenth属性直接赋值会改变数组大小

索引越界不会提示异常，会改变数组大小

以上两点会改变数组大小，undefined会对空位填充

**数组的函数：**

Slice() 函数相当于字符串的substring() ,截取部分数组，范围前闭后包

Push()函数 在数组末尾增加元素 pop() 函数 将数组的末尾元素弹出，并返回该元素

Unshift() 在数组头部添加元素 shift() 弹出数组第一个元素

Sort() 对元素进行排序，直接在原数组中进行，按照[默认顺序](#_高阶函数)排序

Reverse() 将数组中元素进行翻转

Splice() 可以从指定索引开始删除若干元素，然后再添加若干元素

**var** arr = ['Microsoft', 'Apple', 'Yahoo', 'AOL', 'Excite', 'Oracle'];

*// 从索引2开始删除3个元素,然后再添加两个元素:*

arr.splice(2, 3, 'Google', 'Facebook'); *// 返回删除的元素 ['Yahoo', 'AOL', 'Excite']*

arr; *// ['Microsoft', 'Apple', 'Google', 'Facebook', 'Oracle']*

*// 只删除,不添加:*

arr.splice(2, 2); *// ['Google', 'Facebook']*

arr; *// ['Microsoft', 'Apple', 'Oracle']*

*// 只添加,不删除:*

arr.splice(2, 0, 'Google', 'Facebook'); *// 返回[],因为没有删除任何元素*

arr; *// ['Microsoft', 'Apple', 'Google', 'Facebook', 'Oracle']*

concat() 数组连接函数 将若干个数组连接起来返回新的数组

join() 将数组元素连接起来，参数接受连接符号，如果元素不是字符串，将自动转换为字符串后再连接

**\*使用字符串与数组连接会直接将数组内容转换为字符串显示**

var arr = ['小明', '小红', '大军', '阿黄'];

console.log('欢迎'+arr+'同学！');

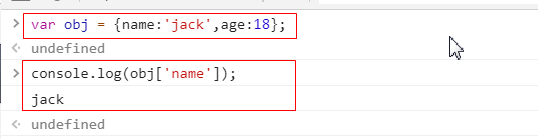
欢迎小明,小红,大军,阿黄同学！

### 对象

无序集合数据类型

键为字符串类型，值为任意类型

访问键的值：



对象属性中的键如果包含特殊字符（如“-”）,需要用‘’括起来：

**var** xiaohong = {

name: '小红',

'middle-school': 'No.1 Middle School'

};

访问属性不能用.来访问，使用[‘xxx’]

xiaohong['middle-school']; // 'No.1 Middle School'

xiaohong['name']; // '小红'

xiaohong.name; // '小红'

访问对象不存在的属性会返回 undefined

对象属性支持动态添加，obj.name=’xxx’ ,如果obj中原来没有name属性，那么这个语句会为obj添加属性name,并且值为xxx

使用in 检测对象中是否包含该属性

‘name’in obj //true

不过要小心，如果in判断一个属性存在，这个属性不一定是xiaoming的，它可能是xiaoming继承得到的：

'toString' **in** xiaoming; // **true**

因为toString定义在object对象中，而**所有对象最终都会在原型链上指向object**，所以xiaoming也拥有toString属性。

要判断一个属性是否是xiaoming自身拥有的，而不是继承得到的，可以用hasOwnProperty()方法：

var xiaoming = {

name: '小明'

};

xiaoming.hasOwnProperty('name'); // **true**

xiaoming.hasOwnProperty('toString'); // **false**

## ES6新增数据类型

传统js对象的键只支持字符串，引入新的数据类型以支持其他数据类型作为键

### Map

Map是一组键值对的结构，具有极快的查找速度。检索速度不会因为数据量增大而变慢

**var** m = **new** Map([['Michael', 95], ['Bob', 75], ['Tracy', 85]]);

m.get('Michael'); *// 95*

**var** m = **new** Map(); *// 空Map*

m.set('Adam', 67); *// 添加新的key-value*

m.has(‘Bob’) //是否存在键Bob

m.delete(‘Bob’) //删除键 再次调用get返回undefined

键一旦重复，会发生值覆盖

### Set

存储不重复的键

**var** s1 = **new** Set(); *// 空Set*

**var** s2 = **new** Set([1, 2, 3]); *// 含1, 2, 3*

s.add(4);

添加和删除方法与Map相同

### Map和Set的遍历 Iterable

遍历Array可以采用下标循环，遍历Map和Set就无法使用下标。为了统一集合类型，ES6标准引入了新的iterable类型，Array、Map和Set都属于iterable类型。

**var** a = ['A', 'B', 'C'];

**var** s = **new** Set(['A', 'B', 'C']);

**var** m = **new** Map([[1, 'x'], [2, 'y'], [3, 'z']]);

**for** (**var** x of a) { *// 遍历Array*

console.log(x);

}

**for** (**var** x of s) { *// 遍历Set*

console.log(x);

}

**for** (**var** x of m) { *// 遍历Map*

console.log(x[0] + '=' + x[1]);

}

更好的迭代方式是使用Iterator内置的forEach方法--- ES5.1开始引入

**1.数组**

var a = ['A', 'B', 'C'];

a.forEach(function (element, index, array) {

// element: 指向当前元素的值

// index: 指向当前索引

// array: 指向Array对象本身

console.log(element + ', index = ' + index);

});

**2.Set**

Set与Array类似，但Set没有索引，因此回调函数的前两个参数都是元素本身

3.Map

Map的回调函数参数依次为value、key和map本身：

中断循环：forEach循环的中断可以转变思路，先使用filter过滤掉不需要循环的元素，在将剩余的元素进行forEach循环

## Strict模式

在strict模式下运行的JavaScript代码，强制通过var申明变量，未使用var申明变量就使用的，将导致运行错误。防止不使用var声明全局变量造成操作结果互相影响。

## 条件判断

### If 判断括号内布尔值的原则

Javascript将 null 、undefined 、0 、NaN 、 和空字符串’’视为false,其他值视为true

### for…in循环

for( var element in obj) 可以遍历对象的所有属性（包括继承过来的属性）

要过滤掉对象继承的属性，用hasOwnProperty()实现

for(var u : obj){

if( obj.hasOwnProperty(u) ){

//得到对象属性（过滤掉了继承而来的属性）

}

}

## 函数

Js函数两种定义方式

function f1(a){ … }

var f1 = function(a){ … };

两种方式等价，第二种后面用分号结束

调用函数时传入参数可以少于或多于定义的个数

按位置对应，没有传入的接收为undefined，超过的用不到

函数中应该对参数检查

**function** abs(x) {

**if** (**typeof** x !== 'number') {

**throw** 'Not a number';

}

**if** (x >= 0) {

**return** x;

} **else** {

**return** -x;

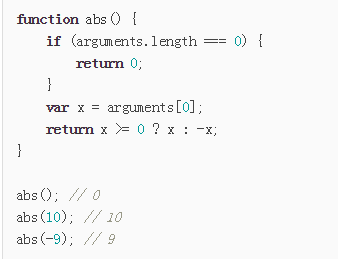
}

}

### arguments 关键字

在方法内部起作用，表示函数接受的参数，类似数组但不是数组，最常用的用法是判断函数接受的参数数量

可以利用arguments来获取参数即使函数没有定义任何数量的参数



### ES6 新增rest参数

因为js函数在调用时可以传入任意数量的参数，那么为了方便获取除了定义之外的参数数组，ES6引入了rest参数

function testRest(a,b,…rest){}

在rest前面要加…标识，类似于可变参数，传入的参数先绑定a,b然后以数组的形式绑定到rest中，如果rest什么都没有接收到返回空数组，而不是undefined

请用rest参数编写一个sum()函数，接收任意个参数并返回它们的和：

function sum(...rest) {

if(rest.length<=0){

return 0;

}

return rest.reduce(function(a,b){

return a+b;

});

}

### 嵌套函数中变量的作用域

Js中函数可以嵌套定义，内部函数可以访问外部函数的变量，反之不可

如果外部函数定义的变量与内部重名，那么变量的值是不影响的

function foo() {

var x = 1;

function bar() {

var x = 'A';

console.log('x in bar() = ' + x); // 'A'

}

console.log('x in foo() = ' + x); // 1

bar();

}

foo();

x in foo() = 1  
x in bar() = A

这说明JavaScript的函数在查找变量时从自身函数定义开始，从“内”向“外”查找。如果内部函数定义了与外部函数重名的变量，则内部函数的变量将“屏蔽”外部函数的变量。

Js引擎在解析函数的时候，会将函数中所有变量的定义提升到最上面，提前引用未声明的变量不会报错，但值是undefined

**在函数内部定义变量时，严格遵守“在函数内部首先申明所有变量”这一规则**

最常见的做法是用一个var申明函数内部用到的所有变量：

**function** foo() {

**var**

x = 1, *// x初始化为1*

y = x + 1, *// y初始化为2*

z, i; *// z和i为undefined*

*// 其他语句:*

**for** (i=0; i<100; i++) {

...

}

}

### 全局作用域

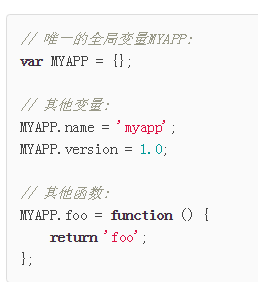
在函数外定义的变量是全局变量，任何函数都可以调用这个变量，全局变量会绑定到window对象上，所以 var a=’xxx’ ，也可以使用window.a来访问，进一步来说，所有定义的函数也属于全局变量，以变量方式var foo = function () {}定义的函数实际上也是一个全局变量，因此，顶层函数的定义也被视为一个全局变量，并绑定到window对象

所以可以用window.foo调用函数，也可以调用父页面window.parent.foo来调用父页面的js函数。

### 名字空间

全局变量会绑定到window上，不同的JavaScript文件如果使用了相同的全局变量，或者定义了相同名字的顶层函数，都会造成命名冲突，并且很难被发现。

减少冲突的一个方法是把自己的所有变量和函数全部绑定到一个全局变量中。例如：



将变量放入唯一的名字空间中，大大减少全局变量冲突的可能

### (ES6) let块级变量与const常量

使用var定义的变量在函数内都存在作用域，这样在函数内比如循环体内的变量将会在循环结束时仍然可以引用，使用let关键字声明的变量将声明一个块级作用域

ES6之前，js无法声明值不变的常量，ES6新增关键字const来定义常量，改变值不会报错牡但是没有效果。

### (ES6)解构赋值

通过解构赋值语法可以实现同时为多个变量赋值

var [x,y,z] = [‘java’,’javascript’,’ES6’];//变量需要使用[]括起来

var [x,[y,z]] = [‘java’,[‘c’,’c++’]];

var [,,z] = [‘structs’,’hibernate’,’mvc’];//忽略前两个元素

var person = {

name: '小明',

age: 20,

gender: 'male',

passport: 'G-12345678',

school: 'No.4 middle school',

address: {

city: 'Beijing',

street: 'No.1 Road',

zipcode: '100001'

}

};

var {name, age, passport} = person;

var {name,age,address:{city,street}};

//address不是变量，而是为了让city和street获得属性值

address; *// Uncaught ReferenceError: address is not defined*

如果对应属性不存在，变量被赋值undefined,如果要使用的变量名与属性名不一致，通过以下方法：

var {name,userage:age}

提供默认值，避免undefined

var {name,single=true}

变量提前声明的话会报语法错误，使用以下形式定义

var name,age;

（｛name,age｝ = {name:’小明’,x:’语句’}）

**使用场景**

Var x=1,y=2;

[x,y] = [y,x];

**快速获取当前页面的域名和路径：**

**var** {hostname:domain, pathname:path} = location;

**如果一个函数接收一个对象作为参数，那么，可以使用解构直接把对象的属性绑定到变量中。**例如，下面的函数可以快速创建一个Date对象：

**function** buildDate({year, month, day, hour=0, minute=0, second=0}) {

**return** **new** Date(year + '-' + month + '-' + day + ' ' + hour + ':' + minute + ':' + second);

### 方法

定义在对象内部的函数称为方法

var obj = {

name : ‘有方法的对象’,

birthday:’1995’

getAge :function(){

var year = new Date().getFullYear();

return year – this.birthday;

}

}

This 关键字指代当前对象 使用obj.方法名（）才能保证this有正确的指向

‘stict’模式下 this有可能指向 undefined 否则指向window对象

可以通过函数自带的方法apply来指定this的指向

getAge.apply(obj,[参数])；

另一个与apply()类似的方法是call()，唯一区别是：

* apply()把参数打包成Array再传入；
* call()把参数按顺序传入。

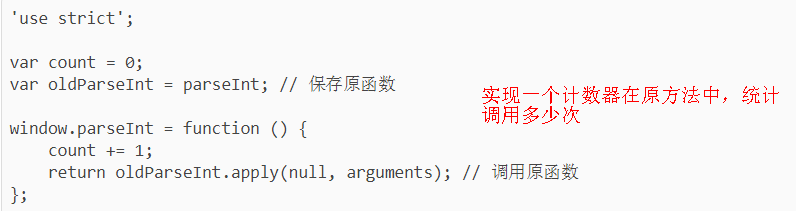
比如调用Math.max(3, 5, 4)，分别用apply()和call()实现如下：

Math.max.apply(null, [3, 5, 4]); *// 5*

Math.max.call(null, 3, 5, 4); *// 5*

对普通函数调用，我们通常把this绑定为null。

因为js中对象是动态的，所有可以动态的将对象内部的函数指向新的函数，也就是说可以在对象方法中加入新的处理逻辑



### 高阶函数

函数中接受另一个函数作为参数，这个函数称为高阶函数。

* **map** map(callback(currentvalue,index,array){ … });

接受函数依次对每一个元素进行处理，例如将所有元素的首字母大写

* **reduce**

接受函数依次对相邻元素进行处理，例如累加、累乘等操作

* **filter**

对数组元素进行过滤，callback 中return为false的过滤掉

array.filter(function(element,index,self){

Element 当前元素

Index 索引 0，1，2…

Self array数组本身

});

去重

array.filter(function(element,index,self){

return self.indexOf(element) === index;

});

* **sort**

直接用数组调用sort()方法排序，结果会与预想的相反

Sort()方法默认按照ASCLL码值来排序，并且数组元素为数字时会将数字转换为字符串然后按照ASCLL排序，总之，使用sort方法一定写好排序规则作为回调.

通常规定，对于两个元素x和y，如果认为x < y，则返回-1，如果认为x == y，则返回0，如果认为x > y，则返回1

Sort方法会对数组元素进行修改

* **Array对象内置高阶函数**

**every()** 判断数组中所有元素是否满足测试条件

arr.every(function(x){ return s.length>0; });//true 每个元素长度是否都大于零

**find()** 查找数组中符合条件的第一个元素，没有返回undefined.

**findIndex()** 查找数组中符合条件的第一个元素的索引，没有返回-1

**forEach()** 遍历数组

var arr = ['Apple', 'pear', 'orange'];

arr.forEach(console.log); // 依次打印每个元素

### 闭包

闭包函数是一种携带状态的函数，可以隐藏变量防止外部访问，

闭包函数将函数作为返回值

闭包就是能够读取其他函数内部变量的函数---阮一峰

闭包是个函数，而它「记住了周围发生了什么」。表现为由「一个函数」体中定义了「另个函数」

function outer () {

...

function inner () {

...

}

}

function foo(x) {

var tmp = 3;

return function (y) {

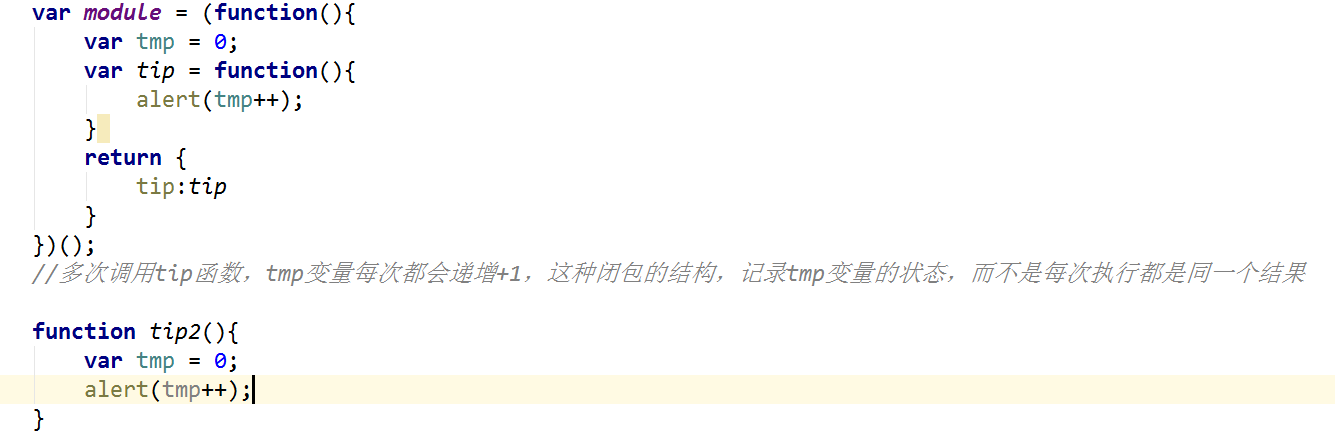
alert(x + y + (++tmp));

}

}

var bar = foo(2); // 将foo中的赋值给变量bar,bar 现在是一个闭包函数，可以访问tmp变量的值

bar(10);



### (ES6)箭头函数

箭头函数，类似与java的lambda表达式

在箭头函数中，this是根据上下文决定，不必再使用var that = this指定

箭头函数和匿名函数有个明显的区别：箭头函数内部的this是词法作用域，由上下文确定。

call()函数和apply函数，js中每个函数都提供call和apply函数，两个函数作用相同都接受两种参数，第一个参数是函数上下文绑定的对象，除此之外，call函数接收函数参数列表，与单独调用函数时传入参数相同；apply接受参数数组

https://github.com/lin-xin/blog/issues/7

在箭头函数中（ES6提供），因为this是词法作用域，所以调用call和apply时第一个参数就无效了

### (ES6) generator

在函数体中使用 yield 关键字定义返回语句，可以多次获得返回值，相当于保持了函数中局部变量的状态，函数使用function\*定义，使用f（）获取到的是generator对象，调用generator对象有两种方法，使用 .next()获取 ｛value:返回值/undefined，done：false/为true时表示函数执行结束｝

第二种方法使用for…of方法循环获取值,有多少值循环多少次

编写自增id函数

function\* next\_id(){

var I = 1;

while(true){

yield i++;

}

}

调用

*//调用generator函数的两种方式***console**.log(*getGeneratorId2*());*//直接调用获得generator对象 getGeneratorId2 {<suspended>}***console**.log(*getGeneratorId2*().next())*//next方法获得返回值及状态 {value: 1, done: false}  
  
//使用for of 循环***for**(**var** genValue of *getGeneratorId2*()){  
 **console**.log(genValue);*//1 2 3 4 ...*}

### 关于for…in和for…of

结论for…in 适合遍历对象，for…of适合遍历数组等*iterable对象*

ES6标准引入了新的iterable类型，Array、Map和Set都属于iterable类型

具有iterable类型的集合可以通过新的for ... of循环来遍历。

for ... of循环是ES6引入的新的语法。

for (var key in obj){  
 //console.log(key+"--->"+obj[key]);  
}

for(var i of arr){  
 //console.log(i);//只会遍历数组内的值，不会将属性遍历出来，ES6提供for...of循环主要解决了for...in的这个问题  
}  
//////////////////////////////////////////////////////////////////////////////////////////////////////  
//不建议使用for...in遍历数组  
for(var j in arr){  
 //console.log(j);//使用for...in输出的是索引,并且自定义属性也会遍历出来  
 //console.log(arr[j]);//虽然可以获得数组的所有值，但是数组的自定义属性也会遍历出来  
}  
  
for(var k of obj){  
 console.log(k);//报错，obj不属于iterable对象 for...of主要遍历iterable对象  
}

## 内置标准对象

### Date 时间

**var** now = **new** Date();

now; *// Wed Jun 24 2015 19:49:22 GMT+0800 (CST)*

now.getFullYear(); *// 2015, 年份*

now.getMonth(); *// 5, 月份，注意月份范围是0~11，5表示六月*

now.getDate(); *// 24, 表示24号*

now.getDay(); *// 3, 表示星期三*

now.getHours(); *// 19, 24小时制*

now.getMinutes(); *// 49, 分钟*

now.getSeconds(); *// 22, 秒*

now.getMilliseconds(); *// 875, 毫秒数*

now.getTime(); *// 1435146562875, 以number形式表示的时间戳*

js中使用get接受到的月份值是从0开始的，0表示一月

### RegExp 正则表达式

^ 行开头

$ 行结束

字符类型表达式

\d 数字

\w 字母或数字

. 任意字符

\s 匹配空格或者Tab

长度表达式

\* 任意个（包括0个）

+ 至少1个

？ 0个或1个

{n} n 个

{n,m} n-m个

如果要匹配'010-12345'这样的号码呢？由于'-'是特殊字符，在正则表达式中，要用'\'转义，所以，上面的正则是\d{3}\-\d{3,8}。

字符或数字范围表达式

[0-9a-zA-Z\\_] 匹配字母数字或下划线

A|B 匹配A或B

正则表达式创建

第一种 var reg = /^\d{0,3}\\_[a-z]+$/;

第二种 var reg = new RegExp(‘\d{0,3}\\\_[a-z]+’); //通过new创建，注意\的转义，两个\相当于一个

var re = /^\d{3}\-\d{3,8}$/;

re.test('010-12345'); // **true**

re.test('010-1234x'); // **false**

re.test('010 12345'); // **false**

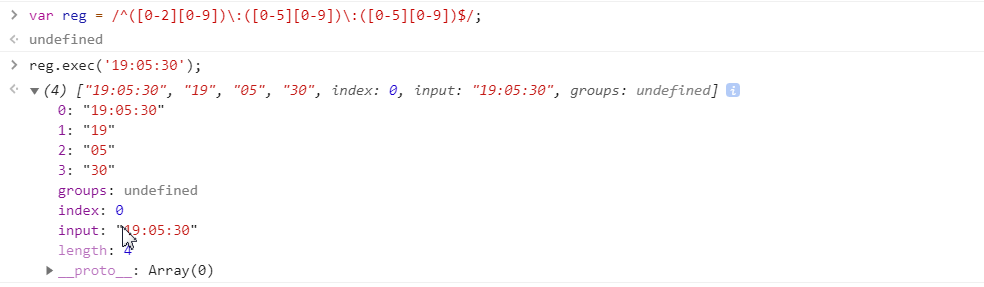
正则表达式除了用于字符串匹配，还可以用于**字符串拆分**

'a b c'.split(' '); // ['a', 'b', '', '', 'c']

'a b c'.split(/\s+/); // ['a', 'b', 'c']

'a,b;; c d'.split(/[\s\,\;]+/); // ['a', 'b', 'c', 'd']

使用正则表达式对字符串进行分组



使用括号来定义子串，然后调用exec()方法获得分组之后的数组，数组中第一项为匹配到的字符串，后面若干项为子串

贪婪匹配

分组过程中尽可能会匹配更多，以？结尾可以使分组采用非贪婪匹配

**var** re = /^(\d+)(0\*)$/;

re.exec('102300'); *// ['102300', '102300', '']*

**var** re = /^(\d+?)(0\*)$/;

re.exec('102300'); *// ['102300', '1023', '00']*

### JSON ---js对象表示

在JSON中，一共就这么几种数据类型：

* number：和JavaScript的number完全一致；
* boolean：就是JavaScript的true或false；
* string：就是JavaScript的string；
* null：就是JavaScript的null；
* array：就是JavaScript的Array表示方式——[]；
* object：就是JavaScript的{ ... }表示方式。
* 要输出得好看一些，可以加上参数，按缩进输出：
* JSON.stringify(xiaoming, null, ' ');
* 第二个参数用于控制如何筛选对象的键值，如果我们只想输出指定的属性，可以传入Array：
* JSON.stringify(xiaoming, ['name', 'skills'], ' ');
* 还可以传入一个函数，这样对象的每个键值对都会被函数先处理：
* **function** convert(key, value) {
* **if** (**typeof** value === 'string') {
* **return** value.toUpperCase();
* }
* **return** value;
* }

值被转换大写

如果我们还想要精确控制如何序列化小明，可以给xiaoming定义一个toJSON()的方法，直接返回JSON应该序列化的数据：

**var** xiaoming = {

name: '小明',

age: 14,

gender: true,

height: 1.65,

grade: null,

'middle-school': '\"W3C\" Middle School',

skills: ['JavaScript', 'Java', 'Python', 'Lisp'],

toJSON: **function** () {

**return** { *// 只输出name和age，并且改变了key：*

'Name': **this**.name,

'Age': **this**.age

};

}

};

JSON.stringify(xiaoming); *// '{"Name":"小明","Age":14}'*

JSON.parse()还可以接收一个函数，用来转换解析出的属性

var obj = JSON.parse('{"name":"小明","age":14}', function (key, value) {

if (key === 'name') {

return value + '同学';

}

return value;

});

console.log(JSON.stringify(obj)); // {name: '小明同学', age: 14}

## 附

var height = parseFloat(prompt('请输入身高(m):')); //接受输入

<https://www.liaoxuefeng.com/wiki/1022910821149312/1023021271742944> 高阶函数