**JavaScript 概述**

**1.什么是JavaScript**

1.JavaScript是一种具有面向对象能力的解释性程序设计语言(不需要编译),基于对象和事件驱动并且具有相对安全性的客户端脚本语言。

2.Javascript不需要在一个语言环境下运行，只需要支持它的浏览器即可，它的主要目的是验证发往服务器的数据，增强web互动，加强用户体验。

**2.JavaScript特点**

1.**松散性，**JavaScript语言核心于C,C++,Java相似，但是它是以一种松散性的语言，**它的变量不具有明确的类型**;

2.**对象属性,**JavaScript中的对象的属性名可以为任意的属性值。很像哈希表或关联数组,而不像Java，C++中的对象;

3.**继承机制，**JavaScript中的面向对象继承机制是基于原型的和C++及Java中的继承不同;

**3.JavaScript的核心**

一个完整的JavaScript由三个不同部分组成；

1.核心(ECMAScript)：语法、类型、语句、关键字、操作符、对象等;

2.文档对象模型(DOM)

3.浏览器对象模型（BOM）

**4.javaScript的使用**

1.<script></script>标签是在html页面插入js的主要方法；

2.一般情况，js最好另存一个.js文件,通过src属性引入<script src="common.js"></script>，这样维护性高,可缓存(加载一次，无序加载)，方便未来拓展。如果用了src属性引入了外部的js,在<script></script>的js就无效了;

3.javaScript代码从上往下执行；

**JavaScript语法**

**1.语法规则**

1.变量、函数名、操作符区分大小写；

2.表示符（变量，函数，属性的名字）定义规则：

a.第一个字符必须是一个字母，下划线，或美元符号；

b.其他字符可以是字母，下划线，美元符号，数字；

c.不能把关键字，保留字，true，false，null等作为标识符;

3.保留字和关键字不能作为标识符；

a.关键字：程序中已经开始使用的字符(如var);

b.保留字：在javaScript中还没有特殊的用途，但是在将来可能被用做关键字;

**2.注释**

//单行注释

/\*多行注释\*/

**3.变量**

全局变量可以不声明var 函数内变量必须声明var，在定义全局变量时加或不加var关键字没什么影响；但在定义局部变量时如果不加var关键字javascript解释程序会将其解释为全局变量。

1.ECMAScript的变量是松散性的;使用var 关键字来定义变量;

 var 变量;

2.undefined:定义了变量,但是没有为变量没有初始化，系统会 给一个特殊的值undefined,表示未定义;

3.ECMAScript属于弱类型语言，可以同时改变不同类型的量，但是这样做以后维护起来比较麻烦;

var box='wangtao';

   box=100;

console.log(box);

4.重复使用var声明一个变量，只不过是一个赋值操作， 不会报错,没有必要;

var a =10;

var a =11;

5.没有var关键字的变量和var关键字声明的变量有一定的区别和作用范围;

a.方法外部的为全局变量，方法内部的为局部变量。

<script>

 var part = "tom";

 test();

 function test(){

  var local ="joe";

  console.log(part);

  console.log(local);

 }

 console.log(part);

 console.log(local);*//local is not defined*

</script>

b.在方法内部使用var声明的变量为局部变量，当方法使用了定义在方法内部不加var声明的变量（初始化后的没有使用var声明的变量），该变量为全局变量。

<script>

var part = "tom";

test();

function test(){

 local ="joe";

 console.log(part);

 console.log(local);

}

console.log(part);

console.log(local);

 </script>

c.在定义全局变量时加或不加var关键字没什么影响。

<script>

  var part = "tom";

   p1 = "joe";

  test();

  function test(){

    local="rom";

    console.log(p1);

   console.log(part);

   console.log(local);

  }

  console.log(p1);

  console.log(part);

  console.log(local);*//local is not defined*

 </script>

d.在定义局部变量时如果不加var关键字javascript解释程序会将其解释为全局变量

<script>

var part = "tom";

test();

function test(){

 local ="sax"；

 console.log(part);

 console.log(local);

}

console.log(part);

console.log(local);

 </script>

总结：全局变量可以不声明var 函数内变量必须声明var，在定义全局变量时加或不加var关键字没什么影响；但在定义局部变量时如果不加var关键字javascript解释程序会将其解释为全局变量。[https://www.jb51.net/article/58794.htm](https://www.jb51.net/article/58794.htm" \o "https://www.jb51.net/article/58794.htm" \t "_blank)

6.当每条语句不再同行的时候可以省略分号，但是不建议这样做；

var a =10

var b =11

console.log(a+"--"+b)

7.可以使用一条语句定义多个变量,只要把每个变量(初始化或不初始化均可)用逗号隔开即可;但是为了可读性，最好是另起一行;

var box='wangtao',

     age=18,

     height;

**JavaScript数据类型**

[JavaScript](http://localhost:9093/itour/work/detail?id=234)

阅读  最后发布于 2022-09-26 17:03

**1.JavaScript数据类型**

1.ECMAScrip中提供了5中简单数据类型：  
**Undefined、Null、Boolean、Number、String**

2.一种复杂类型：**Object;**

3.ECMAScript**不支持创建自定义类型机制** ;

**2.typeof操作符**

1.typeof操作符用来**检测变量的数据类型** ,对于值或变量会返回如下字符串。

|  |  |
| --- | --- |
| **字符串** | **说明** |
| undefined | 未定义 |
| boolean | 布尔值 |
| string | 字符串 |
| number | 数值 |
| object | 对象或null |
| function | 函数 |

2.typeof可以操作变量,也可以操作常量;

<script>

var box='wangtao';

console.log(typeof box);

console.log(typeof 'wangtao');

console.log(typeof("wangtao"));

</script>

tip:虽然可以这样使用typeof(box),但是typeof是操作符而非内置函数；函数在ECMAScript中是对象,不是一种数据类型;

**3.Undefined**

1.Undefined的类型只有一个值undefined。**在使用var声明变量，但没有对其初始化时，这个变量的值为undefined（会自动赋值为undefined）**;

<script>

  var uname;

  console.log(uname);

</script>

2.**未初始化的变量和未定义的变量是不一样的;**

<script>

  var uname;

  console.log(uname);*//undefined*

  console.log(username);*//报错：Uncaught ReferenceError: username is not defined*

</script>

3.如果两个变量(uname,username)的typeof返回undefined,他们的值一个是undefined，一个报错,他们的类型都是Undefined;所以声明变量尽可能初始化，避免这种问题;

<script>

var uname;

  console.log(typeof uname);*//undefined*

  console.log(typeof username);*//undefined*

  console.log(uname);*//undefined*

  console.log(username);*//报错：Uncaught ReferenceError: username is not defined*

</script>

**4.Null**

1.Null类型只要一个值null,**它表示该变量初始化为null。**当检测null值可以知道这个对象是否分配了对象引用;

var box=null;

 if(box!=null){

  console.log("box对象已经存在");

 }

2.**undefined派生自null,规定对他们的相等性测试返回true;**

<script>

 console.log(undefined==null);*//true*

</script>

3.因为undefined和null两个值的比较是相等的,所有未初始化的变量和赋值为null的变量会相等。可以采用typeof变量的类型进行比较；

<script>

var box;

var car = null;

console.log(box == car);*//true*

console.log(typeof box == typeof car); *//false*

</script>

**5.Boolean**

**1.Boolean类的值：true，false;true不一定等于1，false不一定等于0；**

<script>

var b=true;

console.log(b==1);*//true*

console.log(b===1);*//fasle  恒等（值和数据类型相等返回true）*

console.log(typeof b ==typeof 1);*//false*

</script>

2.JavaScript区分大小写，True和False不是Boolean类型的值;

<script>

 console.log(typeof True);*//undefined*

 console.log(typeof False);*//undefined*

</script>

3.虽热Boolean类型的值只有true和false；但是ECMAScript中所有的类型的值都有与这两个Boolean值等价的值，要将一个值转换为其对应的Boolean值，可以使用转型函数Boolean()强制转换;

<script>

 var hello="hello";

 var h=Boolean(hello);

 console.log(typeof h);*//boolean*

 console.log(h);*//true*

</script>

**在实际使用中使用隐式转换(在if条件语句里面的条件判断);**

<script>

 var hello="hello";

  if(hello){

   console.log(hello);

  }

</script>

4.**其他类型转boolean的规则**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **转换为true的值** | **转换为false的值** |
| Boolean | true | false |
| String | 任何非空字符串 | 空字符串 |
| Number | 任何非0数字（包括无穷大） | 0和NAN |
| Object | 任何对象 | null |
| Undefined |  | undefined |

**6.Number**

**Number类型包含了两种数值：整型和浮点型;**为了支持各种数值类型，ECM定义了不同的数值格式

1.由于保存浮点数值需要的内存空间比整数值大两倍，**因此ECMAScript会自动将可以转换为整型的浮点数转为整型;**

<script>

var a=8.;*//小时点后面没有值，转换为8*

var b=12.0;*//小数后面为0，转换为12；*

console.log(a);*//8*

console.log(b);*//12*

</script>

2.对于过大或过小的数值，可用科学计数法表示，用e表示该数值的前面10的指数次幂

<script>

var a =4.12e3 *//4120;*

var b=0.0000000412;*//4.12e-8*

console.log(a);*//4120;*

console.log(b);*//4.12e-8*

</script>

**3.虽然浮点数的最高精度是17位小数，但是算术运算中可能不精确**

console.log(0.1+0.2);*//0.30000000000000004*

4.浮点数的范围在：Number.MIN\_VALUE;- Number.MAX\_VALUE之间;如果超出了浮点数的范围,就会出现Infinity(正无穷)或-Infinity(-正无穷),使用isFinite()判断是否超出了浮点数的范围;

<script>

var max=100e1000;*//Infinity*

var min=-100e1000;*//-Infinity*

console.log(isFinite(max));*//fasle*

</script>

**5.NAN 非数值 Not a number;**

<script>

var box1=0/0 ;

var box2=12/0;

var box3=12/0\*0;

console.log(box1);*//NaN*

console.log(box2);*//infinity*

console.log(box3);*//NaN*

</script>

a.通过Number.NaN得到NaN，任何与NaN进行运算的结果均为NaN，NaN与自身不相等（不与任何值相等）

console.log(Number.NaN);*//NaN*

console.log(NaN+1);*//NaN*

console.log(NaN==NaN);*//false*

**b.ECMAScript提供了isNaN():判断这个值是不是NaN**

console.log(isNaN(NaN));*//true*

console.log(isNaN(25));*//false*

console.log(isNaN('25'));*//false*

console.log(isNaN('lee'));*//true*

console.log(isNaN(true));*//false 可以转为1*

c.isNaN()也适用于对象，在调用的isNaN()函数过程中，首先会调用valueOf(),然后确定返回值是否能够转换成数字，如果不能，则基于这个返回值在调用toString()测试返回值;

**6.非数值转换位数值**

|  |  |
| --- | --- |
| **方法** | **说明** |
| Number() | 可以用在任何数据类型; |
| parseInt() | 字符串转为数值类型 |
| parseFloat() | 字符串转为数值类型 |

Number(true);//1

Number(25);//25

Number(null);0 空对象返回0

Number(undefined)NaN;

**7.String**

**1.String（字符串）可由双引号或单引号表示,但是必须成对出现，不能穿插使用，否则出错;**

2.String类型包含了一些特殊的转义符\n 换行 \t 制表 \b 空格 \r 回车 \ 斜杠 \' 单引号 \" 双引号;

3.ECMAScript中的字符串是不可变的，一旦创建就不能改变;

4.toString()方法可以包值转为字符串,toString()一般不传参,但是在数值转换为字符是可以传参;

5.如果在转型之前不知道变量是否是null或undefined的情况下，我们可以使用转型函数String(),这个函数可以将任何类型的值 转为字符串；

 var box=null;

console.log(String(box));

**tip**:如果值有toString()方法，则调用该方法并返回结果，如果是null或undefined，则返回"null"或"undefined"。

**8.Object**

**1.ECMAScript的对象是一组数据和功能的集合,对象可以同new操作符创建;**

var box=new Object();

var box=null;

ar box={};*//对象自变量的创建方法*

2.Object()是对象构造,如果对象初始化不需要传递参数，可以不写括号，但是不推荐；

3.Object()可以任意传参，可以传数值，字符串，布尔值，而且还可以进行相应的计算;

var box=new Obejct(2);

var age=box+2;

console.log(age);

4.可以使用new Object()来表示一个对象，我们也可以使用new操作符来创建其他类型的对象；

var box=new Number(5);

var box1=new String('wangtao');

**JSON和JavaScript对象的区别**

**1.JavaScript对象**

1.JavaScript对象是一组数据和功能的集合。

**2.JavaScript对象对键值要求**

1.键名可允许不加、加单引号、加双引号;

2.值没有限制;

3.最后一个属性后面允许有逗号;

**3.JSON**

1.JSON 是一种语法，用来序列化对象、数组、数值、字符串、布尔值和 null 。它基于 JavaScript 语法，但与之不同：JavaScript不是JSON，JSON也不是JavaScript。

2.JSON是一种轻量级的数据交换格式，采用完全独立的语言的文本格式，是理想的数据交换格式。

3.JSON以键值的形式 {"key1":"value1","key2":"value2"}；

**4.JSON对键值要求**

1.键名必须用双引号

2.值只能是数值（10进制）、字符串（双引号）、布尔值和null，数组或者符合JSON要求的对象，不能是函数、NaN, Infinity, -Infinity和undefined;

3.最后一个属性后面不能有逗号;

4.前导0不能用，小数点后必须有数字;

**3.JavaScript对象实例**

*//创建一个js对象*

var person ={"id":1,"name":"mike",'age':25};

*//创建一个空js对象*

var p ={}

*//创建一个数值js对象*

var box=new Number(5);

*//创建一个字符串js对象*

var box1=new String('wangtao');

*//创建一个数组js对象*

var arr  = new Array("1","2","3");

*//JSON格式的js对象*

var data1 ={"id":1,"name":"mike","age":25};

*//JSON格式的JS字符串*

var person='{"id":"001","name":"mk"}';

*//JSON格式的JS数组*

var people=[

 {"id":"001","name":"mk"},

 {"id":"001","name":"mk"}

];

**4.JavaScript对象和JSON字符串之间的转换**

1.JSON格式的JavaScript对象--->JSON字符串

*//JSON格式的js对象*

var data =｛"id":1,"name":"mike","age":25};

*//将JSON格式的js对象转为JSON字符串*

var jsonStr=JSON.stringify(data);

console.log(jsonStr);

console.log(typeof jsonStr);*//string*

2.JSON格式的字符串转--->JSON格式的JavaScript对象

var jsonStr ='{"id":1,"name":"mike","age":25}';

var p=JSON.parse(jsonStr);

console.log(p);

console.log(typeof p);*//object*

# JavaScript中for循环

## 1.for

### 1.特点

1.方便使用下标遍历并可以控制循环起点;

2.支持break和continue语法;

3.不能遍历对象;

### 2.遍历Array

var arr = [

 {"id":"1","name":"mike","job":"SALEMAN"},

 {"id":"2","name":"amy","job":"Cashier"},

 {"id":"3","name":"joe","job":"MANAGER"},

 {"id":"4","name":"tom","job":"MANAGER"},

 {"id":"5","name":"tim","job":"MANAGER"},

 {"id":"5","name":"scott","job":"Cashier"}

  ];

for(var i=0;i<arr.length;i++){

*//获取数组中对象*

 var emp = arr[i];

*//获取数组对象中的属性值*

 console.log(emp.id+"--"+emp.name);

 console.log(emp["id"]+"--"+emp["name"]);

 console.log(arr[i].id+"--"+arr[i].name);

  }

### 3.break和continue语法支持

var arr = [

 {"id":"1","name":"mike","job":"SALEMAN"},

 {"id":"2","name":"amy","job":"Cashier"},

 {"id":"3","name":"joe","job":"MANAGER"},

 {"id":"4","name":"tom","job":"MANAGER"},

 {"id":"5","name":"tim","job":"MANAGER"},

 {"id":"5","name":"scott","job":"Cashier"}

 ];

var m\_arr =[];

 for(var i=0;i<arr.length;i++){

 var emp = arr[i];

 if(emp["job"]!="MANAGER"){

 continue;

 }

 m\_arr.push(emp);

 if(m\_arr.length>=2){

 break;

 }

 }

 console.log(m\_arr)

## 2.for/in

### 1.特点

1.大部分用于遍历对象,遍历对象简洁方便；

2.支持break和continue语法;

### 2.迭代对象

var person = {"id":"1","name":"mike","age":"18"};

for(p in person){

 var key = p;

 var value = person[key];

 console.log(key+"--"+value);

}

### 3.支持break和continue语法;

var arr = [

 {"id":"1","name":"mike","job":"SALEMAN"},

 {"id":"2","name":"amy","job":"Cashier"},

 {"id":"3","name":"joe","job":"MANAGER"},

 {"id":"4","name":"tom","job":"MANAGER"},

 {"id":"5","name":"tim","job":"MANAGER"},

 {"id":"5","name":"scott","job":"Cashier"}

 ];

var c\_arr = [];

        for(index in arr){

         var emp = arr[index];

         if(emp["job"]!="Cashier"){

       continue;

      }

      c\_arr.push(emp);

      if(c\_arr.length>=2){

       break;

      }

        }

        console.log(c\_arr);

### 3.迭代Array

var arr = [

 {"id":"1","name":"mike","job":"SALEMAN"},

 {"id":"2","name":"amy","job":"Cashier"},

 {"id":"3","name":"joe","job":"MANAGER"},

 {"id":"4","name":"tom","job":"MANAGER"},

 {"id":"5","name":"tim","job":"MANAGER"},

 {"id":"5","name":"scott","job":"Cashier"}

];

for(index in arr){

 var employee = arr[index];

*//console.log(employee);*

*//访问对象两种方式*

 var id = employee.id;

 var name = employee["name"];

 console.log(id+"-->"+name);

}

## 3.for/of

### 1.特点

1.支持break和continue语法;

2.不能遍历对象；

### 2.支持break和continue语法

var d\_arr = [];

for(index in arr){

 var emp = arr[index];

 if(emp["job"]!="Cashier"){

  continue;

 }

 d\_arr.push(emp);

 if(d\_arr.length>=2){

  break;

 }

}

console.log(d\_arr);

### 3.遍历Array

var arr = [

 {"id":"1","name":"mike","job":"SALEMAN"},

 {"id":"2","name":"amy","job":"Cashier"},

 {"id":"3","name":"joe","job":"MANAGER"},

 {"id":"4","name":"tom","job":"MANAGER"},

 {"id":"5","name":"tim","job":"MANAGER"},

 {"id":"5","name":"scott","job":"Cashier"}

 ];

for(emp of arr){

   var id = emp.id;

   var name = emp["name"];

 console.log(id+"-->"+name);

 console.log(emp);

}

## 4.forEach

### 1.特点

1.遍历简洁,不关心集合下标的问题；

2.不支持break和continue语法；

3.不能遍历对象；

### 2.遍历Array

var arr = [

 {"id":"1","name":"mike","job":"SALEMAN"},

 {"id":"2","name":"amy","job":"Cashier"},

 {"id":"3","name":"joe","job":"MANAGER"},

 {"id":"4","name":"tom","job":"MANAGER"},

 {"id":"5","name":"tim","job":"MANAGER"},

 {"id":"5","name":"scott","job":"Cashier"}

 ];

arr.forEach(function(value,index,arr){

     var v = value;

     console.log(v.job);

   })

# JavaScript toFixed()方法

## 1.语法

number.toFixed(x)

## 2.作用

number.toFixed(x)，将Number类型的数值四舍五入为指定小数位数的数字。

## 3.参数值

x:必需。规定小数的位数，是 0 ~ 20 之间的值，包括 0 和 20，有些实现可以支持更大的数值范围。如果省略了该参数，将用 0 代替。

## 4.返回类型

返回小数点后有固定的 x 位数字的字符串。

## 5.number.toFixed(x)中遇到问题

number.toFixed(x)默认使用银行家舍入法，即四舍六入五取偶,是IEEE754标准的推荐舍入标准.

1.被修约的数字小于5时，该数字舍去；

console.log("8.834: "+8.834.toFixed(2));*//8.83*

console.log("8.844: "+8.844.toFixed(2));*//8.84*

2.被修约的数字大于5时，则进位；

console.log("8.866: "+8.866.toFixed(2));*//8.87*

3.被修约的数字等于5时，要看5前面的数字，若是奇数则进位。若是偶数则将5舍掉，即修约后末尾数字都成为偶数，若5的后面还有不为“0”的任何数，则此时无论5的前面是奇数还是偶数，均应进位.

console.log("8.885: "+8.885.toFixed(2));*//8.88*

console.log("8.875: "+8.875.toFixed(2));*//8.88*

console.log("8.8850: "+8.8850.toFixed(2));*//8.88*

console.log("8.8854: "+8.8854.toFixed(2));*//8.89*

console.log("8.8855: "+8.8855.toFixed(2));*//8.89*

console.log("8.8856: "+8.8856.toFixed(2));*//8.89*

## 6.解决方案

自定义方法，覆盖之前toFixed()的使用数学中的四舍五入

Number.prototype.toFixed=function(len){*//2*

 var s;

 var s1=this+"";*//1.8656*

 var temp=Math.pow(10,len);*//10的len次幂*

*//将参数\*10的len次幂1.8656\*100=186.56-->187*

*//Math.round(x)最接近x的整数*

 s=Math.round(this\*temp);

 return s/temp

};

# JavaScript处理小数运算

## 1.JavaScript在处理小数的时候会存在不准确的问题。

console.log(0.1+0.2);*//0.30000000000000004*

console.log(0.3-0.1);*//0.19999999999999998*

console.log(0.1\*0.2);*//0.020000000000000004*

console.log(0.7/0.1);*//6.999999999999999*

## 2.解决方案

解决方法：将小数变为整数来处理。

### 1.加法

1.先将所有的小数乘为整数；

2.待加减运算执行完之后再除去对应的 m 的值，将其变为小数输出;

function accAdd(arg1,arg2){

 var r1,r2,m;

*//获取arg1,arg2的小数点后的长度;*

 try{

 r1=arg1.toString().split(".")[1].length;

 }catch(e){

 r1=0;

 }

 try{

 r2=arg2.toString().split(".")[1].length;

 }catch(e){

 r2=0;

 }

 m=Math.pow(10,Math.max(r1,r2));

 return (arg1\*m+arg2\*m)/m;

}

3.给Number类型增加一个accAdd方法，调用起来更加方便。

Number.prototype.accAdd = function (arg){

 return accAdd(arg,this);

}

4.调用

console.log(0.1.accAdd(0.2));

或

console.log(accAdd(0.1,0.2));

### 2.减法

1.在加法的基础上做减法

function accsub(arg1,arg2){

 return accAdd(arg1,-arg2);

}

2.给Number类型增加一个accsub方法，调用起来更加方便。

Number.prototype.accsub = function (arg){

 return accAdd(this,-arg);

}

3.调用

console.log(0.3.accsub(0.1));

或

console.log(accsub(0.3,0.1));

### 3.乘法

1.25 \*8.2 = 10.25

125 \* 82 =10250/1000

function accMul(arg1,arg2){

 var m=0,s1=arg1.toString(),s2=arg2.toString();

 try{

 m+=s1.split(".")[1].length

 }catch(e){}

 try{

 m+=s2.split(".")[1].length

 }catch(e){}

 return Number(s1.replace(".",""))\*Number(s2.replace(".",""))/Math.pow(10,m)

}

给Number类型增加一个mul方法，调用起来更加方便

Number.prototype.mul = function (arg){

 return accMul(arg, this);

}

### 4.除法

function accDiv(arg1,arg2){

   var t1=0,t2=0,r1,r2;

   try{

    t1=arg1.toString().split(".")[1].length;

   }catch(e){}

   try{

    t2=arg2.toString().split(".")[1].length;

   }catch(e){}

   with(Math){

    r1=Number(arg1.toString().replace(".",""))

    r2=Number(arg2.toString().replace(".",""))

    return (r1/r2)\*pow(10,t2-t1);

   }

给Number类型增加一个div方法，调用起来更加方便

Number.prototype.div = function (arg){

  return accDiv(this, arg);

  }

# JavaScript正则表达式

## 1.正则表达式

正则表达式(regular expression)描述了一种字符串匹配的模式（pattern）。

## 2.正则表达式能做什么

1.判断给定的字符串是否符合正则表达式的过滤逻辑。

2.可以通过正则表达式，从字符串中获取我们想要的特定部分。

## 3.语法

1.语法

var reg =/pattern/[modifiers];

或

var patt=new RegExp(pattern,modifiers);

2.说明

pattern：正则表达式

modifiers：表达式,用于指定全局匹配、区分大小写的匹配和多行匹配

## 4.元字符

正则表达式由一些普通字符和一些元字符（metacharacters）组成。普通字符包括大小写的字母和数字，而元字符则具有特殊的含义。

|  |  |
| --- | --- |
| **元字符** | **说明** |
| \ | 在非特殊字符之前的反斜杠表示下一个字符是特殊的. |
| ^ | 匹配输入的开始. |
| $ | 匹配输入的结束. |
| {n} | n是一个正整数，匹配了前面一个字符刚好发生了n次. |
| {n,m} | n 和 m 都是整数。匹配前面的字符至少n次，最多m次。如果 n 或者 m 的值是0， 这个值被忽略. |
| [xyz] | 一个字符集合。匹配方括号中的任意字符，包括转义序列。可以使用破折号（-）来指定一个字符范围。对于点（.）和星号（\*）这样的特殊符号在一个字符集中没有特殊的意义。他们不必进行转义，不过转义也是起作用的. |
| [^xyz] | 一个反向字符集。匹配任何没有包含在方括号中的字符。可以使用破折号（-）来指定一个字符范围。任何普通字符在这里都是起作用的. |
| \* | 匹配前一个表达式0次或多次。等价于 {0,}. |
| + | 匹配前面一个表达式1次或者多次。等价于 {1,}. |
| ? | 匹配前面一个表达式0次或者1次。等价于 {0,1}. |
| . | 小数点匹配除换行符之外的任何单个字符. |
| x|y | 匹配‘x’或者‘y’. |
| \d | 匹配一个数字,等价于[0-9]. |
| \D | 匹配一个非数字字符,等价于[^0-9]. |
| () | 分组使用的字符为“(”和“)”，即左括号和右括号。每一个子表达式都可以当做一个整体来处理。（ 小括号就是括号内看成一个整体 ,中括号就是匹配括号内的其中一个） |

## 5.表达式（modifiers）

|  |  |
| --- | --- |
| **表达式** | **说明** |
| g | 全局标记 g 指定将该表达式应用到输入字符串中能够查找到的尽可能多的匹配 |
| i | i标记指定不区分大小写 |

## 6.常用正则表达式

1.大于等于0的整数

var ruler=/^\d+$/;

2.大于等于0的小数(保留小数点后两位)和整数

var ruler=/^([1-9]\d\*|0)(\.\d{2})?$/;

3.小数(保留小数点后两位)和整数,允许负数

var ruler=/^(\-|\+)?([1-9]\d\*|0)(\.\d{2})?$/;

4.1-100以内的小数(保留小数点后两位)和整数;

var ruler=/^(([1-9]\d)(\.\d{2})?)$|^100$/;

5.非负整数和非负小数(保留一到两位小数)

var ruler=/^([1-9]\d\*|0)([.][0-9]{1,2})?$/;

6.非负非零的整数和小数(保留一到两位小数)

var ruler=/^(([1-9]\d\*)(\.\d{1,2})?)$|^(0\.\d{1,2})$/

7.0到100之间(0,100)的整数和小数(运行保留一位或两位),不包括0和100

var ruler=/^(([1-9]\d{0,1})|(\d{1,2}\.\d{1,2}))$/;

## 7.测试使用

<script>

var ruler=/^(([1-9]\d{0,1})|(\d{1,2}\.\d{1,2}))$/;

var num = 100

console.log(ruler.test(num));

</script>