DOM：JavaScript操作网页上元素API,如操作标签等。

BOM：JavaScript操作浏览器的部分功能的API。针对浏览器进行的操作，如弹窗、前进、后退、关闭等。browser

# js基础

## js语法规范

### 1.每行代码结束加 ；号

编写Javascript注意语法规范，一行代码结束后必须在代码最后加上;  
如果不写分号，浏览器会自动添加，但是会消耗一些系统资源；  
此外，容易加错分号，所以在开发中分号必须要写。

### 2.严格区分大小写

在JavaScript中是严格区分大小写的

comfirm("Hello，JavaScript！"); // 正确

COMFIRM("Hello，JavaScript！"); // 错误

### 3.自动忽视多个空格和换行

JavaScript中会忽略多个空格和换行

console.log

(

"Hello，JavaScript！"

);

### 4.注释

单行注释： //

多行注释： /\*。。。。。。\*/

单行注释可以嵌套单行注释、多行注释  
// 北京  
// 北京 // 南京  
// /\* 北京 \*/

多行注释可以嵌套单行注释  
/\*  
// author：jack  
// intro：It's good  
\*/

## js引入

分为行内式、业内式、外链式

### 行内式

行内式直接写在标签内部：

<button onclick=”alert(‘今天天气很好’)”>今天天气很好</button>

### 业内式

放入<script></script>标签内部

一般是写在body的最末端。等待所有body加载完毕。

如果想写在前面，需要等待所有body元素加载完毕。

语句：

<script>

window.onload=function(){

要执行的js语句

}

</script>

最好是放在body最后面。

### 外链式

也是放在body后面

<script type=”text/html” src=”js文件地址”></script>

即可。

type=”text/html”可省略

## js输出方式

### **1.alert（）直接弹窗输出**

阻断式操作，不点击不会继续往下走，可以当做断点使用。

### **2.console.log/warn/error控制台输出**

<script>

console.log("Hello，JavaScript！");

console.warn("警告输出！");

console.error("错误输出！");

</script>

警告输出前有一个黄色感叹号

错误输出前有个×号。

### **3.prompt**

在网页中弹出输入框，一般用于接收用户输入的信息

<script>

var s = prompt("Hello，JavaScript！"); //将输入值赋给变量s

</script>

### 4.confirm

在网页中弹出提示框，显示信息，该方法一般与if判断语句结合使用

<script>

confirm("Hello，JavaScript！");

</script>

比alert的弹出框 多一个取消选项，可用于逻辑判断中的‘否’。

## js变量/关键字/标识符

### 1.常量

数值常量

字符串常量：使用单引号或者双引号括起来的

布尔常量：true false

转义字符：

’\n’ ‘\t’ ‘\i’等

### 2.变量

#### ①定义变量

定义变量(声明变量), 任何变量在使用之前，必须先进行定义

如果定义了多个变量，就会为这多个变量分别分配不同的存储空间。

格式： var 变量名称;

var name;

var name, age, sex;

#### ②变量赋值

name = “hello world”;

name1 = name2;

最好在等号左右各加一个空格。

#### ③变量初始化

变量第一次赋值，可以称为“初始化”

##### 先定义,后初始化

var name;

name= "撩课学院";

##### 定义的同时进行初始化

var name = "撩课学院";

##### 其它表现形式

// 部分初始化

var num1, num2, num3= 100;

// 完全初始化

var num1 = num2 = num3 = 100;

##### 如果定义后没有初始化

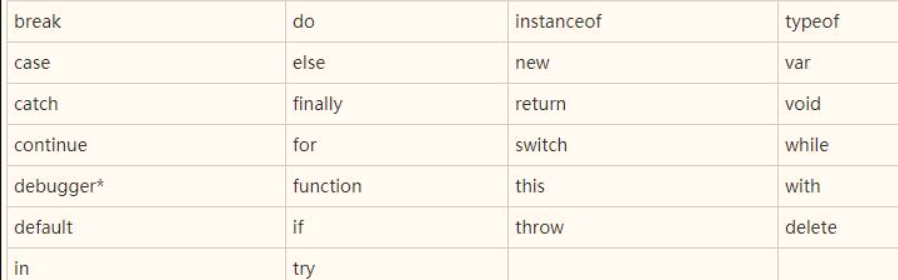
var num;

console.log(num); // undefined 返回的是undefined类型。

### 3.关键字和保留字

严格区分大小写

关键字：



保留字：

JavaScript预留的关键字，他们虽然现在没有作为关键字，但在以后的升级版本中有可能作为关键字



### 4.标识符

即变量名

严格区分大小写，

标识符中可以含有字母 、数字 、下划线\_ 、$符号

标识符不能以数字开头

标识符不能是ES中的关键字或保留字

严格区分大小写，比如name和Name是2个不同的标识符

标识符一般都采用驼峰命名法

## js数据和数据类型转换

### 复杂数据类型的浅复制

分为简单数据类型和复杂数据类型。

简单数据类型放在栈区，体积小，速度快。

复杂数据类型放在堆区，堆区不能直接访问，需要通过栈区访问。堆区数据的地址保存在栈区。所以堆区访问速度要慢于栈区。

对于复杂数据类型，其变量名指向的是栈区，栈区中保存的是指向数据的地址。在通过地址寻找数据。

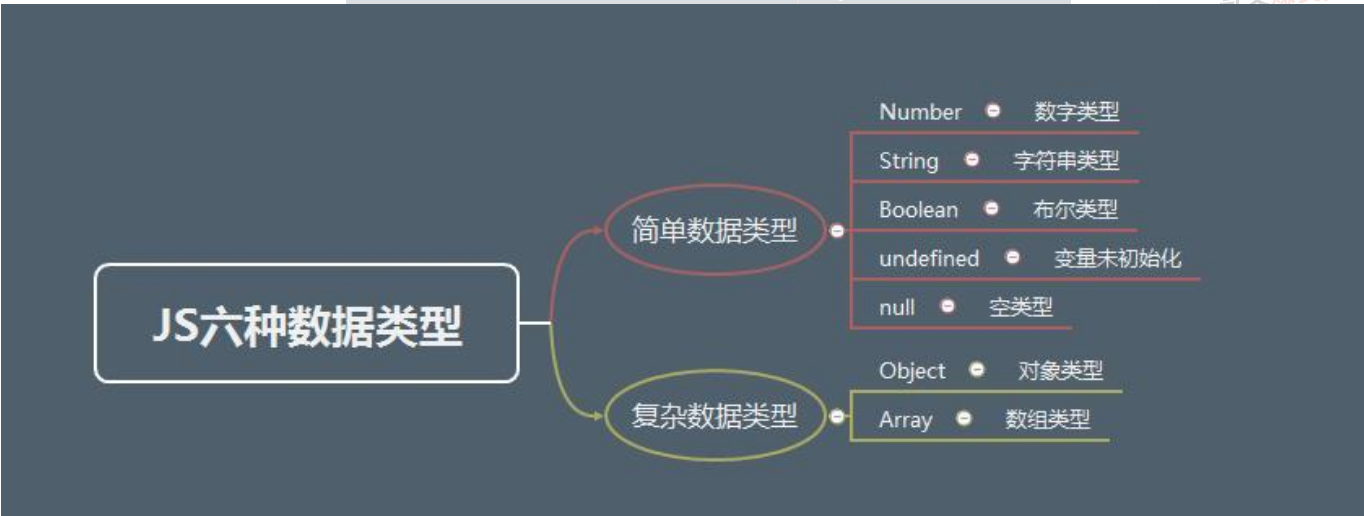
更改数据时，同样通过栈区地址找到堆区数据进行直接修改，地址并没有变化，但是改地址指向的内存中保存的数据发生了变化。

因此，对于复杂数据类型进行赋值时，

obj = obj2

当obj2改变时，obj跟着改变。这是因为obj和obj2中保存的地址都没有改变，但是obj2修改时，即通过obj2修改了堆区的数据。结果obj也跟着改变。

目前共6种数据类型：



### 1.查看数据类型

使用typeof 、instanceOf

* 会将检查的结果以字符串的形式返回给我们

console.log(typeof 100); // number

var name = "旋之华";

console.log(typeof name ); // string

### 2.number 数字类型

#### 基本性质

1 .最基本的数据类型  
2. 不区分整型数值和浮点型数值  
3.能表示的最大值是±1.7976931348623157乘以10的308次方  
能表示的最小值是±5 乘以10的-324次方  
4.包含十六进制数据，以 0x开头 0到9之间的数字，a(A)-f(F)之间字母构成。 a-f对应的数字是10-15  
5. 八进制直接以数字0开始，有0-7之间的数字组成。

6.在JS中所有的数值都是Number类型(整数和小数)

var num1= 123;

var num2= 3.14;

console.log(typeof num1); // number

console.log(typeof num2); // number

7.最大值、最小值、无穷大、无穷小

由于内存的限制，ECMAScript 并不能保存世界上所有的数值

Number是js的内置对象，最大值等是其内置属性。

// 最大值：Number.MAX\_VALUE

console.log(Number.MAX\_VALUE); // 1.7976931348623157e+308

// 最小值：Number.MIN\_VALUE

console.log(Number.MIN\_VALUE); // 5e-324

// 无穷大：Infinity, 如果超过了最大值就会返回该值

console.log(Number.MAX\_VALUE + Number.MAX\_VALUE); // Infinity

// 无穷小：-Infinity, 如果超过了最小值就会返回该值

console.log(typeof Infinity); // number

console.log(typeof -Infinity); // number

8.NaN 非法数字（Not A Number）  
JS中当对数值进行计算时没有结果返回，则返回NaN

var str = NaN;

console.log(typeof str ); // number

9. Number类型注意点

// JS中整数的运算可以保证精确的结果

var counter = 120 + 20;

console.log(counter ); // 140

// 在JS中浮点数的运算可能得到一个不精确的结果

var counter = 100.1 + 20.1;

console.log(counter ); // 120.200000000000003

### 3.string类型

使用单引号或者双引号进行包裹。

相同引号不能相互嵌套，不同引号可以相互嵌套。

// 正常的情况

var str1 = "hello";

var str2 = `world`;

console.log(typeof str1 ); // string

console.log(typeof str2); // string

// 不能一单一双，必须成对出现

var str3 = `hello world"; // 错误

// 相同引号不能嵌套，不同引号可以嵌套

var str3 = "hello "world""; // 错误

var str4 = `hello `world``; // 错误

var str5 = "hello 'world'"; // 正确

var str6 = `hello "world"`; // 正确

4.bool型

布尔型也被称为逻辑值类型或者真假值类型

布尔型只能够取真（true）和假（false）两种数值, 也代表1和0，实际运算中true=1,false=0

var bool1 = true;

var bool2 = false;

console.log(typeof bool1); // boolean

console.log(typeof bool2); // boolean

虽然Boolean 类型的字面值只有两个，但 ECMAScript 中所有类型的值都有与这两个 Boolean 值等价的值

任何非零数值都是true, 包括正负无穷大, 只有0和NaN是false  
任何非空字符串都是true, 只有空字符串是false  
任何对象都是true, 只有null和undefined是false

var bool = Boolean(0);

console.log(bool ); // false

var bool = Boolean(1);

console.log(bool ); // true

var bool = Boolean(-1);

console.log(bool ); // true

var bool = Boolean("hello");

console.log(bool); // true

var bool = Boolean("");

console.log(bool ); // false

var bool = Boolean(undefined);

console.log(bool ); // false

var bool = Boolean(null);

console.log(bool8); // bool

### 4. Null和Undefined

#### ①Undefined

Undefined这是一种比较特殊的类型，表示变量未赋值，这种类型只有一种值就是undefined

undefined本身就是一个值，可以对一个变量赋值undefined。

name = undefined;

var num;

console.log(num); //结果是undefined

undefined是Undefined类型的字面量

前者undefined和10, "abc"一样是一个常量

后者Undefined和Number,Boolean一样是一个数据类型

需要注意的是typeof对没有初始化和没有声明的变量都会返回undefined。

var str1 = undefined;//可以对一个变量赋值undefined

console.log(typeof str1 ); //结果是undefined

var str2;

console.log(typeof str2); //结果是undefined

#### ②Null

Null 类型是第二个只有一个值的数据类型，这个特殊的值是 null

从语义上看null表示的是一个空的对象。所以使用typeof检查null会返回一个Object

var test1= null;

console.log(typeof test1);//object

undefined值实际上是由null值衍生出来的，所以如果比较undefined和null是否相等，会返回true

var str1 = null;

var str2 = undefined;

console.log(str1 == str12);

console.log(str1=== str2);

### 5.== 与===

== 是值比较，不管类型是否一样，只要值是一样的就行。

如 5 == ‘5’ 返回true

=== 是完全比较，既比较值也比较类型。

### 6类型转换

#### ①toString( )/String( )

将除Null和Undefined以外类型的数据转换为字符串类型

toString()不会影响到原变量，它会将转换的结果返回

var num = 12306;

var str= num .toString();

var str2 = String(num)

console.log(str); // 12306

console.log(typeof str); // string

var num1 = true;

var str1 = num1.toString();

console.log(str1 ); // true

console.log(typeof str1 ); // string

注意:null和undefined没有toString()方法，如果强行调用,则会报错

var num2 = undefined;

var str2 = num2.toString(); // 报错

console.log(str2 );

var num3 = null;

var str3= num3.toString(); // 报错

console.log(res4);

var num4 = NaN;

var str4 = num4.toString();

console.log(str4); // NaN

console.log(typeof str4); // String

#### ②+’’

任何数据和 + ‘’ 连接到一起都会转换为字符串, 其内部实现原理和String()函数一样

var num1 = 100;

var res1 = num1 + "";

console.log(res1); // 100

console.log(typeof res1); // string

var num2 = true;

var res2 = num2 + "";

console.log(res2); // true

console.log(typeof res2); // string

var num3 = undefined;

var res3 = num3 + "";

console.log(res3); // undefined

console.log(typeof res3); // string

var num4 = null;

var res4 = num4 + "";

console.log(res4); // null

console.log(typeof res4); // string

undefined 和null也可以使用。

#### ③‘-’

字符也可以做减法运算，得到数字类型。

“2”-1

“2”-“1”

2-“1”

得到的都是数字1

#### ③Number( )函数

##### 字符串 转 数字

如果是数字字符串，则直接转为数值。

如果包含其他字符，则转换为NaN

如果是空字符串或者全是空格的字符串，则转换为0

var str1 = "666";

var res1 = Number(str1);

console.log(res1); // 666

console.log(typeof res1); // number

var str2 = "itlike123";

var res2 = Number(str2);

console.log(res2); // NaN

var str3 = "";

var res3 = Number(str3);

console.log(res3); // 0

var str4 = " ";

var res4 = Number(str4);

console.log(res4); // 0

##### undefined 转 数字

转为NaN

var str6 = undefined;

var res8 = Number(str6);

console.log(res8); // NaN

##### null转数字

转为0

var str5 = null;

var res7 = Number(str5);

console.log(res7); // 0

##### 布尔转数字

true转为1，false转为0

var bool1 = true;

var res5 = Number(bool1);

console.log(res5); // 1

var bool2 = false;

var res6 = Number(bool2);

console.log(res6); // 0

#### ④parseInt()函数和parseFloat()函数

* Number()函数中无论混合字符串是否存在有效整数都会返回NaN, 利用parseInt()/parseFloat()可以提取字符串中的有效整数
* parseInt()和parseFloat()的区别是前者只能提取整数,后者可以提取小数

##### parseInt( )提取字符串中的整数

字符串第一位是有效数字，则开始提取, 直到遇到无效数字，如果第一位不是有效数字, 什么都提取不到, 会返回NaN

有两个参数：

第一个参数是要转换的字符串，第二个参数是要转换的进制（默认为十进制）

* var str7 = "300px";
* var res9 = parseInt(str7);
* console.log(res9); // 300
* var str8 = "300px250";
* var res10 = parseInt(str8);
* console.log(res10); // 300
* console.log(parseInt("abc123")); //返回NaN，如果第一个字符不是数字或者符号就返回NaN
* console.log(parseInt("")); //空字符串返回NaN，Number("")返回0

##### parseFloat提取字符串中的小数

字符串第一位必须是有效数字，否则什么都提取不到，返回NaN。从第一位开始提取，知道碰到不是数字为止。只提取出一个小数点，碰到第二个小数点也会自动结束。

不支持第二个参数，只能解析10进制数

如果解析的内容里只有整数，解析成整数

var str9 = "20.5px";

var res11 = parseInt(str9);

console.log(res11); // 20

var str10 = "20.5.5.5px";

var res12 = parseFloat(str10);

console.log(res12); // 20.5

##### 对非String使用

对非String使用parseInt()或parseFloat(), 会先将其转换为String然后在操作

var str11 = true;

var res13 = parseInt(str11); // 这里相当于parseInt("true");

console.log(res13); // NaN

var res14 = Number(str11);

console.log(res14); // 1

## js运算符

### 作用

#### 按照功能划分：

算术运算符

位运算符

关系运算符

逻辑运算符

#### 按照操作数个数划分

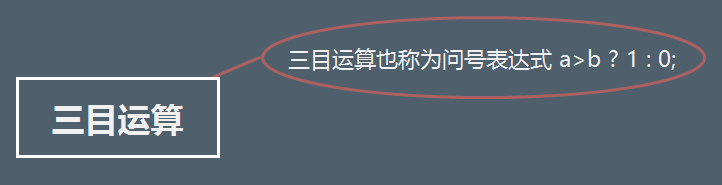
##### 单目运算

只有一个操作数 如 : i++ !

##### 双目运算

有两个操作数 如 : a+b;

##### 三目运算



三目运算也称为问号表达式 a>b ? 1 : 0;

### 运算符的优先级

#### 优先级

JavaScript中,运算符的运算优先级共分为15 级。1 级最高,15 级最低。

在表达式中,优先级较高的先于优先级较低的进行运算。

先计算优先级高的

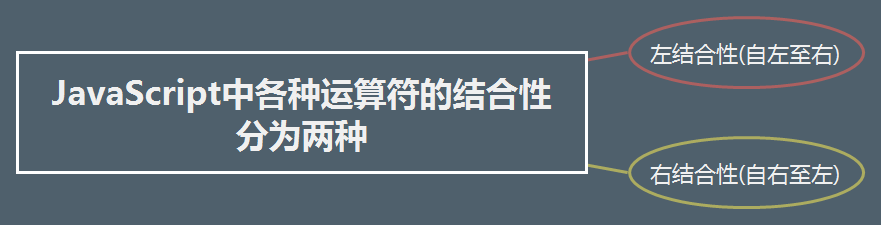
优先级相同则先左后右计算

### 运算符概览



### 运算符的结合性

JavaScript中各种运算符的结合性分为两种

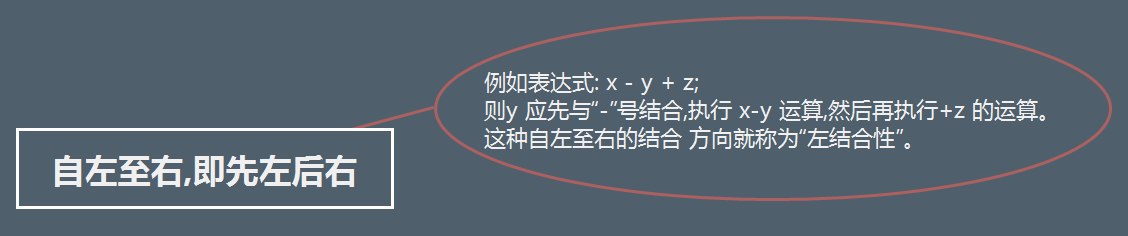


左结合性(自左至右)

右结合性(自右至左)

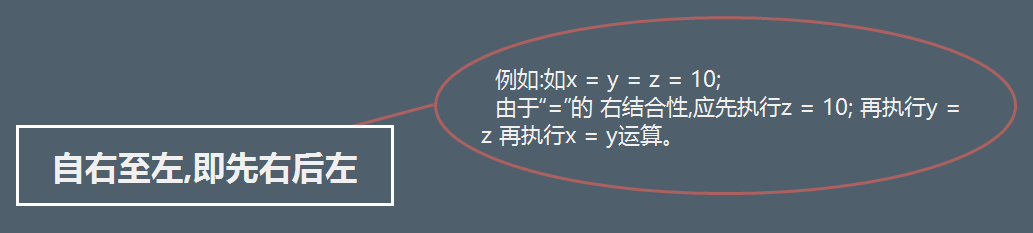
#### 左结合性

自左至右,即先左后右

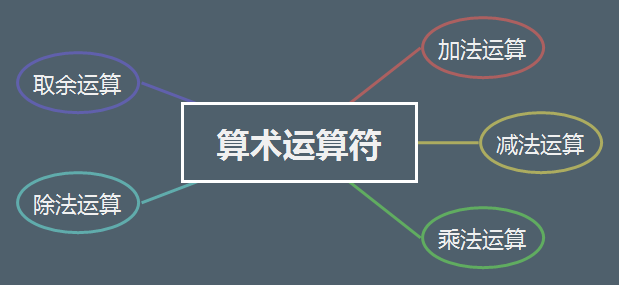


#### 右结合性

自右至左,即先右后左



### 算术运算符



#### 加法运算

##### 格式: Number1 + Number2

##### 规律

非Number类型的值进行运算时（除字符串外），会将这些值转换为Number然后在运算

任何的值和字符串做加法运算，都会先转换为字符串，然后再和字符串做拼串的操作

除字符串外，任何值和NaN做运算都得NaN。字符串与NaN是拼接的关系。

var s = ’ss’;

var t = NaN;

console.log(s+t) //ssNaN

var result = true + 1;  
console.log(result); // 2  
result = true + false;  
console.log(result); // 1  
result = 2 + null;   
console.log(result);// 2

var result = 1 + NaN;  
console.log(result); //NaN

var result = 10 + "123";  
console.log(result); // 10123  
result = 1 + "true";  
console.log(result); // 1rue

#### 减法运算

##### 格式: Number1 - Number2

##### 规律

非Number类型的值进行运算时，会将这些值转换为Number然后在运算

任何值和NaN做运算都得NaN

字符串做减法也会转换为Number

var result = 2 - "1";  
console.log(result); // 1  
result = "2" - "1";  
console.log(result); // 1

#### 乘法运算

##### 格式: Number1 \* Number2

##### 规律

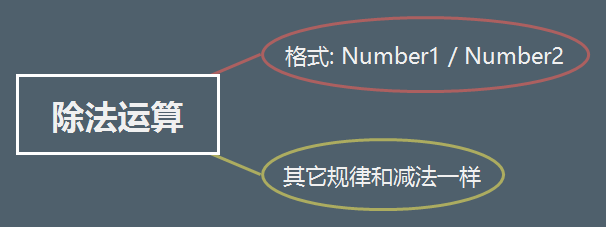
其它规律和减法一样

非Number类型的值进行运算时，会将这些值转换为Number然后在运算

任何值和NaN做运算都得NaN

字符串做乘法也会转换为Number

#### 除法运算



##### 格式: Number1 / Number2

##### 规律

其它规律和减法一样

非Number类型的值进行运算时，会将这些值转换为Number然后在运算

任何值和NaN做运算都得NaN

字符串做乘法也会转换为Number

#### 取余运算%

##### 格式: Number1 % Number2

m%n 求余,相当于m/n 获取余数，若分母为0，则返回NaN

##### 规律：

其它规律和减法一样

非Number类型的值进行运算时，会将这些值转换为Number然后在运算

任何值和NaN做运算都得NaN

字符串做取余也会转换为Number

‘3’ % ’2’ //1

#### 正负运算符

##### 正号

+ 正号不会对数字产生任何影响

对于非Number类型的值,会将先转换为Number（包括字符串），然后再运算（隐形类型转换）

var bool = true;  
var res = +bool;  
console.log(res); // 1  
  
var str = "123";  
res = +str;  
console.log(res); // 123  
  
var str2 = "123abc";  
res = +str2;  
console.log(res); // NaN, 所以内部不是调用parseInt, 而是Number()函数  
  
var temp = null;  
res = +temp;  
console.log(res); // 0

##### 负号

- 负号可以对数字进行负号的取反

对于非Number也存在和+号一样，也会有隐形类型转换。

### 赋值运算符

#### 简单赋值运算符 =

##### 格式: 变量 = 数据

// 将等号右边的常量100赋值给左边的变量num  
var num = 100;  
  
// 将等号右边的变量num中存储的值赋值给左边的变量value  
var value = num;

赋值运算符左边只能是变量

多个赋值运算符可以组成 赋值表达式, 赋值表达式具备右结合性

// 从右至左计算  
// 先将10赋值给变量c, 然后将变量c中存储的值赋值给变量b  
// 然后将变量b中存储的值赋值给变量a, 最后a,b,c存储的都是10  
a = b = c = 10;

#### 复合赋值运算符

+= 加后赋值 变量+=表达式 如:a+=1;即a=a+1

-= 减后赋值 变量-=表达式 如:a-=1;即a=a-1

\*= 乘后赋值 变量=表达式 如:a=1;即a=a\*1

/= 除后赋值 变量/=表达式 如:a/=1;即a=a/1

%= 取模后赋值 变量%=表达式 如:a%=1;即a=a%1

##### 复合赋值表达式运算

先等号右侧算完，再进行复合运算。

格式: 变量 复合赋值运算符 表达式; 如: a \*= 1 + 2;

由于赋值运算符是右结合性, 所以会先计算等号右边, 然后再进行复合运算

var value = 5;  
value \*= 2 - 2; // 等价于 value = 5 \* (2 - 2);  
console.log(value); // 0  
// 设想如果先计算复合运算符结果是什么?  
value = (5 \* 2) - 2 = 10 - 2 = 8

#### 自增运算符

i++

++i

i--

--i

和c语言的一样。

### 关系运算符

#### 关系运算符概览



#### 示例

var a, b;  
 a = 20;  
 b = 20;  
 console.log(a > b); // false  
 console.log(a < b); // false  
 console.log(a >= b); // true  
 console.log(a <= b); // true  
 console.log(a == b); // true  
 console.log(a != b); // false

#### 数值与非数值的比较

对于非数值进行比较时，会将其转换为数字然后在比较

console.log(1 > true); //false  
console.log(1 >= false); //true  
console.log(1 > "0"); //true  
console.log(1 > null); //true

#### 字符串之间的比较

如果符号两侧的值都是字符串时，不会将其转换为数字进行比较, 而会分别比较字符串中字符的Unicode编码

#### null、undefined 、NaN比较

console.log(null == 0); // false  
console.log(undefined == 0); // false  
// 永远不要判断两个NaN是否相等  
console.log(NaN == NaN); // false  
  
/\*  
 \* 可以通过isNaN()函数来判断一个值是否是NaN  
 \* 如果该值是NaN则返回true，否则返回false  
 \*/  
var num = NaN;  
console.log(isNaN(num)); // true  
  
// undefined 衍生自 null, 所以返回true  
console.log(null == undefined); // true;  
console.log(null === undefined); // false;  
  
// == 判断值是否相等  
// == 会进行数据类型转换  
console.log("123" == 123); // true  
// === 判断值和类型时候同时相等  
// === 不会进行数据类型转换  
console.log("123" === 123); // false

### 逻辑运算符

#### &&(与运算)

##### 格式: 条件A && 条件B

逻辑与运算过程

总是先判断条件A是否成立，所以一般将为false的条件放前面。

如果条件A成立，接着再判断条件B是否成立：如果条件B成立，“条件A && 条件B”的结果就为true，如果条件B不成立，结果就为false

如果条件A不成立，就不会再去判断条件B是否成立：因为条件A已经不成立了，不管条件B如何结果肯定是false

##### 注意点

对于非Boolean类型的数值, 逻辑与会自动将其转换为Boolean类型来判断

如果条件A不成立, 则返回条件A的数值本身

如果条件A成立, 不管条件B成不成立都返回条件B数值本身

var result = "123" && "abc";  
console.log(result); // "abc"  
result = "123" && 0;  
console.log(result); // 0  
result = null && 0;  
console.log(result); // null

#### ||(或运算)逻辑或

##### 格式: 条件A || 条件B

总是先判断条件A是否成立，所以一般把结果为true的放前面。

如果条件A成立，就不会再去判断条件B是否成立：因为条件A已经成立了，不管条件B如何结果肯定是1，也就是true

如果条件A不成立，接着再判断条件B是否成立：如果条件B成立，“条件A || 条件B”的结果就为true，如果条件B不成立，结果就为false

##### 注意点

对于非Boolean类型的数值, 逻辑或自动会将其转换为Boolean类型来判断

如果条件A不成立, 则不管条件B成不成立都返回条件B数值本身

如果条件A成立, 则返回条件A的数值本身

var result = null || 0;  
console.log(result); // 0  
  
result = "123" || "abc";  
console.log(result); // "123"  
result = "123" || 0;  
console.log(result); // "123"

#### ！ 逻辑非

##### 格式: ! 条件A

##### 注意点

对一个值进行两次取反，它不会变化

对非布尔值进行元素，则会将其转换为布尔值，然后再取反

所以, 要想将其它类型转换为Boolean类型除了Boolean()函数, 还可以使用 !!数值;

!!数值;的形式,实现原理和Boolean()函数一样

### 逗号运算符

在JavaScript中逗号“,”也是一种运算符,称为逗号运算符。 其功能是把多个表达式连接起来组成一个表达式, 称为逗号表达式。

一般形式形式:　表达式1，表达式2，… …，表达式n;

例如: a = 1 + 1,b = 3 \* 4, c = 10 / 2;

#### 求值过程

表达式1，表达式2，… …，表达式n;

逗号表达式的运算过程是：先算表达式1，再算表达式2，依次算到表达式n

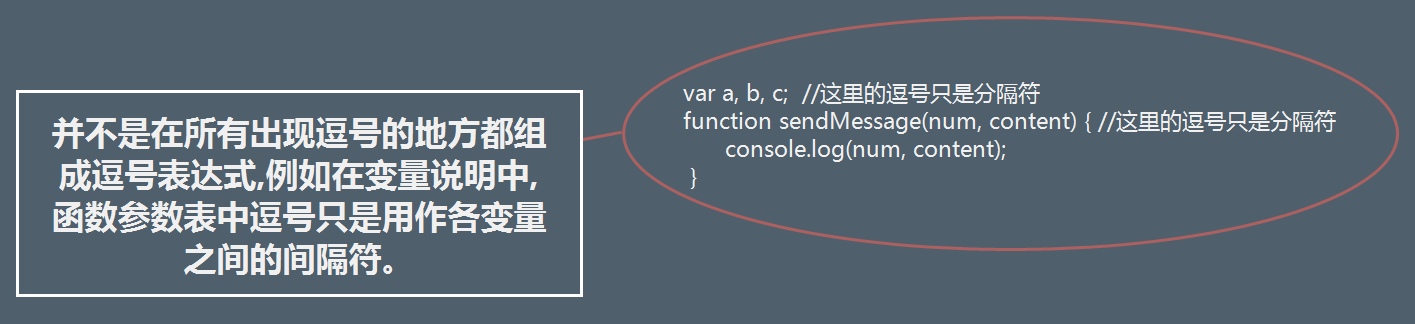
整个逗号表达式的值是最后一个表达式的值

var a, b, c, d;  
/\*  
1.先计算表达式1, a = 2  
2.再计算表达式2, b = 12  
3.再计算表达式3, c = 5  
4.将表达式3的结果返回给d  
\*/  
d = (a = 1 + 1,b = 3 \* 4, c = 10 / 2);  
console.log(d); // 5

#### 使用注意

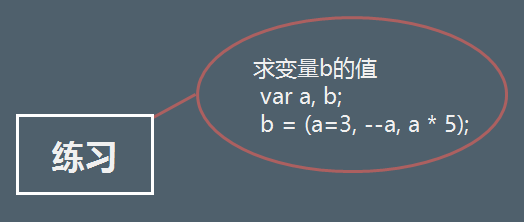
程序中使用逗号表达式,通常是要分别求逗号表达式内各表达式的值,并不一定要求整个逗号表达式的值。

并不是在所有出现逗号的地方都组成逗号表达式,例如在变量说明中,函数参数表中逗号只是用作各变量之间的间隔符。



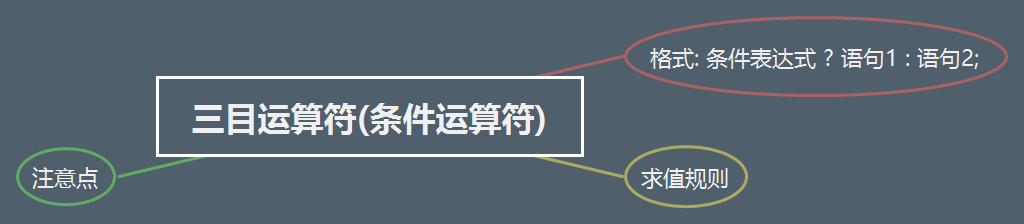
var a, b, c; //这里的逗号只是分隔符  
function sendMessage(num, content) { //这里的逗号只是分隔符  
 console.log(num, content);  
 }

练习



求变量b的值  
 var a, b;  
 b = (a=3, --a, a \* 5);

### 三目运算符(条件运算符)



#### 格式: 条件表达式 ? 语句1 : 语句2;

#### 求值规则

如果条件表达式为true，则执行语句1，并返回执行结果

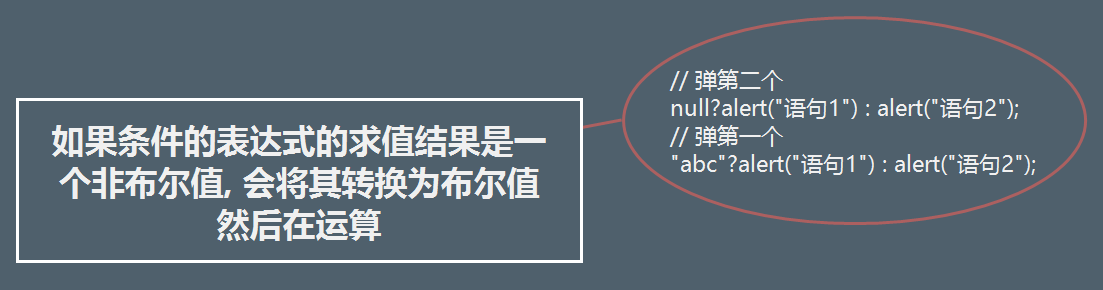
如果条件表达式为false，则执行语句2，并返回执行结果

// 弹第一个  
true?alert("语句1") : alert("语句2");  
// 弹第二个  
false?alert("语句1") : alert("语句2");

#### 注意点

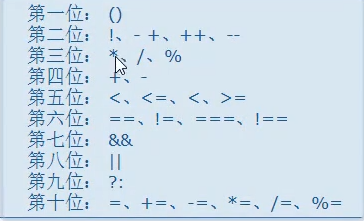
条件运算符?和:是一对运算符,不能分开单独使用

如果条件的表达式的求值结果是一个非布尔值, 会将其转换为布尔值然后在运算



// 弹第二个  
null?alert("语句1") : alert("语句2");  
// 弹第一个  
"abc"?alert("语句1") : alert("语句2");

### 优先级



## js流程控制

### 1.if语句

如果条件表达式为真,执行语句块1,否则不执行。

如果前面有满足条件执行，则不会再对后面的语句进行判断。

if语句也可以嵌套使用。  
if(条件表达式) { 语句块1 } 后续语句;

if(age>40)

{

console.log("老年人");

}else if(age>25){

console.log("中年人");

}else if(age>18){

console.log("成年人");

}else{

console.log("未成年");

}

特点

这么多大括号中只有一个大括号中的内容会被执行

当执行到后面条件的时候证明前面的条件不满足

else if可以只有一个, 也可以有多个, 但是都必须添加在if和else之间

注意事项

如果只有一条语句时if后面的大括号可以省略

if(age > 18)

console.log("成年人");

注意:受到if管制的只有紧跟其后的那条语句

开发中尽量不要省略大括号

if(false)

console.log("语句1");

console.log("语句2"); // 语句2会被输出

分号“;”也是一条语句, 空语句

if(4 > 6);

{

console.log("4 > 6");

}

// 输出结果: 4 > 6, if管制的紧跟其后的那个分号

// {}在JS中仅仅代表告诉系统里面的语句是一个整体, 没有别的用途,不会影响代码的执行

if else是一个整体， else匹配if的时候匹配离它最近的一个if

if(0)

if(1)

console.log("A");

else

console.log("B");

else

if (1)

console.log("C");

else

console.log("D");

对于非Boolean类型的值,会先转换为Boolean类型后再判断

if (0) {

console.log("itlike\n"); // 不会被执行

}

```

判断变量与常量问题

但凡遇到比较一个变量等于或者不等于某一个常量的时候，把常量写在前面

// if(a = 0) // 错误写法, 但不会报错

if (0 == a) {

console.log("a的值是0\n");

}else{

console.log("a的值不是0\n");

}

/\*

本来是先判断变量a存储的值是否等于0,

但是如果不小心写成a=0,代表把0存储到a中.

为了避免这种问题判断是否相等时把常量写前面,

因为等号左边只能是变量, 所以如果少写一个等号会报错\*/

if语句可以嵌套使用

var num = 20;

if (num >= 10)

{

if(num <= 30)

{

console.log("一个10~30之间的数")

}

}

### 2.switch

switch(num){

case 1:

console.log("壹");// 输出壹

break; // 跳出switch语句, 后面代码不会执行

case 2:

console.log("贰");

break;

default:

console.log("非法数字");

break;

}

通过判断num的值，来执行语句。都不符合的话，就执行default，default可以省略。同样，按照从从上往下的顺序执行，如果上面有条件满足，就会执行条件中的语句，执行完毕后直接跳出switch，不会继续比较下去。

### 3.循环

#### ①while循环

while ( 条件表达式 ) {

语句1;

语句2;

....

}

可以使用break跳出循环。

#### ②do..while循环

与while不同的是，会先执行一遍语句，然后再进行判断。

格式

do {

语句1;

语句2;

....

} while ( 条件 );

执行流程

首先会执行一次循环体中的语句

接着判断while中的条件表达式是否为真，如果为真再次执行循环体中的语句

重复上述过程，直到条件不成立就结束while循环

特点

不管while中的条件是否成立，循环体中的语句至少会被执行一遍

应用场景

口令校验

#### ③for循环

for(①初始化表达式；②循环条件表达式；④循环后操作表达式)

{

③执行语句；

}

for(var i = 0; i < 10; i++){

alert(i); // 输出0-9

}

#### ④break

跳出循环或者switch语句。

#### ⑤continue

跳出本次循环，开始下次循环。

#### for..in

用于遍历对象属性方法

for key in object

### 4. ；问题

分号代表语句结束。

if （i == 1）;

代表if语句没有结束。

while(1);

永远的死循环。

## js数组

数组是一种数据类型, 属于对象

### 1.数组创建

#### 方式一：

使用 Array 构造函数

创建一个空数组

var arr1 = new Array();

创建一个长度为30的数组

var arr2 = new Array(30);

创建一个包含2个字符串的数组

var arr3 = new Array("旋之华","yann");

#### 方式二：

使用数组字面量

创建一个空数组

var arr4 = [];

创建一个内容为30的数组

var arr5 = [30]; // arr5中只有一个元素，30

创建一个包含2个字符串的数组

var arr6 = ["撩课学院","www.itlike.com"];

### 2.数组常用操作

#### ①数组长度

arr.length

#### ②获取数组中的元素

arr[index] : index从0开始，如果超出索引，不会报错，而是返回undefined。

### 3.常用api

join()

push()和pop()

shift() 和 unshift()

sort()

reverse()

concat()

slice()

splice()

indexOf()和 lastIndexOf()

forEach()

map()

filter()

every()

some()

reduce()和 reduceRight()

#### ①join

arr.join(sep)：以sep为分割，将数组内容组成一个字符串。默认为逗号分割。

返回新的字符串，不会改变原数组。

#### ②push( )/pop( )

push:接收任意数量的内容。将内容添加到数组的最后，并返回添加后数组长度。也可以接收数组，但是将会作为一个元素放入arr中，相当于数组嵌套。

pop：不接受参数，删除数组最后一个元素，并返回该元素。

#### ③shift( )/unshift( )

shift和pop类似，但是是对头部操作

unshift和push相似，但是也是对头部操作。

#### ④reverse( )

arr.reverse() 将数组进行倒置。是在原数组上改动，将改变原数组。

#### ⑤contact（）

arr.contact(s) :将括号中的内容与数组进行拼接，返回新拼接的数组。s参数可以为数组，数组将会被拆开，然后拼接入arr中。

#### ⑥indexOf / lastindexOf

indexOf 接收两个参数，要查找的项和表示查找起点的索引。查到第一个，就停止查找，返回索引。如果没找到，返回 -1

其中，查找起点可省略，默认为0.

lastindexOf为从后往前查找。查找起点默认为数组尾部。

若找到，则返回索引。如果没有找到，则返回-1.

arr=[1,2,3,4]

arr.indexOf(4,1)

#### ⑦sort

对数组进行排序。采用冒泡排序。

默认按照升序排列，最小值在前。返回一个新数组。

通过回调函数设置降序排列：

arr.sort(function(a,b){

return b-a;

})

return a-b 为降序，

return b-a 为升序。

#### ⑧slice切片

返回指定开始结束index的切片。返回区间为左闭右开区间，不包括结束index。传入值也可以为负数，和python一样。

接收1或者2个参数，如果是一个参数，则返回从该参数开始到数组末尾的切片。

#### ⑨splice

用于删除、插入和替换

##### 删除：

可以删除任意数量的项，只需指定 2 个参数：要删除的第一项的位置和要删除的项数。例如， splice(0,2)会删除数组中的前两项。

##### 插入：

可以向指定位置插入任意数量的项，只需提供 3 个参数：起始位置、 0（要删除的项数）和要插入的项。例如，splice(2,0,11,12)会从当前数组的位置 2 开始插入11和12。

##### 替换：

其实就是先删除再插入。

可以向指定位置插入任意数量的项，且同时删除任意数量的项，只需指定 3 个参数：起始位置、要删除的项数和要插入的任意数量的项。插入的项数不必与删除的项数相等。例如，splice (2,1,4,6)会删除当前数组位置 2 的项，然后再从位置 2 开始插入4和6。

##### 清空数组

arr.splice(0);

也可以直接给数组进行赋值空：

arr= []

#### 10.forEach

对数组进行遍历循环，对数组中的每一项运行给定函数（回调函数）。

arr.forEach(function…)

将arr中的每一项分别给括号中的回调函数，有多少个元素，调用回调函数多少次。

格式：

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

回调函数也可以提前写好。

value为元素值，index为元素的索引。

#### 11.map( )映射

“映射”，对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组。类似于python，回调函数也可以接收参数为value和index。

var arr = [1, 2, 3, 4, 5];  
var arr2 = arr.map(function(item){  
return item\*item;  
});  
console.log(arr2); //[1, 4, 9, 16, 25]

#### 12.filter( )

“过滤”功能，数组中的每一项运行给定函数，返回满足过滤条件组成的数组。类似python。回调函数接收参数和foreach一样，为value和index

var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  
var arr2 = arr.filter(function(x, index) {  
return index % 3 === 0 || x >= 8;  
});   
console.log(arr2); //[1, 4, 7, 8, 9, 10]

#### 13.every

遍历数组，如果所有都满足回调函数设置的条件，则返回一个true，只要有一个不满足，就返回false。回调函数可以接收两个参数，value和index，如果只有一个形参，则为value。

var arr = [1, 2, 3, 4, 5];  
var arr2 = arr.every(function(x) {  
return x < 10;  
});   
console.log(arr2); //true  
var arr3 = arr.every(function(x) {  
return x < 3;  
});   
console.log(arr3); // false

#### 14.some

和every一样，但是是或的关系，只要有一个满足，就返回true

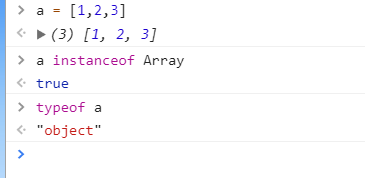
#### 15.instansceof

判断两个对象类型是否相同。

格式 A instansceof B 返回布尔值，类型相同为true，不同为false

b可以直接为Array等类型名。

与typeof区别：



typeof 只能判断是否为object类型。

#### 16.Array.isArray( )

Array.isArray(arr) 判断arr是否为数组，返回布尔值。

#### 17.toString( )

arr.toString( ) :将数组所有组合为一个字符串，用元素之间用逗号隔开。

#### 18.valueOf( )

返回数组对象本身。

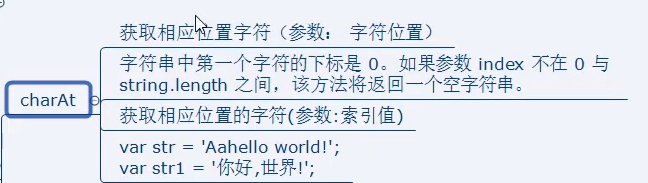
数组 = 数组.valueOf()

## js字符串的相关操作

### 1.转换为字符串 +/toString/String

### 2.返回某一位置的字符

#### ①charAt



只分字符

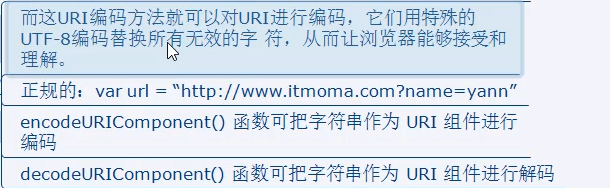
#### ②charCodeAt( )

返回相应位置字符的编码。默认返回Unicode码，是数字。

### 3.根据字符返回位置indexOf/lastIndexOf

和数组的用法一样。

### 4.uri编码与解码



encodeURICompnent(str)为编码

decodeURICompnent(url)为解码

### 5.截取字符串slice()、substring()、substr()

slice()  
第一个参数代表开始位置,第二个参数代表结束位置的下一个位置,即顾头不顾尾。截取出来的字符串的长度为第二个参数与第一个参数之间的差;若参数值为负数,则将该值加上字符串长度后转为正值;若第一个参数等于大于第二个参数,则返回空字符串.

substring()  
第一个参数代表开始位置,第二个参数代表结束位置的下一个位置，顾头不顾尾;若参数值为负数,则将该值转为0;两个参数中,取较小值作为开始位置,截取出来的字符串的长度为较大值与较小值之间的差.

substr()  
第一个参数代表开始位置,第二个参数代表截取的长度

### 6.split字符串拆分

### 7.contact/+字符串拼接

str1.contact(str2) :返回一个新字符串，str1和str2都不变。

+ ： 拼接。

### 8.转换大小写

返回新的字符串，原字符串不变。

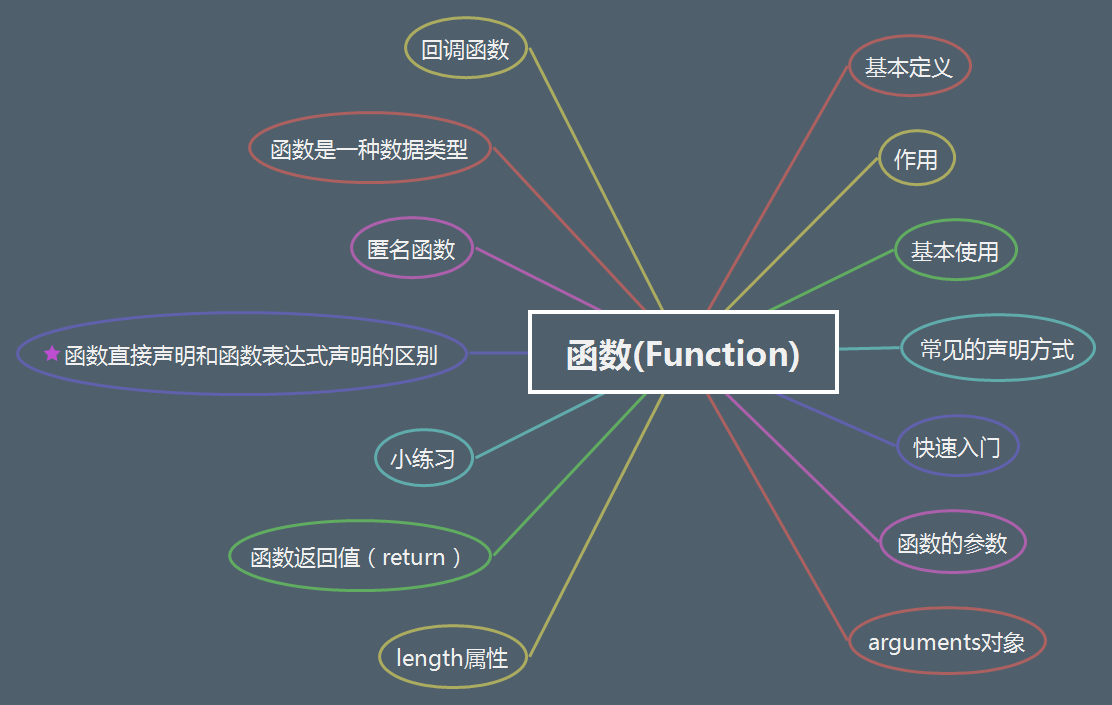
toUpperCase :str.toUpperCase()

toLowerCase

### 3.字符串字符长度

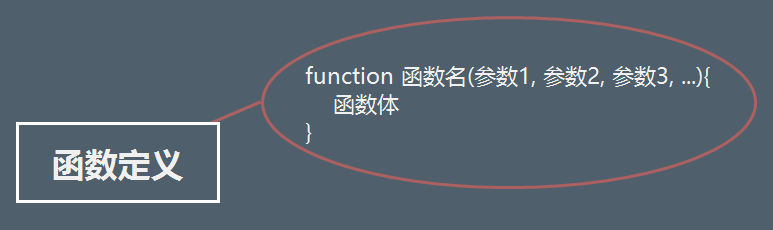
str.length

## js函数

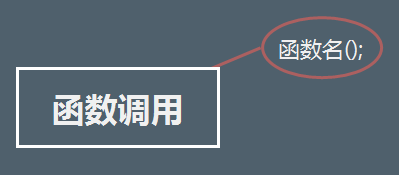


### 基本使用

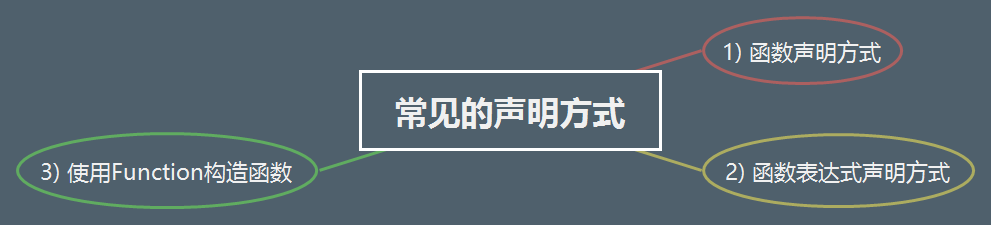
#### 函数定义



#### 函数调用



### 常见的声明方式



#### 1) 函数声明方式

function add(num1,num2){  
 return num1+num2;  
 }

#### 2) 函数表达式声明方式

var add= function(num1,num2){  
 return num1+num2;  
 };

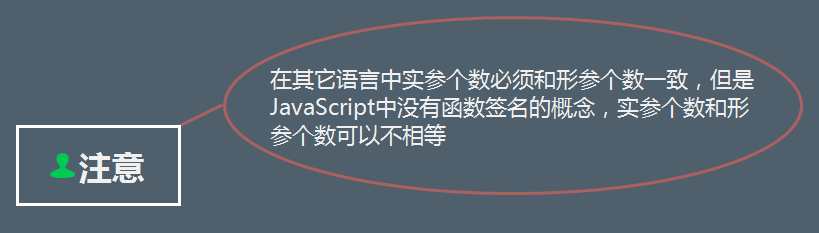
#### 3) 使用Function构造函数

var add = new Function('num1','num2','return num1+num2');

不推荐使用, 主要用于面向对象时理解"函数就是对象, 函数名就是指针"这一概念

### 函数的参数

#### 注意



在其它语言中实参个数必须和形参个数一致，但是JavaScript中没有函数签名的概念，实参个数和形参个数可以不相等。

如果没有传入需要的参数，则函数调用该参数时为调用的为undefined。传入的所有参数都保存在arguments对象中。

### arguments对象

包含了传入函数中的所有参数, arguments并不是一个数组，只是与数组相似, 除了拥有length属性，数组的所有属性和方法都不具备。类数组，可以使用数组的访问方式进行访问。

arguments对象还有一个名叫callee的属性, 该属性是一个指针, 指向拥有这个arguments对象的函数;

### length属性

函数形参的个数

### 函数返回值（return）

当一个函数被调用，通常会从函数的开始执行到结束。如果想提前结束该函数的执行可以使用return语句，return语句后面的所有语句将永远不会执行。

一般return用于返回结果。

#### 注意

如果函数没有显示的使用 return语句 ，那么函数有默认的返回值：undefined

如果函数使用 return语句，那么跟再return后面的值，就成了函数的返回值

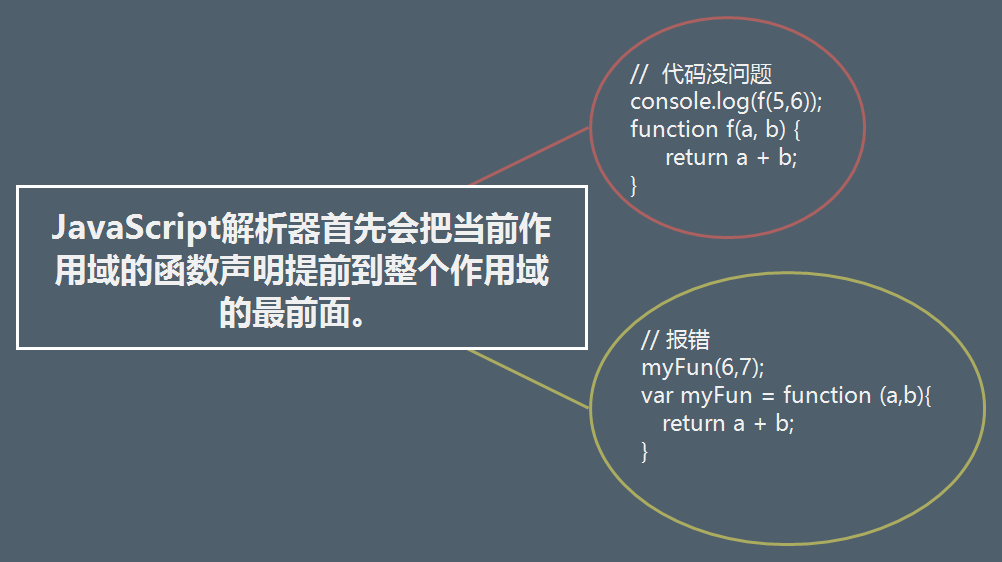
如果函数使用 return语句，但是return后面没有任何值，那么函数的返回值也是：undefined

推荐的做法是要么让函数始终都返回一个值，要么永远都不要返回值。

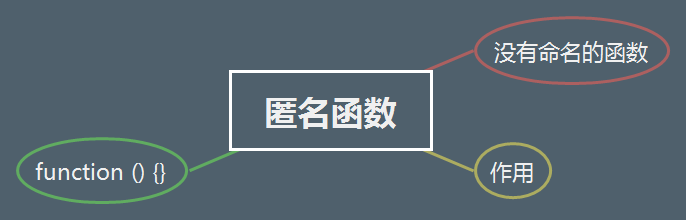
### 函数直接声明和函数表达式声明的区别

JavaScript解析器首先会把当前作用域的函数声明提前到整个作用域的最前面。

类似的还有变量的声明。解析器会把当前作用域的变量声明提到最前面，单是对变量的赋值则按照程序写的顺序进行。



### 匿名函数

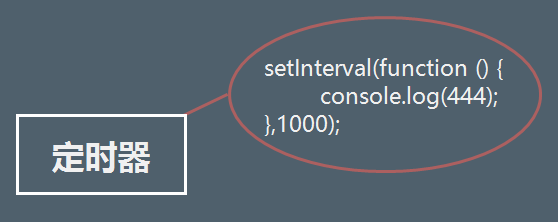


没有命名的函数

#### 作用

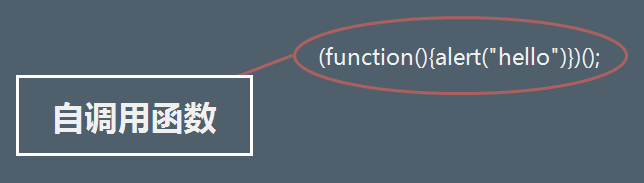
用在绑定事件的时候

#### 定时器



#### 自调用函数

注意括号的位置



(function(){alert("hello")})();

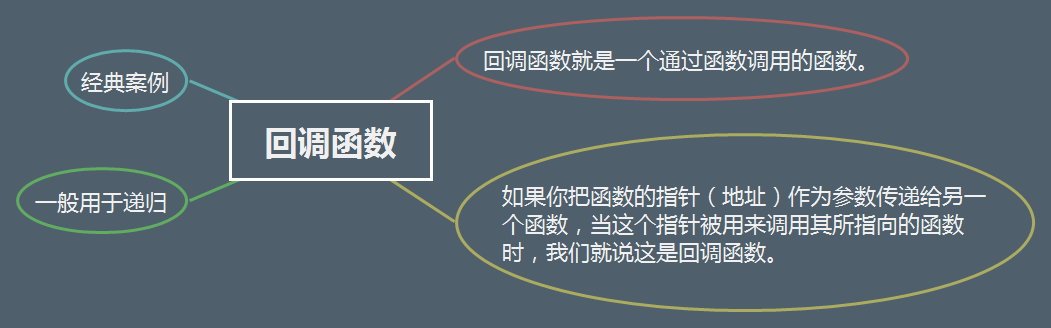
function () {}

### 函数是一种数据类型

console.log(typeof fn);  
console.log(typeof fn());

归根结底，它还是属于object

### 回调函数



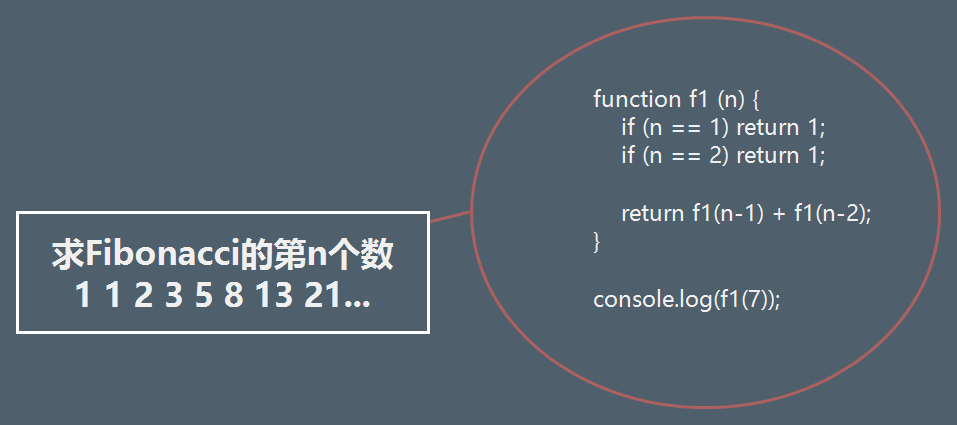
函数可以通过函数名赋值操作进行传递。

回调函数就是一个通过函数调用的函数。

如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。

function fn(num1,num2,demo){  
 return demo(num1,num2);  
 }  
  
 //定义四个规则：加减乘除  
 function test1(a,b){  
 return a+b;  
 }  
 function test2(a,b){  
 return a-b;  
 }  
 function test3(a,b){  
 return a\*b;  
 }  
 function test4(a,b){  
 return a/b;  
 }  
  
 console.log(fn(10,5,test1));  
 console.log(fn(10,5,test2));  
 console.log(fn(10,5,test3));  
 console.log(fn(10,5,test4));

#### 一般用于递归



function f1 (n) {  
 if (n == 1) return 1;  
 if (n == 2) return 1;  
  
 return f1(n-1) + f1(n-2);  
 }  
  
 console.log(f1(7));

## js作用域

类似于python作用域。

注意：

变量可以不用var来声明。

如 a=1

这只直接赋值的变量为全局变量，不管在什么函数中定义，都可以全局使用。

var a,b,c

b，c为全局变量，a不是。

## js对象/内置对象

js是基于对象的语言。

封装、继承、多态

不能很好的实现继承和多态，但是可以借助某些手段实现。

基本对象相当于python的字典。键值对。

对象中可以定义属性和方法。

### 定义与调用

#### 字面量定义

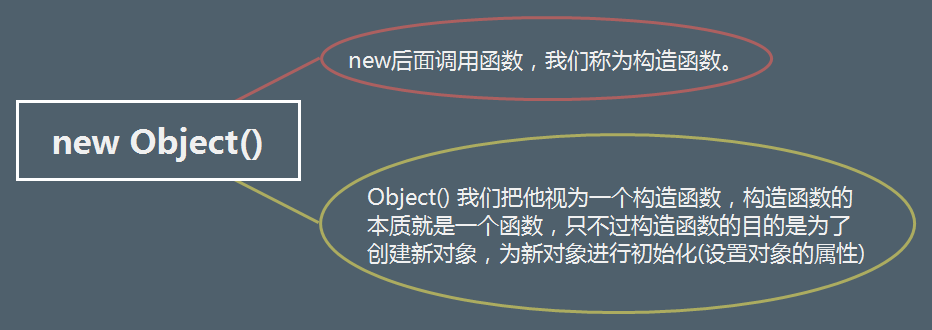
var p = {  
 name : "旋之华",  
 age : 28,  
 sex : true,  
 sayHi : function() {  
 console.log(this.name);  
 }  
 };

p.sayHi(); //对象p调用sayHi()方法，所以sayHi()中的this是对象p

#### 构造函数

能够产生对象的函数叫做构造函数。

通过构造函数创建对象。



构造函数一般首字母大写。

##### 方式一（不常用）

function Person() {

var obj = new Object;

//属性

obj.name = null;

obj.age = null;

obj.study = function(){

console.log(this.name + ' is studing')

}

return obj;

}

var stu1 = Person();

stu1.name = 'xiaoming'

stu1.study()

##### 方式二（常用）

function Dog(name,age){

this.name = name;

this.age = age;

this.eat = function(food){

console.log(this.name + ' is eatting '+ food);

}

}

var hask = new Dog('hask',5);

hask.eat('shit')

或者参数使用对象方式：

function Dog(arg){

this.name = arg.name;

this.age = arg.age;

this.eat = function(food){

console.log(this.name + ' is eatting '+ food);

}

}

hask = new Dog({name:'hask',age:5});

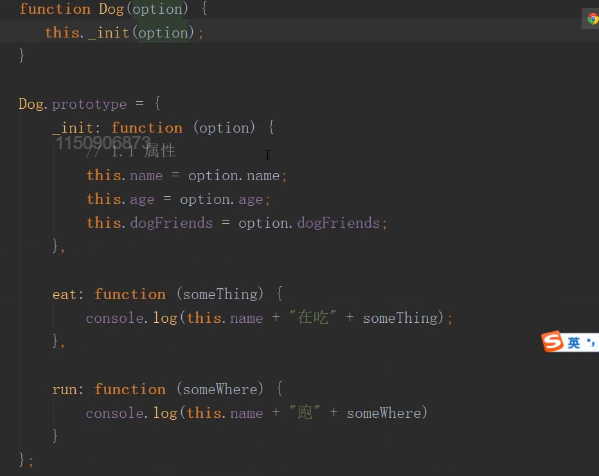
hask.eat('shit')

gold = new Dog({name:'gold',age:4})

console.log(gold == hask)

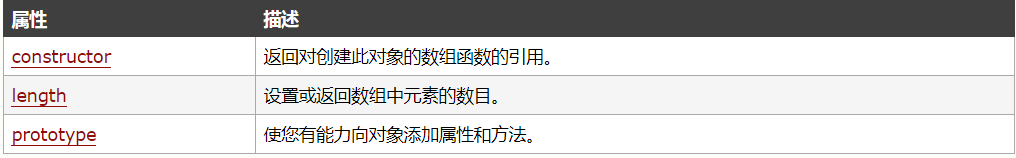
##### 方式三，常用

或者利用prototype来将属性和方法隔开：（常用）



#### 构造器(constructor) 和 原型属性 (prototype)

对象的构造函数内部，有两个默认属性：constructor和prototype。



constructor属性即返回到底是由哪个构造函数产生的该对象。即返回产生该对象的类。

prototype可以向构造函数中添加属性和方法。即动态的向类中添加属性和方法。由此生成的对象中，也含有该属性和方法。这种方法对于动态添加前的实例的对象也可以调用动态天添加的方法。

Dog.prototype.run = function(){

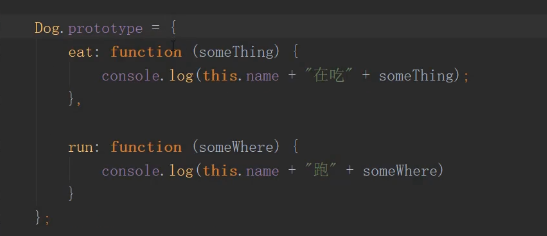
console.log('running')

}

Dog.prototype.gender = 'male';

动态的向构造函数中添加属性和方法。

也可以通过对象的方式进行改变：这种添加方法，在此之前实例化的不能够调用改变后的方法。添加后的实化的不能够调用添加前的方法。



通过constructor获取构造函数，然后再添加方法：

hask.constructor.prototype.walk =function(){

console.log('haha')

}

gold.walk()

注意，即使gold对象是在动态添加方法之前进行的实例化，后面添加的方法也可以使用。

#### 动态添加/改变对象增加属性

直接增加：

obj.atr = ‘hello’

obj[key] = ….

obj.func2= function(){

}

或者直接赋值函数名。

同理，对已有属性或方法进行赋值，可以动态改变属性和方法。

#### 调用

常规的属性，可以使用 . 来调用。

ob.name

ob.func()

也可以通过中括号来调用。

ob[‘name’]

ob[‘func’]()

#### 调用的属性或方法不存在时

对对象不存在的属性和方法进行调用时，

如果属性不存在，会返回undefined，

如果方法不存在，会报错。

### 嵌套

对象可以进行嵌套。对象里面可以嵌套对象、array等所有类型。

### json

JSON的属性必须用双引号引号引起来，对象字面量可以省略

{  
 "name" : "旋之华",  
 "age" : 18,  
 "sex" : true,  
 "sayHi" : function() {  
 console.log(this.name);  
 }  
};

可以存放数组、字典、字符串、数字等

JSON本质上是一种数据交换格式

JSON有两种结构：对象 和 数组，两种结构相互组合从而形成各种复杂的数据结构。

### 遍历对象

使用for..in对对象进行遍历。  
for(var key in obj) {  
 console.log(key + ":" + obj[key]);  
}

key是键，可以通过key取值。

### 内置对象



#### 1.全局对象

    属性:

         Infinity     指定一个正负无穷大的数值

         NaN       指定一个 “非数字” 值

         undefined    指定一个未被赋值的变

  方法:

      decodeURI()       为加密的URI进行解码

       decodeURIComponent() 为加密的URI组件解码

       encodeURI()       将字符串加密为URI

       encodeURIComponent() 将字符串加密为URI组件

       escape(string)      加密一个字符串

       使用escape()对一个字符串进行解码

       eval\_r(string)       判断一个字符串并将其以脚本代码的形式执行

       isFinite(number)     检测一个值是否为一个有限数字,返回True或False

       isNaN(val)      检测一个值是否不是一个有限数字

       Number()         将一个对象的值转换为一个数字

       parseFloat(string)    将一个字符串解析为一个浮点数字

       parseInt(string)     将一个字符串解析为一个整数,不是四舍五入操作,而是切尾，也可以对数字使用，向0的方向取整。

       String(object)      将一个对象值转换为一个字符串

       number(object)

#### 2.Number

   属性:

      MAX\_VALUE       The largest possible value a number in JavaScript can have 1.7976931348623157E+308

      MIN\_VALUE       The smallest possible value a number in JavaScript can have 5E-324

      NaN          Equal to a value that is not a number.

      NEGATIVE\_INFINITY A value that is less than MIN\_VALUE.

      POSOTIVE\_INFINITY A value that is greater than MAX\_VALUE.

      prototype       A static property of the Number object

   方法:

      toString       Returns a string representing the specified object

      valueOf()      返回数学对象的原始值

  toFixed(n) : 保留n位小数，会自动四舍五入

#### 3.Boolean

   属性:

      constructor 所建立对象的函数参考

      prototype    能够为对象加入的属性和方法

  方法:

      toSource()    显示对象的源代码

      toString()    将布尔值转换为字符串,并且返回结果

      valueOf()    返回布尔对象的原始值

4.String

   属性:

      constructor 所建立对象的函数参考

      prototype    能够为对象加入的属性和方法

      length      返回字符串的字符长度

  方法 :

      anchor("name")用来把字符串转换为HTML锚点标记内(<A NAME=>)

      big()      把字符串中的文本变成大字体(<BIG>)

      blink()     把字符串中的文本变成闪烁字体(<BLINK>)

      bold()      把字符串中的文本变成黑字体(<B>)

      fixed()     把字符串中的文本变成固定间距字体,即电报形式(<TT>)

      fontcolor(color)设置字符串中文本的颜色(<FONT COLOR=>)

      Fontsize(size) 把字符串中的文本变成指定大小(<FONTSIZE=>)

      italics()    把字符串中的文本变成斜字体(<I>)

      Link(url)用来把字符串转换-HTML链接标记中(<A HREF=>)

      small()     把字符串中的文本变成小字体(<SMALL>)

      strike()     把字符串中的文本变成划掉字体(<STRIKE>)

      sub()      把字符串中的文本变成下标(subscript)字体((SUB>)

      sup()      把字符串中的文本变成上标(superscript)字体(<SUP>)

      charAt(index) 返回指定索引处的字符

      charCodeAt(index)返回一个整数,该整数表示String对象中指定位置处的字符的Unicode编码

      concat(string2)连接两条或多条字符串

      fromCharCode(num1, num2, …, numN)获取指定的Unicode值并返回字符串

      indexOf(searchString, startIndex) 返回字符串中第一个出现指定字符串的位置

      lastlndexOf(searchString, startIndex) 返回字符串中最后一个出现指定字符串的位置

      match(regex) 在字符串中查找指定值

      replace(regex, newString)将字符串中的某些字符替换成其它字符

      search(regex) 针对某执行值对字符串进行查找

      slice(startIndex, endIndex)将部分字符抽出并在新的字符串中返回剩余部分

      split(delimiter)将字符串分配为数组

      substr(startIndex, length) 从startIndex取,取length个字符

      substring(startIndex, endIndex) 从startIndex和endIndex之间的字符,不包括endIndex

      toLowerCase() 把字符串中的文本变成小写

      toUpperCase() 把字符串中的文本变成大写

      toSource()    显示对象的源代码

      valueOf()    返回字符串对象的原始值

#### 5.Array

   属性:

      constructor 所建立对象的函数参考

      prototype    能够为对象加入的属性和方法

      index      For an array created by a regular expression match, the zero-based index of the match in the string.

　　input      For an array created by a regular expression match, reflects the original string against which the regular expression was matched.

　　length      获取数组元素的个数,即最大下标加1

   方法(13):

　　concat(array1,arrayn)将两个或两个以上的数组值连接起来,合并后返回结果

　　join(string) 将数组中元素合并为字符串,string为分隔符.如省略参数则直接合并,不再分隔

　　pop()      移除数组中的最后一个元素并返回该元素

　　push(value) 在数组的末尾加上一个或多个元素,并且返回新的数组长度值

　　reverse()    颠倒数组中元素的顺序,反向排列

　　shift()     移除数组中的第一个元素并返回该元素

　　slice(start, deleteCount, [item1[, item2[,...[,itemN]]]]) 返从一个数组中移除一个或多个元素,如果必要,在所移除元素的位置上插入新元素,返回所移除的元素

　　sort(compare Function) 在未指定排序号的情况下,按照元素的字母顺序排列,如果不是字符串类型则转换成字符串再排序,返回排序后的数组

　　splice()     为数组删除并添加新的元素

　　toSource()    显示对象的源代码

　　toString()    将数组所有元素返回一个字符串,其间用逗号分隔

　　unshift(value)为数组的开始部分加上一个或多个元素,并且返回该数组的新长度

　　valueOf()    返回数组对象的原始值

#### 6.Object

   属性:

　　constructor Specifies the function that creates an object's prototype.

　　prototype    Allows the addition of properties to all objects.

#### 7.Function

   属性:

　　arguments    An array corresponding to the arguments passed to a function.

　　arity      Indicates the number of arguments expected by the function.

　　caller      Specifies which function called the current function.

　　prototype    Allows the addition of properties to a Function object.

   方法:

　　toString     Returns a string representing the specified object.

#### 9.Date

<http://www.w3school.com.cn/js/jsref_obj_date.asp>

创建 Date 对象的语法：

var myDate=new Date() //myDate是当前使劲

如果 var date = new Date(‘2019/08/10 18:25:15’) //date为某一时间点。

date - myDate 直接获取两个时间差的时间戳。

**注释：**Date 对象会自动把当前日期和时间保存为其初始值。

   属性(1):

　　constructor      所建立对象的函数参考

　　prototype       能够为对象加入的属性和方法

   方法(43):

　　getDay()        返回一周中的第几天(0-6)，星期天为0。

　　getYear()       返回年份.2000年以前为2位,2000(包含)以后为4位

　　getFullYear()     返回完整的4位年份数

　　getMonth()      返回月份数(0-11)

　　getDate()       返回日(1-31)

　　getHours()       返回小时数(0-23)

　　getMinutes()      返回分钟(0-59)

　　getSeconds()      返回秒数(0-59)

　　getMilliseconds() 返回毫秒(0-999)

　　getUTCDay()      依据国际时间来得到现在是星期几(0-6)

　　getUTCFullYear()    依据国际时间来得到完整的年份

　　getUTCMonth()     依据国际时间来得到月份(0-11)

　　getUTCDate()      依据国际时间来得到日(1-31)

　　getUTCHours()     依据国际时间来得到小时(0-23)

　　getUTCMinutes()    依据国际时间来返回分钟(0-59)

　　getUTCSeconds()    依据国际时间来返回秒(0-59)

　　getUTCMilliseconds()依据国际时间来返回毫秒(0-999)

　　getTime()       返回从1970年1月1号0:0:0到现在一共花去的毫秒数

　　getTimezoneoffset() 返回时区偏差值,即格林威治平均时间(GMT)与运行脚本的计算机所处时区设置之间相差的分钟数)

　　parse(dateString)   返回在Date字符串中自从1970年1月1日00:00:00以来的毫秒数

　　setYear(yearInt)    设置年份.2位数或4位数

　　setFullYear(yearInt)设置年份.4位数

　　setMonth(monthInt) 设置月份(0-11)

　　setDate(dateInt)    设置日(1-31)

　　setHours(hourInt) 设置小时数(0-23)

　　setMinutes(minInt) 设置分钟数(0-59)

　　setSeconds(secInt) 设置秒数(0-59)

　　setMilliseconds(milliInt) 设置毫秒(0-999)

　　setUTCFullYear(yearInt) 依据国际时间来设置年份

　　setUTCMonth(monthInt) 依据国际时间来设置月(0-11)

　　setUTCDate(dateInt)     依据国际时间来设置日(1-31)

　　setUTCHours(hourInt)    依据国际时间来设置小时

　　setUTCMinutes(minInt) 依据国际时间来设置分钟

　　setUTCSeconds(secInt)    依据国际时间来设置秒

　　setUTCMilliseconds(milliInt)依据国际时间来设置毫秒

　　setTime(timeInt)    设置从1970年1月1日开始的时间.毫秒数

　　toGMTString()     根据格林威治时间将Date对象的日期(一个数值)转变成一个GMT时间字符串,如:Weds,15 June l997 14:02:02 GMT

　　toUTCString()     根据通用时间将一个Date对象的日期转换为一个字符串

　　toLocaleString()    把Date对象的日期(一个数值)转变成一个字符串,使用所在计算机上配置使用的特定日期格式

　　toSource()       显示对象的源代码

　　toString()       将日期对象转换为字符串

　　UTC(yyyy, mm, dd, hh, mm, ss, msec)返回从格林威治标准时间到指定时间的差距,单位为毫秒

　　valueOf()       返回日期对象的原始值

#### 10.Math

<http://www.w3school.com.cn/jsref/jsref_obj_math.asp>

   属性:

　　constructor      所建立对象的函数参考

　　prototype       能够为对象加入的属性和方法

　　E           欧拉常量,自然对数的底(约等于2.718)

　　LN2          2的自然对数(约等于0.693)

　　LN10          10的自然对数(约等于2.302)

　　LOG2E         以2为底的e的对数.(约等于1.442)

　　LOG10E         以10为底的e的对数(约等于0.434)

　　PI           ∏的值(约等于3.14159)

　　SQRT1\_2        1/2(0.5)的平方根(即l除以2的平方根,约等于o.707)

　　SQRT2         2的平方根(约等于1.414)

    方法:

　　abs(x)     返回数字的绝对值

　　acos(x)    返回数字的反余弦值

　　asin(x)    返回数字的反正弦值

　　atan(x)    返回位于-PI/2 和 PI/2 的反正切值

　　atan2(y,x) 返回（x,y）位于 -PI 到 PI 之间的角度

　　ceil(x)    返回 x 四舍五入后的最大整数

　　cos(x)     返回一个数字的余弦值

　　exp(x)     返回 E^x 值

　　floor(x)    返回 x 四舍五入后的最小整数

　　log(x)     返回底数为E的自然对数

　　max(x,y)    返回 x 和 y 之间较大的数

　　min(x,y)    返回 x 和 y 之间较小的数

　　pow(x,y)    返回 y^x 的值

　　random()    返回位于 0 到 1 之间的随机函数

　　round(x)    四舍五入后取整

　　sin(x)     返回数字的正弦值

　　sqrt(x)    返回数字的平方根

　　tan(x)     返回一个角度的正切值

　　toSource() 显示对象的源代码

　　valueOf() 返回数学对象的原始值

# js进阶动画

## BOM及常用内置方法

BOM浏览器对象，即window对象。定义的全局变量全都是window的属性。

所有全局的函数都是window的方法。

### 方法

#### 1.设置浏览器跳转

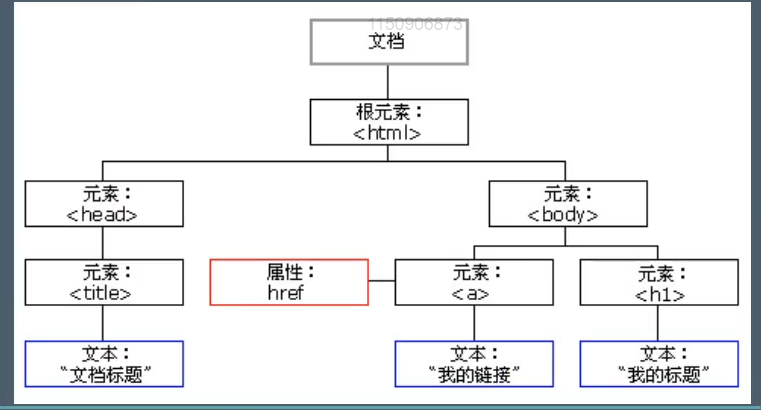
window.location.href = 地址

设置浏览器跳转

## DOM及常用内置属性方法

DOM对象，包含于BOM对象，是window的一部分。

DOM树：



整个文档对象，window.document获取整个文档对象。

常用的对象方法：

document;  
document.childNodes;  
document.head;  
document.body;  
document.title = "测试";  
document.close():关闭用document.open()打开的输出流，并显示选定的数据。

document.open():打开一个流，以收集document.write和document.writeln的输出。

document.write()向文档写html表达式或js代码

document.writeln和write一样，但是会自动在末尾加换行。

### Dom作用：

1.找元素/对象

2.设置元素属性

3.设置元素样式

4.动态添加/删除元素

5.事件-触发响应

### 方法

#### 1.获取整个DOM

document.childNodes：获取document中所有的子节点

#### 2.document.head

#### 3.document.body

#### 4.document.title

## 事件

### js常用事件



onchange:可以用于监听上传文件。

### 绑定事件

box.onclilk = 匿名函数

或者将函数写好，然后赋值给事件。

function fn(){

}

box.onclick = fn //注意，只是函数名，没有括号。

## 选择器

通过选择器获取标签对象。

### 常用选择器

getElementById：根据指定元素的id属性返回元素

getElementsByName：返回所有指定name属性的元素

getElementsByTagName：返回所有指定标签名字的元素

getElementsByClassName：返回所有指定类名的元素

querySelector：根据选择器规则返回第一个符合要求的元素，具体用法和querySelector用法一样，区别就是只获取第一个复合要求的元素。

querySelectorAll：根据选择器规则返回所有符合要求的元素

### querySelectorAll:

参照css选择器。

#### 1. ID选择器

使用ID选择器时，需在前面添加“#”，区分大小写，语法如下：

document.querySelector('#id'); //等同于document.getElementById('id');

#### 2. 元素选择器

元素选择器通过指定的标签查询元素，此时querySelectorAll等同于getElementsByTagName，语法如下：

document.querySelectorAll('a'); //获取页面上的所有a元素并返回元素

#### 3. 类选择器

使用元素的样式类获取一个或一类元素，样式名字前使用“.”（英文句号）开头，语法如下：

document.querySelectorAll('.btn'); //获取所有样式类中包含btn类名的元素

#### 复合选择器

div.bt

与的关系，选择类名为bt的div

#### 4. 分组选择器/并列选择器

document.querySelectorAll('a,p'); //获取页面上所有a元素和p元素，并通过一个列表返回

document.querySelectorAll('.btn,.txt'); //获取页面上所有包含btn和txt样式类名的元素

#### 5. 属性选择器

获取页面上包含指定属性的元素，属性名称可以是元素原生属性和用户自定义属性，语法如下：可以不要前面的元素名

document.querySelectorAll('a[target="\_blank"]'); //获取页面上所有target属性为\_blank的a元素

document.querySelectorAll('img[data-id]'); //获取页面上所有带有自定义属性data-id的img元素



#### 6. 后代选择器

主要用于选择作为某元素后代的元素，规则左边的选择器一端包含两个或多个用空格分隔的选择器，如div a可以理解为查找所有被div包围的所有a元素，语法如下：

document.querySelectorAll('div a'); //获取页面上所有被div包含的a元素

document.querySelectorAll('div .btn'); //获取页面上所有被div包含的带有btn样式类名的元素

#### 7. 子元素选择器/直接后代选择器

后代选择器会将元素底下的所有相关元素都搜索出来，如果想进一步缩小范围，可以使用子元素选择器，只会选择某个元素的一级子元素，子元素用“>”（大于号）表示，代码如下：

<html>

<div id="first">

<div></div>

<div></div>

</div>

</html>

<script>

document.querySelectorAll('html>div'); //只返回一个id为first的div元素

</script>

#### 8. 相邻兄弟选择器（比较少用）

选择紧接在另一个元素后的元素，而且两者有相同的父元素，相邻兄弟选择器使用“+”（加号），代码如下：

<div>

<div></div>

<div></div>

</div>

<p id="p1"></p>

<p id="p2"></p>

<script>

document.querySelectorAll('div+p'); //只返回一个id为p1的p元素

</script>

#### 9. 伪类选择器/序号选择器

“:first-child”表示选择元素的第一个子元素，“:last-child”表示选择元素的最后一个子元素，“:nth-child(n)”表示选择元素的第n个子元素。“:first-child”的使用例子，代码如下：

<div>

<p id="p1"></p>

<p id="p2"></p>

</div>

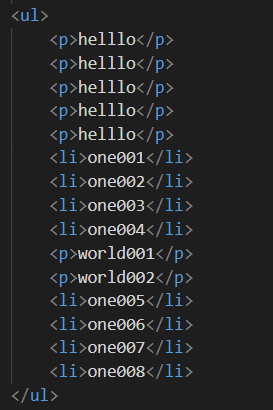
<script>

document.querySelectorAll('p:first-child'); //只返回一个id为p1的p元素

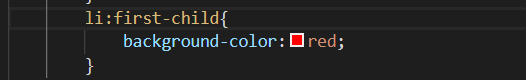
</script>

找到该冒号前选择器的同类别的所有标签，按照冒号后的指示进行选取。如果冒号前的选择器为标签名，则还要同时满足匹配标签。如果不能同时满足，则选取为空。

如：



使用



选取时，则匹配为空。

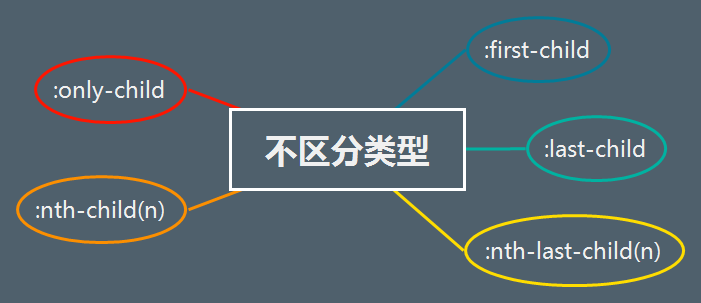
原因：li标签的同级标签中第一个为p标签，不能满足li标签且其是第一个的条件，所以匹配到为空。

同理，对于

li:nth-child(odd)奇数选择也同样不会选择第一个，因为不能同时满足。只会选取在整个整体同级标签排序中是奇数且是li标签的标签。

而是用区分类型的标签时，则不存在这种问题。他先选取所有该类型的同级标签，然后再进行排序、查找。

##### 不区分类型



###### :first-child

选中同级别中第一个标签

li:first-child

###### :last-child

选中同级别中最后一个标签

###### :nth-last-child(n)

选中同级别中倒数第n个标签

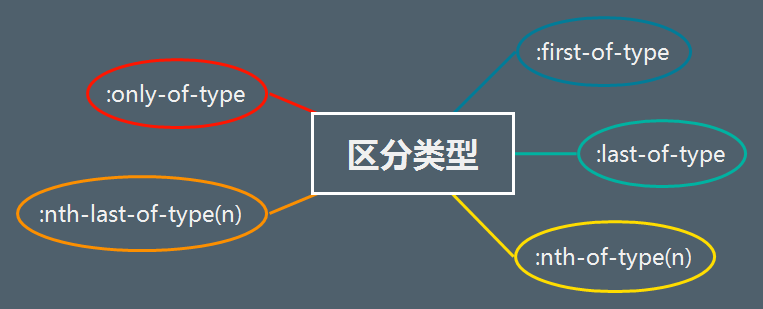
###### :nth-child(n)

选中同级别中第n个标签

###### :only-child

选中同级别中唯一子元素标签

##### 区分类型



###### :first-of-type

选中同级别中同类型第一个标签

li:first-of-type

###### :last-of-type

选中同级别中同类型最后一个标签

###### :nth-of-type(n)

选中同级别中同类型第n个标签

###### :nth-last-of-type(n)

选中同级别中同类型倒数第n个标签

###### :only-of-type

选中同级别中唯一同类型子元素标签

##### 奇偶选择



###### :nth-child(odd)

选中级别中所有奇数

###### :nth-child(even)

选中同级别中所有的偶数

###### :nth-child(xn+y)

x,y是用户自定义的 n为计数器,从0开始递增选择, 到元素的总个数停止

### 获取父节点

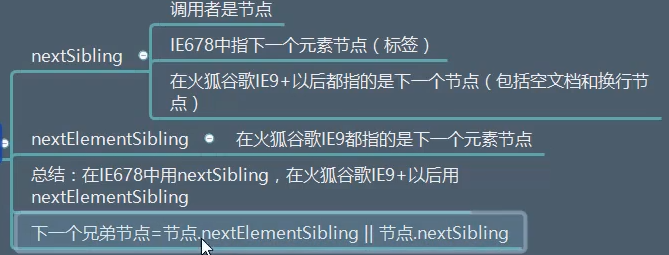
ele.parentNode //没有括号

ele为元素对象，返回该元素对象的父节点。一个元素只有一个父节点

### 获取兄弟节点

2.nextSibling/nextElementSibling

获取紧挨自己的下一个同级节点



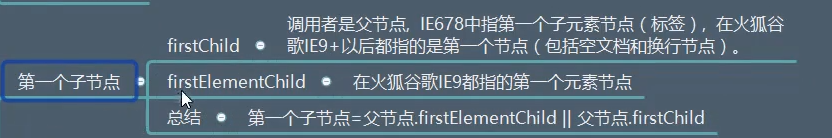
3.previousSibling/preciousElementSibling

获取紧挨自己的上一个同级节点

### 获取子节点

调用者是父节点

1.第一个子节点 firstChild/firstElementChild



2.最后一个子节点 lastChild/lastElementChild

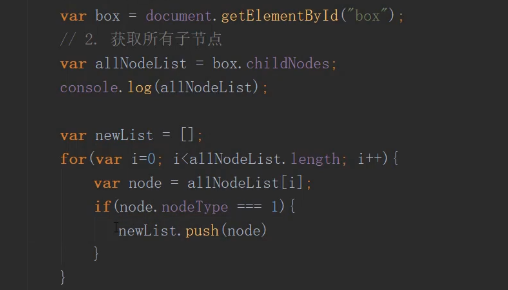
3.childNodes

返回所有子节点

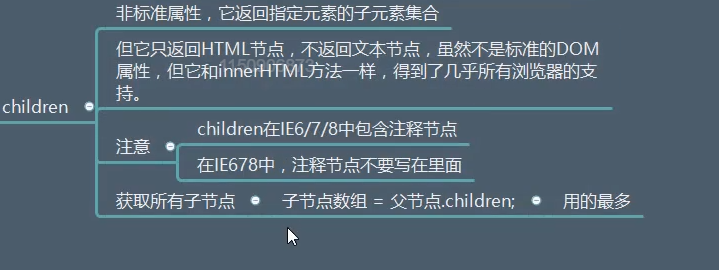


nodeTpye == 1 表示的是元素节点。

挑选元素子节点：



4.children



## 节点操作 增删改查 内容

### 1.创建节点

document.createElement("img"); 创建一个标签  
box.appendChild(img); 将标签插入到某个位置。

box.insertBefore(img, btn);

### 2.删除节点

删除某子节点 removeChild() box.removeChild(img) 参数为标签对象。

remove：完全删除被选元素。包含标签本身和其子标签。img.remove()

var btn = document.getElementById("btn");  
word.parentNode.removeChild(word); // 自杀  
 // btn.remove();

### 3.克隆节点

img.cloneNode(false) 返回该对象的浅拷贝。只拷贝一层，不拷贝子元素

img.cloneNode(true) 返回该对象的深拷贝，拷贝所有，包括子元素。

### 追加节点appendChild()

appendChild()

概念：把要添加的节点添加到指定父级里面的最后面，所以也叫追加

使用方式：父级节点.appendChild( 要添加的节点 )

### 插入节点insertBefore()

insertBefore()

(1)、概念：把要插入的节点添加到指定父级里面的指定节点之前

(2)、使用方式：父级节点.insertBefore( 要插入的节点，指定节点 )

### 使用字符串创建节点

### div = document.createElement('div')

### a = '<a href = "www.baidu.com">百度一下</a>'

div.innerHTML = a

这就将a标签插入到div中。

### 4.标签内容的获取

#### object.innerText

获取object中的所有文本，嵌套的标签也只是获取标签中的文本内容，而不是标签。

#### object.innerHTML

获取object标签中的所有内容（本标签两个尖括号之间的内容，不包括该标签的属性），包括被嵌套的标签。

#### object.value/val

不加括号。主要针对单标签内容的获取。如input/单选中用户输入的内容。如上传文件的路径是保存在input.value中的。

## 节点属性操作

### 1.获取节点属性

获取：getAttribute(属性名称)

对于元素自带的一些属性，也可以通过 对象.属性 的格式来获取属性，这是对象的基本用法。

### 2.设置节点属性

setAttribute(属性名称，属性值) 字符串形式

### 3.删除节点属性

删除：removeAttribute(名称)

### 4.常用的一些属性

img.src 与 img.getAttribute(‘src’)的不同：img.src获取的地址会自动补全，包括前面的http等网络地址。而getAttribute则是src里面是什么就获取什么。

a.href：获取超链接地址。

img.alt 获取alt值

## js定时器

### 基本概念



func可以以双引号引用，也可以不用双引号。后面为1000ms

在js中的定时器分两种：1、setTimeout() 2、setInterval()

#### 1.setTimeOut()

只在指定时间后执行一次

//定时器 异步运行

function hello(){

alert("hello");

}

//使用方法名字执行方法

var t1 = window.setTimeout(hello,1000);

var t2 = window.setTimeout("hello()",3000);//使用字符串执行方法

window.clearTimeout(t1);//去掉定时器

#### 2.setInterval() setTimeout() clearInerval()

在指定时间为周期循环执行

//实时刷新时间单位为毫秒

var time=setInterval('refreshQuery()',8000);

/\* 刷新查询 \*/

function refreshQuery(){

console.log('每8秒调一次')

}

setTimeout(function(){  
 //console.log(time)  
   
 window.clearInterval(time)  
},5000)

两种方法根据不同的场景和业务需求择而取之，

一般情况下setTimeout用于延迟执行某方法或功能，只执行一次。

setInterval则一般用于刷新表单，对于一些表单的假实时指定时间刷新同步，动画效果等。

setinterval会返回一个值，代表该函数

获取这个值后，把他传给clearinterva，来停止setinterval的运行。

#### 开启定时器

直接在事件函数中设置即可。

button.onclick = function(){

var t = setInterval(func,1000);

}

func为函数，也可以直接写匿名函数，或者写字符串形式 ”func()”。

## this

this相当于python中的self ，在哪个对象中，就指代哪个对象。

有两个例外：

1.事件源中this是指向事件源的元素对象。

box.onclick = function(){

console.log(this)

}

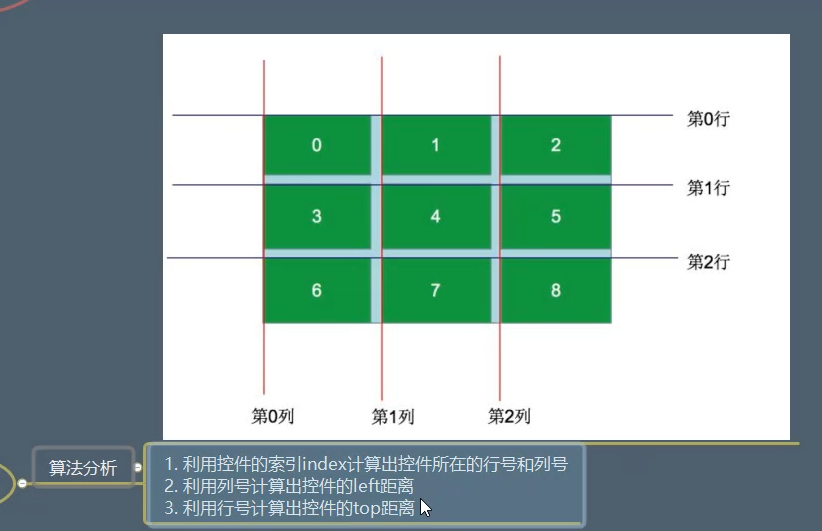
虽然这个函数在全局中，但是这个this是指向box的，指代box对象，而不是window对象。

2.定时器对象

## 常用特效：

### 1.Tab选项卡

### 2.九宫格算法



大盒子相对定位，子盒子绝对定位。根据每个盒子的index，来算出其位置。

### 3.dom的增删改查

### 4.倒计时

使用date类获取当前时间和设置的终点时间

两个时间相减，得到时间差（毫秒）。

利用时间差即可，采用定时器不断更新时间差即可。

当前时间：

curent\_time = new Date();

终点时间：

last\_time = new Date(“2020/01/01 12:20:25”)

### 5.时钟

利用css的transform:rotateZ 来旋转时针。

transform-origin可以用来设置旋转的中心。