**Js的功能：**

1、实现网页动效

2、实现数据验证

3、制作简单的网页游戏

4、实现网页的一些功能（内容的读写）

5、记录用户信息（cookie）

6、后台的应用

7、分辨浏览器

**历史**

1992 蒂姆 伯纳斯 李 html http

1995 NetScape liveScript+Sun java javaScript 和sun公司合作改名

为javaScript

1997 ECMA ECMAScript

1998 ECMAScript 1.0

1999 ECMAScript 2.0

2001 ECMAScript 3.0

2007 ECMAScript 3.1==>5

2015 ECMAScript 6.0==2015

2016 ECMAScript 2016

Es5 IE8以上 以下es3 一般用插件 JQuery

**注释**：行注释ctrl+/

快注释ctrl+shift+/

**组成部分**

ECMAScript

js基础语法

变量 运算符 流程控制 函数 数组 对象...

BOM浏览器对象模型（用js操作浏览器）+DOM文档对象模型（用js操作

文档）页面效果

Ctrl+alt+L 自动调整书写格式

**准备工具：**火狐，谷歌

**引用方式：**

**1、域名或重定向**

a标签 from标签

<a href="javascript:alert(1)">连接</a>

<form action="javascript:alert(2)">

<input type="submit">

</form>

**2、在事件的后面调用**

div标签内 onclick

<div onclick="alert(3)"></div>

**3、嵌入script标签的方式**

<script>

alertI(2);

</script>

**4、外部引用**

<script src=“”></script>

**输出工具进行调试：**

Alert() 终止代码执行

Consloe.log() 控制台输出

Document.write() 可以写标签

**变量**：存储数据的一个容器

得到变量通过 关键字 var let

Var 声明变量

Var 变量名字=赋值

不赋值时 变量为undefined

使用var声明变量的过程会被优先解析(解析声明过程 只在当前块中)

Alert(aa)

Var aa=1;

**变量的声明和赋值四种形式**

声明赋值

var num=2;

先声明后赋值

var aa;

aa=3;

声明多个变量后赋值

var name,age,sex;

name="dtt";

age=20;

sex="women";

声明多个变量同时赋值

var a1=2,a2=4,a3=5;

**变量的命名规范:**

1、必须以字母 下划线- $开头

后面可以跟字母 数字 下划线 $

2、不能使用关键字（var let）和保留字

3、javascrip 自身的命名习惯

驼峰命名法 studentAverageGrade getElementsByClassName

首字母大写 Object() Function() Array()

4、命名要有意义

**let和var的区别：**

**（覆盖问题）**

1、var可以重新声明（重新声明不赋值还是原先的值）也可以重新赋值（原先的值被

覆盖掉）；可以优先解析

2、Let在es6版本中新增的；可以重新赋值 不能重新声明；

let aa=2; let aa=3;是错误的；

不能优先解析

3、es6中新增的还有const 声明常量的，不能重新声明 不能重新赋值，声明的

同时必须赋值；

使用var声明变量的过程会被优先解析(解析声明过程 只在当前块中)

**Javascript的数据类型**

**1、初始类型**（栈区里保存）栈区是用来保存短的有固定的长度限制

1>数值

1 数字

2 小数

3 负数

4 进制数（十六进制 0x11 八进制011或者0o11二进制 0b11）

5 科学计数法（2e+3=2\*10的三次方）

6 Number.MAX\_VALUE Number.MIN\_VALUE 最大值 最小值

2>字符串

1 “”

2 ‘’

3 “ ’’ ”

4 ‘ “” ’

5 es6 模板字符串 ``

3>undefined undefined

4>布尔值 true(1) false(0)

5>空类型 null

一个占位符 用来清空对象

Typeof 得到某一个数据的类型

“number” ”string” “undefined” “boolean” “object” 引用类型“object”

6>symbol()————唯一的值

**2、引用类型**（堆区里保存）堆区 没有固定的长度 在栈区里保存地址来

通过地址查找

1>函数

2>数组

3>普通的对象

**运算符**

**1、算数运算符**

+ - \* / % ++ -- （es6\*\*）

var num1=10;

var num2=10;

var he=num1+num2;

console.log(he); 20

var cha=num1-num2;

console.log(cha); 0

var cheng=num1\*num2;

console.log(cheng) 100

var chu=num1/num2;

console.log(chu); 1

var num3=20;

var num4=num3++; //先赋值后运算

console.log(num3); 21

console.log(num4); 20

var num5=++num3; //先运算后赋值

console.log(num3); 22

console.log(num5); 22

console.log(100%3);

console.log(2\*\*5); 2的五次方 结果32

字符串类型和任意其他类型进行相加的运算 结果都为连接之后的新字符串

NaN Not a Number

**2、关系运算符(比较运算符)**

> < >= <= == === != !==

对符号两边的数值进行比较 得到一个布尔值

\*两边是数值：

var num1=10;

var num2=20;

var r=num1!=num2;

console.log(r);

\*数值和字符串

能被转化为数值 比较大小

转化不了 返回false

\*两边都是字符串：

按ASCII码比较 97 --a 65--A 48--0

只要是字符串和字符串比较 是从第一位开始比较ASCII码的大小

直到能比较出大小为止

\*== 只判断值是否相等

=== 不仅判断值是否相等 还判断类型是否相等

\*undefined和null是相等的 undefined==null true

**3、赋值运算符**

= += -= \*= /=

val1+=10; 相当于 val1=val1+10;

var val2; val2+=val1; val2没有被赋值相当于undefined 所以输出值为NaN

**4、逻辑运算符**

&&与 ||或 !非

1>对于符号两边的布尔值进行运算 得到另一个布尔值

&&True true true

True false false

False false false

同真才为真

||

True true true

True false true

False false false

有真就为真

!

True false

False true

2>逻辑运算符的两边不是布尔值

假值：0 NaN “” null false undefined

&&

真值1 真值2 输出 真值2 （判断第二个值为真才为真）

假值1 假值2 输出 假值1

真值 假值 输出 假值 （需要判断第二个值才知道真假）

||

真值1 真值2 输出 真值1 （有真就为真 不需要往后判断）

真值 假值 输出 真值

假值1 假值2 输出 假值2 （需要判断第二个值才知道真假）

!

真值 false

假值 true

**5、一元运算符**

+ - 正号 负号 typeof new delete

**6、三元运算符**

布尔值？当布尔值为真的时候表达式的值：当布尔值为假的时候表达式的值

var a=10;

var b=20;

var c=**a>b?a:b;**

console.log(c);

**7、特殊运算符**

, 声明多个变量的时候进行分割

() 算术优先级的作用 执行函数

**流程控制**

**1、顺序结构**

代码默认执行顺序（var会被优先解析）

**2、选择结构**

1、分支结构

\*有选择的执行某一块代码

2、条件结构

\*单路分支 if(){}

\*双路分支 if(){}else{}

\*多路分支 if(){}else if(){}

\*嵌套分支 花括号里还可以进行分支结构的书写

\*Switch(值的名称){

Case “1”:document.write(“星期一”);

Break;

Case “2”:document.write(“星期二”);

Break;

Default:alert(“”) （相当于else）

}

parseInt() 把字符串转化为数值

**3、循环结构**（重复执行某一段代码）

**\*for**

For(循环的控制语句){循环具体要执行的内容}

1、重复的执行内部的代码

2、有一个不断变化的值可以使用

1~100累加的和

var ***sum***=0;  
 for(var ***i***=1;***i***<=100;***i***++){  
 ***sum***=***sum***+***i***;  
// sum+=i;  
  
 }console.log(***sum***);

1~100 所有奇数  
 for (var ***i***=0;***i***<=100;***i***++){  
 if(***i***%2==1){  
 console.log(***i***)  
 }  
  
 }

ul的循环  
 document.write("<ul>")  
 for(var ***i***=0;***i***<10;***i***++){  
 document.write("<li>"+(***i***+1)+"</li>")  
 }  
 document.write("</ul>")  
 document.write("<ul>")  
 for(var ***i***=0;***i***<10;***i***++){  
 document.write(`<li>${***i***+1}</li>`)  
 } 只有在模板字符串（反引号）下 ${}  
 document.write("</ul>")

//十行十列表格  
 document.write("<table border='1'>");  
 for(var ***i***=0;***i***<10;***i***++){  
 if(***i***%2==0){  
 document.write("<tr>");  
 }else{document.write("<tr

bgcolor='pink'>");

}  
 for(j=0;j<10;j++){  
 document.write("<td>000</td>")  
 }  
 document.write("</tr>");  
 }  
 document.write("</table>");

//鸡兔同笼  
// 头36 脚100  
 var ***n***=0;  
 loop1://给循环命名的名字  
 for(var ***i***=0;***i***<=36;***i***++){  
 for(var ***j***=0;***j***<=36;***j***++){  
 if(***i***+***j***==36&&2\****i***+4\****j***==100){  
 console.log(`鸡的个数是${***i***},

兔的个数是${***j***}`)  
 break loop1;符合条件后跳出

loop1循环  
 }  
 ***n***++;//查看循环的次数  
 }  
 }

console.log(***n***);//在控制台输出循环的次数

增加断点，在控制台选source 之后鼠标点击行刷新后就能查看运行过程。再次点击

取消断点

**\*while**

While(一个判断语句){具体要执行的语句}

1、完全可以模拟for循环

2、可以执行次数不固定的判断

**\*do while**

Do()while{}

先执行一次后判断

Break：跳出整个循环，继续循环后面的代码。

Continue：跳出当前这一次循环，继续进行下一次（满足条件的前提下）

指定跳出哪一个循环

**函数**

封装起来可以被重复调用的代码块

函数的好处：1、调用方便

2、维护方便

3、程序简洁

4、其他

**函数的声明方式：**

1、function 关键字 function 函数名(){} \*会被优先解析 命名规范同上

2、匿名函数 字面量 var 名称=function(){}

3、实例化构造函数 var date=new Date() div.onclick=function(){} ...

**函数的调用方式：**

1、函数名() 变量名()

2、事件后调用

3、自调用 (function fn3(){alert})();

**参数**：例hanshu.html

定义在函数当中的，用来控制函数执行细节的一个变量

1. 实参：在调用函数时给他的值
2. 形参：在定义函数时使用的变量（本身无意义 在调用时赋值有意义）

注：形参和实参 从前往后一一对应

参数的个数可以是多个

参数的数据类型可以是任意的类型

如果没有传实参 对应的形参undefined

默认值(缺省值)

function *table*(row,col,color){//形参  
 */\*if(row==undefined){  
 row=10;  
 }  
 if(col==undefined){  
 col=10;  
 }  
 if(color==undefined){  
 color="#fff";  
 }\*/* row=row||10;  
 col=col||10;  
 color=color||"#fff"  
 document.write(`<table border='1' bgcolor=`+color+`>`);  
 for(var i=1;i<=row;i++){  
  
 document.write(`<tr>`)  
  
 for(var j=1;j<=col;j++){  
 document.write(`<td>${i}${j}</td>`);  
 }  
 document.write("</tr>")  
 }  
 document.write("</table>");  
}  
*table*(10,10,"red" //实参);  
*table*(5,6,"#fff");  
*table*();

**返回值**

函数运行之后产生的一个结果

1、没有定义返回值的时候

函数运行的结果就是undenfined

2、return是可有可无的

3、返回值只会有一个 可以有多个 但只会返回一个值

4、return之后的代码都不会再继续执行

function *pf*(a){  
 var r=a\*a;  
 var rs=a\*a\*a;  
 return r;  
 return rs;//return只能执行一次且return之后的代码不继续执行  
 alert(r);  
}  
alert(*pf*(4));

**Arguments (参数) 对象**

\*保存所有实参的信息

\*通过arguments.length　参数的个数

\*Arguments[0] 访问具体的第几个参数

\*arguments.callee 访问函数自身 (function(){})();

**函数重载**

根据函数的个数和数据类型的不同，执行不同的函数体

模拟函数重载

function *sum*() {  
 var he=0;  
 if(arguments.length==1) {  
 for (var i = 1; i <= arguments[0]; i++) {  
 he += i;  
 }  
 }else{  
 for(var j=0;j<arguments.length;j++){  
 he+=arguments[j];  
 }  
 }return he;  
 }console.log(*sum*(10,10))  
  
  
function *say*(val) {  
 if(typeof val=="string"){  
 console.log(`名字是${val}`)  
 }else if(typeof val=="number"){  
 console.log(`年龄${val}`)  
 }  
}  
*say*("dtt")  
*say*(20)

**函数的作用域**

一段代码起作用的范围

\*环境 宿主环境 (浏览器)

执行环境（定义作用域的地方）

·全局环境（整个html页面）

·函数环境 （函数的内部）

\*全局变量 和 局部变量

定义在全局环境中的变量 称为全局变量

定义在函数环境中的变量 称为局部变量

变量的作用域：定义变量的环境当中的任意位置

·保护全局变量不受污染

·方便对于变量的命名

·函数的参数也是局部变量

**作用域链**

多个函数重叠嵌套的时候，他们的作用域也会相互嵌套，在最内层访问变量，

会沿着作用于的嵌套顺序，从当前的环境开始去找这个变量，直到找到为止。

这种访问变量的方式称为作用域链

var ***ee***=40;  
function *fn6*() {  
 var ee=30;  
 function *fn7*() {  
 var ee=20;  
 function *fn8*() {  
 var ee=10;  
 alert(ee);  
 }*fn8*();  
 }*fn7*();  
}  
*fn6*();

**预解析顺序**

·首先按照script块 依次执行

·每一块中，优先解析 var function

·解析所在的环境

·按照流程执行

var ***val***=10;  
function *fun*(val) {  
 alert(val);//undenfined  
 var val=20; 若将var换成let,上边弹出error会出现暂时性死区  
 alert(val);//20  
}

**ES6 块级作用域**

If(){} for(){} {} function(){}

在块级作用域中，用let声明的变量只能在当前的作用域中访问 const相同

闭包函数，用var

if(true){  
 var ***aa***=1;  
}alert(***aa***);//弹出  
  
if(true){  
 let ***lt***=1;  
}alert(lt);//报错

*{  
 var num1=10;  
 let num2=20;  
 console.log(num1,num2);//10 20  
}  
 console.log(num1);  
 console.log(num2);//num2 is not defined\*/*

**函数的特殊应用**

1、**回调函数**

把一个函数作为另外一个函数的参数使用，这个函数就称为回调函数

1. **递归函数** 一定要有限制条件 不然就进入了死循环

\*在函数的内部调用自身

\*条件一定要有改变

\*在函数内部一般有判断

\*函数的参数一定要有改变

1. **闭包函数**

在函数的外部访问函数内部声明的变量 这种形式称为闭包。

内存一直被占用

**ES6中函数新增的语法**

1、**箭头函数** 匿名函数的简写

一般应用在写回调函数的地方

两个参数

无参数

一个参数

弹出一个值

1. **参数的默认值**

**内置顶层函数**

内置：在js内部定义好的 顶层：拥有全局的作用域

Number() 将其他的数据类型转化为数值类型

\*string 可以转换 “123” “” “0x11”

不能转换 “abc” “123abc”

\*布尔值 true 1 false 0

\*undenfined NaN

\*null 0

parsetInt() 将字符串转化为整数类型

\* “123” 123

\* “123.123” 123

\* “123abc” 123

\* “abc123” NaN

\* “0x11” 17

第二个参数：以几进制的的方式解析第一个参数

var ***num***=11;  
console.log(parseInt(***num***,16));//11的16进制

ParseFloat() 浮点类型

除了小数点可以识别 和parseint相同

Boolean() 将任意的数据类型转化为布尔值

除了 0 “” null undefined false NaN返回值是false之外 其他都为真

String() 将任意的数据类型转化为字符串

isNaN() 判断这个值能不能转换为一个数 Not a Number

如果用number()对于一个值转化的结果是NaN 那么isNaN对这个数

的判断结果就是true

**数据类型的转换**

**1、数据类型的强制转换** Number() parseint() parseFloat() string() boolean()

**2、数据类型的隐式转换**

Var num=”10”-”5”; var num2=ture-false; var r=”12”>10;var r=0||1;

Var r= 1?1:2;

ob二进制 0x十六进制 0八进制

Use strict;语法使用严格模式 放在块的开始位置

**数组** 存储一组相关数据的容器

**数组的声明/创建**

1、var arr=[];

2、var arr=new Array();

结果完全相同

**数组的赋值**

1、在声明的同时赋值

\*Var arr=[1,2,3,4,5];

\*Var arr=new Array(1,2,3,4,5);

如果只有一个参数的话这个数字表示的是数组的长度

2、通过下标赋值

Arr[0]=1;arr[1]=2;

**数组的lenght** 数组中下标最大的数下标加一

**数组的下标** 数组中每一个数对应的位置 从0开始

**数组的访问** 如果要访问一个数，可以通过下标去访问数组的某一个数

Console.log(arr[0]);

**数组的遍历/枚举**

For(){} while(){}

**二维数组** 数组里面所有的元素都是数组就是二维数组

**对象：**数组对象 字符串对象 数学对象 正则对象 日期对象 BOM 浏览器对象

dom 文档对象 div p h1 元素对象 事件对象 表单对象 xhr对象

**概念**：

\*在日常生活中所见的一些客观的事物、抽象的规则、计划、事件。

\*每一个对象都是属性的无序集合体

属性 值普通的数据类型

方法 值为函数

**类**：类是具有相同或者相似属性的对象的抽象描述，类的具体化/实例化就是对象

通过构造函数实现对类的定义的 new 实例化构造函数得到对象

**对象的声明/创建：**

1、JavaScript 原生对象描述法 JAON

2、自己定义构造函数 通过实例化构造函数得到对象

3、实例化 object js自动提供的 内置顶层函数

对象属性添加：

注：·属性名 英文单词

·属性值 可以是任意的数据类型

\*创建时直接添加 var obj={name:”zhangsan”};

·属性之间用“，”隔开

·属性和值可以用一个变量来表示

var color=”white” var phone={color,band:”huanwei”}

\*创建之后添加

·对象.属性=值；

·对象[变量]=值；

**对象属性的访问**：1对象.属性

2对象.方法()

3 对象[变量]

**对象的遍历**：for(var i in obj){}

i代表的属性名字

**对象的删除**：delete 对象.属性

Delete 对象[变量]

Delete 对象.方法

Instanceof 判断某个对象是否是由某个构造函数实例化的结果

constructor 属性 对象的当前构造函数

Null 用于清空对象 obj=null; 继续留在内存当中，javascript垃圾回收机制会在

某一时刻从内存中彻底删除这个对象

**继承：**

Array的原型式Array.prototype

Array.prototype的原型是object.prototype

概念：在javascript当中，每一个对象都会自动获得另一个对象的所有的属性和方

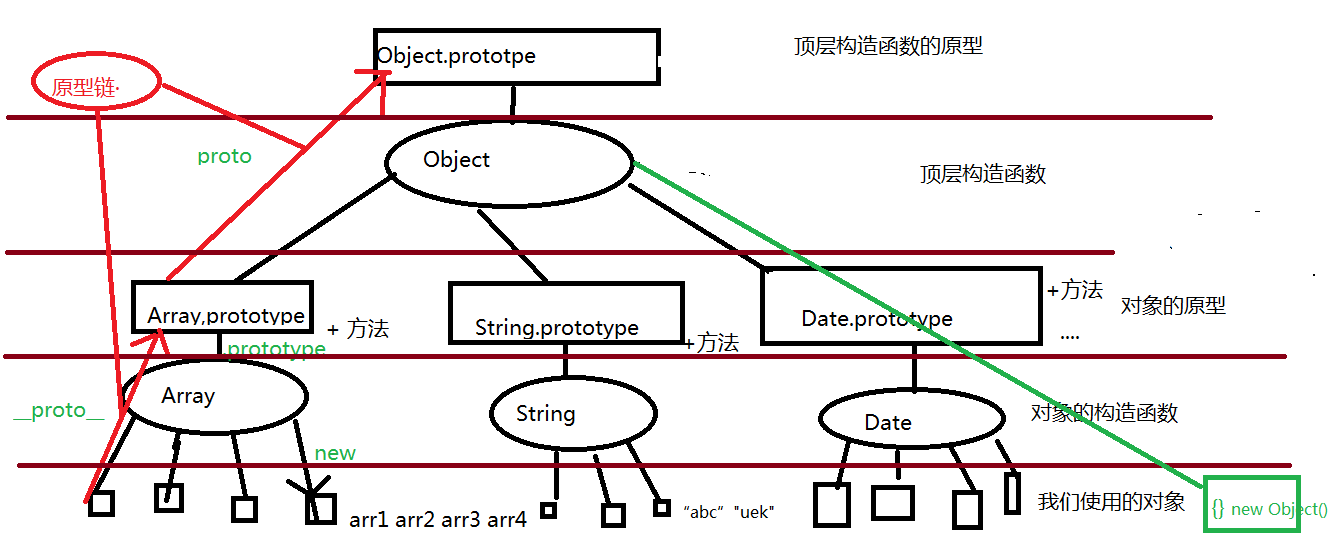
法，这种行为称为继承，被继承对象称为当前对象的原型。

继承的方式有两种：

第一种：通过指定当前对象的构造函数prototype值为原型对象，就能实

现继承

第二种：对象冒充



**This:**

在每一个函数里面都有的

\*当函数为构造函数的时候，this指的就是当前构造函数实例化的对象

\*在一般函数当中 this指的是window对象

\*在作为方法的函数当中，this指的当前在调用这个方法的对象

\*当函数在调用的时候使用 call apply调用的时候 this指的就是传入的

第一个参数

call 需要传参数时直接加“，”继续写。

Apply 把参数放置到一个数组里面来写

**数组对象：**

在ECMAScipt3

\*增加删除的

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 参数 | 返回值 | 功能 |
| Push | 一个或多个元素 | 新数组的长度 | 在数组的末尾添加 |
| Pop | 无 | 被删除的元素 | 在数组的末尾删除 |
| Unshift | 一个或者多个元素 | 新数组的长度 | 在数组开始位置添加 |
| Shift | 无 | 被删除的元素 | 在数组的开始位置删除 |
| Splice | 1、位置2、删除的个数 之后都是要添加的元素 | 被删除的元素组成的数组 | 万能的添加 删除 替换 |
| Join | 连接符 | 转化后的字符串 | 将数组转化为字符串 |
| Slice | 位置一个参数：表示从当前位置到结束 两个参数：就是选中的范围 | 截取到的数组 | 从数组中截取一段组成新数组 |
| Sort | 回调函数 | 排序之后的数组 | 对数组排序 |
| Concat | 数组 | 连接之后的新数组 | 让两个数组连接 |

在ECMAScipt5

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 参数 | 返回值 | 功能 |
| ForEach | callback | undenfined | 遍历数组中的每一个数 |
| Filter | callback | arr | 过滤数组 |
| Map | callback | arr | 映射新数组 |
| Some | callback | boolean | 判断数组当中是否有符合条件的元素 |
| every | callback | boolean | 判断数组当中每一个是否都符合条件 |
| indexOf | Iteam | Number这个数出现的第一个位置 | 查找某一元素的位置 返回当中某个元素第一次出现的位置 |
| LastindexOf | Iteam | Number出现的最后一个位置 | 返回当中某个元素最后一次出现的位置 |
| Reduce | callback | \* | 对元素当中每个元素从左到右进行累加操作 |
| Reduceright | Callback | \* | 对元素当中每个元素从右到左进行累加操作 |
| Reverse | 无 | arr | 将数组中的元素顺序颠倒 |

在ECMAScipt6

|  |  |  |  |
| --- | --- | --- | --- |
| Find | Callback | Item | 在数组中找到某个值并取出 |
| findIndex | Callback | Index | 在数组中找到某个值得位置并返回 |
| CopyWithin | 1、目标位置2、赋值的开始位置3、赋值的结束位置 | Undenfined | 从数组中复制一段放置到另一个位置 |
| Fill | 1、填充的内容2，3填充的起始位置和结束位置 | 填充后的数组 | 快速填充指定长度的空数组 |
| Incloudes | Item | Boolean | 判断数组当中是否有某个元素 |

**对象的分类：**

**本地对象**

内置对象

Math 数学对象

全局对象 window

Array()

String()

Funxtion()

Bollean()

Number()

RegExp()

Date()

**宿主对象** (浏览器)

BOM 浏览器对象模型

History

Loaction

navigater

DOM 文档对象模型

Document对象

其他标签元素

Can i use 网站 兼容性问题

**字符串对象，属性：**

Length 字符串长度

【】 访问到字符串当中对应下标的值；

Construction 访问构造函数

获取

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 参数 | 返回值 | 功能 |
| charAt | Number | 字符串 | 获取指定位置的字符串 |
| charCodeAt | Number | Number | 获取指定位置字符串的ASCII码 |
| string.formCharCode | Number | String | 根据指定的一个或多个ASCII码，得到对应的字符串 |

查找

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 参数 | 返回值 | 功能 |
| indexOf | string | number | 查找某个字符串在当前字符串当中第一次出现的位置 |
| lastIndexOf | string | number | 查找某个字符串在当前字符串当中最后一次出现的位置 |
| replace | Str1,str2 | string | 查找到字符串当中的str1并替换为str2 |
| includes | str | boolean | 判断当前字符串当中是否包含某个元素 |
| startsWith | str | boolean | 判断当前字符串是否是以某个字符串开始的 |
| endsWith | str | boolean | 判断当前字符串是否是以某个字符串结束的 |

截取

slice :和数组一样

substring：与slice区别，不能传负值

substr：截取指定位置、指定长度的字符串

转换

toLowerCase:将大写字母转换为小写字母

toUpperCase:将小写字母转换为大写字母

**split：将字符串转化为数组**

第一个值 分隔符 第二个值 限制最大的长度

trim:去除字符串两端的空格

**Math对象：**

不需要实例化的一个

|  |  |
| --- | --- |
| Math.abs() | 取绝对值 |
| Math.round() | 四舍五入取整 |
| Math.floor() | 向下取整 |
| Math.ceil() | 向上取整 |
| Math.max() | 取最大值 |
| Math.min() | 取最小值 |
| Math.random() | 取随机数 |
| Math.pow() | 求X得Y次幂 |
| Math.sqrt() | 求平方根 |
| Math.sin() | 求正弦值 |
| Math.cos() | 求余弦值 |
| Math.tan() | 求正切值 |
| Math.PI() | Π值 |
| Math.random()\*范围+最小值 | 任意的随机数 |
| 角度\*Math.PI/180 | 角度和弧度的转换 |

**解构赋值 （ES6）**

根据特定的匹配模式，从数组或者对象当中提取值给变量赋值的一种形式

**基本用法：**两边的模式匹配就可以完成赋值

var [a,b,c]=[1,2,3];  
console.log(a,b,c);  
var [a,[b],c]=[1,[2],3];  
console.log(c,b,a);

**空值的处理：**使用的变量可以是空值，会自动赋值为undefined

var [a,b]=[4,5,6];  
console.log(a,b);  
var [,a,b]=[4,5,6];  
console.log(a,b);  
var [a,b,c]=[4,5];  
console.log(a,b,c);  
var [a,b,c]=[,4,5];  
console.log(a,b,c);

**默认值：**变量可以通过=添加默认值，如果在赋值表达式的右侧没有对应值或者对应值为undefined，则会使用默认值

var [q=2,p]=[22,3];  
console.log(q,p);

**结合扩展运算符：**可以完成对数组的赋值，被扩展的变量只能是最后一个变量

var [q,...p]=[9,8,5,4,3];  
console.log(q,p);

**DOM文档对象模型**

**document属性**

url:当前页面地址

body：获取到body标签

head：获取到head标签

title：当前title的内容

all：获取所有标签元素

images：获取所有图片元素集合

forms：获取所有的表单对象集合

links：获取所有的链接集合

**方法：**

**document.之后写**

querySelector() : 通过传递一个选择器获取到某一个元素对象，如果选择器能选中许多个，则选取第一个。如果选择不到，则会得到null；

querySelectorAll() : 通过传递一个选择器获取到一个元素对象集合。如果获取到一个，也是一个集合；如果一个也获取不到，得到一个空集合。

getElementsByClassName()：通过传递类名获取到一个元素集合，获取不到，也是一个空集合

getElementsByTagName()： 通过标签名获取元素集合

getElementById()： 通过ID名来获取元素集合

addEventListener(): 用来给元素添加事件

*animationend 帧动画*

offsetWidth: 一个元素在页面当中所占的实际宽度（内宽，内间距，边框）

offsetHeight: 一个元素在页面当中所占的实际高度

offsetLeft: offsetTop:

如果前辈元素没有定位属性，相对于文档的坐标

如果前辈元素有定位属性，就是相对于前辈元素的坐标

**事件：——用户对页面所做的一些操作或者是浏览器自身的一些行为**

鼠标事件：

onclick：单击

ondblclick：双击

onmousedown: 鼠标按下的时候触发

onmouseup: 鼠标抬起的时候触发

onmouseover: 鼠标移动上去

onmouseout: 鼠标移走

onmousemove: 鼠标移动事件，只要鼠标在元素上移动，就会一直触发

表单事件：

onfocus: 获得焦点

onblur: 失去焦点

页面/元素的滚动条位置发生改变的时候触发的事件：（可以直接给window对象添加）

scroollTop

scroollLeft

**Window对象的方法**

获取元素css属性的方法（在任何地方设置的，包括默认值）：getComputerStyle(div).height

**普通的元素对象：**

innerHTML: 获取或者设置内容 （console.log来获取）识别标签

textContent：获取或者设置文本内容 不识别标签

style: 获取或者设置行内样式集合

value：获取或者设置表单元素的value

id：获取或者设置元素的ID值

className：获取或者设置元素类名

classList：此属性是一个对象，包含类名信息

add(): 给元素添加一个类名

remove(): 从元素对象身上移除一个类名

toggle(): 切换某个元素类名的添加和移除状态

contains(): 判断是否包含某个类名

**获取浏览器宽高**

documentElement.clientWidth

documentElement.clientHeight

window.innerHeight

window.innerWidth

事件流：父元素的事件子元素也会继承

**获取当前可视窗口值**

scroollTop

scroollLeft

document.body.scrollTop获取可视窗口值top值（谷歌浏览器）

document.documentElement.scrollTop获取可视窗口值top值（火狐浏览器）

兼容性处理：

Var obj=document.body.scrollTop==0?document.documentElement : document.body；

**a链接不使用时如何处理：**

<a href="javascript:"></a>

**固定的侧面导航如何紧贴内容：**

position: fixed;  
left:50%; 用百分比来定位，定位到中间  
bottom: 50px;  
margin-left: 600px; 相对于中间移动

**节点**

每一个标签都是一个节点  
 每一个内容是一个节点  
 每一个标签的属性也是节点  
 注释也是节点  
 Document整个文档也是一个节点

**节点属性：**

**实现节点之间的互相访问的属性**：

parentNode：获取父元素

childNodes：获取子节点们（默认包括空格text）

firstChild：访问第一个子节点

lastChild：访问最后一个子节点

nextSibling：访问下一个兄弟节点

nextElementSibling：访问下一个元素兄弟节点

previousSibling：访问上一个兄弟节点

previousElementSibling：访问上一个元素兄弟节点

**每一个节点的信息属性：**

|  |  |  |  |
| --- | --- | --- | --- |
| 节点 | nodeType | nodeName | nodeValue |
| 元素节点 | 1 | 大写的标签名 | null |
| 属性节点 | 2 | 属性名（例：class） | 属性值 |
| 文本节点 | 3 | #text | 文本内容 |
| 注释节点 | 8 | #comment | 注释的内容 |
| 文档节点 | 9 | #document | null |

属性节点的获取：例：ul.attributes[0].nodeType

Console.log(ul.attributes);

namedNodeMap{0:class,1:style,length:2}

**节点方法：**

**创建一个元素**：

document.createElement()

**插入到页面**：

document.body.appendChild()

var d=document.createElement("div");  
// d.style.width="100px";  
// d.style.height="100px";  
// d.style.background="red";

以上三步单独添加样式  
 d.style.cssText="width:100px;height:200px;background:red"; //添加行内样式

或者现在css中设置一个类名和属性，JS指定类名 d.className=”item”;

document.body.appendChild(d);

ele.insertBefore(要插入的新元素，被插入的元素) //ele 表示父元素

**删除元素**：

ele.removeChild(要被删除的元素)

**替换一个元素**：

ele.replaceChild（新元素，被替换的 ojnononjo元素）

**克隆节点**：

cloneobj=obj.cloneNode（）

传入参数true 表示也要克隆子节点

不传参数，没有节点

Web安全色 ：（十六进制）0 3 6 9 c f

**ES6**

**Symbol**

//Symbol() 独一无二的值  
var s=Symbol();

**对象的解构赋值**

//通过对象进行解构赋值  
var obj2={myname:"zhangsan",ninling:20};  
//第一种写法  
var {myname,ninling}={myname:"zhangsan",ninling:20};  
console.log(myname,ninling); //zhangsan 20  
//第二种写法  
var {myname:n,ninling:a}=obj2;  
console.log(n,a);

**set**

//set和map  
//两个特殊的对象  
 //set 类似于数组 只不过当中保存的都是不重复的值  
 //new set() 实例化得到  
var se=new Set();  
console.log(se);  
//add() 添加某个内容 返回值是对象本身所以可以链式调用  
se.add("hua").add(5).add("jin");  
console.log(se);  
console.log(se.size);  
//size 表示个数  
//小案例  
var arr=[1,2,3,4,5,3,3,2,2,4,5,3,6,7,5,5,88];  
var newset=new Set();  
arr.forEach(function (v) {  
 newset.add(v)  
});  
console.log(newset);  
//delete（） 删除一个内容 返回值是是否删除成功 true/false  
newset. delete(1);  
console.log(newset);  
//has() 返回是否包含某一个内容  
console.log(newset.has(88));  
//支持forEach()遍历  
newset.forEach(function (v) {  
 console.log(v);  
});  
//entries() keys() values()  
for(var i of newset.entries()){  
  
}  
//更高级的遍历 for of

**Map()**

//map 类似于对象 只不过属性名可以是任意类型的值  
console.clear();  
var m=new Map();  
//set() 添加属性  
m.set(1,2); //属性名，属性值  
var arr=[78,67,54];  
m.set(true,1).set("hua",2).set(null,3).set(arr,5);  
console.log(m);  
//get() 通过属性名访问属性值  
console.log(m.get(true));  
console.log(arr);  
//clear() 清空所有内容  
//delete() 进行删除  
//forEach() 进行遍历  
//has() 进行判断

### BOM浏览器对象模型

**Window对象**

**属性：**

window.innerWidth

window.innweHeight

**获取浏览器分辨率**

window.screen.width

window.screen.height

**获取浏览器距屏幕左边的值**

window.screenLeft (IE浏览器/谷歌浏览器)

window.screenX (火狐浏览器/谷歌浏览器)

**获取浏览器距屏幕上边的值**

window.screenTop

window.screenY

window.top 当前页面的页面的顶层窗口的window对象 iframe

谁被引用，在谁上面设置

**方法：**

alert() 弹出

prompt() 输入

confirm() 带有提示信息的确认窗口，可以点击确认，或者取消

setInterval()

clearInterval()

setTimeout()

clearTimeout()

getComputedStyle() 获取CSS属性

**历史记录对象(history)**

**属性：**

Length 表示当前历史记录的条数

**方法：**

foreward() 跳转到历史记录的前一条

back() 跳转到历史记录的上一条

go() 1,下一条； -1 上一条； 0，当前页（刷新页面）

**地址栏对象location() ——保存地址各种细节**

http:// 协议（应用层）

localhost 当前域名 127.0.0.1 每一个ip地址对应一个域名

:63342 端口号

/WUIF1612/4.22/2.html 路径

?zh=123456 查询字符串

#top 锚链接

**访问：**

location.protocol 访问协议

location.hostname 访问域名

location.host 域名+端口号

loaction.origin 协议+域名+端口号

location.prot 端口号

location.pathname 路径

location.search 查询部分

location.hash 锚链接地址

location.herf 访问整个地址

**方法：**

location.reload() 刷新网页

location.assign(“http://www.baidu.com”) 跳转到指定地址

location.replace(“http://www.baidu.com”) 跳转到指定地址，不会留下历史记录

**事件对象：**

在事件发生的时候，产生一系列关于事件触发详细情况的一个对象

e.screenX e.screenY 鼠标点击位置，距离屏幕左上角的坐标

e.clientX e.clientY 距离浏览器窗口左上角的坐标

e.pageX e.pageY 距离文档左上角的坐标

e.offsetX e.offsetY 距离事件源左上角的坐标

**键盘事件：**

e.keyCode 键盘对应的键盘码

e.ctrlKey e.shiftKey e.altKey 判断某个键是否被按下的状态

事件的添加

1.on+事件名

2.添加事件监听

//addEventListener(事件名称，事件处理程序)，现代浏览器

attachEvent(on事件名称,事件处理程序)ie8以下,

3.事件的移除

removeEventListener(事件名称,事件处理程序)

detachEvent();

4.一般对于兼容性的处理方案

//写一个函数

//判断浏览器

//执行各自对应的操作

**事件流：**

当某个元素身上的事件触发的时候，整个页面以及元素所有的前辈元素都会按照特定的方式响应这个事件，这种事件传播的流程就称之为事件流。

**冒泡型事件流：**

\* 默认存在在任何一个事件之中。

\* 触发顺序：从最明确的事件源（正在点击的）到最不明确的事件源（从小到大）依次触发。

**捕获型事件流：**

\* addEventListenter(event,fn,true)才会触发捕获型事件流

\* 触发顺序：从最不明确的事件源到最明确的事件源（从大到小），依次触发

\* 捕获事件流触发完成之后，依然还会触发冒泡

**事件流的利用与阻止：**

**利用：**例: 拖拽，轮播图，事件委派

**阻止事件流的传播：**

\* e.stopPropagation() 现代浏览器

\* e.cancelBubble=true; IE浏览器

**\*（加一个判断，利用hover.js 函数。解决事件流问题）**

**事件委派案例（e.target）：要实现子元素效果时，利用事件流给其父辈设置事件，对直接事件源进行判断**

//事件委派案例  
 var box=document.querySelector(".box"); //大盒子  
 var item=document.querySelector(".item"); //子元素  
 var add=document.querySelector(".add"); //添加按钮  
 add.onclick=function () {  
 var div=document.createElement("div"); //创建div  
 div.className="item"; //为div指定类名  
 box.appendChild(div); //将div加入大盒子  
 };  
 box.onclick=function (e) { //大盒子添加事件  
// console.log(e.target); //点击谁，e.target指谁  
 if(e.target.className=="item"){ //直接事件源  
 e.target.style.background="#000";  
 }  
 }

Border-collapse：collapse； （边框合并）

**移动端事件**

新增事件：

\*onclick

\*ontouchstart 表示触摸开始，类似onmousedown

\*ontouchmove 表示触摸移动的效果，类似于onmousemove

\*ontouchend 表示事件结束，类似于onmouseup

触摸点胡集合：

\*e.touches 所有触摸点集合

\*e.targetTouches 当前事件源身上所有触摸点的集合

\*e.changedTouches 当前事件源所有正在改变的触摸点的集合

**日期对象：**

本地对象，（在ECMAScript中内置的）

所有关于当前日期的信息

得到日期对象

New Date() 当前时刻的日期对象

New Date() （年，月，日，时，分，秒）得到一个具体的

console.log(date.getFullYear()); //2017 获取正确年份  
console.log(date.getMonth()); //3 0~11 获取月份  
console.log(date.getDate()); //26 获取日期  
console.log(date.getHours()); //17 获取24小时  
console.log(date.getMinutes()); //13分钟 获取分钟  
console.log(date.getSeconds()); //47 获取秒  
console.log(date.getMilliseconds());//138 0~999 获取毫秒  
console.log(date.getDay()); //3 0~6 获取星期  
console.log(date.getTime()); //1970年1月1日到现在所经过的毫秒数（）  
console.log(date.getTimezoneOffset());  
console.log(date.getUTCHours()); //国际相同时间  
  
date.setFullYear(2019);  
console.log(date)

**正则对象：**

**什么是正则表达式：**

一个用来描述或者匹配一系列符合某个语法规则的字符串的语言。

**应用场合：**

数据验证，文本替换，内容检索，过滤内容

可以理解为：执行字符串函数无法完成的特殊的匹配、拆分、替换功能

**语法：**

**创建正则表达式：**

\*实例化构造函数 var reg=new RegExp();

\*字面量 var reg2=/uek/;

**定界符：**

\* / 正则开始和结束

\* ^ 匹配字符串的开始位置

\* $ 匹配字符串的结束位置

**原子：——正则当中最小的一个单位**

\* \s 空格

\* \S 除了空格之外的任意字符

\* \d 任意一个0到9的数字 [0-9]

\* \D 除了数字之外的任意字符 [^0-9]

\* \w 字母，数字，下划线 [a-zA-Z0-9\_]

\* \W 除了字母，数字，下划线以外的任意字符 [^a-zA-Z0-9\_]

**原子表：**

\* [ ] 匹配任意一个

var reg=/^[a-z0-9]{6,10}$/; //6-10个小写字母和数字组成的字符串

\* [^ ] 排除

**原子分组：**

\* （） 把一部分原子通过括号包裹成一个整体，在后面调用；

**反向引用：**

\* \1 \2 \3 表示当前位置的内容必须和第n个括号的内容一样 反向引用

\* （？:） 取消反向引用

//判断开始和结尾字母相同  
var strr="a12b";  
var reg1=/^([a-zA-Z]).\*\1$/;  
console.log(reg1.test(strr));  
var strr2="1ab234ab";  
var reg2=/^(?:.)([a-zA-Z]{2}).\*\1$/;  
console.log(reg2.test(strr2));

**量词：——定义个数（必须跟在原子/原子表/原子组/字母之后）**

\* \* 匹配0个或多个

\* {1,} 表示1个以上

\* {6,11} 6到11

\* {6} 6个

\* + 匹配1个或多个

\* ？ 匹配0个或1个

**正则的贪婪与吝啬：(跟在量词之后——替换或解析)**

\* 使用量词去匹配的时候，是尽量多的去匹配的（默认）

\* 尽量少的匹配，在量词的后面加一个？

**元字符：——表示特殊功能的符号**

\* \ 转义

\* | 或者

\* . 代表任意字符

\* - （连字符）表示范围

**模式修正符：——定义匹配模式（写在结束/之后）**

\* g 全局的匹配

\* i 不区分大小写

**方法：**

\* test() 判断某一个字符串是否能被当前正则匹配

\* exec() 解析字符串当中被正则匹配到的部分，返回结果到一个数组当中。如果是全局模式的正则，下一次解析会接着上一次解析的位置继续找，如果找不到，返回null；

获取所有标签  
var reg=/^([a-z]{1,10}|h[1-6])$/;  
获取英文名字 Kobe Bryent  
var reg=/^[A-Z][a-z]{1,10}\s[A-Z][a-z]{1,10}$/;  
判断一个类名选择器  
var reg=/^\.[a-zA-Z\_][0-9a-zA-Z\_\-]{1,}$/;  
判断一个身份证号码  
var reg=/^([1-9][0-9]{17}|[1-9][0-9]{16}[X])$/;  
var reg=/^(\d{18}|\d{17}[xX])$/;  
判断一个十六进制颜色  
var reg=/^#([0-9a-fA-F]{3}|[0-9a-fA-F]{6})$/;

**字符串方法中的正则：**

\* replace() 替换

\* match()

\* search() 类似于indexOf 参数是正则

str.search("d") 或者 str.search(/\d/); 两种传参

\* split()

str.split(/[\s\.,]+/); //分割

var str="yang jin rui,haha zui.la ";  
console.log(str.split(/[\s\.,]+/)); //分割

**表单对象：**

获取：

**表单的获取：**

var form=document.querySelector("form");  
var form2=document.getElementsByName("myform")[0]; //getElements代表多个，所以要加下标  
var form3=document.forms[0];  
var form4=document.forms.myform;  
var form5=document.myform;

**表单控件的获取：**

var input1=document.getElementsByName("text")[0];  
var input2=form.elements[0];  
var input3=form.elements.text;  
var input4=form.text;

事件：

表单控件事件：

\* onblur() 失去焦点

\* onfocus() 获得焦点

\* onchange 内容改变并失去焦点

\*

**方法：**

\* focus() 自动获取焦点

\* blur() 自动失去焦点

表单事件：

\* onsubmit() 提交之后触发

\* onreset() 重置之后触发

\* submit() 提交表单

\* reset() 重置表单

属性：

\* form action name method

\* type name value placheholder autofocus(自动获取焦点) pattern(正则判断) maxLength step min max checked

\* selected size muliple rows cols readOnly disabled required;

\* option标签: text（文本内容）

\* select标签： selectedIndex当前选中的option的索引值

**客户端存储**

客户端存储主要方便一些App离线使用，目前常见的客户端存储方式有cookie/localStorage/sessionStorage.

Cookie:

\* 存储在用户浏览器当中的一个字符串，能够用来在不同的页面之间传递数据

\* cookie的保存是按照域名来划分的，不同的域名下，不能访问cookie

\* cookie保存的数据量小于4kb或者20个键值对

\* 不同的浏览器保存的cookie也是各自独立的；

\* 操作

\* 获取 document.cookie

\* 设置 document.cookie=”name=hua”;(键值对)——“属性名=属性值”

\* 设置过期时间

\* 默认的生存周期 session 打开网页 到 关闭网页这段时间之内

\* expires过期时间 国际标准时间来设置 expires=过期时间 date.toGMTString();

\* 获取，删除

（h5新增的）window对象，不用实例化

localStorage:

\* 和cookie的区别：

\* 保存的数据是永久的，除非用户手动操作删除。

\* localStorage，保存的数据量5MB-10MB

\* 相同点

\* 在同一个域名下共同使用一个localStorage，不可跨浏览器

sessionStorage：

\* 和localStorage的区别：

\* 保存的数据只能存在在一次会话当中

以上两者操作一样：（对象）

\* 设置：

\* localStorage.key=value;

\* localStorage.setItem(key,value)

\* 获取：

\* localStorage.key

\* localStorage.getItem(key)

\* 删除：

\* localStorage.removeItem(key) //删除某一个，需传值

\* localStorage.clear() //清空

Es5新增：

JSON.parse(): //把字符串转化为json

JSON.stringify(): //把json转化为字符串

**视频播放器**

Video <!--autoplay自动播放 controls播放控件-->  
<video src="video/1.mp4" autoplay controls>  
 <source src="video/1.mp4"> <!--多个格式兼容各个浏览器-->  
 <source sre="video/1.ogg">  
</video>

API:

\* 方法

\* load（） //重载

\* play（） //播放

\* pause（） //暂停

\* 事件

\* oncanplay //当前视频准备好播放触发事件

\* ontimeupdate //进度条（播放位置）改变，触发事件

\* 属性

\* video.duration //总共的时间

\* video.currentTime //变化的时间

**画布canvas：**行内块元素

宽高设置为行内样式

JS设置中（操作）：

\* 获取绘制对象

\* canvas.getContext(“2d”);

\* 绘制形状 默认形状：矩形

\* fillRect() 绘制一个填充矩形

\* strokeRect() 绘制一个描边矩形

\* clearRect() 清空一个矩形区域

\* (x,y),width,height

\* canvas的坐标系

\* 默认左上角0 0；向右X轴增加 ，向下 Y 轴增加。

\* 样式设置

\* fillstyle=””; //画笔颜色

\* strokestyle=””; //描边颜色 （内外都描）

颜色设置：颜色名/16进制/透明度

\* 路径

\* beginPath() 开启新的路径（清空旧的路径）

\* moveTo(X,Y) 从哪个位置开始（开始坐标）

\* lineTo(X,Y) 画到哪个位置（线的结束坐标）

\* closePath() 闭合路径

\* fill() 把路径闭合的区域填充满颜色

\* stroke() 对路径描边

\* arc（x,y,r,s,e,boolean） 绘制圆弧(角度顺时针走)

(坐标，半径，起始角度，结束角度，顺时针绘制还是逆时针绘制，默认顺时针（false）)

\* quadraticCurveTo(cx,cy,x,y) 绘制弧线

(cx cy 控制点坐标（圆弧的左上角）； x y 弧线结束位置坐标)

\* rect(x,y,width,height); //快速绘制矩形路径

\* 线条的属性设置

huabi.beginPath();  
huabi.lineWidth=8; //线的长度  
huabi.lineJoin="miter"; //线的连接拐点样式  
//round圆角 bevel平角 miter尖角（默认）  
huabi.miterLimit=2; //限定拐点交接延伸长度 默认10；超出设置的值，会自动裁剪为平角  
huabi.lineCap="butt"; //线条两端点样式  
//round圆帽 square方块 butt平的（默认）

\* 虚线的绘制

\* getLineDash() //得到一个数组

\* setLineDash([4,2]) //虚线设置,传递一个数组,4像素线，2像素空格

\* lineDashOffset() //虚线初始的偏移量

\* 保存和载入

\* save（） //保存画布所有设置的状态lineWidth strokStyle setLineDash

\* restore() //恢复到对应的save的位置的状态

\* 全局透明度

\* globalAlpha=0~1

\* 渐变

\* 线性渐变

//线性渐变  
var linear=cobj.createLinearGradient(200,100,300,200); //颜色渐变(参数为两点坐标)  
linear.addColorStop(0,"red");  
linear.addColorStop(1,"yellow");  
cobj.fillStyle=linear;  
cobj.fillRect(200,100,100,100);

\* 径向渐变

径向渐变 RadialGradient( 小圆圆心横坐标, 小圆圆心纵坐标, 小圆半径, 大圆圆心横坐标, 大圆圆心纵坐标, 大圆半径 )  
 var radial=cobj.createRadialGradient(250,250,0,250,250,100);  
 radial.addColorStop(0,"pink");  
 radial.addColorStop(0.5,"blue");  
 radial.addColorStop(1,"yellow");  
 cobj.fillStyle=radial;  
 cobj.arc(250,250,100,0,Math.PI\*2);  
 cobj.fill();

\* 文字

\* fillText(text,x,y)

\* strokeText(text,x,y)

\* font=“30px 微软雅黑” //文字大小、字体

\* textAlign center //水平居中

\* textBaseline middle //垂直居中

\* 坐标系移动

\* translate(x,y) //坐标系位移（坐标）

\* rotate(rad) //旋转（弧度）

\* scale(x,y) //缩放（坐标）

// requestAnimationFrame(); //动画

// cancelAnimationFrame(); //结束动画

**方法：**

\* getContext() //获取canvas的执行对象（画笔） var cobj=getContext(“2d”);

**函数：**

\* beginPath() //开始绘制路径

\* closePath() //闭合路径

\* stroke() //描边路径

\* fill() //填充闭合路径

\* moveTo(x,y) //设置起点，——参数为：（起始点x坐标，y坐标）

\* lineTo(x,y) //设置终点，即画到哪里。——参数：（结束点x坐标，y坐标）

\* arc(x, y, radius, startAngle, endAngle, anticlockwise)

//绘制圆弧——参数（圆心坐标x，y，半径，开始弧度，结束弧度，顺/逆【默认顺时针false】）

【arc()函数中的角度单位是弧度，不是度数。角度与弧度的js表达式:radians=(Math.PI/180)\*degrees。

\* quadraticCurveTo(cp1x, cp1y, x, y) //cp1x,cp1y为控制点，x,y为结束点。

\* bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

//绘制二次贝塞尔曲线，cp1x,cp1y为控制点一，cp2x,cp2y为控制点二，x,y为结束点。

\* rect(x, y, width, height) //直接快速绘制矩形

**属性：**

\* fillStyle=color //设置填充颜色

\* strokeStyle=color //设置描边颜色

\* 画布数据

\* getImageData(gx,gy,w,h) //获取画布指定区域当中所有的图像数据（要取的坐标起点，及区域宽高）

\* putImageData(数据名称，gx,gy,x,y,w,h)

//将某个图像数据放置到画布（名称，要放置的坐标，放置东西的位置及宽高）

**Git:** 分布式版本控制系统

一个人开发项目

\* 把代码回退到之前的某个版本

\* 实现本地代码和远程服务器代码之间的同步

团队开发

\* 方便团队合作开发代码

\* 提高安全性

\* 及时查看进度

**Github：**

\* 一个网站

一个基于git的代码托管平台，程序员的facebook ，github网站对应的一个应用程序

\* github 有大量的开源项目jquery bootstrap less项目

\* 在github中上上传自己的项目 访问

\* 在github上进行协同开发