## 一 js介绍

### 1.1 起源

94年网景公司开发世界第一框浏览器

95年sun公司开发java语言

因为java太火，所以起名为javascript（script是脚本的意思）

类似：北大青鸟和北大的关系

### 1.2 三大组成部分

|  |  |
| --- | --- |
| ECMAscript | js标准 |
| dom | 通过js操作网页元素 |
| bom | 通过js操作浏览器 |

### 1.3 引入方式

内嵌式：

|  |
| --- |
| <script type="text/javascript">  // js代码  </script> |

外嵌式：

|  |
| --- |
| <script type="text/javascript" src="myjs.js"></script> |

### 1.4 注释和书写规范

|  |
| --- |
| //单行注释 /\*   多行注释  \*/ |

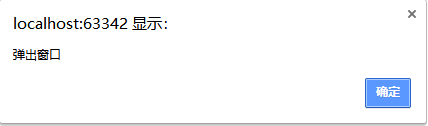
每行代码的句末，要加分号“;”

## 二 js基础

### 2.1输出

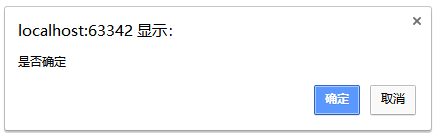
#### 2.1.1 消息框 alert

alert('弹出窗口');



#### 2.1.2 确认框 confirm

ret = confirm('是否确定');

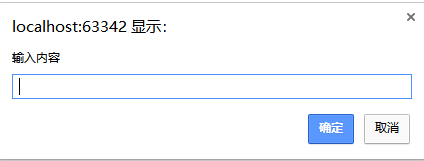


确定：ret = true

取消：ret = false

#### 2.1.3 输入框 prompt

ret = prompt('输入内容')

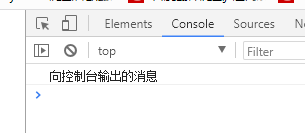


#### 2.1.4 向页面输出消息 document.write

*document*.write('页面显示的信息')

#### 2.1.5 向控制台输出消息 console.log

*console*.log('向控制台输出的消息')



控制台：按f12，点击console

## 三 基本数据类型

### 3.1 数字

#### 3.1.1不同进制表示

十进制表示法：

var *number1* = 1;

十六进制表示法：（0x开头，0-9 a-f）

var *number2* = 0x12;

八进制表示吧：（0开头，0-7）

var *number3* = 012;

#### 3.1.2 不穷大infinity

var *a* = 1 / 0; 🡪 不会报错，且a = infinity（表示无穷大）

### 3.2 字符串

var *string\_1* = '字符串';

### 3.3 布尔

var *boolean\_1* = true;  
var *boolean\_2* = false;

### 3.4 空 null

var empty = null;

### 3.5 未定义 undefined

var undefinedObject = undefined;

undefined和null的区别：

var emptyObject = null; 内存中没有emptyObject这个变量

var undefinedObject = undefined; 虽然这个变量未被定义，但是内存中存在

### 3.6 保留字 NaN

指明：这个对象不是数字

### 3.7 查看对象类型

typeof()

### 3.8 集体变量声明

var numberObject = 1,arrayObject = [],dictObject = {};

多组变量用逗号‘,’ 隔开

## 四 基本数据类型转换

### 4.1 数字转字符串

#### 4.1.1 String

|  |
| --- |
| var numberObject = 123; var stringObject = String(numberObject); 🡪 '123' |

#### 4.1.2 toString

|  |
| --- |
| var *numberObject* = 123; var *stringObject* = *numberObject*.toString(); 🡪 '123' |

### 4.2 字符串转数字

#### 4.2.1 Number

|  |
| --- |
| var stringObject1 = '123.456'; var stringObject2 = '1a2b3c.456';  var numberObject1 = Number(stringObject1); 🡪 123.456 var numberObject2 = Number(stringObject2); 🡪 NaN |

#### 4.2.2 parseInt

|  |
| --- |
| var stringObject1 = '123.456'; var stringObject2 = '1a2b3c.456';  var stringObject3 = 'a12b3c.456';  var numberObject1 = parseInt(stringObject1); 🡪 123 var numberObject2 = parseInt(stringObject2); 🡪 1  var numberObject3 = parseInt(stringObject3); 🡪 NaN |

#### 4.2.3 parseFloat

|  |
| --- |
| var stringObject1 = '123.456'; var stringObject2 = '1a2b3c.456';  var numberObject1 = parseFloat(stringObject1); 🡪 123.456 var numberObject2 = parseFloat(stringObject2); 🡪 1 |

### 4.3 转布尔 Boolean

null、undefined、0 、空字符串 🡪 false

空数组、空字典、空对象 🡪 true

|  |
| --- |
| var nullObject = null; var undefinedObject = undefined; var numberObject = 0; var stringObject = ''; var arrayObject = []; var dictObject = {}; var emptyObject = Object();  alert(nullObject); alert(undefinedObject); alert(numberObject); alert(stringObject); alert(arrayObject); alert(dictObject); alert(emptyObject); |

## 五 运算符

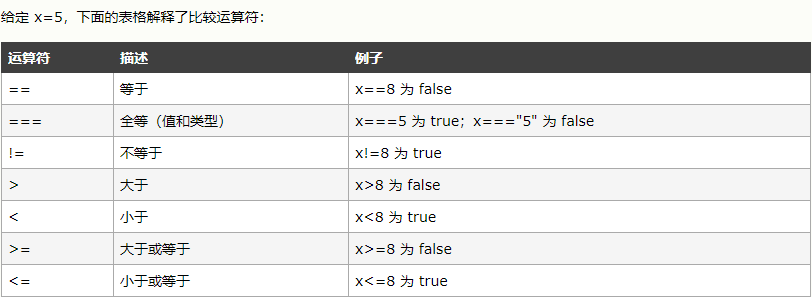
### 5.1 算数运算符



### 5.2 赋值运算符



### 5.3 比较运算符



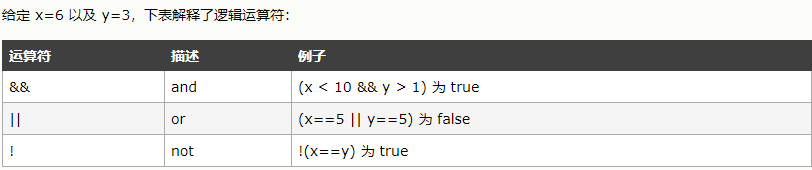
注意：

var x = 5;

var y = '5';

alert(x == y); 🡪true

### 5.4 逻辑运算符



js中的与和或是短路与和短路或

## 六 判断

### 6.1 if

#### 6.1.1 if else

|  |
| --- |
| if (条件){  //执行代码 }else{  //执行代码 } |

#### 6.1.2 js中的“elseif”

|  |
| --- |
| if (条件){  //执行代码 }else if (条件){  //执行代码 } |

### 6.2 三元表达式

条件?【条件成立执行的代码】:【条件不成立执行的代码】;

例如：

|  |
| --- |
| 1 == '1'?alert('条件成立'):alert('条件不成立'); |

### 6.3 switch

|  |
| --- |
| var *a* = '2'; switch (*a*){  case '1':  alert('a = 1');  break;  case '2':  alert('a = 2');  break;  default:  alert('default');  } |

## 七 循环

### 7.1 for

|  |
| --- |
| for (var *i* = 0; *i* < 5; *i* ++){  alert(*i*); } |

相当于：

for i in range(0,5):

print(i)

### 7.2 while

|  |
| --- |
| var *a* = 1;  while (*a* < 5){ alert(*a*); *a* ++;  } |

注：

break 退出当前循环

continuew 进入下一次循环

### 7.3 do while

|  |
| --- |
| var *a* = 1; do {  alert(*a*);  *a* ++; }while(*a* < 1); |

## 八 异常

### 8.1 try catch finally

|  |
| --- |
| try{  vara *a* = 1; }catch (*error*) {  alert(*error*); }finally{  alert('finally'); } |

### 8.2 手动抛异常

|  |
| --- |
| try{  throw '异常原因'; }catch (*error*) {  alert(*error*); }finally{  alert('finally'); } |

## 九 函数

### 9.1普通函数

#### 9.1.1第一种创建方法

|  |
| --- |
| var func = function(*param1*,*param2* = 2){  //do something }; |

#### 9.1.2第二种创建方法

|  |
| --- |
| function func(*param1*,*param2* = 2){  //do something }; |

### 9.2 匿名函数

|  |
| --- |
| *window*.onload = function(){  //do something } |

## 十 事件

### 10.1 事件种类

#### 10.1.1 焦点

|  |  |
| --- | --- |
| 命令 | 说明 |
| onfocus | 元素获得焦点 |
| onblur | 元素失去焦点 |

#### 10.1.2 鼠标

|  |  |
| --- | --- |
| 命令 | 说明 |
| onclick | 鼠标单击对象 |
| ondbclick | 鼠标双击对象 |
| onmousedown | 鼠标按下 |
| onmouseup | 某个鼠标按键被松开 |
| —————————————————— | —————————————————— |
| onmouseover | 鼠标移动某个元素上 |
| onmouseout | 鼠标从某个元素移开 |
| onmousemove | 鼠标移动 |

#### 10.1.3 键盘

|  |  |
| --- | --- |
| 命令 | 说明 |
| onkeydown | 某个键被按下 |
| onkeypress | 某个键被按下且按住 |
| onkeyup | 某个键被弹起 |

#### 10.1.4 加载

|  |  |
| --- | --- |
| 命令 | 说明 |
| onload | 某个页面或图像被完成加载 |
| onabort | 图像加载被中断 |
| onerror | 当加载文档或图像时发生某个错误 |

#### 10.1.5 其他

|  |  |
| --- | --- |
| 命令 | 说明 |
| onchange | 用户改变域的内容 |
| onsubmit | 提交按钮被点击 |
| onreset | 重置按钮被点击 |
| onresize | 窗口或框架被调整尺寸 |

### 10.2 事件的调用

#### 10.2.1 html中调用

|  |
| --- |
| <script type="text/javascript">  var func = function(){  alert('事件被触发了');  } </script>  ———————————————————————————————————————  <body>  <input type = 'text' onfocus = 'func()'> </body> |

#### 10.2.2 js中调用

|  |
| --- |
| <script type="text/javascript">  *window*.onload = function(){  var inputElement = *document*.getElementById('input\_id');  inputElement.onblur = function(){  alert('事件触发了');  }  } </script>  ———————————————————————————————————————  <body>  <input type="text" id="input\_id"> </body> |

注：html是超文本，按顺序执行，所以要得到<input>的对象，必须等<input>加载完，js才能得到它，window.onload等到html加载完成后，才触发的函数。

### 10.3 与表单提交的处理

#### 10.3.1 通过js提交表单

|  |
| --- |
| <script type="text/javascript">  *window*.onload = function(){  var formElement = *document*.getElementById('form\_id');  var buttonElement = *document*.getElementById('button\_id');  buttonElement.onclick = function(){  formElement.submit();  }  }; </script>  ———————————————————————————————————————  <body>  <form action = '//www.baidu.com' id = 'form\_id'>  <input type = 'text' name = 'username'>  <input type = 'button' value = 'js提交按钮' id = 'button\_id'>  </form> </body> |

#### 10.3.2 修饰submit

|  |
| --- |
| <script type = "text/javascript">  *window*.onload = function(){  var formElement = *document*.getElementById('form\_id');  formElement.onsubmit = function(){  alert('submit事件发生了');  return false; // return false阻止submit提交  }  }; </script>  ———————————————————————————————————————  <body>  <form action = '//www.baidu.com' id = 'form\_id'>  <input type = 'text' name = 'username'>  <input type = 'submit' value = 'submit提交'>  </form> </body> |

注：阻止事件的发生，用return false

## 十一 Number

### 11.1 转字符串 toString()和toLocaleString()

|  |
| --- |
| var *a* = 3; b = *a*.toString(2); *console*.log(b); --> '11'  c = *a*.toString(3); *console*.log(c); --> '3' |

### 11.2 设置有效位数toFixed()

|  |
| --- |
| var *a* = 300.173; b = *a*.toFixed(1); *console*.log(b); --> '300.2' |

### 11.3 指数计数法 toExponential()

|  |
| --- |
| var *a* = 300; b = *a*.toExponential(); *console*.log(b); --> 3e+2  b = *a*.toExponential(1); *console*.log(b); --> 3.0e+2 |

### 11.4 格式化指定长度 toPrecision()

|  |  |
| --- | --- |
| var *a* = 123456789; b = *a*.toPrecision(6); *console*.log(b); --> 1.23457e+8 |  |

### 11.5 命令总结

|  |  |
| --- | --- |
| 方法 | 说明 |
| toString(radix) | 数字转字符串（默认转成10进制的） |
| toLocaleString() | 数字转字符串（本地格式） |
| toFixed(fractionDigits) | 数字转字符串（指定有效小数位） |
| toExponential(fractionDigits) | 转为指数计数法（指定有效小数位） |
| toPrecision() | 数字格式化指定长度 |

## 十二 String

### 12.1 大小写

|  |  |
| --- | --- |
| 命令 | 说明 |
| big() | 转大写（通过加入<big>标签） |
| small() | 转小写（通过加入<small>标签） |
| toLowerCase() | 转小写 |
| toUpperCase() | 转大写 |
| toLocaleLowerCase() | 转小写（本地格式） |
| toLocaleUpperCase() | 转大写（本地格式） |

例如：

|  |
| --- |
| var *a* = 'abcABC'; *console*.log(*a*.big()); --> '<big>abcABC</big>' *console*.log(*a*.small()); --> '<small>abcABC</small>' *console*.log(*a*.toLowerCase()); --> 'abcabc' *console*.log(*a*.toUpperCase()); --> 'ABCABC' *console*.log(*a*.toLocaleLowerCase()); --> 'abcabc' *console*.log(*a*.toLocaleUpperCase()); --> 'ABCABC' |

### 12.2 索引与切片

|  |  |
| --- | --- |
| 命令 | 功能 |
| split(str) | 分割字符串为数组 |
| slice(start,end) | 切片（数组和字符串都可以操作） |
| substring(start,end) | 切片（只能操作字符串） |
| substr(start,length) | 切片（起始索引和指定长度） |
| charAt(index) | 返回指定索引的字符 |
| charCodeAt(index)（了解） | 返回指定索引字符的Unicode编码 |

#### 12.2.1 分割 split()

简单例子：

|  |
| --- |
| var *a* = '1,2'; ret = *a*.split(','); *console*.log(ret); --> ['1','2'] |

分割get请求的url，得到参数：

|  |
| --- |
| var *url* = 'http://www.baidu.com/?username=name1&password=passwd1'; params = *url*.split('?')[1]; key\_value\_s = params.split('&'); for (var *i* = 0 ; *i* < key\_value\_s.length ; *i* ++){  *console*.log(key\_value\_s[*i*].split('=')[1]); } |

#### 12.2.2 切片 slice()、substring()、substr()

|  |
| --- |
| var *a* = '01234'; a\_slice = *a*.slice(0,2); *console*.log(a\_slice); --> '01' a\_substring = *a*.substring(0,2); *console*.log(a\_substring); --> '01' a\_substr = *a*.substr(2,2); *console*.log(a\_substr); --> '23' |

#### 12.2.3 索引字符 charAt()、charCodeAt()

|  |
| --- |
| var *a* = '01234'; a\_charAt = *a*.charAt(0); *console*.log(a\_charAt); --> '0' a\_charCodeAt = *a*.charCodeAt(0); *console*.log(a\_charCodeAt); --> 48 （注意是Number类型） |

### 12.3 检索

|  |  |
| --- | --- |
| 命令 | 功能 |
| indexOf(str) | 从前向后查找str对应的索引 |
| lastIndexOf(str) | 从后向前查找str对应的索引 |

例如：

|  |
| --- |
| var *a* = '110'; a\_indexOf = *a*.indexOf('1'); *console*.log(a\_indexOf); --> 0 a\_lastIndexOf = *a*.lastIndexOf('1'); *console*.log(a\_lastIndexOf); --> 1 |

### 12.4 正则

|  |  |
| --- | --- |
| 命令 | 说明 |
| match | 返回匹配成功的字符串（返回第一个） |
| search | 返回搜索成功的字符串（返回第一个索引） |
| replace | 替换（替换第一个） |

### 12.5 连接字符串

|  |  |
| --- | --- |
| 命令 | 说明 |
| concat(strs) | 连接多个字符串（也可以操作数组） |
| "" + "" | 字符串直接相加 |

例如：

|  |
| --- |
| var *str1* = 'a'; var *str2* = 'b'; var *str3* = 'c'; str4 = *str1*.concat(*str2*,*str3*); str5 = *str1* + *str2* + *str3*; alert(str4); --> 'abc' alert(str5); --> 'abc' |

### 12.6 锚（了解）

|  |  |
| --- | --- |
| 命令 | 说明 |
| anchor() | 创建锚 |

### 12.7 样式（了解）

|  |  |
| --- | --- |
| 命令 | 说明 |
| fontsize() | 指定尺寸显示字符串 |
| fontcolor() | 指定颜色显示字符串 |
| bold() | 粗体显示字符串 |
| italics() | 斜体显示字符串 |
| blink() | 显示闪动字符串 |
| link() | 字符串显示为链接样式 |
| fixed() | 以打印机文本方式显示字符串 |

## 十三 RegExp

### 13.1 创建

#### 13.1.1 attribute可取值

|  |  |
| --- | --- |
| attribute | 说明 |
| i | 执行对大小写不敏感的匹配 |
| g | 全局匹配 |
| m | 执行多行匹配 |

#### 13.1.2 两种创建方式

|  |
| --- |
| var myPattern1 = new RegExp(pattern,attribute);  var myPattern2 = /【pattern】/【attribute】 |

例如：

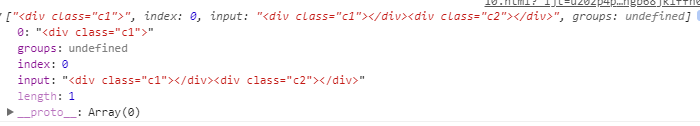
|  |
| --- |
| var *pattern\_1* = new *RegExp*('(<div.\*?>)','g'); var *pattern\_2* = /(div.\*?)/g; |

### 13.2 RegExp对象方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| compile（了解） | 编译正则表达式 |
| exec() | 检索正则表达式，返回object对象 |
| test() | 检索正则表达式，有：true，没有：false |

#### 13.2.1 exec()

|  |
| --- |
| var *a* = '<div class="c1"></div><div class="c2"></div>'; var *pattern*= /<div.\*?>/g; ret = *pattern*.exec(*a*); *console*.log(ret);  alert(ret[0]);  alert(ret.index);  alert(ret.input); |



注：exec()方法中的pattern虽然加了g，表示全局搜索，单还是匹配到第一个就返回

#### 13.2.2 test()

|  |
| --- |
| var *a* = '<div class="c1"></div><div class="c2"></div>'; var *pattern*= /<div.\*?>/g; ret = *pattern*.test(*a*); *console*.log(ret); |

### 13.3 String对象的4中方法

|  |  |
| --- | --- |
| 命令 | 说明 |
| search() | 返回匹配成功的第一个索引 |
| match() g | 返回匹配成功的字符串（返回第一个） |
| replace() g |  |
| split() |  |

#### 13.3.1 search()

|  |
| --- |
| var *a* = '<div class="c1"></div><div class="c2"></div>'; var *pattern* = /<div.\*?>/g; var *ret* = *a*.search(*pattern*); *console*.log(*ret*); |

#### 13.3.2 match()

|  |
| --- |
| var *a* = '<div class="c1"></div><div class="c2"></div>'; var *pattern*= /<div.\*?>/g; var *ret* = *a*.match(*pattern*); *console*.log(*ret*); --> ['<div class="c1">','<div class="c1">'] |

#### 13.3.3 replace()

|  |
| --- |
| var *a* = '<div class="c1"></div><div class="c2"></div>'; var *pattern*= /<div.\*?>/g; var *ret* = *a*.replace(*pattern*,'==='); *console*.log(*ret*); |

#### 13.3.4 split()

看Stirng中的split()

## 十四 数组array

### 14.1 创建

|  |  |
| --- | --- |
| 命令 | 说明 |
| var arrayObject = new Array(); | 创建一个空数组 |
| var arrayObject = []; | 同上 |
| var arrayObject = new Array(1,2,3); | 创建并赋值 |
| var arrayObject = [1,2,3]; | 同上 |
| var arrayObject = new Array(size); | 创建数组，并用添加size个undefined |

注：new可以不写

### 14.2 对象属性

|  |  |
| --- | --- |
| 属性 | 解释 |
| length | 数组中元素的数目 |
| constructor（了解） | 返回对创建次对象的数组函数的引用 |
| prototype（了解） | 有能力对对象添加的属性和方法 |

例如：

|  |
| --- |
| var arrayObject = new Array(10); console.log(arrayObject.length); --> 10 |

### 14.3 对象方法

#### 14.3.1 进出栈

|  |  |
| --- | --- |
| 命令 | 说明 |
| pop() | 右侧出栈 |
| push() | 右侧进栈 |
| shift() | 左侧出栈 |
| unshift() | 左侧进栈 |

例如：

|  |
| --- |
| var arrayObject = [1,2,3]; console.log(arrayObject.pop()); arrayObject.push(4); console.log(arrayObject.shift()); arrayObject.unshift(5); console.log(arrayObject) |

#### 14.3.2 顺序

|  |  |
| --- | --- |
| 命令 | 说明 |
| sort() | 从小到大排序 |
| reverse() | 翻转 |

例如：

|  |
| --- |
| var *arrayObject* = [1,4,2]; *console*.log(*arrayObject*.sort()); *console*.log(*arrayObject*.sort().reverse()); |

#### 14.3.3 切片

|  |  |
| --- | --- |
| 命令 | 说明 |
| slice(start,end) | 包括start，不包括end |

例如：

|  |
| --- |
| var *a* = [0,1,2,3]; alert(*a*.slice(0,1)); --> [0] |

#### 14.3.4 替换

|  |  |
| --- | --- |
| 命令 | 说明 |
| splice(start,end,objs) | 删除从start开始到end索引对应的元素，然后把objs添加进去 |

例如：

|  |
| --- |
| var *a* = [0,1,2,3]; *a*.splice(0,1,'a','b','c'); alert(*a*); --> ['a','b','c',2,3] |

#### 14.3.5 拼接

|  |  |
| --- | --- |
| 命令 | 说明 |
| concat(arrays) | 拼接数组 |

例如：

|  |
| --- |
| var arrayObject\_1 = [1,2,3]; var arrayObject\_2 = [4,5,6]; var arrayObject\_3 = [7,8,9]; var arrayObject\_4 = arrayObject\_1.concat(arrayObject\_2,arrayObject\_3); alert(arrayObject\_4); --> [1,2,3,4,5,6,7,8,9] |

#### 14.3.6 转字符串

|  |  |
| --- | --- |
| 命令 | 说明 |
| toString() | 转字符串 |
| toLocaleString() | 转字符串（本地格式） |
| join(str) | 用str拼接字符串 |

例如：

|  |
| --- |
| var *arrayObject* = ["1",2,3]; arrayObject\_string = *arrayObject*.toString(); arrayObject\_localeString = *arrayObject*.toLocaleString(); arrayObject\_join = *arrayObject*.join('--');  *console*.log(arrayObject\_string); --> '1,2,3' *console*.log(arrayObject\_localeString); --> '1,2,3' *console*.log(arrayObject\_join); --> '1--2--3' |

## 十五 Date

### 15.1 创建日期对象

|  |
| --- |
| var *date* = new *Date*();  console.log(date); |

### 15.2 设置与取值

|  |  |
| --- | --- |
| 属性 | 关键字 |
| 年（四位数） | FullYear |
| 月（0-11）（一月：0,十二月：11） | Month |
| 日（1-31） | Date |
| 小时（0-23） | Hours |
| 分钟（0-59） | Minutes |
| 秒钟（0-59） | Seconds |
| 毫秒（0-999） | Milliseconds |
| 星期（0-6） | Day |
| 时间戳毫秒（1970.1.1—今） | Time |

|  |  |
| --- | --- |
| 功能 | 命令 |
| 本地时间取值 | get【关键字】() |
| 本地时间设置 | set【关键字】() |
| 世界时间取值 | getUTC【关键字】() |
| 世界时间设置 | setUTC【关键字】() |

注意： 有getTime()和setTime()

没有getUTCTime()和setUTCTime()

补充命令：

parse() 返回1970年1月1日到指定日期的毫秒数

UTC() 返回1970年1月1日到指定日期的毫秒数（世界时间）

getTimezoneOffset() 返回本地时间与世界时间的分钟差

例如：

|  |
| --- |
| var *date* = new *Date*(); year = *date*.getFullYear(); month = *date*.getMonth(); date\_ = *date*.getDate(); hours = *date*.getHours(); minutes = *date*.getMinutes(); seconds = *date*.getSeconds(); milliseconds = *date*.getMilliseconds(); day = *date*.getDay();  *console*.log(year,month,date\_); *console*.log(hours,minutes,seconds,milliseconds); *console*.log(day); |

### 15.3 转字符串

|  |  |
| --- | --- |
| 命令 | 说明 |
| toString() | Date转字符串 |
| toTimeString() | Date中的时间转字符串 |
| toDateString() | Date中的日期转字符串 |
| toLocaleString() | Date转字符串（本地时间规则） |
| toLocaleTimeString() | Date中的时间转字符串（本地时间规则） |
| toLocaleDateString() | Date中的日期转字符串（本地时间规则） |

例如：

|  |
| --- |
| var *date* = new *Date*(); date\_toString = *date*.toString(); date\_toTimeString = *date*.toTimeString(); date\_toDateString = *date*.toDateString(); date\_toLocaleString = *date*.toLocaleString(); date\_toLocaleTimeString = *date*.toLocaleTimeString(); date\_toLocaleDateString = *date*.toLocaleDateString();   *console*.log(date\_toString); *console*.log(date\_toTimeString); *console*.log(date\_toDateString); *console*.log('-------------------------'); *console*.log(date\_toLocaleString); *console*.log(date\_toLocaleTimeString);  *console*.log(date\_toLocaleDateString); |

## 十六 Math

### 16.1 对象属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| E | e（≈2.718） |
| PI | Ω（≈3.14） |
| LN2 | 2的自然对数（≈0.693） |
| LN10 | 10的自然对数（≈2.302） |
| LOG2E | 2为底e的对数（≈1.414） |
| LOG10E | 10为底e的对数（≈0.434） |
| SQRT2 | 2的平方根（≈1.414） |
| SQRT1\_2 | 2的平方根的倒数（≈0.707） |

例如：

|  |
| --- |
| e = *Math*.E; pi = *Math*.PI; ln2 = *Math*.LN2; ln10 = *Math*.LN10; log2e = *Math*.LOG2E; log10e = *Math*.LOG10E; sqrt2 = *Math*.SQRT2; sqrt1\_2 = *Math*.SQRT1\_2;  *console*.log(e); *console*.log(pi); *console*.log(ln2); *console*.log(ln10); *console*.log(log2e); *console*.log(log10e); *console*.log(sqrt2); *console*.log(sqrt1\_2); |

### 16.2 对象方法

#### 16.2.1 算数运算

|  |  |
| --- | --- |
| 命令 | 说明 |
| exp(x) | e的指数 |
| log(x) | x的自然对数 |
| sqrt(x) | 平方根 |
| abs(x) | 绝对值 |
| max(x,y) | 取最大值 |
| min(x,y) | 取最小值 |
| round(x) | 四舍五入 |
| ceil(x) | 上舍入 |
| floor(x) | 下舍入 |

#### 16.2.2 三角函数

|  |  |
| --- | --- |
| acos(x) | 反余弦 |
| asin(x) | 反正弦 |
| atan(x) | 反正切 |
| cos(x) | 余弦 |
| sin(x) | 正弦 |
| tan(x) | 正切 |

#### 16.2.3 随机数

|  |  |
| --- | --- |
| 命令 | 说明 |
| random() | 0-1的随机数 |

## 十七 Function（了解）

### 17.1 顶层函数

#### 17.1.1 编码

|  |  |
| --- | --- |
| 命令 | 说明 |
| decodeURI() | 解码某个编码的URI |
| decodeURIComponent() | 解码某个编码的URI组件 |
| encodeURI() | 把字符串编码为URI |
| encodeURIComponent() | 把字符串编码为URI组件 |
| escape() | 对字符串进行编码 |
| unescape() | 对escape()编码的字符串解码 |

#### 17.1.2 运行字符串（重要）

|  |  |
| --- | --- |
| 命令 | 说明 |
| eval | 把字符串作为脚本来执行 |

#### 17.1.3 转换类型

|  |  |
| --- | --- |
| 命令 | 说明 |
| Number() | 把对象转成数字 |
| parseInt() | 转成整数 |
| parseFloat() | 转成浮点数 |
| String() | 转成字符串 |

#### 17.1.4 getClass()

|  |  |
| --- | --- |
| 命令 | 说明 |
| getClass() | 返回一个JavaObject的JavaClass |

#### 17.1.5 infinite和NaN的判断

|  |  |
| --- | --- |
| 命令 | 说明 |
| isFinite() | 检查是否是无穷大 |
| isNaN() | 检查是否是数字 |
|  |  |

### 17.2 顶层属性

|  |  |
| --- | --- |
| 命令 | 说明 |
| Infinity | 无穷大 |
| java | 代表java.\*，包层级的一个JavaPackage |
| NaN | 指示某个值是不是数字 |
| Packages | 根JavaPackage对象 |
| undefined | 指示未定义的值 |

## 十八 Object

### 18.1 创建

|  |
| --- |
| var a = new Object(); （new可以不写） |

### 18.2 特殊用法

|  |
| --- |
| var *a* = *Object*(); *a*.func = function(){  // do something }; *a*.array = [1,2,3]; alert(*a*.array[1]); |

## 十九 json

JSON 语法是 JavaScript 对象表示法语法的子集。

•数据在名称/值对中

•数据由逗号分隔

•花括号保存对象

•方括号保存数组

例如：

|  |
| --- |
| var *myJson* = {  "employees": [  { "firstName":"Bill" , "lastName":"Gates" },  { "firstName":"George" , "lastName":"Bush" },  { "firstName":"Thomas" , "lastName":"Carter" }  ] }; |

## 二十一 DOM

### 20.1 dom树

|  |
| --- |
|  |

### 20.2 节点

#### 20.2.1 节点的含义

根据 W3C 的 HTML DOM 标准，HTML 文档中的所有内容都是节点：

•整个文档是一个文档节点

•每个 HTML 元素是元素节点

•HTML 元素内的文本是文本节点

•每个 HTML 属性是属性节点

•注释是注释节点

#### 20.2.2 父节点、同胞节点、子节点

|  |
| --- |
| <div id = 'id1' class = 'class1'>  <div id = 'id1\_1' class = 'class2'>  </div>  <div id = 'id1\_2' class = 'class2'>  <a href = '//www.baidu.com'>baidu</a>  <a href = '//www.taobao.com'>taobao</a>  </div>  <div id = 'id1\_3' class = 'class2'>  <a href = '//www.vip.com'>vip</a>  </div> </div> |

对于 <div id = "id1\_2">来说：

父节点（parent）：

parentNode： <div id = 'id1'>

同胞节点（sibling）：

上一个同胞节点previoussibling： <div id = 'id1\_1'>

下一个同胞节点nextsibling： <div id = 'id1\_3'>

子节点（child）：

第一个子节点firstChild： <a href = '//www.baidu.com'>

最后一个子节点lastChild： <a href = '//www.taobao.com'>

没有父节点的节点叫做根节点（root）

### 20.3 常用属性

|  |  |
| --- | --- |
| 命令 | 说明 |
| parentNode（或者parentElement） | 节点的父节点 |
| childNodes（注意换行也是一个子节点） | 节点的子节点 |
| firstChild | 第一个子节点 |
| lastChild | 最后一个子节点 |
| attributes | 节点的属性节点 |
| innerHTML | 节点内的内容 |
| innerText | 节点内的文本 |
| nodeName | 节点名字 |
| nodeValue | 节点属性值 |
| nodeType | 节点属性 |

#### 20.3.1父节点和子节点 parentNode和childNodes

|  |
| --- |
| element\_id1\_2 = *document*.getElementById('id1\_2'); element\_id1\_2\_parentNode = element\_id1\_2.parentNode; element\_id1 = *document*.getElementById('id1'); *console*.log (element\_id1 === element\_id1\_2\_parentNode);  element\_id1\_2\_childNodes = element\_id1\_2.childNodes; *console*.log(element\_id1\_2\_childNodes); |

#### 20.3.2 属性节点attributes

|  |
| --- |
| element\_id1\_2 = *document*.getElementById('id1\_2'); attributes = element\_id1\_2.attributes; *console*.log(attributes); |

#### 20.3.3 节点内容 innerHTML（重点）

|  |
| --- |
| element\_id1\_2 = *document*.getElementById('id1\_2'); html = element\_id1\_2.innerHTML; *console*.log(html); element\_id1\_2.innerHTML = '<a href = "#">空链接</a>'; html = element\_id1\_2.innerHTML; *console*.log(html); |

#### 20.3.4 节点文本 innerText（区别innerHTML）

|  |
| --- |
| element\_id1\_2 = *document*.getElementById('id1\_2'); text = element\_id1\_2.innerText; *console*.log(text); |

### 20.4 常用方法

#### 20.4.1 得到节点

|  |  |
| --- | --- |
| 命令 | 说明 |
| getElementById() | 根据id返回节点 |
| getElementsByTagName() | 根据标签名字返回节点集合 |
| getElementsByClassName() | 根据class返回节点集合 |

例如：

|  |
| --- |
| element\_div\_id1 = *document*.getElementById('id1'); element\_div\_s = *document*.getElementsByTagName('div'); element\_div\_class2 = *document*.getElementsByClassName('class2');  *console*.log(element\_div\_id1); *console*.log(element\_div\_s); *console*.log(element\_div\_class2); |

#### 20.4.2 创建

|  |  |
| --- | --- |
| 命令 | 说明 |
| createAttribute() | 创建属性节点 |
| createElement() | 创建元素节点 |
| createTextNode() | 创建文本节点 |
| ——————————————————— | ——————————————————— |
| appendChild() | 添加子节点 |
| removeChild() | 删除子节点 |
| replaceChild() | 替换子节点 |
| insertBefore(插入的节点，引用的节点) | 在指定子节点前插入新的子节点 |
| insertAfter(插入的节点，引用的节点) | 在指定子节点后插入新的子节点 |
| ——————————————————— | ——————————————————— |
| getAttribute() | 返回指定的属性值 |
| setAttribute() | 把指定属性设置或改为指定的值 |
| setAttributeNode() | 设置属性节点 |

例子1（添加）：

|  |
| --- |
| var new\_element = *document*.createElement('a'); new\_element.setAttribute('href','#');  var text\_node = *document*.createTextNode('新的链接'); new\_element.appendChild(text\_node);  element\_div\_1\_1 = *document*.getElementsByClassName('class2')[0];  element\_div\_1\_1.appendChild(new\_element); |

例子2（添加）：

|  |
| --- |
| var new\_element = *document*.createElement('a');  var attribute = *document*.createAttribute('href'); attribute.value = '#';  var text\_node = *document*.createTextNode('新的链接');  new\_element.setAttributeNode(attribute); new\_element.appendChild(text\_node);  element\_div\_1\_1 = *document*.getElementsByClassName('class2')[0]; element\_div\_1\_1.appendChild(new\_element); |

例子3（添加）：

|  |
| --- |
| element\_div\_1\_2 = *document*.getElementsByClassName('class2')[1]; var new\_element = *document*.createElement('a'); var textNode = *document*.createTextNode('新的链接'); new\_element.appendChild(textNode); new\_element.setAttribute('href','#');  element\_div\_1\_2.insertBefore(new\_element,element\_div\_1\_2.firstChild); |

例子4（删除）：

|  |
| --- |
| element\_div\_1\_2 = *document*.getElementsByClassName('class2')[1]; element\_div\_1\_2.removeChild(element\_div\_1\_2.childNodes[1]); |

例子5（删除）：

|  |
| --- |
| element\_a = *document*.getElementsByTagName('a')[0]; element\_a.parentNode.removeChild(element\_a); |

## 二十二 BOM

### 22.1 window尺寸和windos.screen

|  |  |
| --- | --- |
| 命令 | 说明 |
| window.innerHeight | 浏览器窗口内置高度 |
| window.innerWidth | 浏览器窗口内置宽度 |
| ——————————————————— | ——————————————————— |
| window.screen.availHeight | 可用的屏幕高度（可以不写window） |
| window.screen.availWidth | 可用的屏幕宽度（可以不写window） |

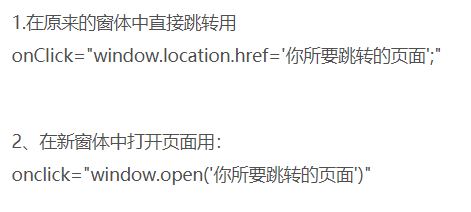
### 22.2 window.location

|  |  |
| --- | --- |
| 命令 | 说明 |
| location.hostname | 返回web主机的域名 |
| location.pathname | 返回当前页面的路径和文件名 |
| location.href | 返回整个页面的url |
| location.port | 返回web主机的端口号 |
| locaiton.protocol | 返回时用的web协议（http或者https） |
| location.assign() | 加载新文档 |
| location.open() | 打开新的网页 |

### 22.3 跳转

#### 22.3.1 location.href

|  |
| --- |
| *location*.href = '//www.baidu.com'; |



#### 22.3.2 window.open()

|  |
| --- |
| *window*.open('//www.baidu.com'); |

#### 22.3.3 location.assign()

|  |
| --- |
| *location*.assign('//www.baidu.com'); |

#### 22.3.4 延时跳转

|  |
| --- |
| var func = function(){  *location*.href = '//www.baidu.com'; }; setTimeout(func,3000); |

### 22.4 history

|  |  |
| --- | --- |
| 命令 | 说明 |
| history.back() | 后退 |
| history.forward() | 前进 |

### 22.5 popupAlert

|  |  |
| --- | --- |
| 命令 | 说明 |
| alert() | 警告框 |
| confirm() | 确认框 |
| prompt() | 输入框 |

### 22.6 timing

|  |  |
| --- | --- |
| 命令 | 说明 |
| setTimeOut(【js语句】,【毫秒】) | 计时 |
| clearTimeOut(【计时器】) | 清除计时 |

例如：

|  |
| --- |
| var func1 = function(){  t = setTimeout('alert("弹出内容")',2000); }; var func2 = function(){  clearTimeout(t); }  ————————————————————————————————————  <input type="button" value = 'setTimeOut' onclick = 'func1()'> <input type="button" value = 'clearTimeOut' onclick = 'func2()'> |

## 二十三 this关键字

<https://blog.csdn.net/mutouafangzi/article/details/76563005>