&times**;** 乘号×

&lt; <箭头

&gt; >箭头

this 内部对象，只能在函数内部使用

**.toFixed(**num**);** 在数字(类型必须是number)后面调用，num为小数为，有四舍五入的功能，得到一个字符串

例 15.222.toFixed(2);

**parseInt();** 解析一个字符串，并返回一个整数 (多用于取整) (会自动滤掉非数字)

parseInt(11.2);

**parseFloat(**string**);** 解析一个字符串，并返回一个浮点数 (会自动滤掉非数字)

typeof(); 数据类型判断(返回类型是字符串)

isNaN(x);判断x是否**非数字 返回值布尔类型**

**.**focus(); 自动获得焦点

x.value;获取x的值

x.value=5; 把5赋值给x

**函数的执行**

手动调用:

sum();

事件驱动:

格式： 元素**.**事件 = 函数名;

buton**.**onclick = sum; (无括号)

console.log(); 控制台中输出

document.write(); 在html中输出 可以使用,传输多个参数显示出来或者用+字符串拼接

document.writeln();在html中输出并自动换行 同上

# 数组

数组属于：引用数据类型

数组创建方式

字面量（直接量）(推荐)

var arr = [1,2,3];

使用构造函数创建

var arr = new Array();//创建一个空数组

属性

arr.length 数组的长度 后面没有()

方法

arr.unshift(添加的元素); 往数组开头添加一个或多个元素，返回数组新的长度

arr.push(添加的元素); 往数组尾部添加一个或多个元素，返回数组新的长度

arr.shift(); 删除数组第一个元素，返回删除的元素

arr.pop(); 删除数组最后一个元素，返回删除的元素

arr.sort(); 将数组中的元素排序，并返回排序后的数组 (返回原数组)

默认以字符串的排列方式（转换成ASCII码进行对比

按数字大小排序 var numArr = new Array(12,23,1,4,23,34,2,5);

numArr**.sort(function(a,b){return a-b;});( 正为>否则负0为=)**

for (var i = 0; i<numArr.length; i++) {

document.write(numArr[i] +"<br>");

} 页面输出 1 2 5...

(倒序： numArr.sort(function(a,b){**return b-a;**}); )

[javascript\Array\14默认排序法.html](javascript/Array/14默认排序法.html)

arr.reverse(); 将数组中的元素颠倒顺序，返回逆序后的数组(在原数组改变)

arr.slice(start[,end]); 截取，返回数组的片段或子数组，从start开始到end(不包括end所对应的元素) 如果省略end参数，则截取到数组的最后一项 返回的新的数组

arr.splice(start,deleteNum,…items) (); 在数组中插入、删除、替换的通用方法

start 起始索引位置

deleteNum要删除的数量

items插入的元素（可以是多个 用逗号分隔）

**在原数组上操作**

arr.join(separator); 返回字符串值(新的)，其中包含了连接到一起的数组的所有元素

separator为分隔符，默认为逗号 **‘’**为无分隔

arr.concat(); 返回一个新数组，这个新数组是由调用这个方法的数组和参数组成

参数可以是多个

Array.isArray(需要判断的数组);判断是否为数组 (ECMAScript5(ES5)新方法)

迭代方法(ES5方法)

arr.forEach(fn) 遍历方法，for循环没有太大差别，比for循环方便 (类数组不能使用该方法)

不能用break、continue 可以分别用return false、return true 代替

arr.map(fn) 返回每次函数调用的结果组成的数组 (返回新数组)

[javascript\ES5&String\数组的es5方法.html](file:///C:\Users\Administrator\AppData\Roaming\Microsoft\Word\javascript\ES5&String\数组的es5方法.html)

arr.filter(fn) 返回该函数会返回 true 的项组成的数组，利用这个方法可对数组元素进行过滤筛选 (返回新数组)

arr.every(fn) 如果该函数对每一项都返回 true，则返回true

arr.some(fn) 如果该函数对任何一项返回 true，则返回true

以上方法都对数组中的每一项运行给定函数fn,，函数中有三个形参分别为

- item：数组中的每一项,

- index：遍历过程中对应的索引值,

- array：对数组的引用

归并方法(ES5方法)

这两个方法都会迭代数组中的所有项，然后生成一个最终返回值。

reduce(fn,initVal)

reduceRight(fn,initVal) (从右往左查找)

[javascript\ES5&String\数组的es5方法-归并.html](file:///C:\Users\Administrator\AppData\Roaming\Microsoft\Word\javascript\ES5&String\数组的es5方法-归并.html)

fn(prev,cur,index,array): fn是每一项调用的函数，函数接受4个参数分别是

prev：前一次返回值，

cur：当前值，

index：索引值，

array：当前数组，

函数需要返回一个值，这个值会在下一次迭代中作为prev的值

initVal: 迭代初始值（可选），如果缺省，prev的值为数组第一项

# 字符串String

String类是不可变的,对String类的任何改变,都是返回一个新的String类对象

创建一个字符串

方式一：字面量（推荐）

var str = '城市套路深，我想回农村';

方式二：构造函数

//PS：用new产生的变量都是引用类型的变量，也叫对象

var str = new String('我不是黄蓉，我不会武功');

str**.length();** 表示字符串的长度，只读（只能读取）

str**.charAt(3);** //获取下标为3的字符

方法

str**.indexOf(****"**需查找的内容**");** 从开头向后查找字符串第一次出现的位置,如有**返回**所在的

**索引值否则**返回**-1**(可用于数组)

str**.lastIndexOf("**需查找的内容**");** 从尾部向前查找字符串第一次出现的位置，如有**返回**所在的

**索引值否则**返回**-1**(可用于数组)

str**.search(str|regExp);** 查找字符串第一次出现的位置 (支持正则indexOf不支持)

str**.match(str|regExp);** 匹配字符串，返回一个数组

index:匹配字符所在的索引

input:表示整个字符串的引用

str**.replace(str|rgExp, '替换内容');** 替换字符串

这里的替换只能执行一次，不能够进行全局匹配，如果需要全局匹配，则应使用正则表达式

截取方法

str**.substring(start[,end]);** 不包括end所在字符，end省略表示截取到最后

str**.substr(start[,len]);** 支持负数，len为截取的数量

str**.slice(start,end);** 截取start到end(不包括end)的字符串，**支持负数**

字符串分割

str**.split(分割符);** 根据分割字符(默认逗号)，把字符串拆分成**数组**

大小写转换

str**.toLowerCase();** 转换成小写

str**.toUpperCase();** 转换成大写

ES5新增:

**str[3]** 通过下标获取

str**.trim();** 删除前后所有空格，返回新的字符串

ASCII码和字符集

str**.****charCodeAt(3);** 获取下标为3的字符的ASCII(American Standard Code for \* Information Interchange) == > unicode编码

**String.fromCharCode(94);** //编码转换成字符

[javascript\ES5&String\doc\ES5&String.html](javascript/ES5&String/doc/ES5&String.html)

## 对象 引用数据类型

var obj = new Object(); / var obj = {}; **创建**一个**空对象**

读取一个不存在的属性，返回undefined

**创建**对象**并写入属性**

var superStar = {

// 属性:属性值**,**

name:'lanmeng',}

创建完对象后再**设置属性** 两种方式

obj.marry = false;

obj[**'**weight**'**] = 60; (没有引号会被当作变量，容易报错，根据实际加与否)

获取对象的值：

superstar.weight或者superstar[weight]**当属性为变量时必须使用**第二种方式

**for…in** 遍历(也可用于数组，注意遍历顺序是随机的，随着引擎不同而不同)

如 for(var attr in superStar){

// 遍历的过程会把每一个属性赋值给attr

// 遍历的属性会以写入的顺序进行打印

console.log(attr);

}

[javascript\Array\doc\数组.html](javascript/Array/doc/数组.html)

## Math 对象方法

属性：

**Math.E** //自然对数底，即e的值

**Math.PI** //3.1415926

方法：

**Math.round();** 四舍五入取整

例 Math.round(0.6);

**Math.ceil();** 向上取整 它的值**大于或等于**这个浮点数

**Math.floor();** 向下取整，它的值**小于或等于**这个浮点数

例 Math.floor(0.60) 为 0

Math.floor(-5.9) 为 6

**Math.random()**;随机得到一个0-1(不含1)的数 返回值double

例 var a = Math.random();

**Math.max(num1, num2,…);** //返回较大的数

**Math.min(num1, num2,…);** //返回较小的数

**Math.abs(num);** 绝对值

**Math.pow(x,y);** x的y次方

**Math.sqrt(num);** 开平方根

**Math.sin(**radian**);**

**Math.cos(**radian**);**

**Math.tan(**radian**);**

角度与弧度的转换 弧度=角度\*Math.PI/180

勾股定理 在直角三角形中，斜边的平方等于直角边的平方和

[javascript\Math&Date\doc\Math&Date.html](javascript/Math&Date/doc/Math&Date.html)

## 日期Date

创建当前时间的日期和时间

var d = new **D**ate();//得到的是代码执行时的时间（本地时间）

创建指定日期的时间和日期

var d = new **D**ate("2015/08/22");

var d = new **D**ate(56521211021);//参数为距1970-1-1零时的毫秒数

获取/设置时间 注意大小写

获取年月日

**.getFullYear()/setFullYear(2014); 年**

**.getMonth()/setMonth(8); 月** 注意：月份是从0开始(0-11)

**.getDate()/setDate(25); 日**

获取星期

**.getDay();** 0-6:星期天-星期一 没有set方法

获取时分秒 都有s

**.getHours()/setHours(); 时**

**.getMinutes()/setMinutes(); 分**

**.getSeconds()/setSeconds(); 秒**

**.getMilliseconds()/setMilliseconds(); 毫秒**

UTC时间：在以上方法的get/set后加UTC, 如：

**.set/getUTCFullYear();**

**.get/setUTCHours();**

**…**

日期处理 都没有s

**.getTime();** 获取某个日期自1970年1月1日0时以来的毫秒数

**.setTime();** 修改日期的毫秒数，对应的日期时间也会修改

**.toDateString();** 以特定的格式显示星期几、月、日和年

**.toTimeString();** 以特定的格式显示时、分、秒和时区

**.toLocaleDateString();** 以特定地区格式显示年、月、日

**.toLocaleTimeString();** 以特定地区格式显示时、分、秒

ES5方法：

**Date.parse(“2015-08-24”);** 返回指定日期距1970-1-1零时的毫秒数

PS：转换格式默认支持2015-08-24或2015/08/24

**Date.now();**  返回执行这行代码时距1970-1-1零时的毫秒数

[javascript\Math&Date\doc\Math&Date.html](javascript/Math&Date/doc/Math&Date.html)

## 延迟与定时器

setTimeout(fn,200); 两百毫秒后执行fn这个函数（只执行一次）,返回一个id标识

clearTimeout(timeoutID); 清除指定id标识的延迟操作

setInterval(fn,30); 每隔30毫秒执行一次fn这个函数,返回一个id标识

clearInterval(intervalID); 清除指定id标识的定时器操作

例 var timer = setTimeout(function(){

//2s后执行这里的代码

},2000);

//清除

clearTimeout(timer);

延迟定时器的执行是异步的

[javascript\Math&Date\doc\Math&Date.html](javascript/Math&Date/doc/Math&Date.html)

# BOM

BOM 是Browser Object Model（浏览器对象模型）的缩写，提供与浏览器窗口进行交互的对象

## window对象

它是BOM的核心, 是最顶层的对象，所有对象都是通过它延伸出来的

**全局作用域**

定义在全局环境下的变量都会成为window对象的属性

全局方法：在window对象下的方法 (如alert();，setInterval();)

把变量定义在函数体里，可以有效减少全局环境下的变量冲突，避免污染全局环境

通过var在全局作用域下声明的变量用delete无法删除(不加var通过隐式声明可删除)

var obj = {name:'xxx'}

**删除对象的属性**用**delete**：

delete obj.name;

在函数内部不用var声明的变量会成为全局变量，即window对象的属性

window对象可以在代码中省略，如window.alert()可以写成alert();

**window对象下的属性**

常用属性

浏览器窗口尺寸

window.**innerWidth/innerHeight;** 表示浏览器窗口”**可视区域**”**尺寸**

**outerWidth/outerHeight;** 表示**整个**浏览器**窗口**的**尺寸**

滚动相关

window.**scrollX/****scrollY;** 获取浏览器窗口滚动条**滚动过的距离**

**scrollTo(x,y);** **设置**浏览器**滚动距离**

**scrollBy(xnum,ynum);** **设置基于当前位置滚动的距离**，可以为负数

常用方法(了解)

系统对话框

**.alert(**msg**);** 弹出**对话框**



**.confirm(**msg**);** 弹出**确认框**，返回布尔值



**.prompt(**msg,default**);**  弹出**输入框**，确定返回消息/取消返回null



\*\*三个方法都是中断代码的执行

**open(**'url',name,options**);** **打开**一个**新窗口**并返回新 window 对象

name:不命名会每次打开新窗口，命名的第一次打开新窗口,之后在这个窗口中加载

options为字符串：'width=400,height=400,top=200,left=200'

opener父窗口对象，通过open方法打开的窗口才有opener对象

**close();** **关闭open的窗口**

**print();** 调出打印对话框

属性对象

document(核心) 文档对象，让我们可以在js脚本中直接访问页面元素(DOM)

**history**(重要): 历史对象,包含窗口的浏览历史，可以实现后退

属性：

history**.length** 返回浏览器**历史列表**中的 **URL 数量**。

方法：

history**.back();** 加载 history 列表中的前一个 URL(**后退**)

history**.forward();** 加载 history 列表中的下一个 URL(**前进**)

history**.go();** **加载** history **列表中**的某个**具体页面**，支持负数。

history.go(2)：向前两个页面；

history.go(-2)：后退两个页面

**location**(重要):

BOM最有用的对象之一，保存着当前窗口中加载文档的相关信息，还提供一些导航功能，它是个很特别的对象，既是window的属性，也是document的属性

属性：

location**.hash=** 设置或返回从井号 (#) 开始的 URL（锚）==>**哈希值**。

location**.**host= 设置或返回主机名和当前 URL 的端口号。

location**.**hostname= 设置或返回当前 URL 的主机名。

location**.href=** 设置或返回**完整的 URL**。

location**.**pathname= 设置或返回当前 URL 的路径部分。

location**.**port= 设置或返回当前 URL 的端口号。

location**.**protocol= 设置或返回当前 URL 的协议。

location**.search=** **设置或返回从问号 (包含?) 开始的 URL（查询部分）**。

(PS：修改以上属性(hash除外)都会刷新当前页面，并生成历史纪录)

[javascript\BOM\04页面之间传参.html](javascript/BOM/04页面之间传参.html)

encodeURI() 转码

var d=**decodeURI(**需解码的内容**) 解码**

[javascript\BOM\04list.html](javascript/BOM/04list.html)

方法：

**reload()** **重新加载**当前文档，带参数**true**表示**不使用缓存**刷新页面。

navigator(了解):

导航对象, 包含所有有关访问者浏览器的信息，通常用于检测浏览器类型

appName 浏览器名称

appVersion 浏览器版本

platform 操作系统

userAgent 用户代理信息，通过该属性可以获取浏览器及操作系统信息

geolacation 获取地理位置信息

window对象常用事件

**.onload**=fn**;** **页面资源**全部**加载完成**后**触发**这个事件

**.onbeforeunload**=fn**;** 在**页面关闭**时**触发**这个事件

**.onscroll**=fn**; 滚动**窗口**滚动条**时**触发**

**.onresize**=fn**;** **窗口大小改变**时**触发**

获取html元素 (3种方式)：

**document.documentElement****.** ? **; (推荐)**

document.getElementsByTagName('html')[0];

document.body.parentNode;

**document.body.** ? **;** 获取body元素

**document.documentElement.scrollTop** 代表垂直的滚动条，向下滚动的距离

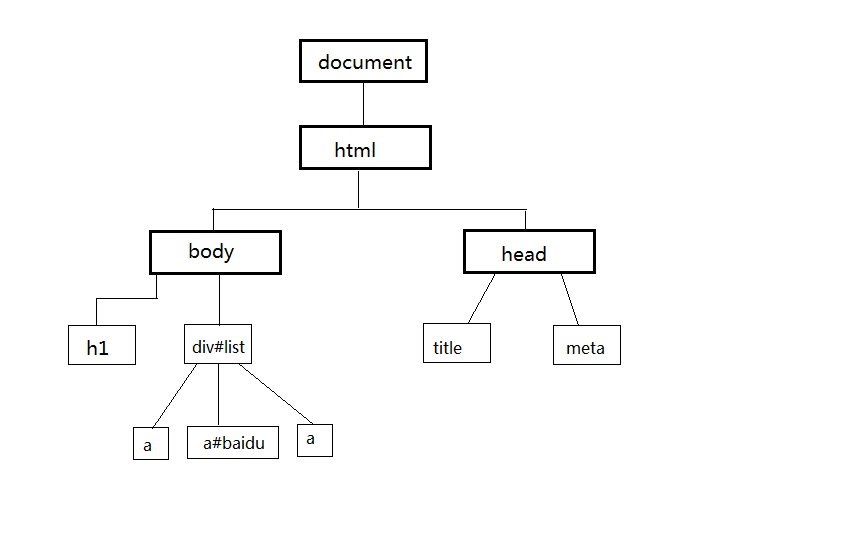
（FF、Opera 和IE）

**document.body.scrollTop** 代表**垂直**的滚动条，向下**滚动的距离**

（chrome和IE的quirks mode兼容模式）

# DOM

DOM是Document Object Model（文档对象模型）的缩写，是针对HTML和XML文档的一个API。DOM描绘了一个层次化的节点树，通过DOM，我们可以访问所有的HTML元素，连同它们所包含的文本和属性。可以对其中的内容进行修改和删除，同时也可以创建新的元素



## 节点类型

节点类型与对应**nodeType**值(用于判断获取到的元素属于什么类型节点)：

元素节点 <==> 1

文本节点 <==> 3

属性节点 <==> 2

## 节点获取

### document.getElementBy*Id*(id)

通过 **ID**获取元素的节点（速度最快）

*必须通过document调用*

返回DOM节点对象，如果id不存在返回null

document.getElementsBy*Name*()[ ]

通过**name**属性获取元素节点列表 (一般用于表单)

*必须通过document调用*

返回类数组，如果tagname不存在返回空数组[ ]

.getElementsBy*ClassName*()[ ]

通过class**类名**获取节点列表

返回类数组，如果tagname不存在返回空数组[ ]

.getElementsBy*TagName*(tagname)[ ]

通过**标签名**获取元素节点列表

返回类数组，如果tagname不存在返回空数组[ ]

document.images[ ]; 获取页面的图片(返回类数组)

注意: 如果确认元素存在, 但是返回null或[]，一定是代码执行顺序的问题

绑定事件时，页面一定要存在这个元素

【 页面事件绑定数量越多，性能越差

优化：**事件委托** (把本该自己处理的事件交给父元素处理)

[javascript\Event\09event对象的兼容性问题.html](javascript/Event/09event对象的兼容性问题.html) 】

利用节点关系获取其他节点 (一般用于获取文本节点，否则用元素节点操作)

获取父级节点

ele**.parentNode** 得到ele元素的**父节点**

获取子节点

ele**.childNodes** 得到ele元素的**全部子节点**列表（返回类数组，注意过滤文本节点）

ele**.firstChild** 获得ele元素的**第一个子节点**  (注意可能获取到文本节点的问题)

ele**.lastChild** 获得ele元素的**最后一个子节点** (注意可能获取到文本节点的问题)

获取兄弟节点

ele**.nextSibling** 获得ele元素的**下一个兄弟节点** (注意可能获取到文本节点的问题)

ele**.****previousSibling** 得到ele元素的**上一个兄弟节点** (注意可能获取到文本节点的问题)

## 节点操作

节点属性

**.nodeName** 返回**节点**的**名称**，根据其类型。

**.nodeType** 返回**节点**的**类型**。

**.nodeValue** 返回**节点**的**值**（元素节点的nodeValue为null）

节点方法

创建：

var ele = **document.createElement(****'**需增加的标签名**');** **创建**一个**元素节点**

var ele = **document.createTextNode('**文本**');** **创建**一个**文本节点**

var ele = **new Image();** **创建一张图片节点**

插入：

parent**.****appendChild('**需插入的子节点**');** 向**节点的子节点**列表的**结尾添加**新的**子节点**

parent**.insertBefore(new,node);** 在指定的**子节点**node**前插入新**的**子节点**new。

对页面已存在节点的处理

(如果元素已经存在页面上,appendChild()/insertBefore()都会移动元素)

[javascript\DOM\05创建节点.html](javascript/DOM/05创建节点.html)

复制&替换

var cloneE=需克隆的元素**.cloneNode(boolean);** **复制节点**，为true是深复制(克隆包括子元素)。

parent**.replaceChild(**new,old**);** 将old节点**替换**为new**节点**

[javascript\DOM\09印钞机.html](javascript/DOM/09印钞机.html)

删除：

parent.removeChild(ele); 删除（并返回）当前节点parent的指定子节点ele。

判断：

parent.hasChildNodes(); 判断当前节点是否拥有子节点,返回布尔值

ele.hasAttribute(attr); 判断当前元素是否拥有attr属性，返回布尔值

以上parent表示父级元素

## 元素节点的操作

常用属性 (可以通过点语法或方括号访问)

**.tagName** 获取元素**元素**的**标签名**

**.id(=)** 设置/获取**元素id**属性

**.name(=)** 设置/获取**元素name**属性

**.style(=)** 设置/获取元素的**内联样式**

**.className(=)** 设置/获取**元素**的**class**属性

**.innerHTML(=)** 设置/获取**元素的内容**（包含html代码）

**.outerHTML(=)** 设置或获取**元素及其内容**（包含html代码）

**.innerText(=)** 设置或获取位于**元素标签内**的**文本**

**.outerText(=)** 设置(包括标签)或获取(不包括标签)**元素的文本**

根据元素关系获取其他元素 (不存在获取到文本节点的问题)

**.****parentElement;** 获取**父**级**节点元素**

**.children;** 获取元素的**全部子节点元素** (返回类数组类型)

**.firstElementChild;** 获取**第一个子节点元素**

**.lastElementChild;** 获取**最后一个子节点元素**

**.previousElementSibling;** 获取**前一个节点元素**(IE8以下不支持)

**.nextElementSibling;** 获取**下一个节点元素**(IE8以下不支持)

元素方法 (适用所有的属性)

ele**.****getAttribute(attr);** //**获取**元素的**属性值** attr属性（自定义属性获取）

ele**.setAttribute(attr,val);** //**设置**元素的**属性** attr属性 val属性值(自定义属性规范:data-\*)

(可以设置自定义属性并会显示在html上，如果不在html上显示可以直接用**点**语法设置自定义属性)

ele**.removeAttribute(attr);** //**删除属性**attr

ele**.hasAttribute(attr);** //**判断**是否存在**属性**attr (注意这里类名是class而不是className)

盒模型相关

**.****offsetTop;** 当前元素**离**<**定位父级**>元素**顶部**的**距离**。

**.offsetLeft;** 当前元素**离**<**定位父级**>元素**左边**的**距离**。

以上两个属性如果没定位的父级，则相对于根元素html的距离

**.offsetWidth;** 当前**元素**的**宽度**（border + padding + content即**自border以内**）

**.offsetHeight;** 当前**元素**的**高度**（border + padding + content即**自border以内**）

获取css样式（非内联样式）

得到当前元素计算后的所有样式 (**点**语法只能获取内联样式)

var style=**getComputedStyle(**ele(,pseudo)**);** （标准）(window.getComputedStyl可用来判断

是否是IE8以下的浏览器)

ele:要获取样式的元素

pseudo:伪元素样式字符(可选)，可获取伪元素样式

ele**.currentStyle;** （IE8-）

(注意：两种方法互不兼容且会报错 , 要同时兼容IE8及其他浏览器要做判定)

用法：[javascript\DOM\14getComputedStyle.html](javascript/DOM/14getComputedStyle.html)

## table对象

table对象属性&方法

**rows** 返回包含表格中所有行的一个数组

**tBodies** 返回包含表格中所有 tbody 的一个数组

**.insertRow(index);** 在表格中的指定位置**插入**一个新**行** (特殊:index为**-1**表示在末尾插入)

**.deleteRow(index);** 从表格**删除**一**行**

Index 索引 起始值0

tr对象属性&方法

**.cells;** 返回包含表格中所有单元格的一个数组

**.rowIndex;** 返回该行在表中的位置(第一行位置为0)

**sectionRowIndex** 返回在 tBody 、tHead 或 tFoot 中行的位置。

**.insertCell(index);** 在一行中的指定位置插入一个空的列 (特殊:index为**-1**表示在末尾插入)

**.deleteCell(index);** **删除**行中的指定的**单元格**

Index 索引 起始值0

td/th对象属性&方法

**cellIndex** 返回单元格在表格行的单元格集合中的位置。

[javascript\DOM\18动态生成随机背景色表格.html](javascript/DOM/18动态生成随机背景色表格.html)

# 事件

格式 元素.on+事件名 = 事件处理程序;

例 div.onclick = function(){}

事件分类：

鼠标事件

**onclick** 当用户**点击**某个对象时调用的事件。

**ondblclick** 当用户**双击**某个对象时调用的事件。

**onmousedown** 鼠标按钮被**按下**。

**onmouseup** 鼠标按键被**松开**。

**onmouseover** 鼠标**移到某元素**之上。

**onmouseout** 鼠标从某元素**移开**。

**onmousemove** 鼠标被**移动时**触发。

**onmouseenter** 在鼠标光标从元素外部移动到元素范围之内时触发。这个事件不冒泡

**onmouseleave** 在位于元素上方的鼠标光标移动到元素范围之外时触发。这个事件不冒泡，

**oncontextmenu** 鼠标**右键菜单展开时**触发。

[javascript\Event\01事件分类.html](javascript/Event/01事件分类.html)

PS：click = mousedown + mouseup dblclick = click\*2; (短时间内两次单击)

执行顺序：mouseover=>mouseenter; mouseout => mouseleave

键盘事件

**onkeydown** 某个键盘按键被**按下**。

**onkeyup** 某个键盘按键被**松开**。

**onkeypress** 键盘**<字符键>**被**按下,**而且如果按住不放的话，**会重复触发**此事件。

UI事件

**onload 页面**元素（包括图片多媒体等）**加载完**成后(注意，有缓存时不会执行)

**onbeforeunload** 用户**退出**页面。(必须要有return语句并且return后要有值才生效)

**onresize** 窗口或框架被**重新调整大小**。

表单事件

**onselect** 输入框文本被**选中**。

**onblur** 元素**失去焦点**时触发。

**onfocus** 元素**获得焦点**时触发。

**onchange** 元素**内容**被**改变**，**且失去焦点时**触发。

**onreset** **重置按钮**被**点击**。

**onsubmit** **确认按钮**被**点击**。

**oninput** **输入字符时**触发

[javascript\Event\03表单事件.html](javascript/Event/03表单事件.html)

其他事件

暂无

# Event对象

event对象：事件执行过程中的状态，用来保存当前事件的信息对象

获取event对象

标准：事件处理函数的第一个参数

即 div.onclick = function(**e**){}

IE8- window.event;

兼容写法: e = e || window.event;

Event对象详情

鼠标/键盘事件Event属性

event**.button** 返回当事件被触发时，哪个**鼠标按钮被点击**。

W3C标准

**0** 代表鼠标按下了**左键**

**1** 代表按下了**滚轮**

**2** 代表鼠标按下了**右键**

IE8-（IE8以下的浏览器）

1鼠标左键， 2鼠标右键， 3左右同时按， 4滚轮， 5左键加滚轮，

6右键加滚轮， 7三个同时

event**.**which/**keyCode**

对于keypress事件，该属性声明了被敲击的键生成的 Unicode 字符码(ascii码)

对于keydown和keyup事件，它指定了被敲击的键的虚拟键盘码。虚拟键盘码可能和使用的键盘的布局相关。

[javascript\Event\07愤怒的小鸟.html](javascript/Event/07愤怒的小鸟.html)

event**.altKey** 返回当事件被触发时，”ALT” 是否被按下。

event**.ctrlKey** 返回当事件被触发时，”CTRL” 键是否被按下。

event**.shiftKey** 返回当事件被触发时，“Shift” 键是否被按下。

光标位置信息

event**.clientX** /**clientY** 光标**相对**于浏览器**可视区域**的位置，也就是浏览器坐标。

event**.screenX**/**screenY** 光标指针**相对**于电脑**屏幕**的水平/垂直坐标。

event**.pageX**/**pageY** 鼠标**相对**于**文档**的位置（包括滚动条滚动的距离，即：clientX+window.scrollX，IE不支持）

event**.offsetX**,**offsetY** 光标**相对**于**事件源对象**的相对偏移量。

事件源对象：触发事件的对象

Event公共属性

event**.type** 被触发的事件的类型(一般用于判断)

event**.target** 返回触发此事件的元素（事件的目标元素）。

event**.currentTarget** 其事件处理程序当前正在处理事件的那个元素

IE事件Event属性（ie8-）

event**.**cancelBubble 如果事件句柄想阻止事件传播到包容对象，必须把该属性设为 true

（同stopPropagation()）。

event**.**srcElement 对于生成事件的 Window 对象、Document 对象或 Element 对象的引用（同target）。

兼容写法var target = e.target || e.srcElement;

event**.**returnValue 如果设置了该属性，它的值比事件句柄的返回值优先级高。把这个属性设置为fasle，可以取消发生事件的源元素的默认动作（同preventDefault()）。

[javascript\Event\09event对象的兼容性问题.html](javascript/Event/09event对象的兼容性问题.html)

事件冒泡

在一个对象上触发某类事件（如onclick事件），那么click事件就会沿着DOM树向这个对象的父级传播，从里到外，直至它被处理程序处理，或者事件到达了最顶层（document/window）

事件源对象：触发事件的元素

标准：event.target

IE8-：event.srcElement

事件源对象在事件传播过程中不会改变

阻止事件的传播

标准：event**.stopPropagation()**;

IE8-：event.cancelBubble = true;

1）不是所有的事件都能冒泡。以下事件不冒泡：blur、focus、load、unload…。

2）冒泡到最顶层的目标不同。大部分浏览器到window对象，IE8-到document对象

[javascript\Event\11自定义右键菜单.html](javascript/Event/11自定义右键菜单.html)

阻止浏览器默认行为

(如：链接跳转、表单提交、右键菜单、图片拖拽…)

标准：event**.preventDefault()**;

IE8-：event.returnValue = false;

注意：阻止事件冒泡只能阻止同类事件

[javascript\Event\10返回顶部.html](javascript/Event/10返回顶部.html)

# 事件监听与捕获(反冒泡)

事件监听器

格式：元素**.addEventListener(**事件名**,**事件处理函数**,**是否捕获**);**

例 target.addEventListener("click", fn, false);

可以绑定多个处理函数在一个对象上, 执行顺序按照绑定的顺序来

不同元素事件执行顺序跟html结构有关

相同元素事件执行顺序跟绑定先后有关

第三个参数是否使用捕获(反向冒泡)，**默认false**，为**冒泡**

IE8-的事件监听器：

格式：元素**.attachEvent(**on+事件名**,**事件处理函数**);**

例 target.attachEvent("onclick",fun);

可以绑定多个函数在一个对象上, 执行顺序按照绑定的反序

移除

标准：**removeEventListener(**"click"**,**fn**,**true**);**传入的参数fn要跟添加时一样(同一个函数)，需要**三个参数一致**,**否则不能移除事件**

(用on+事件名可以使用覆盖的方式移除，如ele.onclick = null;)

IE8-：detachEvent**(**"onclick",fun**);** 传入的参数fun要跟添加时一样，否则不能移除事件

[javascript\Event\14事件移除.html](javascript/Event/14事件移除.html)

注意：

页面事件绑定数量越多，越影响性能（速度越慢）

# 拖拽效果

拖拽的原理

鼠标按下且移动鼠标时，改变当前元素的top,left值

拖拽的思路

1、给目标元素添加onmousedown事件

(拖拽的前提是目标元素设置css定位)

记录按下鼠标位置与元素左上角的偏移量offsetX，offsetY

2、当onmousedown发生以后，此刻给document添加onmousemove事件

(意味着此刻鼠标在网页的移动都将改变目标元素的位置)

在onmousemove事件中，设定目标元素的left和top：

(公式)目标元素的left = 鼠标的clientX – offsetX

(公式)目标元素的top = 鼠标的clientY– offsetY

3、给目标元素添加onmouseup事件，清空document的onmousemove事件

(意味着拖动过程中鼠标在网页的任意位置松开鼠标，都会放弃拖拽的效果)

[javascript\Event\15拖拽效果.html](javascript/Event/15拖拽效果.html)

[javascript\Event\16图片拖拽.html](javascript/Event/16图片拖拽.html)

[javascript\Event\17轨迹回放.html](javascript/Event/17轨迹回放.html)

# Cookie

网络知识

通信协议

通信规则，设备与设备之间通信时共同遵守的规则

**TCP/UDP/IP**

IP： Internet Protocol（网络之间互连的协议），规定了计算机在因特网上进行通信时应当遵守的规则

IP地址：4个字节, 一共32位 ，用来标识设备在网络中的位置

子网掩码: 255.255.255.0 确认两台计算机是否处于同一网段

TCP：Transmission Control Protocol（传输控制协议）

TCP面向连接的协议（通信之前必须先建立连接）

TCP相对可靠，它建立连接的过程称为3次握手

经历3次握手，才能建立连接

所有的消息传送，需要对方确认送达

UDP：面向数据报的协议 (不可靠的协议)，如果TCP比作是打电话，那么UDP就是在发短信

无需建立连接，发送消息也无需对方确认

无法保证数据的发送顺序，以及准确率

UDP通常用于视频、语音等通信（丢掉了一帧画面是无所谓的）

**http/https**: 超文本传输协议HyperText Transfer Protocol，基于TCP协议的一种高级协议, 用于客户端和服务器直接的通信

http的特点是，请求完成后就立即断开与服务器的连接

缺点：通信使用明文（不加密），内容可能会被窃听；不适用特定的Web服务器，如：聊天室，消息广播

**Socket**：是一种通信模式，客户端与服务端一直保持着连接，用于随时传输数据

cookie的概念

cookie 是客户端与服务器端进行通讯使用的一个能够在浏览器本地化存储的技术

PS：chrome不支持本地文件的cookie读写

cookie的应用场景

七天免登陆：7天内访问网站无需输入密码

[javascript\Cookie\02七天免登陆.html](javascript/Cookie/02七天免登陆.html)

购物车信息：添加到购物车后，取到购物车页面，商品信息依然存在

[javascript\Cookie\04购物车商品列表.html](javascript/Cookie/04购物车商品列表.html)

商品浏览记录：用户每浏览一个商品就会保留商品的浏览记录

cookie的设置和获取

**document.cookie = '**name=laoxie**';** **设置**

var cookies = document.cookie**;** **获取**

一次只能写入一个cookie

获取时得到所有cookie，多个cookie之间用分号+空格（'**;** ‘）来隔开

cookie的组成

cookie由键值对形式的文本组成，完整格式：

**document.cookie = name=value[;expires=date][;path=路径][;domain=域名][;secure];**

[ ]表示该值是可选

**name**=value 为你要保存的**键值对**(必选)

利用decodeURI/decodeURIComponent解码中文字符

利用json保存多条信息

**expires**=date 表示cookie的**失效时间**, 默认是浏览器关闭时失效(可选)

例 设置7天内生效的cookie

var date = new Date();

date.setDate(date.getDate() +7);

document.cookie = "user=laoxie;expires=" + date;

[javascript\Cookie\02七天免登陆.html](javascript/Cookie/02七天免登陆.html)

**path**=路径 访问**路径**, 默认为当前文件所在目录(可选)

cookie只能在设置的路径及它的子目录下使用

**domain**=域名 访问**域名**, 限制在该域名下访问(可选)

没有设置则默认为当前域名

**secure** **安全设置**, 如果设置了则必须使用https协议才可获取cookie(可选)

\* json字符串的格式要求：

\* 属性名必须使用双引号

\* 字符串必须使用双引号

\* json字符串中不能含有注释

\* 属性值只能是：字符串、数字、布尔值、对象、数组 (不能是函数)

\* **JSON.stringify(**obj**);** 把对象/数组**转换成json字符串**

\* **JSON.parse(**json**);** \* 把**json字符串转换成对象**

[**javascript\Cookie\04购物车商品列表.html**](javascript/Cookie/04购物车商品列表.html)

cookie的删除

利用设置过期时间达到删除的效果。

cookie的封装

封装cookie的增删改查

XSS攻击

跨域脚本攻击(Cross Site Scripting)

# 正则表达式

正则表达式(regular expression)是一个描述字符模式的对象。能够进行强大的“模式匹配”和“文本检索与替换”功能。前端往往有大量的表单数据校验的工作，采用正则表达式会使得数据校验的工作量大大减轻

创建正则表达式

使用RegExp构造函数

var reg= **new RegExp(' ','ig');**

第一个参数模式“字符串”

第二个参数可选，模式修饰符

**i**: case-insensitive，表示**忽略大小写**

**g**: global，表示**全局匹配**

**m**: multiline，表示**多行匹配**

用直接量方式直接声明

**var reg = / /gi;**

注意：**直接量**是字符匹配，**不支持变量**

正则表达式

支持正则表达式的字符串方法

**search** 返回**第一次匹配**时所在的**索引值**,如果匹配不到则返回-1

**match** 默认**匹配**字符串，**返回**一个**数组**

0:所匹配的字符

index:匹配第一个字符所在的索引

input:对字符串的引用

全局匹配(g)，返回一个匹配所有字符串数组

如果匹配不到则返回null

**replace** **替换**字符串

**split** **分割**(把字符串拆分成数组)

正则表达式的属性和方法

正则表达式**. test('**需测试的字符串**');** 用<正则表达式>**测试**<字符串>**是否匹配**,

返回true/false

正则表达式**. exec('**需测试的字符串**'); 返回第一个匹配**的项，没有则返回null但是当

连续调用exec时，则每次的返回值都是下一个匹配项

( 字符串**.match(**正则表达式**);** 字符串执行匹配正则表达式规则的方法

注意：exec和match返回的都是数组，**match会返回所有符合条件的匹配项**，并以数组形式返回 。test和exec是正则表达式的方法，match是字符串方法 )

匹配规则

[javascript\正则表达式\doc\正则表达式手册.htm](javascript/正则表达式/doc/正则表达式手册.htm)

所有字母和数字都是按照字面量进行匹配,和字符串匹配等效 如/good/gi

字符类（只记小写字母即可）

* **.** : 除换行以外的字符
* \w : 代表数字或字母或下划线
* \W : 非数字字母和下划线字符
* \d : 数字
* \D : 非数字
* \s : 代表一个空格
* \S : 空格以外的字符
* \b : 匹配一个单词边界，也就是指单词和空格间的位置(对中文无效)
* \B : 匹配非单词边界。

PS:以上所有字符类都只是匹配“一个”字符

特殊符号

^ $ . \* + ? = ! : | \ / () [] {}

* []: 代表任意“**单个字符**” ,**里面**的内容**表示**“**或**”的关系
  + -: 代表范围
  + ^: 代表非
* (): 表示**分组**（n是以最左边括号出现的顺序排列）
  + $1: 表示第一个分组
  + $n: 表示第n个分组（$不能写在正则表达式里）
  + \n: 在正则分组后面使用，表示对第n个分组的引用(一定要写在正则表达式里)

PS: 编写的正则分组数量越少越好，分组越多性能越差，(?: )表示为一个整体但不分组可优化性能

* | : 表示**或者**
* **锚点**定位
  + ^: 表示以什么开头
  + $: 表示以什么结尾
* 表示**数量**，对前一个字符计数，
  + \*: 代表0个或0个以上 <===>{0,}
  + +: 代表1个或1个以上 <===>{1,}
  + ?: 代表0个或1个 <===>{0,1}
  + {}:

\d{5}: 匹配5个数字

\d{5,10}: 匹配5个到10个数字

\d{5,}: 匹配5个或5个以上的数字

PS: 1）数量词**\***,**+**,**{5,}**，会**尽可能多**的去**匹配**结果（**贪婪**）

2）在(包括数量词)后面加一个**?**表示**尽可能少**的去**匹配**结果（**非贪婪**）

google,goooogle ==> /go+/

[javascript\正则表达式\02表单验证.html](javascript/正则表达式/02表单验证.html)

# ECMAScript5(ES5)

支持ES5的浏览器

Opera 11.60+

Internet Explorer 9+（IE9不支持严格模式）

Firefox 4+

Safari 5.1+

Chrome 13

加载事件

DOM文档加载的步骤：

1、解析HTML结构。

2、加载外部脚本和样式表文件。

3、解析并执行脚本代码。

4、DOM树构建完成。//DOMContentLoaded

5、加载图片等外部文件。

6、页面加载完毕。//window.onload

document事件

readystatechange事件

**document.onreadystatechange = function(){**

**if(document.readyState === 'complete'){**

**}**

**}**

DOMContentLoaded事件

**document.addEventListener('DOMContentLoaded'**,**function(){**

//DOM树构建完成后执行

**})**

PS:以上事件用以取代window.onload事件

[javascript\ES5&ES6\01延迟代码执行.html](javascript/ES5&ES6/01延迟代码执行.html)

ES5的严格模式(strict mode)

严格模式((更可靠，更安全)IE6-9不支持)：

消除javascript语法的一些不合理，不严谨的地方，减少一些怪异行为；

消除代码运行的一些不安全之处，保证代码运行的安全；

提高编译器效率，增加运行速度；

为未来新版本的javascript做好铺垫；

用法

**“use strict”;**

全局：针对整个js文件, 放在脚本文件的第一行，则整个脚本都将以”严格模式”运行

局部：针对单个函数, 放在函数体的第一行，则整个函数以”严格模式”运行。

执行限制

不使用var声明变量严格模式中将不通过

删除系统内置的属性会报错

delete不可删除属性的对象时报错，(如：var声明的全局变量（会自动称为window的属性）)

对象有重名的属性将报错， ( 如：var obj={name:"小王",name:'王大锤'} )

函数有重名的参数将报错 ( 如： function sum(a,a,b){} )

arguments严格定义为参数, 不允许对arguments赋值, 禁止使用arguments.callee

函数必须声明在顶层，不能写在条件判断语句或for循环语句中

如 var arr = [10,2,3,50];

if(arr.length>3){

function sum(){//报错

}

}

JSON对象方法

JSON.parse(text [, fnReviver]); 将json字符串转换成对象/数组

text：json字符串

fnReviver：带有key和value参数的函数，一般用于过滤和转换返回值。

如 var res = JSON.parse('{"a": 10, "b": 20,"c":15}', function(key, value){

if(value < 15){

return undefined;

}else{

return value\*value

}

});

[javascript\ES5&ES6\03JSON对象的方法.html](file:///D:\h5_1701\javascript\ES5&ES6\03JSON对象的方法.html)

JSON.stringify(value [, replacer [, space]]); 将数组/对象转换成标准的json字符串

replacer：带有key和value参数的函数，一般用于过滤和转换返回值

space参：数字，表明了json字符串缩进的空格数

获取元素节点

(document).querySelector(‘cssselector’); 获取匹配选择器的第一个元素节点，返回DOM节点

(document).querySelectorAll(‘cssselector’); 获取匹配选择器的所有元素，返回数组

Function方法

.bind(); 用于将当前函数和指定对象绑定，返回一个新的函数

兼容低版本浏览器写法：

if (!Function.prototype.bind) {

Function.prototype.bind = function(context) {

var self = this,

args = Array.prototype.slice.call(arguments);

return function() {

return self.apply(context, args.slice(1).concat(arguments));

}

};

}

[javascript\ES5&ES6\05bind方法.html](javascript/ES5&ES6/05bind方法.html)

获取class列表属性

classList js 的方法

元素.classList.**length;** class的长度

元素.classList.**add(‘’);** 添加class方法

元素.classList**.remove();** 删除class方法

元素.classList**.contains();** 判断是否存在class方法

元素.classList**.toggle();** 切换class方法(如果有，则删除,如果无，则添加

toggle = contains,add,remove的结合)

[javascript\ES5&ES6\06操作类名的方法.html](javascript/ES5&ES6/06操作类名的方法.html)

data自定义属性

dataset

dataset.name (即设置data-name)

dataset.nameFirst (即设置data-name-first)

[javascript\ES5&ES6\07自定义属性.html](javascript/ES5&ES6/07自定义属性.html)

# ES6

变量声明

let 代码块内的变量声明

1）变量声明不会提升

2）let不允许相同作用域内多次声明同一变量

3）块级作用域 (大括号内)

[javascript\ES5&ES6\09打印按钮索引值.html](javascript/ES5&ES6/09打印按钮索引值.html)

const 常量声明

1）声明后无法修改值 (声明常量约定：名称都为大写)

2）块级作用域 (大括号内)

3）const不允许相同作用域内多次声明同一变量

解构赋值

ES6允许对数组和对象中提取值，对变量进行赋值，这被叫做“解构”

数组

1）var [a,b,c] = [1,2,3] //a=1,b=2,c=3

2）var [a,[[b],c]] = [10,[[20],30]] //a=10,b=20,c=30

3）var [a,...b] = [1,2,3,4] //a=1,b=[2,3,4];...表示剩余参数

对象

1）var {a,b}={a:’html’,b:’css’}

2）变量必须与对象属性名相同，否则为undefined

3）如果变量名与属性名不相同，则必须写成以下格式才能取到值

var {a:test} ={a:’html’,b:’css’} //test=>html

解构失败

var [a]=[],[b]=1,[c]=’jiegou’,[d]=false //a,b,c,d都得到undefined

指定默认值

var [a=true]=[];

var {a=10} = {};

解构同样适用于let和const

PS:解构只能用于数组和对象，如果解构不成功，变量会返回undefined，但如果对undefined和null解构则会报错（因为其它原始类型的值都可以转为相应的对象。但是，undefined和null不能转为对象）

解构用途

交换变量值

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

函数返回多个值

function example(){

return [1,2,3]

}

//接收

var [x,y,z] = example();

定义函数形参

函数的参数定义方式, 不用再考虑参数的顺序

function test({x,y,z}){}

//传参

test({x:10,y:20,z:30})

//参数可以设置默认值

fuction test({x=10,y=20,z}){}

[javascript\ES5&ES6\10变量的解构.html](javascript/ES5&ES6/10变量的解构.html)

字符串扩展

字符串方法

**.includes();** **判断**是否**包含**某个**字符**，返回布尔值

如 'html5'.includes('html');//true

**.startsWith/endsWith(); 判断**是否以某一**字符开头**/**结尾**

如 let str='google';

str.startsWith('g'); //true

str.endsWith('le'); //true

**.repeat(n);** 得到**字符串重复**n次后的结果，n可以为小数，但不能为负数

如 'laoxie'.repeat(2);//laoxielaoxie

字符串模板, 使用反引号 ` 表示

通过一种更加美观、更加方便的方式向字符串中插入变量，而不用拼接字符串

模板字符串中所有的空格、新行、缩进，都会原样输出在生成的字符串中

格式 **`${变量|函数}`**

如 **`**你好，我的名字叫${username},接下来是我的自我介绍：${introduce()}**`**

数组扩展

静态方法

**Array.from();** 将**类数组**(array-like)**转**化为**数组**并返回

如 var list = Array.from(document.getElementsByTagName("li"));

**Array.of();** 将其参数转化为数组

如 Array.of(1, 2, 3); // [1, 2, 3]

原型方法

Array.prototype.copyWithin()

格式：arr**.copyWithin(**target, start [, end=arr.length]**);**

最后一个参数为可选参数，省略则为数组长度。该方法在数组内复制从start(包含start)位置到end(不包含end)位置的一组元素覆盖到以target为开始位置的地方

如 var arr = [1,2,3,4,5,6,7,8,9,0];

arr.copyWithin(2,6,8); //[1,2,7,8,5,6,7,8,9,0]

Array.prototype.find()

格式：array**.find(**callback**);**

返回数组中符合条件的第一个元素，如果没有则返回undefind

如 [1, 2, 3, 4].find(function(item, index, arr) {

return item > 2;

}) // 3

[1, 2, 3, 4].find(function(item, index, arr) {

　 　return item > 4;

}) // undefined

Array.prototype.findIndex()

findIndex()方法与find()方法用法类似，返回的是第一个符合条件的元素的索引，如果没有则返回-1

如 [1, 2, 3, 4].findIndex(function(el, index, arr) {

return el > 2;

}) // 2

[1, 2, 3, 4].findIndex(function(el, index, arr) {

　 　return el > 4;

}) // -1

遍历

**for..of**

for…of跟for-in的区别很明显，就是**直接取值**，而**不**再**取下标**了，

与forEach()不同的是，它可以正确响应break、continue和return语句

for-of循环不支持普通对象

for-of循环也可以遍历其它的集合：DOM节点、字符串、Set/Map集合

是最简洁、最直接的遍历数组元素的语法，且避开了for-in循环的所有缺陷

如 var arr = [10,12,18,30]

for (var value of arr) {

console.log(value);

}

[javascript\ES5&ES6\12循环语句.html](javascript/ES5&ES6/12循环语句.html)

对象扩展

Object.assign(obj1,obj2,…objN); 合并对象(可用于复制对象(为浅拷贝对象里的对象不会被拷贝)如果要深拷贝可以用JSON方法 如 var res = JSON.parse(JSON.stringify(obj)); )

只支持浅拷贝（对于引用类型，只拷贝引用），忽略不可枚举属性

如 Object.assign({a:1},{b:2},{b:4,c:3}); //{a:1,b:4,c:3}

箭头函数=>

省略了function、return关键字和大括号。

格式：标识符=>表达式

参数与返回值

箭头函数内部没有constructor方法，也没有prototype，所以不支持new操作

零个参数用 () 表示

//传统写法

var sum = function(){

return 10 + 10;

}

//es6箭头函数

var sum = **() =>** 10+10;

函数只有一个参数（可省略括号）

//传统函数写法

var test = function(x){

return x+2;

}

//使用箭头函数

var test = **x=>**x+2;

多个参数

// ES5

var total = values.reduce(function (a, b) {

return a + b;

}, 0);

// ES6

var total = values.reduce(**(a, b) =>** a + b, 0);

函数中包含多行代码时用代码块括起来

ES6中的规则是，紧随箭头的{被解析为块的开始，而不是对象的开始

// ES5

$("#confetti-btn").click(function (event) {

playTrumpet();

fireConfettiCannon();

});

// ES6

$("#confetti-btn").click(event => {

playTrumpet();

fireConfettiCannon();

});

使用了块语句的箭头函数不会自动返回值，需要**使用return**语句将所需值**返回**

当使用箭头函数返回一个普通对象时，需要将对象包裹在小括号里

//传统写法

var createPerson = function(){

return {name:'laoxie',age:18}

}

// ES6

var createPerson = ()=>{name:'laoxie',age:18}; // 这样写会报Bug！

var createPerson = ()=>({name:'laoxie',age:18});

默认参数

var func1 = (x = 1, y = 2) => x + y;

func1(); // 得到 3

剩余参数

var func2 = (x, ...args) => { console.log(args) };

func2(1,2,3); // 输出 [2, 3]

箭头函数中的this值

箭头函数没有它自己的this值，箭头函数内的this值继承自外围作用域

Symbol数据类型

ES6引入了一种新的原始数据类型Symbol，表示独一无二的值，一旦创建后就不可更改，且不能对它设置属性。

// 没有参数的情况

var s1 = Symbol();

var s2 = Symbol();

s1 === s2 // false

// 有参数的情况

var s1 = Symbol("foo");

var s2 = Symbol("foo");

s1 === s2 // false

Symbol函数前不能使用new命令，否则会报错

Symbol函数可以接受一个字符串作为参数，表示对Symbol实例的描述，主要是为了标识和区分，对调式非常有用

Symbol值不能与其他类型的值进行运算

用途

作为属性名的Symbol

var mySymbol = Symbol();

// 第一种写法

var a = {};

a[mySymbol] = 'Nani';

// 第二种写法（注意加方括号，否则回被当作普通属性）

var a = {

[mySymbol]: 'Nani'

};

// 第三种写法

var a = {};

Object.defineProperty(a, mySymbol, { value: 'Nani' });

// 以上写法都得到同样结果

a[mySymbol] // "Nani"

Symbol值作为对象属性名时，不能用点运算符。

不能用for…in、for…of遍历，也无法用Object.keys()、Object.getOwnPropertyNames()获取

常用方法

Symbol.for() 首先在全局中搜索已登记的Symbol值，如果有，就返回这个Symbol值，否则就新建并返回一个以该字符串为名称的Symbol值

直接用Symbol()方法创建的Symbol值不会被登记

let one = Symbol("laoxie");

let two = Symbol.for("laoxie");

//由于创建了两个Symbol值，所以他们不相等

console.log(one===two);//false

Symbol.keyFor() 获取被登记的Symbol值

直接使用Symbo()创建的Symbol值的键不会被登记，所以也就获取不到

用途

给对象创建私有属性，而且不用担心属性名冲突

Map集合（即映射）

let map = new Map();

map.set("S0", "张三");

map.set("S1", "李四");

map.set("S2", "王五");

获取某一个元素 ：

map.get("s2"); //王五

循环遍历，配合解构赋值

for(let [key,value] of map){

console.log(key,value);

}

方法

Map.prototype.keys() 获取所有键

Map.prototype.values() 获取所有值

Map.prototype.entries() 获取所有键值对，返回类数组

Set集合

本质上就是对数组的一种包装

let imgs = new Set();

imgs.add(1）;

imgs.add(1);

imgs.add(5);

imgs.add("5");

imgs.add(new String("abc")）;

imgs.add(new String("abc")）;

//打印的结果： 1 5 '5' 'abc' 'abc'

Set集合是默认去重复的，但前提是两个添加的元素严格相等，所以5和”5”不相等，两个new出来的字符串不相等

SET集合没有提供下标方式的访问，因此只能使用for来遍历。由于Set集合本质上还是一个map，因此会有以下几种奇怪的遍历方法

var imgs = new Set(['a','b','c']); //根据KEY遍历

for(let item of imgs.keys()){

console.log(item);

}

//a

//b

//c

//根据VALUE遍历

for(let item of imgs.values()){

console.log(item);

}

//a

//b

//c

//根据KEY-VALUE遍历

for(let item of imgs.entries()){

console.log(item);

}

//['a','a']

//['b','b']

//['c','c']

//普通for...of循环

for(let item of imgs){

console.log(item);

}

//a

//b

//c

# 动画Animation

运动原理

不断改变对象的属性产生动画的效果

运动分类

匀速运动

速度保持不变的运动

加速运动

速度不断增加的运动

减速运动

速度不断减小的运动

抛物线运动

水平方向速度不断减小，垂直方向速度不断增加

缓冲运动

一开始速度很快，然后慢下来，直到停止

圆周运动

我具体在以某点为圆心半径为r的圆周上的运动叫“圆周运动”

盒模型

offset

offsetTop: 当前元素离<定位父级>元素顶部的距离。

offsetLeft: 当前元素离<定位父级>元素左边的距离。

以上两个属性如果没定位的父级，则相对于根元素html的距离

offsetWidth: 当前元素的宽度（border + padding + content）

offsetHeight: 当前元素的高度（border + padding + content）

client

clientTop: 当前元素上边框的宽度（border:top）

clientLeft: 当前元素左边框的宽度（border:left）

clientWidth: 当前元素宽度（padding + content，不包括border）

clientHeight: 当前元素高度（padding + content，不包括border）

scroll

scrollTop: 当前元素<**垂直**滚动条>**滚动过的距离**

scrollLeft: 当前元素<**水平**滚动条>**滚动过的距离**

scrollWidth: 当前元素滚动**宽度**（不包括边框）

scrollHeight: 当前元素滚动**高度**（不包括边框）

PS: scrollWidth,scrollHeight浏览器间有兼容性问题

与滚动相关的window对象属性/方法：

scrollX,scrollY：获取滚动条**滚动过的距离**

(document.body.scrollTop、document.documentElement.scrollTop

都可以利用scrollTop获取页面滚动过的距离，但都有兼容性的问题，如火狐前者无效)

scroll(x,y)/scrollTo(x,y)：**设置**滚动条**滚动距离**

scrollIntoView(): 把当前元素滚动到可见区域

[javascript\动画\06返回顶部效果（缓冲运动）.html](javascript/动画/06返回顶部效果（缓冲运动）.html)

# 动画Animation2

链式运动

动画的排队效果，上一个动画完成后才执行下一个动画效果

无缝滚动

1、复制初始状态下可见区域内的图片并放到最后 (可见区域有几张就复制几张)

2、当复制的图片都滚动到可见区域时，立即把图片定位到初始状态

# 面向对象

描述一个对象

Cat: 猫

属性：fur

方法：eat; miaow

创建对象

直接量

**var** student **= {**id**:**10**,**name**:**'小明',age:18**};**

new关键字

**var** student **= new Object();**

student.id = 10;

student.name = '王铁锤';

student.age = 18;

缺点：使用同一个接口创建很多对象，会产生大量的重复代码

封装工厂函数

为了减少重复代码，对上述代码进行封装

自定义构造函数

function Student(name, age){

this.name = name;

this.age = age;

this.sayName = function(){alert(this.name);};

}

var s1 = new Student("王铁锤", 18);

构造函数与普通函数的区别

唯一区别：**调用方式不同**

任何函数，只要通过**new操作符**来调用，它就可以**作为构造函数**；

而任何构造函数，如果**不通过new** 操作符来调用，那它跟**普通函数**无区别。

约定：构造函数名首字母大写

this

JS的关键字, 代表了当前对象, 而当前对象是谁，由函数执行时所处的环境来决定

用**new**关键字执行：this指向生成的**实例**对象

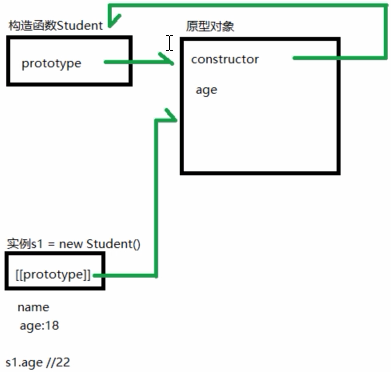
**普通**函数执行 ：this指向**调用**函数的对象

原型对象prototype

我们创建的每个函数都有一个prototype属性，这个属性值指向原型对象

原型对象默认包含一个constructor属性，指向构造函数

任何写在原型对象中的属性和方法都可以让所有对象实例共享



实例

用new关键字生成的对象称为实例，实例会复制构造函数内所有的属性和方法

当调用构造函数创建一个新实例后，该实例的内部将包含一个内部属性[[prototype]]，指向原型对象。这个内部属性可以通过以下方式获取（通过实例得到原型对象）：

在FF,Chrome等浏览器可以通过私有属性\_\_proto\_\_得到；

通过ES5方式去获取：Object.getPrototypeOf(实例)；

构造函数、原型对象和实例的关系

每个构造函数都有一个原型对象（prototype），

原型对象都包含一个指向构造函数的指针（constructor），

而实例都包含一个指向原型对象的内部指针（[[prototype]]）

判断原型和实例的关系（返回布尔值）

constructor: 得到构造函数的引用，一般用于判断该实例是否由某一构造函数生成

实例.constructor == Student //true

instanceof: 检测某个对象是不是某一构造函数的实例，适用于原型链中的所有实例

实例 instanceof Student //true

实例 instanceof Object //true

isPrototypeOf: 判断当前对象是否为实例的原型

原型对象.isPrototypeOf(实例) //true

实际应用

构造函数方法很好用，但是单独使用存在一个浪费内存的问题（所有的实例会复制所有构造函数中的属性/方法）。这样既不环保，也缺乏效率。

**解决方案:构造函数+原型对象**

使用构造函数添加私有属性

使用原型对象添加共享方法

优点：实例对象都有自己的独有属性

实例共享了原型中的方法，最大限度的节省了内存

支持向构造函数传递参数

面向对象其实只做两件事情

1、创建并描述对象

描述属性

描述方法（遵循单一原则：一个方法只做一件事情）

2、操作对象

指挥对象实现某种效果

ES5对象扩展

Object.create(prototype) 以指定的原型创建对象

Object.getPrototypeOf(object) 最后如果想要得到对象原型，可以用此方法

Object.defineProperty(object, propertyname, descriptor) 对指定的对象的一个属性设置属性特性

值属性

configurable 可配置性，控制着其描述的属性的修改，表示能否修改属性的特性

enumerable 可枚举性，表示能否通过for-in遍历得到属性

writable 可写性，表示能否修改属性的值

value 数据属性，表示属性的值。默认值为undefined

访问器属性

configurable

enumerable

get 在读取属性时调用的函数。只指定get则表示属性为只读属性。默认值为undefined

set 在写入属性时调用的函数。只指定set则表示属性为只写属性。默认值为undefined

Object.defineProperties(object, descriptors) 对指定的对象的一组属性设置属性特性

Object.getOwnPropertyDescriptor(object, propertyname) 返回属性特性

Object.keys(object) 返回对象所有可枚举属性的名称

Object.getOwnPropertyNames(object) 返回所有属性的名称（哪怕说是不能枚举的属性）

匿名函数

匿名函数与具名函数的区别

匿名函数自动执行

**(**function(){

alert("匿名函数的运行方式一");

}**)()**;

**(**function(){

alert("匿名函数的运行方式二");

}**())**;

使用()将匿名函数包围，然后再调用,这叫做匿名函数的自运行

PS: 把匿名函数变成一个表达式，就能实现匿名函数的调用

**-**function(){

console.log('333')

}**()**;

[javascript\面向对象\12匿名函数.html](javascript/面向对象/12匿名函数.html)

# 闭包

闭包是这样一种机制: 函数嵌套函数, 内部函数可以引用外部函数的参数和变量,参数和变量不会被垃圾回收机制所收回.

涉及到几个概念:

1、函数嵌套函数

2、作用域(全局变量和局部变量)

变量的访问规则

3、垃圾回收机制(garbage collection)

js内部不断扫描内存,并清理无引用对象(自动完成);

function aa(){

var num=100;

console.log('aa中访问num：',num);

function bb(){

console.log("bb中访问num：",num);

}

}

aa();

//bb(); //无法直接访问函数内部的函数

闭包的好处

1、可以让一个变量长期驻扎在内存当中不被释放

2、避免全局变量的污染, 和全局变量不同, 闭包中的变量无法被外部使用

3、私有成员的存在, 无法被外部调用, 只可以自己内部使用

结论：

1、闭包是指有权访问另一函数作用域中的变量的函数

2、闭包，可以访问函数内部的局部变量，并让其长期驻留内存

3、由于闭包会携带包含它的作用域(运行环境)，因此会比其他函数占用更多内存，过度使用闭包可能会造成性能问题。

# 原型链

实例与Object原型对象之间的链条称为原型链

原型模式的访问机制（原型搜索机制）

1. 读取实例对象的属性时，先从实例对象本身开始搜索。如果在实例中找到了这个属性，则返回该属性的值；
2. 如果没有找到，则继续搜索实例的原型对象，如果在原型对象中找到了这个属性，则返回该属性的值
3. 如果还是没找到，则向原型对象的原型对象查找，依此类推，直到Object的原型对象（最顶层对象）；
4. 如果再Object的原型对象中还搜索不到，则抛出错误；

重置原型对象

重置原型对象，可以一次性给原型对象添加多个方法

function Popover(){}

Popover.prototype = {

show:function(){},

hide:function(){}

}

注意覆盖、识别问题

内置原型对象

使用内置原型可以给已有构造函数添加方法

数组/字符串/数字等方法调用原理

扩展内置方法

对象属性的遍历与判断

for…in：遍历对象中的所有属性, 无论该属性存在于实例中还是原型中

in：只要通过对象能够访问到属性就返回true, 无论该属性存在于实例中还是原型中

if(name in s1){

}

实例.hasOwnProperty(属性)：检测一个属性是存在于实例中(true)，还是存在于原型中(false)

检测一个属性是否存在于原型中：!obj.hasOwnProperty(name) && (name in obj)

# 继承

继承是面向对象中一个非常重要的特征。指的是：子类继承父类的属性和方法。

可以通过继承的方式, 在父类的属性和方法基础上, 让子类也拥有这些属性和方法, 并可以扩展。

继承的特点:

1. 子类拥有父类所有的属性和方法（代码复用）；

2. 子类可以扩展自己的属性和方法（更灵活）；

3. 子类可以重写父类的方法

原型链继承 prototype

原型链属性(方法)的查找过程：实例–>原型对象–>原型对象… –> Object.prototype

PS:原型链继承法一般用于继承方法

借用构造函数

**call**：使用父类函数的call（子类对象,参数1，参数2，参数3......） 最常见

格式：构造函数.call(this,参数1,参数2,参数3......)

**apply**：使用父类函数的apply（子类对象，[参数1，参数2，参数3]）

格式：父类构造函数.apply(子类当前对象,[参数1,参数2,...])//第二个参数为数组

call与apply的唯一区别：传参方式不同

借用构造函数法只能继承构造函数里的属性/方法，不能继承原型对象里的方法

借用构造函数法一般用于继承属性

组合继承

继承属性：借用构造函数

继承方法：原型链继承

组合继承是最常用的继承模式。

缺点：多次执行父类构造函数，在原型对象中生成多余的属性

原型式继承

原理：先创建了一个临时性的构造函数，然后将传入的对象作为这个构造函数的原型，最后返回了这个临时类型的一个新实例

function object(o){

function F(){}

F.prototype = o;

return new F();

}

**Object.create()**：ES5版本的原型式继承

寄生组合继承法

继承属性：借用构造函数

继承方法：原型式继承

# ES6 Class

ES6提供了更接近传统语言的写法，引入了Class（类）这个概念，作为对象的模板。通过class关键字，可以定义类

class Person {

constructor(name,age) {

this.name = name;

this.age = age;

}

getInfo() {

return `我叫${this.name},今年${this.age}岁`;;

}

}

写在类里面的方法实际是给Person.prototype添加方法

constructor方法是类的默认方法，通过new命令生成对象实例时，自动调用该方法。如果没有constructor方法，则得使用默认的constractor方法

继承

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

}

class Man **extends** Person {

constructor(name, age, gender) {

//this.gender = gender; // 报错

**super**(name, age);

this.gender = gender; // 正确

}

}

子类继承了父类，在子类构造函数中必须调用super方法。

子类的constructor方法没有调用super之前，不能使用this关键字，否则报错，而放在super方法之后就是正确的。

静态方法 static

class Person {

constructor(){

this.name = 'laoxie',

this.age = 18;

}

**static** getInfo(){

return this.name

}

say(){

console.log(`Hello everyone, my name is ${this.name}, I'm ${this.age} years old`)

}

}

class Man extends Person {}

静态方法方法不会被实例继承，而是直接通过类来调用Person.getInfo()

父类的静态方法，可以被子类继承Man.getInfo()

# 循环跳转

break：//退出当前整个循环

只能在循环语句中使用

循环体中位于break后的语句不会被执行

在多层循环嵌套中，一个break语句只向外跳一层

continue：//跳过本次循环，继续下一次循环

只能在循环语句中使用，

结束本次循环（即跳过循环体中下面尚未执行的代码），接着执行下次循环。

label：给循环代码添加标识

break和continue后如果带标识，则跳到标识所在循环

例 *outer:*for(var i=0;i<10;i++){

inner:for(var j=0;j<10;j++){

times++;

console.log(i,j);

if(i===j){

continue *outer*;

}

}

}

# 数据类型转换

基本数据类型转换：利用内置函数进行转换

| **值(a)** | **转换为** |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 字符串String(a) | 数字Number(a) | 布尔值Boolean(a) |
| undefined | => | ‘undefined’ | NaN | false |
| null | => | ‘null’ | 0 | false |
| true | => | ‘true’ | 1 |  |
| false | => | ‘false’ | 0 |  |
| ”“ | => |  | 0 | false |
| “1.2” | => |  | 1 | true |
| “one” | => |  | 1 | true |
| 0 | => | “0” |  | false |
| -0 | => | “0” |  | false |
| NaN | => | “NaN” |  | false |
| Infinity | => | “Infinity” |  | true |
| -Infinity | => | “-Infinity” |  | true |
| 1 | => | “1” |  | true |

强制类型转换

Number();

String();

Boolean();

## 隐式转换

如果运算不能进行下去，内部就会尝试进行数据类型的转换

支持隐式转换的运算：逻辑运算、关系运算、数学运算

===、恒等于/全等于，比较的时候要求值和类型都相等（不会进行类型隐式转换）

!==、不全等于

# 关系运算符比较规则:

1. 数字和数字比较, 直接比较大小

2. 数字和字符串比较, 字符串转换为数字后再比较

3. 字符串和字符串比较, 进行字符的ASCII码值比较

# 一元运算

a++ 自增1 返回值是原本的值 先运算再自增

++a 自增1 返回值是自增后的值 先自增再运算

a--、--a 同上

# 三元运算

格式：条件 ? 条件成立代码 : 条件不成立代码

var a=20;

var b = 50;

var sum = a>b ? a-b : a+b;

# 进制

二进制

var num = **0b**10101101;

八进制

var num = **0o**0123;

十进制

十六进制

var num = **0x**ef9302;

**.toString(**2/8/16**);** 十进制转换成其他进制

例 var res = (88).toString(2); //转成2进制

**parseInt(**num,2/8/16**);** 其他进制转换成十进制

例 var res = parseInt(255, 16); //十六进制转十进制

其他转其他：先用parseInt转成十进制再用toString转到目标进制

# Switch语句

switch(表达式/值) {

case 表达式/value1:

//如果表达式的值**恒等于**表达式/value1，代码从这里开始执行

break;

default:

如果以上条件都不成立，默认执行这里的代码

}

假如default不在末尾，则要加上break，否则一直向下执行

do {

//不管条件是否成立，先执行一次这里的代码，再进行条件判断，如果条件依然成立，则再次执行这里的代码，依此类推

//所以这里一般会伴随着条件的更新

} while(条件)**;**

(结尾的分号)