JavaScript高级程序设计笔记

## 1、JavaScript的组成

核心（ECMAScript），文档对象模型（DOM），浏览器对象模型（BOM）。

## 2、DOM

1）、组成：DOM核心（DOM Core）和DOM HTML，

## 3、JavaScript中DOm对象

## 4、位移运算符

### （一）>>：算术右移

将左侧数据的二进制值向右移动右侧数值的位数，右侧空白由0补充。例：14=00001110>>2，=》000011=3

### （二）<<：算术左移

将左侧数据的二进制值向左移动右侧数值的位数，忽略被移出的位(向右添加右侧右侧数值的补0位数)。例：14=00001110<<2，=》00111000=56

### （三）>>>：逻辑右移

将左侧数据的二进制值向右移动右侧数值的位数，忽略被移出的位，左侧空位补0。14=00001110>>2，=》00000011=3

## 5、DOM操作

### （一）js中的CSS选择器

A、document.querySelector()//返回第一个匹配到的元素。B、document.querySelectorAll()//返回的所有匹配到的元素。显示B方法中的结果，以数组方式。例：query= document.querySelectorAll('div')；query[0]显示第一条内容。

### （二）编辑文本节点

创建文本节点：document.createTextNode(),通过appendChild()追加显示文本内容。

例：var txtNode=document.createTextNode("新建的文本节点");

document.getElementById("div1").appendChild(txtNode);

### （三）编辑元素节点

创建元素节点：document.createElement("p");通过innerText或textCount添加元素的文本内容。由appendChild()追加新的元素节点。

var ele = document.createElement("option");

ele.innerText = "测试添加option节点";

### （四）插入节点

appendChild()，将一个节点插入到调用节点的最后面；

insertBefore(param1,param2)，param1表示新插入的节点，param2表示插入到指定某节点前面，如果不添加参数2，则默认在调用节点的最后面。

如果是获取了页面上的另一元素，则可以直接将元素（param1）移动到指定位置（param2）前面。

var node = document.getElementById("nodeTest");

var newNode2 = document.getElementById("tel");

node.appendChild(newNode);

node.insertBefore(newNode2,node.firstChild);

### （五）删除和替换节点

removeChild()，replaceChild(param1,parm2)。

### （六）JS中操作CSS元素

通过style属性读取及设置

document.getElementById('p1').style.color='pink';

### （七）Attr节点的属性

attribute属性，非element对象返回null，element一般返回Attr对象。Attr对象时一个比较特殊的节点，通过name和value获取属性名称和值。

例：

<div id="div1">

<img src="1big.jpg" alt="我是一张图片" class="imgClass" id="img1" onclick="fun1()" />

</div>

document.getElementById("img1").attributes[0].name；

//输出 onclick 注意，通过索引器访问是写在右面在排前面，从0开始

document.getElementById("img1").attributes.src.value；//输出1big.jpg

## 6、JS中Array数组中的排序方法

### （一）sort()/sort(fucOrder)

带参数和不带参数的排序不一样。

不带参数：不指定排列顺序，JavaScript 脚本将数组元素转化为字符串，然后按照字母顺序进行排序，数字则是按照首字的数字排序（例：[1,34,132,245]，排序后：[1,132,245,34]）

带参数：按照函数方法计算排序，该函数中必须包含（a，b）两个值进行比较，也必须返回一个值以表示两个参数之间的关系。若a在b之前出现(a<b),则返回小于零的值（-1）；若a在b之后出现(a>b),则返回大于零的值(1)；若a，b相等，则返回零。

### （二）sort(fucOrder):

fucOrder表示函数的名称，不需要传递参数，但是fucOrder函数中必须有参数进行比较，

例：function fucOrder（a,b）{

var temp;

if (a > b ){ temp=1;}

else if( a < b ){ temp=-1;}

else { temp=0;}

return temp;

}

或者：

function compare(v1,v2){

return v2-v1; //降序，v1-v2升序

}

### （三）reverse()

将按照数组的索引号顺序将数组中的元素完全颠倒。

计算输出九九乘法表格式：

function test(){

for (var m = 1; m < 10; m++) {

var array = new Array();//每循环一次就新建一个数组

for (var n = 1; n <= m; n++) {

var result;

var product = m \* n;

result = (n + "\*" + m + "=" + product);

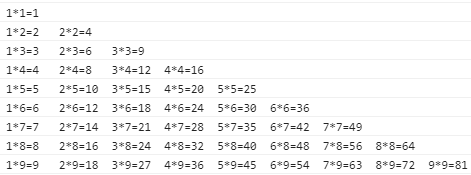
array.push(result);

}

console.log(array.toString().replace(/,+/g,"\t"));//全局多次匹配+（加号），用空格替换

}

}



## 7、堆栈和队列操作方法

### （一）堆栈(LIFO)

“后进先出”顺序填充数据。

符合堆栈的添加和删除数据的方法是：push()和pop()。从数组的最后一个元素操作。push()末尾添加元素，pop()末尾删除元素。

### （二）队列(FIFO)

“先进先出”顺序填充数据

符合队列的添加和删除数据的方法是：shift()和unshift()。从数组的第一个元素操作。unshift()首位添加元素，shift()首位删除元素。

## 8、splice(start,delete,arg3,…,argN)方法

添加和删除数组元素，各参数解析：start=开始删除的数据下标，delete=要删除的数据个数，arg表示要添加的新数据，新数据的位置从start开始，旧数据往后顺移。

（1）splice(1,0)：当delete为0时，不进行任何删除操作。

（2）splice(1,1)：[3,43,15,74]，操作之后[3,15,74],delete非0的时候，删除的数据下标是从start到start+delete。（删除）

（3）splice(1,2,10, 20,30):[3,43,15,74],操作后：[3,10,20,30,15,74]。（替换）

（4）splice(1,0,11,22,33)：当delete=0时不进行操作，则本次为插入新数据的操作，从下标为1的位置开始插入。原数组：[3,43,15,74]，结果:[3,11,22,33,43,15,74]。（插入）

## 9、slice(start,end)

基于当前数组中的一或多个项创建一个新数组，不影响原来的数组。从下标为start到下标为end的之间的元素被取出新建一个数组。

例：var colors = ['red', 'green', 'blue', 'yellow', 'purple'];

colors.slice(1, 3); //从下标1到下标3之间的元素数组:['green', 'blue', 'yellow']

colors.slice(1); //返回从下标1到数组结束位置的数组:['green', 'blue', 'yellow', 'purple']

colors.slice(1, 0); //end的位置比start位置小，则返回空数组:[]

## 10、join()/join(stirng)方法，将数组转换为字符串

（1）join():直接将数组转换成字符串。默认用逗号隔开。

（2）join('-')：根据分隔符将数组转换成字符串。

## 11、split()：分割字符串，返回字符数组

可选两个参数，第一个参数是分隔符号，第二参数可选，表示返回数组的最大长度。

例：var str='hello wonderful word';

str.split('');//分割符号并没有，所以返回的是将每个单词的字符单独分割:'h,e,l,l,o.......'

str.split(' ');//分隔符号是' '空格，根据空格分割字符串：‘hello,wonderful,word’

str.split(' ',2);//分隔符号是' '空格，根据空格分割字符串：‘hello,wonderful’

## 12、contact()

拼接两个数据成新的数组，不影响原来的数组。

## 13、关于setTimeout()和setInterval()的区别

### （一）setTimeout(表达式，延迟时间)

设定事件超时控制，只会运行一次表达式。实现原理是setTimeOut只能保证在指定时间之后将其放入队列中，并不能保证是么时候执行，而执行JS是在线程空闲时，自行从队列中取出并执行。

### （二）setInterval（表达式，交互时间）

设定时间间隔，每隔设置的时间则会运行一次表达式，实现周期往复触发某个事件。

也可将setTimeout()包含于被执行的函数中（表达式中），达到循环运行的效果：

例：setTimeout("testTime()", 1000);

var i = 0;

function testTime() {

i++;

setTimeout("testTime()", 1000);

//alert("执行次数：" + i);

}

setInterval("testTime()", 1000);

自动切换内容或手动切换内容实例：

//自动显示flash动画

var playIndex = -1;

function autoShowFlash() {

//定时器

timer = setInterval("showFlash()", 5000);

$('.cont3 .playLeft .playIntro .playContains').hover(

function () {//第一个函数表示鼠标悬停时调用的函数

clearInterval(timer);

}, function () {//第二个函数表示鼠标离开时调用的函数

timer = setInterval("showFlash()", 5000);

});

}

function showFlash() {

var playList = new Array("ssq", "k3", "ggl", "zfzx", "fc3D");

if (playIndex >= -1 && playIndex < playList.length) {

playIndex++;

if (playIndex >= playList.length) {

playIndex = 0;

}

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

if (playList[playIndex] == playList[i]) {

$(".cont3 .playLeft .playIntro .p\_" + playList[i]).addClass("currentPlay");

$("#" + playList[playIndex]).show();

} else {

$(".cont3 .playLeft .playIntro .p\_" + playList[i]).removeClass("currentPlay");

$("#" + playList[i]).hide();

};

}

}

}

## 14、Math对象的常见方法

（1） Math.floor(num)：返回小于等于num的最大整数；（向下舍入:2.6=>2）

（2）Math.random()：返回0至1间的随机数；

（3）Math.round(num)：返回最接近num的整数；（标准舍入:2.6=>3）

（4）Math.abs(num)：返回num的绝对值。

（5）Math.ceil(num)：返回大于等于num的最大整数;（向上舍入:2.6=>3）

## 15、迭代方法

### （一）every()

返回Boolean值，对数组中的每一项运行给定函数，如果该函数对每一项的返回值都为true，则返回true；（判断）

例：num.every(function (item,index,array){ return 函数});item表示每一项，return 后面的函数表示计算函数，如：item>3,数组array中每一项都大于3，则返回true。

### （二）filter()

返回该函数返回值为true的项组成的数组。（获取满足条件的结果集）

例：num.filter(function (item,index,array){ return 函数});整体返回一个数组。如：[2,4,5,7,1]，函数：item>3，返回的数组：[4,5,7]。

### （三）map()

返回在原数组的对应项运行传入函数的结果，也是一个数组。

例：[3,5,6,10]，每一项乘以2。

num.map(function (it,id,array){ return it \* 2})

### （四）forEach()

循环，与for（）循环一样。

## 16、累计方法

### （一）reduce()

reduce(function (pre,cur,id,array){ return pre \* cur})，循环遍历array数组，最终返回一个值，从第一项开始。

### （二）reduceRight()

方法同上，只是开始是从最后一项，然后倒数第二项......。

例：var test = [2,3,4,5], test.reduce(function (pre,cur,id,array){

return pre + cur});返回值：14

## 17、RegExp()类型（正则表达式）

var expression=/pattern/flags

pattern表示任意的正则表达式，flags表示标志，说明匹配的范围，g表示全局匹配，i表示忽略大小写，m表示多行匹配。

\w：表示[a-zA-Z0-9\_]

元字符：{}(重复计算次数/位数，{n}:n次，{n,}：n次或多次，{n，m}：n到m次)，[](中间添加限制条件，添加^符号，表示非），()(表示将各个限制条件分开设置)，\*(重复零次或多次)，+(重复一次或多次)，?(重复零次或一次)，^(匹配行的开始)，$(匹配行的末尾)，.(匹配除换行符的任意字符)，\(转义字符)



### （一）、实例方法

#### 1、test()

返回 Boolean，查找对应的字符串中是否存在模式。  
var str = "1a1b1c";  
var reg = new RegExp("1.", "");  
alert(reg.test(str)); // true

#### 2、exec()

查找并返回当前的匹配结果，并以数组形式返回。返回的数组包含三个特殊属性：index和input。index表示匹配项在字符串中的位置，input表示应用正则表达式的字符串，lastIndex 当前匹配项结束的位置（index + 当前匹配项的长度）。exec()是RegExp的对象实例方法如果不存在模式，则返回值为null，否则返回的数组总是一个长度为1的数组，其值就是当前的匹配项。

var str = "1a1b1c";

var reg = new RegExp("1.+", "g");// 多次全局匹配除换行符以外的包含1的字符串

var array= reg.exec(str);

alert(array); // 1a1b1c

#### 3、match()

String对象的一个方法

var str = "1a1b1c";

var reg = new RegExp("1.", "g");

alert(str.match(reg));// 1a，1b，1c

### （二）、关键正则表达式的验证实例

#### 1、验证密码强度

密码的强度必须包含大小写字母和数字的组合，不能使用特殊字符，长度在8-10之间。

/^([^!%@#$^&\*()=|{}':;',\[\].<>/?~！@#￥……&\*（）——|{}【】‘；：”“'。，、？])(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,10}$/;

说明：([^!%@#$^&\*()=|{}':;',\[\].<>/?~！@#￥……&\*（）——|{}【】‘；：”“'。，、？])这一串表明不能添加[]中的特殊字符，[]中加入^后可表示非条件。如：[^a-z]:限制在非a-z范围内。

#### 2、验证中文

/^[\u4e00-\u9fa5]{0,}$/。

部分展示条件为/^[\\u4e00-\\u9fa5]{0,}$/，需要将转义字符 '\' 去掉。

#### 3、验证手机号码

/^(13[0-9]|14[5|7]|15[0|1|2|3|5|6|7|8|9]|18[0|1|2|3|5|6|7|8|9])\d{8}$/；145是联通号码，147是移动号码

#### 4、验证金额，保留两位小数

/^[0,9]+(.[0-9]{2})$/

#### 5、验证纯数字

/^[0-9]\*$/或者/^\d\*$/

#### 6、验证邮箱

/^[\w!#$%&'\*+/=?^\_`{|}~-]+(?:\.[\w!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[\w](?:[\w-]\*[\w])?\.)+[\w](?:[\w-]\*[\w])?$/

#### 7、验证字母，数字，下划线(固定只能是字母开头)

/^[a-zA-Z][a-zA-Z0-9\_]{4,15}$/或者/^\w{4,15}$/

## 18、Global对象

eval()：类似一个ECMAScript的解析器，接收的参数只有一个，即要执行的ECMAScript（或JavaScript）字符创。

例：eval('function sum(a,b){ return a \* b}')

sum(2,3);//输出结果6

## 19、Function对象

### （一）apply(func,array)方法

劫持另一个对象的方法，继承另一个对象的属性。

1. \*定义一个人类\*/
2. **function** Person(name,age) {
3. **this**.name=name;
4. **this**.age=age;
5. **this**.grade=grade;
6. }
7. /\*定义一个学生类\*/
8. **function** Student(name,age,grade) {
9. Person.apply(**this**,arguments);
10. **this**.grade=grade;
11. }
12. //创建一个学生类
13. **var** student=**new** Student("qian",21,"一年级");
14. //测试
15. alert("name:"+student.name+"\n"+"age:"+student.age+"\n"+"grade:"+student.grade);
16. //大家可以看到测试结果name:qian age:21 grade:一年级
18. 分析：Person.apply(**this**,arguments);
20. **this**:在创建对象在这个时候代表的是student
22. arguments:是一个数组,也就是[“qian”,”21”,”一年级”];
24. 也就是通俗一点讲就是:用student去执行Person这个类里面的内容,在Person这个类里面存在**this**.name等之类的语句,这样就将属性创建到了student对象里面

### （二）call(func,param1,param2...)

方法跟apply差不多，只是传参不一样。

1. 在Studen函数里面可以将apply中修改成如下:
3. Person.call(**this**,name,age);

（3）求取数组中的最大值和最小值：Math.max.apply(null,arrayList)

Math.min.apply(null,arrayList)。

## 20、构造函数和普通函数的区别

（任何函数，只要通过new操作符调用的函数就是构造函数）

（1）构造函数：a、使用new关键字调用；b、函数内部使用this关键字；c、不需用return返回值类型，默认返回this；d、函数名建议首字母大写。

（2）普通函数：a、不需要new关键字，可直接调用函数名；b、不使用this关键，普通函数中的this相当于window；c、必须有return返回值类型；d、函数名一般是首字母小写。

例：构造函数：function Person(name, age, sex, num) {

this.name = name;

this.age = age;

this.sex = sex;

this.num = num;

}

var student = new Person('Mark', 23, '男', 2017010);

alert(student.name);//返回Mark

普通函数：function personInfo(name, age, sex, num) {

alert("学生信息：\n姓名："+name+"，年龄："+age+"，性别："+sex+"，学号："+num);

}

personInfo('Mark', 23, '男', 2017010);

## 21、JavaScript中的instanceof

instanceof 运算符是用来在运行时指出对象是否是特定类的一个实例。instanceof通过返回一个布尔值来指出，这个对象是否是这个特定类或者是它的子类的一个实例。

## 22、迭代方法和循环的区别

表示“重复”的词有很多，如循环（Loop），递归（recursion），遍历（traversal），迭代（iterate）。循环是最基础的概念，凡是重复执行的一段代码都可以称之为循环，大部分的递归，遍历，迭代都是循环。

循环（loop）：在满足条件的情况下，重复执行一段代码（比如while语句）；

递归（recursion）：一个函数不断调用自身的行为（如：斐波那契数列）；

遍历（traversal）：按照一定的规则访问树形结构（非线性）中的每个节点，而且每个节点只访问一次；

迭代（iterate）：按照某种顺序逐个访问列表（线性）中的每一项（如for语句）；

## 23、闭包

### （一）、原理

官方解释：一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。可理解为定义在函数内部的函数。

特点：（1）、作为函数变量的一个引用，当函数返回时，其处于激活状态。

（2）、一个闭包就是当一个函数返回时，一个没有释放资源的栈区。

简单的说，JavaScript允许使用内部函数--即函数定义和函数表达式位于另一个函数的函数体内（函数中嵌套函数）。而且，这些函数可以访问其所在的外部函数中声明的所有声明、局部变量和参数，当这个包含这些内部函数的外部函数被调用时，就会形成闭包。

### （二）、关于闭包

闭包主要涉及js的几个其他的特性：作用域链，垃圾（内存）回收机制，函数嵌套等。

#### 1、作用域及作用域链：

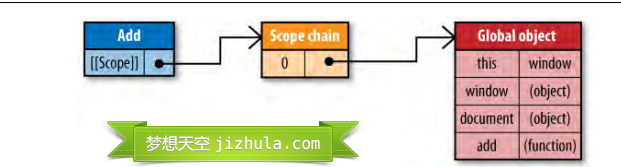
1、作用域：是变量及函数的可访问范围，控制着变量及函数的可见性及生命周期。可分为全局作用域及局部作用域。

全局作用域：A、最外层函数和最外层函数外面定义的变量。B、未定义直接赋值的变量自动声明为全局变量。C、所有window对象的属性。

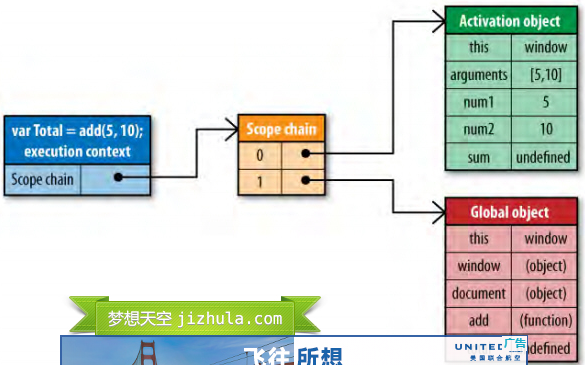
局部作用域：函数内部。

2、作用域链：JavaScript中的一切都是对象，函数对象也拥有可通过代码访问的属性以及可通过JavaScript引擎访问的一系列内部属性，其中[Scope]属性内部包含了函数被创建的作用域中对象的集合，这个集合被称为函数的作用域链，它决定了哪些数据能被函数访问。

函数创建时会创建函数的作用域，它的作用域链会被该函数作用域中可访问的数据对象填充（全局对象包含的全局变量）。



该函数在执行时会创建“运行期上下文”的内部对象，它会形成自己的作用域链，初始化为当前运行函数所包含的对象。它们会被按照在函数中出现的顺序复制到“运行期上下文”作用域链中，组成一个新的“活动对象”，当运行期上下文销毁后，这些活动对象也会销毁。



#### 2、垃圾（内存）回收机制：

其原理就是找出不再使用的变量，释放其占用的内存。垃圾回收机制有两种：标记清除和引用计数。

##### （1）、标记清除：

垃圾回收器在运行的时候会给存储在内存中的所有变量都加上标记（当然，可以使用任何标记方式）。然后，它会去掉环境中的变量以及被环境中的变量引用的变量的标记（闭包）。而在此之后再被加上标记的变量将被视为准备删除的变量，原因是环境中的变量已经无法访问到这些变量了。最后，垃圾回收器完成内存清除工作，销毁那些带标记的值并回收它们所占用的内存空间。

function test(){

var a = 10 ; //被标记，进入环境

var b = 20 ; //被标记，进入环境

}

test(); //执行完毕之后 a、b又被标离开环境，被回收。

##### （2）、引用计数：

跟踪记录每个值被引用的次数。当声明一个变量（a），并将一个引用类型值赋值给该变量时(b=a)，该值的引用次数加一；如果将同一个值再次赋值给另一个变量（c=a），该值的引用次数又加一(a有两次引用)；如果引用该值的一个变量重新获取另一个值，该值的引用次数减一（a有一次引用），直到a的引用次数为零，则可被回收。

但是引用计数容易造成循环引用（如互相引用，引用次数永远不可能为0），导致内存泄漏，这时需要手动解除循环引用，将变量设置为null。

##### （3）、减少垃圾回收：

每次使用new关键字就意味着一次内存分配，最好的处理方法就是：在初始化的时候新建对象，在以后的运用中多次重用这些对象。另外还有以下三种内存分配表达式：{}（创建一个新对象），[]（创建一个新数组）,function (){...}（创建一个新方法，新建方法也会导致垃圾收集）。

1、对象Object优化：

避免使用new关键字来新建对象。最好的方法是将需要返回的数据放入一个全局对象中，并返回此全局对象。这意味着全局对象的值每次都会发生改变，可能导致错误，因此，需要对全局变量进行注释和说明。

有一种方式是保证对象（确保对象prototype上没有属性，prototype是属于函数特有的属性，对象没有该属性，\_\_proto\_\_([[protottype]])属于隐性属性，在对象中存在，表示指向创建这个对象的函数的显示原型，即prototype。）的重复利用，遍历对象的所有属性，再将其清空成为空对象。

// 删除obj对象的所有属性，高效的将obj转化为一个崭新的对象！

cr.wipe = function (obj) {

for (var p in obj) {

if (obj.hasOwnProperty(p))

delete obj[p];

}

};

2、数组Array优化：

将[]赋值给一个数组对象，可将数组对象清空，但这又是创建了一个新的空对象，将原来的数组对象变成一小片内存垃圾。实际上将数组长度赋值为0（arr.length=0）也能达到清空数组的目的，同时也能实现数组重用，减少内存垃圾。

3、方法Function优化：

动态创建方法的地方，就有可能产生垃圾，例如:将方法作为返回值，就是动态创建方法。为解决这个问题，可以先将作为返回值的方法保存起来，运行的时候再调用，达到了重用相同方法的目的。

### （三）、写法及用法

//写法一：直接编写匿名函数形成闭包

function test() {

var name = "Jack";

return (function () {

//contain = "My Name Is " + name;

return document.getElementById('closures').innerHTML += "<div>方法一：匿名函数强制闭包：My Name Is " + name + "</div>";

})();

}

test();

//写法二：利用原型函数

function test2(obj) {

this.obj=obj;

}

test2.prototype.name = function () {

return document.getElementById('closures').innerHTML += "<div>方法二：原型函数直接返回值：My Name Is " + this.obj + "</div>"

}

var t = new test2("Rose");

t.name();

//方法三：声明变量，将函数当做值赋给变量

var test3 = new Object();

test3.info = function (n) {

return document.getElementById('closures').innerHTML += "<div>方法三：声明变量，将函数赋值给变量：My Name Is " + n + "</div>"

}

test3.info("Mark");

//声明空对象，直接返回

var info = {

name: function (n) {

return document.getElementById('closures').innerHTML += "<div>方法四：声明空对象，直接返回给属性：My Name Is " + n + "</div>"

}

}

info.name("Jerry");

//区分prototype与非prototype的调用

var info = function () {

};

info.Show = function () {

document.getElementById('closures').innerHTML += "<div>非prototype方法：静态方法，只能直接类名调用，不能使用this。</div>";

};

info.prototype.Display = function () {

document.getElementById('closures').innerHTML += "<div>prototype方法：非静态方法，只能实例化后调用，可以使用this引用对象自身其他属性。</div>";

};

var d = new info();

d.Display();

info.Show();

info.Show = function () {

document.getElementById('closures').innerHTML += "<div>非prototype方法：静态方法，只能直接类名调用，不能使用this。</div>";

}();//加()可立即执行函数方法

info.prototype.Display = function () {

document.getElementById('closures').innerHTML += "<div>prototype方法：非静态方法，只能实例化后调用，可以使用this引用对象自身其他属性。</div>";

}();//加()可立即执行函数方法

## 24、模拟块级作用域

JavaScript中没有块级作用域，只支持函数作用域。

### （一）块级作用域

任何一对花括号中的语句集都属于一个块，在这之中定义的变量在代码块之外都不可见，称之为块级作用域。在拥有块级作用域的语言中，块级作用域中的变量是不可访问的，而JavaScript不支持块级作用域，只支持函数作用域，所以只要变量属于函数内，花括号（{}）中的变量都可访问。

//C语言

#include <stdio.h>

void main()

{

int i=2;

i--;

if(i)

{

int j=3;

}

printf("%d/n",j);

}//会报错：j未定义，因为j属于if里面的块级作用域的变量，代码块之外无法访问

//js语言

function t5() {

if (true) {

var color = "blue";//使用let可改为局部变量

}

console.log(color);

}

t5();//返回"blue"

### （二）模拟块级作用域

#### 1、匿名函数

在一个函数中定义的变量，当函数调用完后变量会被销毁。故在函数内添加匿名函数可实现块级作用域的功能。

function test6() {

var a = 0;

(function () {

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

var b = a + i;

}

})();//(function(){})()，函数方法外添加括号表示一个块级，后面添加()，表示立即执行这个匿名函数

alert(b);//返回b未定义

}

#### 2、let操作符

在块级{}里面用let定义变量，离开块级{}之后就不能使用变量，模拟实现块级作用域。

function test7() {

var a = 0;

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

let b2 = a + i;

}

alert(b2);//返回b未定义}

## 25、BOM

### （一）window对象

window对象是BOM的核心对象，它扮演着双重角色，既是通过JavaScript访问浏览器窗口的一个接口，又是ECMAScript规定的Global对象。

#### 1、访问框架

window访问frame框架时需要注意每个frame框架都有自己的window对象，并且这些对象保存在frames集合或只能怪，可通过数值索引/框架名称进行访问。

window访问frame框架有以下几个方法：（i表示索引值，name表示框架名称）

window.frames[i]； window.frames[name] ; top.frames[i];

top.frames[name]; frames[i]; frames[name]。

top始终指向最高（最外）层的框架，也就是浏览器窗口。parent指向当前框架的上层框架。self始终指向window，可互换使用。

#### 2、获取位置

不同浏览器对获取位置的方法不同。获取窗口相对屏幕左边和上边的位置，不同浏览器支持不同方法，在IE，Chrome，Opera，Safari中通过screenLeft和screenTop获取位置信息，在FireFox中通过screenX和screenY获取位置信息。

所以在获取信息之前需要判断引用哪种方法。

var leftPos = (typeof window.screenLeft == 'number') ? window.screenLeft : window.screenX;

var topPos = (typeof window.screenTop == 'number') ? window.screenTop : window.screenY;

console.log(" window.screenLeft是什么类型：" + typeof window.screenLeft );//undefined类型

#### 3、获取窗口大小

##### （1）PC端：

可视窗口宽高：

document.documentElement.clientWidth/clientHeight；

window.innerWidth/innerHeight（chrome下包含滚动条）

网页全文实际宽高：

document.documentElement.scrollWidth/scrollHeight（不含滚动条宽度）；

document.documentElement.offsetWidth/offsetHeight(包含滚动条宽度)；

//验证IE9以下（不含IE9）浏览器

var windowW = window.innerWidth;//获取可视窗口宽度，chrome下包含滚动条

var windowH = window.innerHeight;//获取可视窗口高度

var flag = typeof windowW; //获取数据类型，IE9以下不支持innerWidth

if (flag != 'number') {

以下if语句在IE6中有效：

//document.documentElement.clientWidth;正确获取值，body获取的是实际网页内容高度，不包含滚动条的宽度和body，html的margin

if (document.compatMode == 'CSSICompat') {

windowW = document.documentElement.clientWidth; //可视窗口宽度，不包含滚动条宽度

windowH = document.documentElement.clientHeight; //可视窗口高度，不包含滚动条高度

document.getElementById('closures').innerHTML += "<div>IE9以下(不含IE9)标准模式下的可视窗口宽高：" + windowW + "，" + windowH + "</div>";

} else {

windowW = document.body.clientWidth; //实际网页内容高度，不包含滚动条的宽度和body，html的margin

windowH = document.body.clientHeight;

document.getElementById('closures').innerHTML += "<div>IE9以下(不含IE9)混杂模式下的可视窗口宽高：" + windowW + "，" + windowH + "</div>";

}

} else {

document.getElementById('closures').innerHTML += "<div>IE9+及其他webkit内核浏览器的可视窗口宽高：" + windowW + "，" + windowH + "</div>";

}

##### （2）移动端：

可视窗口大小：document.documentElement.clientWidth/clientHeight;

网页实际大小：document.body.clientWidth/clientHeight;

### （二）location对象

location对象既是window对象，又是document对象，window.location和document.location是同一个对象。

pathname：返回URL中的目录或文件名；/view/content

host/hostname:返回服务器名称和端口号/返回服务器名称；www.baidu.com:8088（有端口号就会显示）/www.baidu.com

search:返回？后面的搜索关键字，必须得包含一个或多个字符;

?name=XXX&age=XXX

href:返回完整url；http://localhost:13166/Closures.html?name=123

获取地址栏中的参数信息：

var url = window.location.href;

//获取路径中指定的参数值

function getUrlParam(url, param) {

var pattern = new RegExp("[?&]" + name + "\=([^&]+)", "g");//设置正则表达式

//对url进行筛选,返回的数组是Array实例，但包含两个额外的属性: index 和 input。index 表示匹配项在字符串中的位置，input 表示应用正则表达式的字符串

var matcher = pattern.exec(url);

//matcher的值：["要匹配的字符串", index: 匹配的字符串在原字符串中的位置, input: "原字符串"]

var items = null;

if (matcher != null) {

try {

items = decodeURIComponent(decodeURIComponent(matcher[1]));//对数组中的数据进行解码

} catch (e) {

try {

items = decodeURIComponent(matcher[1]);

} catch (e) {

items = matcher[1];

}

}

}

return items;

}

getUrlParam(url, 'name');

## 26、DOM

### （一）、Node类型

每个节点都有一个nodeType属性，表名节点的类型。一般由12个数值常量表示，任何节点类型必居其一：

NODE.ELEMENT\_NODE(1)（元素类型,a,p标签）；

NODE.ATTRBUTE\_NODE(2)（属性类型，href，style属性）；

NODE.TEXT\_NODE(3)（文本类型）；

NODE.CDATA\_SECTION\_NODE(4)(CDATA区段，可当做Text节点处理)；

NODE.ENTITY\_REFERENCE\_NODE(5)(实体链接)；

NODE.ENTITY\_NODE(6)(实体类型);

NODE.PROCESSING\_INSTRUCTION\_NODE(7)(进程介绍类型)；

NODE.COMMENT\_NODE(8)(注释类型)；

NODE.DOCUMENT\_NODE(9)(文档类型);

NODE.DOCUMENT\_TYPE\_NODE(10);

NODE.DOCUMENT\_FRAGMENT\_NODE(11);

NODE.NOTATION\_NODE(12);

验证节点类型：someNode.nodeType==1；由于IE没有公开Node类型的构造函数，所以直接以someNode.nodeType== NODE.ELEMENT\_NODE会报错。

#### 1、nodeName和nodeValue：

这两的值取决于节点的类型。如果是element元素类型，nodeName保存的值为标签的名字，nodeValue的值为null。

#### 2、节点关系

节点之间有父子关系，同胞关系，每个节点都有childNodes属性，可通过方括号和item()方式访问节点:

var firstChild=someNode.childNodes[0];(someNode.childNodes.item(1))

虽然可通过方括号访问NodeList的值，而且该对象也有length属性，但是其并不是Array的实例，对arguments对象的可通过Array.prototype.slice()方法将其转换为数组，NodeList也可通过该方法进行转换：

Array.prototype.slice.call(someNode.childNodes,0);

每个节点都有parentNode属性，包含在其中的childNodes列表中每个节点互相之间都是同胞节点，通过siblings()方法可以遍历得到同胞节点，previousSibling和nextSibling可以得到前一个及后一个兄弟节点的值。

判断是否有子节点，可通过hasChildNodes()方法判断（返回true/false）。

#### 3、操作节点

**A、appendChild()**：向childNodes末端添加新的子节点。相当于在页面上获取一个节点**移动到**childNodes节点的末端。不支持字符串。

**B、append()**：插入数据，前面是要选择的对象，后面是要插入的元素内容。$("#a").append("content")，为jquery对象;

**C、appendTo()**：插入数据，前面是要插入的元素内容且为jquery对象，后面是要选择插入的对象。

$("<span>appenTo追加数据</span>").appendTo($('#a'));

**D、insertBefore(p1,p2)**：p1参数表示新插入的元素，p2表示要插入的位置。

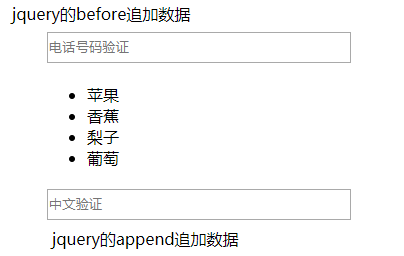
var node = document.getElementById("nodeTest");

var newNode = document.getElementById("chinese");

var newNode2 = document.getElementById("tel");

node.appendChild(newNode);

$("#nodeTest").append("<div>jquery的append追加数据</div>");

 node.insertBefore(newNode2,node.firstChild);

**E、replaceChild(newNode,oldNode)，removeChild()**：

**以上追加元素均在选择元素标签之内**

**F、before()**：在元素前面追加数据（在选择的元素标签之外）

### （二）、DOM2和DOM3

DOM1级主要定义的是HTML和XML文档的底层结构，DOM2和DOM3级则在这个结构的基础上引入更多交互能力，其目的在于扩展DOM API，以曼度操作XML的所有需求，同时提供更好的错误处理及特性检测能力。DOM2和DOM3级分为许多模块（模块之间有某种联系），分别描述了DOM的某个非常具体的子集。

DOM2级核心：在1级核心基础上构建，为节点添加了更多方法和属性。

DOM2级视图：为文档定义了基于样式信息的不同视图。

DOM2级事件：说明了如何使用事件与DOM文档交互。

DOM2级样式：定义了如何以编程方式来访问和改变CSS样式信息。

DOM2级遍历和范围：引入了遍历DOM文档和选择其特定部分的新接口。

DOM2级HTML：在1级HTML基础上构建，添加了更多属性、方法和新接口。

DOM级新增了XPath模块和加载与保存模块。

#### 1、DOM变化

一个是对命名空间的支持，一个是对操作其他web元素的支持。

（1）、其中命名空间的支持主要是针对XML，添加了localName(不带空间名的前缀节点名称),namespaceURI(命名空间URI，未指定的情况下为null),prefix(命名空间前缀)

<s:svg xmlns="http://www.w3.org/2000/svg"> </s:svg>

<!--localName：svg，tageName：s:svg，namespaceURL：http://www.w3.org/2000/svg，prefix：s-->

（2）、Document类型变化中与命名空间无关的方法是importNode()。这个方法的用途是从一个文档中取得一个节点，然后导入另一个文档，而每个节点都有一个ownerDocument属性，表示所属文档。如果调用appendChild()时传入节点属于不同的文档，则会报错，而调用importNode()时传入不同文档的节点会返回一个新节点，这个新节点的所有权归当前文档所有。

importNode()方法与cloneNode()方法非常相似，它接受两个参数：要复制的节点，是否复制节点的布尔值，返回的是原节点的副本。

var newNode = document.importNode(oldNode,true);

document.body.appendChild(newNode);

（3）、DOM2级视图模块增加defaultView属性，指向拥有给定文档的窗口，除IE外所有浏览器均支持该属性，故兼容IE浏览器，可通过parentWindow属性获取。

（4）、框架的变化中，框架和内嵌框架分别用HTMLFrameElement和HTMLIFrameElement表示，它们在DOM2级中有一个新属性，叫contentDocument，该属性包含一个指针，指向表示框架内容的文档对象，只能使用frames集合读取文档对象。在IE8之前不支持框架中的contentDocument属性，但支持contentWidow属性，该属性返回的是window对象，可获取document对象，代码如下：

var iframe = document.getElementById("myIframe");

var iframeDoc = iframe.contentDocument || iframe.contentWindow.document;

#### 2、样式变化

（1）、任何支持style特性的HTML元素在JavaScript中都有一个对应的style属性，其包含所有通过HTML的style特性指定(运用CSS样式有三种方法：link外部引入，<style>标签内部嵌入，style属性指定)的所有样式信息，不包含与外部样式表或嵌入式样式表经层叠而来的样式。对于使用短划线的CSS属性名，必须转换为驼峰大小写形式才能访问(background-image=>backgroundImage)。有一个特殊属性不能进行直接调用和转换，就是float，它是JavaScript的保留字，在IE下使用styleFloat，其他浏览器使用cssFloat。所有属性值必须加上度量单位。

（2）、操作方法，设置样式：

方法一、使用style属性的setProperty(propertyName,value,priority);

参数三是设置优先级(important或者一个空字符串);

myDiv.style.setProperty("background-color", "red", "")

方法二、使用style中"CSS属性对应的style属性"；

var myDiv = document.getElementById("myDiv");

//set the background color

myDiv.style.backgroundColor = "red";

//assign a border

myDiv.style.border = "1px solid black";

方法三、使用style属性的cssText属性。在写入模式下可一次性应用所有变化，重写整个style特性的值，之前通过style特性的指定的样式信息都将丢失。

var myDiv = document.getElementById("myDiv");

myDiv.style.cssText = "width: 25px; height: 100px; background-color: green";

也可通过同样的代码获取样式值：

var myDiv = document.getElementById("myDiv");

alert(myDiv.style.backgroundColor);

（3）、样式层叠问题

虽然style属性可以获取编辑style特性的值，但是并不包含外部引用的样式及内部嵌入式样式，在DOM2级中增强了document.defaultView属性，提供了getComputedStyle()方法，返回一个CSSStyleDeclaration对象（与style属性类型相同），其中包含当前元素的所有计算的样式。

getComputedStyle(elementNode，伪元素（可为空）)：

var myDiv = document.getElementById("myDiv");

var computedStyle = document.defaultView.getComputedStyle(myDiv, null);

alert(computedStyle.backgroundColor); //"red"

alert(computedStyle.border); //部分浏览器显示"1px solid black",IE不支持，因为是综合属性，显示undefined

IE不支持getComputedStyle()方法，但在IE中每个具有style属性的元素还有一个currentStyle属性，是CSSStyleDeclaration的实例，包含当前元素的全部计算后的样式。

var computedStyle = node.currentStyle;

alert(computedStyle.border); //综合属性，不显示值，显示为空

alert(computedStyle.marginLeft); //显示值40px

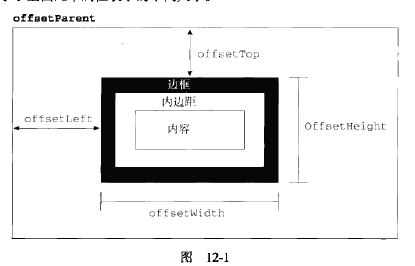
注意：以上的方法只能读取数据，不能修改数据，修改数据还得用style属性。

（4）、元素大小

**A、偏移量：**

元素大小由其高度、宽度决定，包括所有内边距，滚动条及边框大小（但不包括外边距margin），

offsetWidth：元素水平方向占用的空间；offsetHeight：元素垂直方向占用空间大小；offsetLeft：元素左外边框至包含元素的左内边框之间的距离；offsetTop：元素上外边框至包含元素的上内边框之间的距离。

**B、可视区域大小：**

clientWidth：网页可见区域宽；clientHeight：网页可见区域高；

**C、滚动大小：**

scrollHeight/scrollWidth：整个html文件的高/宽，包含内外边距等。

scrollTop：元素顶部距离当前滚动条顶部的位置距离；

scrollLeft：元素左侧距离当前滚动条顶部的位置距离。

在获取宽高度时，通过document.documentElement.scrollHeight/scrollWidth获取，而获取scrollLeft和scrollTop时则需要判断浏览器兼容性，因为IE/FF不支持通过body获取上距及左距，其他chrome浏览器仅支持body获取。

if (document.documentElement && document.documentElement.scrollTop)

{//兼容IE浏览器

positionY = document.documentElement.scrollTop;

} else if (document.body) {

positionY = document.body.scrollTop;// body.scrollTop;

}

#### 3、范围

DOM2级在Document类型中定义了createRange()方法，该方法属于document对象，使用hasFeature()或者直接检测该方法，判断浏览器是否支持范围这个方法。

//返回布尔值

var supportRange = document.implementation.hasFeature("Range", "2.0");

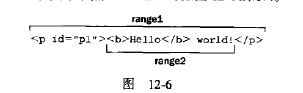
var range2 = (typeof document.createRange == "function");

使用createRange()方法创建范围，该方法是range类型的实例，包含以下方法及属性：startContainer（包含范围起点的节点，即第一个节点的父节点），startOffset（在startContainer中起点的偏移量），endContainer，endOffset。

**范围的选择：**

通过selectNode()或selectNodeContents()来选择文档中的一部分。selectNode()选择整个节点，selectNodeContents()只选择节点的子节点。

var range = document.createRange();

 var node = document.getElementById("nodeTest");

var range1 = range.selectNode(node);

var range2 = range.selectNodeContents(node);

创建复杂的范围得使用setStart()和setEnd()方法，接受两个参数：一个参照节点和一个偏移量值。

**操作范围：**

deleteContens()删除所选择的节点，extractContents()移除选择的节点，返回的是被移除的节点内容，可将它插入其他地方。cloneRnage()复制范围。

**IE中的范围：**

IE中不支持DOM的range范围，但是支持textRange()文本范围，只处理文本内容。document.body.createTextRange()，创建IE中的文本范围。findText()方法，查找指定文本内容，返回布尔值，true为找到指定内容。

var range = document.body.createTextRange();

var isFound = range.findText('aa'); //如何找到返回true，否则返回false

var text = range.text ; //输出aa

IE中创建复杂范围就是一特定的增量向四周移动范围，提供了4个方法，move(),moveStart(),moveEnd()和expand()，接受两个参数：移动单位和移动单位的数量。移动单位是下列一种字符串值：

“character”：逐字符移动；“word”：逐单词（一系列非空格字符）移动；

“sentence”：逐句（一系列以句号，问号或叹号结尾的字符）移动；

“textedit”：移动搭配当前范围选区的开始或结束位置。

通过moveStart()方法规定移动范围的起点，moveEnd()规定终点。

range.moveStart("word", 2);

range.moveEnd("word",3);

### （三）、遍历

遍历即使用NodeIterator或TreeWalker对DOM执行深度优先的遍历。

#### 1、NodeIterator

在IE中不支持NodeFilter。可使用document.createNodeIterator()方法创建实例，它可接受四个参数：root(给定节点为根节点)，whatShow(访问的节点数字代码，决定哪些节点需要搜索)，filter(NodeFilter对象，决定忽略哪些节点)，entityReferenceExpansion(boolean值，是否需要扩展实体引用)。

whatShow参数可取下列数值：

NodeFilter.SHOW\_ALL(所有节点)，NodeFilter.SHOW\_ELEMENT(元素节点)，NodeFilter. SHOW\_ATTRIBUTE(属性节点)，NodeFilter. SHOW\_TEXT (文本节点)等。

filter参数可自定义NodeFilter对象，也可留空（null）。每个NodeFilter对象只有一个方法，即acceptNode()。如果应该访问给定节点，返回NodeFilter.FILTER\_ACCEPT；如果不应该访问给定节点，返回NodeFilter.FILTER\_SKIP。

var iterator = document.createNodeIterator(document, NodeFilter.SHOW\_ALL, null, false);

NodeIterator类型有两个访问节点的方法：nextNode()和previousNode()（下一个节点和前一个节点）。

var iterator = document.createNodeIterator(document, NodeFilter.SHOW\_ALL, null, false);

var node1 = iterator.nextNode();

var node2 = iterator.previousNode();

#### 2、TreeWalker

TreeWalker是NodeIterator的升级版本，除了有基本的nextNode()和previousNode()在内的相同功能外，还提供了在不同方向上的遍历方法。

(1)parentNode()：遍历到当前节点的父节点。

(2)firstChild()：遍历到当前节点的第一个子节点。

(3)lastChild()：遍历到当前节点的最后一个子节点。

(4)nextSibling()：遍历到当前节点的下一个兄弟节点。

(5)previousSibling()：遍历到当前节点的前一个兄弟节点。

TreeWalker的实例创建使用同样的方法createTreeWalker()，接收四个参数。不过filter的返回值有些不同，除了有以上两个返回值外，还增加了NodeFilter.FILTER\_REJECT，表示拒绝进入相应节点。

但是当在NodeIterator中使用时，NodeFilter.FILTER\_SKIP和NodeFilter.FILTER\_REJECT的作用相同：跳过指定节点。在TreeWalker中NodeFilter.FILTER\_REJECT表示跳过相应节点及该节点的整个子树，NodeFilter.FILTER\_SKIP表示跳过相应节点继续前进到子树中的下一个节点。

## 27、DOM扩展

### （一）选择符API

jQuery的核心是通过css选择符查询DOM文档取得元素的引用

1、querySelector()

querySelector()方法接收一个css选择符，并返回与该模式匹配的第一个元素，若没有找到匹配的元素，则返回null。

var query = document.querySelector("#nodeTest"); //返回div#nodeTest下的所有元素内容

var body = document.querySelector("body");//返回body中所有元素内容

var div = document.querySelector("div input");//返回div下满足要求的第一个input元素内容

var li = document.querySelector("ul li.tli");//返回ul下样式第一个li的样式名为tli的li的元素内容

2、querySelectorAll()

querySelectorAll()与querySelector()方法一样，只是返回的值是满足条件的所有值，是NodeList实例。

### （二）HTML5扩展

**1、getElementsByClassName()**

获取符合类名条件的所有元素，返回的是NodeList实例。

**2、classList属性**

classList属性是HTML5新增的操作类名的方式，属于DOMTokenList，有length属性，可通过item()及方括号[]的方式获取内容。该属性还新增以下方法：

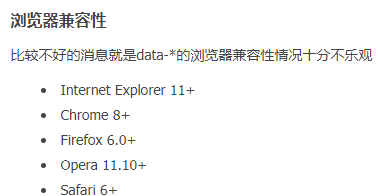
add(value)：新增class样式；contains(value)：是否包含指定样式，返回true/false；

remove(value)：移除指定样式；toggle(value)：如存在指定值则删除，不存在则添加。

node.classList.add("nodeClass");

均属于js方法，jquery中不存在classList属性。

**3、自定义数据属性**

HTML5规定可以为元素添加自定义属性，只是需要以data-做前缀，目的是为元素提供与渲染无关的信息，或提供语义信息。目前该属性的浏览器兼容性比较差：

读取自定义的数据属性，js中使用dataset.属性名的方法，jquery中使用$("选择器").data("属性名")的方法，为解决兼容性问题，可通过getAttribute/setAttribute获取值。

<div id="nodeTest" style="margin-left:40px;" data-myname="Mark">

</div>

var data\_name = node.dataset.myname;//js方法获取

var attr\_name = node.getAttribute("data-myname");/js方法获取，兼容IE11-

var jq\_data = $("#nodeTest").data("myname");//jquery方法获取

var jq\_attr\_name = $("#nodeTest").attr("data-myname");

alert(data\_name);

若出现名称中还包含“-”连接字符的，在取值是改成驼峰形式：

<div id="nodeTest" style="margin-left:40px;" data-student-name="Mark">

</div>

data\_name = node.dataset.studentName;//js方法获取

css样式可以直接写为:

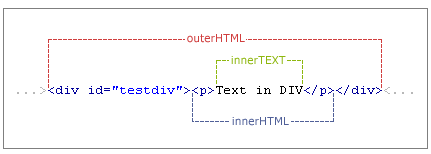
[data-birth-date] {background-color: #0f0; width: 100px; margin: 20px; }

**4、插入标记**

A、innerHTML：返回调用元素的子节点内容（元素，注释，文本所有内容）；

B、outerHTML：返回调用元素本身节点及子节点所有内容；

C、innerText/outerText：返回调用元素的纯文本内容；



### （三）专有扩展

1、文档模式：强制浏览器以某种模式渲染页面，通过<meta>标签设置。

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrom=1"/>

edge模式表示始终使用最新的文档模式渲染页面，也可以使用IE版本号7,8,9等，js中通过documenMode获取当前渲染模式版本。

var mode = document.documentMode;

2、获取两个DOM之间的所有兄弟元素：

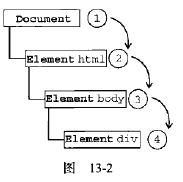
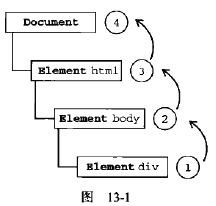
jquery的nextUntil(selector，filter)方法，selector表示何处停止匹配跟随的同胞元素的选择器表达式，filter表示用于匹配元素的选择器表达式。

$("#term-2").nextUntil("dt").css("background-color", "red");//由ID为term-2开始（不包含本身节点），到下一个dt节点为止，期间的背景色改为红色

var term3 = document.getElementById("term-3");

$("#term-1").nextUntil(term3, "dd").css("color", "blue");//由ID为term-1开始，到ID为term-3为止，期间元素为dd节点的文本颜色改为蓝色

## 28、事件

在IE中事件流叫做事件冒泡，即事件开始时由最具体的元素接收（文档中嵌套层次最深的节点），然后逐级向上传播但较为不具体的节点。如点击了某个div节点，具体事件走向如图13-1：

在Netscape Communicator团队提出另一种事件流叫事件捕获。在事件捕获过程汇总，document对象首先接收到click事件，然后事件沿DOM树依次向下，一直到事件的实际目标，即<div>元素，如图13-2。

### （一）事件处理程序

#### 1、DOM0级事件处理程序

使用DOM0级方法指定的事件处理程序被认为是元素的方法，因此，这时候的事件处理程序是在元素的作用域中运行，即程序中的this引用当前元素。

var node = document.getElementById("nodeTest");

node.onclick = function () {

alert(this.id);

}

#### 2、DOM2级事件处理程序

DOM2级事件定义了两个方法，用于处理指定和删除事件处理程序的操作：addEventListener()和removeEventListener()，接受3个参数：eventName，function，boolean；最后boolean值如果是true，表示捕获阶段调用事件处理程序，false表示冒泡阶段调用事件处理程序。需要注意的是，在调用事件名称时不需要加“on”前缀。

var node = document.getElementById("nodeTest");

node.addEventListener("click", function () {

alert(this.id);

}, false);

通过addEventListener添加的事件程序只能通过removeEventListener()方法移除。且传入的参数必须相同。

node.removeEventListener("click", function () {

alert(this.id);

}, false);

removeEventListener()方法中传入的第二个参数完全不同，所以没有效果。所以将代码进行修改：

var handler = function () {

alert(this.id);

};

//DOM2级事件处理程序

node.addEventListener("click", handler, false);

node.removeEventListener("click", handler, false);

将传递的函数方法进行封装，使得传入的参数一致，实现移除添加的监听事件。

IE事件处理程序与DOM2级事件处理程序一样，在使用detachEvent()时，传入的参数相同，将函数进行封装再传入值。

#### 3、IE事件处理程序

IE实现了与DOM中类似的两个方法：attachEvent()和detachEvent()，接受2个参数：eventName和function，由于IE8及更早版本只支持事件冒泡，所以通过attachEvent()添加的事件处理程序都会被添加到冒泡阶段(方法名需加前缀“on”)。(IE11不支持，IE8以上至IE10的执行顺序是编写顺序)

在IE中的事件处理程序与DOM0级方法的主要区别在于处理程序的作用域。DOM0级的事件处理程序作用域是在其所属元素的作用域内运行，而attachEvent方法的作用域是在全局作用域中运行，即this=window。

与addEventListener()方法类似，attachEvent()方法时也可在一个元素上添加多个不同的处理程序，不过attachEvent()方法的执行方式是以相反的顺序被触发。

//IE8(及以下)事件处理程序

node.attachEvent("onclick", function () {

alert(123456);

});

node.attachEvent("onclick", function () {

alert("abcdefg"); });

#### 4、跨浏览器事件处理程序

通过EventUtil对象，调用addHandler()方法来处理浏览器之间的差异。该方法接受3个参数：element，eventName，function。

//跨浏览器事件处理程序

var EventUtil = {

addHandler: function (element, eventName, func) {//添加事件

if (element.addEventListener) {

element.addEventListener(eventName,func,false);//DOM2级

} else if (element.attachEvent) {

element.attachEvent("on" + eventName,func);//IE

} else {

element["on" + eventName] = func;//DOM0级

}

},

removeHandler: function (element, eventName, func) {//移除事件

if (element.removeEventListener) {

element.removeEventListener(eventName, func, false);//DOM2级

} else if (element.attachEvent) {

element.removeEvent("on" + eventName, func);//IE

} else {

element["on" + eventName] = null;//DOM0级

}

}

}

### （二）事件对象

#### 1、DOM中的事件对象

每个事件处理程序在运行时都会传入event对象，event对象包含与创建它的特定事件有关属性和方法，虽然不一样的事件有不同的属性和方法，不过都会有基本的属性及方法（部分IE8及以下不支持）：

要阻止特定事件的默认行为，可使用preventDefault()方法，不过，只有当cancelable属性诶true时，才可使用preventDefault()方法取消默认行为。

var link = document.getElementById("link");

link.onclick = function (e) {

//e.cancelable = true;

e.preventDefault();//取消a链接跳转默认事件

}

target表示事件的目标，包含目标元素的所有信息，可利用其进行css样式的修改(event.target.style.backgroundColor等)。

stopPropagation()方法用于立即停止事件再DOM层次中的传播，即取消进一步的事件捕获或冒泡。

#### 2、IE中的事件对象

IE中event对象属于window，所以需要通过window.event获取。事件运行时所有事件对象都会包含以下属性和方法：

cancelBubble: 默认false,为true可取消事件冒泡(同stopPropagation()),

returnValue：默认true，为false可取消默认事件(同preventDefault()),

srcElement：事件目标，同target。

if (document.documentMode <= 8) {

//IE8(及以下)事件处理程序

node.attachEvent("onclick", function () {

alert(123456);

});

node.attachEvent("onclick", function () {

alert("abcdefg");

});

link.onclick = function () {

//e.cancelable = true;

window.event.returnValue = false;//IE8及以下取消默认事件

}

}

#### 3、跨浏览器的事件对象

通过EventUtil对象实现对跨浏览器的事件进行处理，

//跨浏览器事件处理程序

var EventUtil = {

addHandler: function (element, eventName, func) {

},

//跨浏览器事件对象

getEvent: function (event) {

return event || window.event;//非IE||IE，表示返回结果为true的值,如果event存在，则为true，并返回event

},

getTarget: function (event) {

return event.target || event.srcElement;

},

preventDefault: function (event) {

if (event.preventDefault) {

event.preventDefault();//存在，则调用preventDefault()方法

} else {

event.returnValue = false;//IE8及以下支持

}

}

}

link.onclick = function (e) {//元素调用

var evt = EventUtil.getEvent(e);

EventUtil.preventDefault(evt);

};

### （三）事件类型

#### 1、移动设备的触摸与手势事件

##### （1）、触摸事件

touchstart：当手指触摸屏幕时触发；

touchmove：手指在屏幕上滑动时触发；当滚动事件发生时，可通过preventDefault()阻止滚动；

touches：表示当前跟踪的触摸操作的Touch对象数组。

##### （2）、手势事件

当两个手指触摸屏幕时就会触发手势事件，手势通常会改变显示项的大小，或者旋转显示项，有三个手势事件：

(1)gesturestart：当一个手指已经按在屏幕，另一个手指又触摸屏幕时触发；

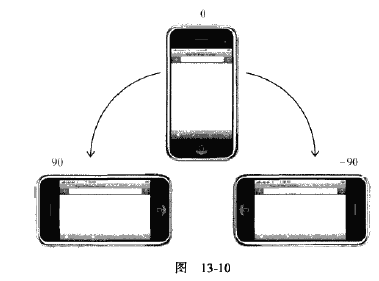
(2)gesturechange：当触摸屏幕的任何一个手指位置发生变化时触发；

(3)gestureend：当任何一个手指从屏幕上移开时触发。

##### （3）、设备事件

**1、orientation事件**

orientation是苹果公司为移动Safari中添加的事件，方便开发人员确定用户核实切换查看模式。window.orientation事件的属性包含3个值：0表示肖像模式（竖直），90表示向左旋转横向，-90表示向右旋转横向。

所有iOS设备都支持orientation事件和window.orientation属性。同时，可以将orientation看成window事件，也可通过body元素的onorientation特性来指定事件处理程序。

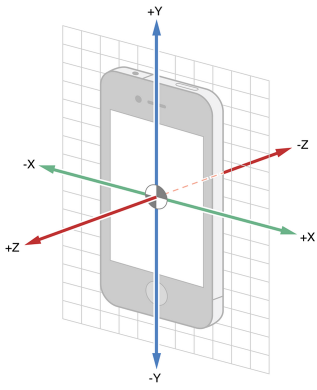
**2、MozOrientation事件**

这是FireFox浏览器开发商特定的事件，可通过window对象触发。

EventUtil.addHandler(window, "MozOrientation", function (event) {

//响应事件

});

此事event事件包含三个属性：x，y，z。这几个属性值介于1到-1之间，表示不同坐标轴上的方向。在静止状态下，x=0，y=0，z=1（设备处于竖直状态）；向右倾斜，x增大；设备向远离用户的方向倾斜，y值会减小；z轴检测垂直加速度，1表示静止不动，在设备移动时值会减小(失重状态下值为0)。只有带加速计的设备才支持MozOrientation事件，但这是一个实验性API，将来可能会被其他事件取代。

##### （4）、HTML5事件

beforeunload事件，在用户切换页面或者刷新页面前阻止这一操作的方法事件，该方法会弹出对话框，返回布尔值，确认用户的操作。

EventUtil.addHandler(window, "beforeunload", function (event) {

event = EventUtil.getEvent(event);//获得事件对象

var msg = '离开页面前请先确认保存案件信息！';

event.returnValue = msg;//是否取消默认事件，兼容IE

return msg;

});

## 29、表单脚本

### （一）表单字段

可以像访问页面的其他元素一样，使用原生DOM方法访问表单元素，通过document.getElementById("IdName")等，除了<fieldset>元素之外，所有表单字段拥有相同的一组属性，如下：

disabled：Boolean值，表示当前字段是否禁用。为防止用户重复提交，可在用户第一次submit（提交）后就禁止按钮提交事件。

tabIndex：表示当前字段的切换(tab)序号。它的值为从1到32767之间的一个值。tabIndex值越小就越最先移动。如果设置值为0，则在所有设置的tabIndex值的元素之后排列；如果设置值为-1，则会忽略该元素，但是在元素的onfocus()和onblur()方法仍然会有效，可通过这个方法，将除了表单字段可实现的获得焦点的方法在其他元素上也实现。

还有form（当前字段所属表单），name，readOnly（是否只读），type，value。

共有方法：focus()方法和blur()方法。这是每个表单字段都有的方法，但是当表单type为hidden时，代码会报错；还有当CSS样式的display和visibility属性隐藏了该字段，也会报错。

HTML5为表单字段新增了autofocus属性，会实现自动聚焦，但是部分浏览器不支持该属性，需要进行验证判断。autofocus属性是一个布尔值属性，在支持的浏览器中返回的值是true，不支持的浏览器中返回的是空字符串。只有不为true时，调用focus()方法，实现向前兼容。

<input type="date" id="birthday" placeholder="出生日期验证" autofocus/>

//验证是否支持autofocus属性

var ele = document.getElementById("birthday");

if (ele.autofocus != true) {

ele.focus();

}

### （二）HTML5约束验证API

1、required属性，表示是否是必填。

2、email和url类型，得到支持最多的类型。

3、数值范围：range，number，datetime，date，month，week，time等。浏览器支持度不是很高。

4、输入模式：pattern属性，需要填写正则表达式，模式额开头与末尾不需要加^和$符号。

<input type="text" pattern="\d+" name="num"/>//提交时自动验证格式

//验证是否支持pattern，返回布尔值

var p = "pattern" in document.createElement("input");

alert(p);

5、检测有效性

所有表单字段都有checkValidity()方法，使用checkValidity()方法可以检测表单中的某个字段是否有效，返回值是布尔值，有效为true，判断依据是前面介绍的约束属性或方法。例如必填字段为空就是无效。也可检测整个表单是否有效，如果表单中有一个字段无效，则返回false。

validity()方法验证字段为何无效，该对象包含一系列属性，每个属性会返回一个布尔值：patternMismatch:与pattern进行匹配，不匹配返回true；rangeOverflow：超过max值，返回true；rangUnderflow：小于min值，返回true；typeMismatch：格式不匹配，返回true；valueMissing：如果标注为required的字段没有值，返回true；valid：如果这里的其他属性都是false，返回true。

### （三）选择框脚本

#### 1、选择框的基本属性和方法：

选择框是通过<select>和<option>元素创建的，除了共有的方法外，HTMLSelectElement类型还提供了下列属性和方法：

**add（newOption，position）：**向控件插入newOption元素，位置在position（数值）之前。

var ele = document.createElement("option");

ele.innerText = "测试添加option节点";

obj.add(ele, 2);

除了add()方法可添加元素外，还可以通过Option构造函数创建，new Option("text","value")；还有使用createElement()方法穿件元素，通过appendChild()方法追加。

**options：**获取控件中所有option元素的HTNLCollection，数组形式，可通过[index]索引号读取元素节点及文本内容。

var obj = document.getElementById("selBooks");

var ops = obj.options[index];

**selectedIndex：**基于0的选中项索引，如果没有选中项，则值为-1。有选中项则返回当前索引值。

**value：**由当前选中项决定。

当没有选中的项，value属性保存空字符串；

当有一个选中项，且value值设置了，则value属性值等于选中项的value特性，即使为空，也一样；

当有一个选中项，没有设置value属性，大部分浏览器中value值等于文本值，但是在IE8及以下的浏览器中，value值为空。

当有多个选中项，则value属性将依据前两条规则取得第一个选中项的值。

#### 2、移除选项的方法有多种：

（1）removeChild(select.options[0]),传入要移除的项；

（2）remove(0)，传入要移除项的下标；

（3）将选项设置为null，select.options[0]=null。

#### 3、移动和重排选项：

（1）appendChild(node)：如果新创的节点，使用该方法可以新增节点；如果获取的是页面本来的节点，则是将该节点移动到选择框的最后。

（2）insertBefore(newNode，position)：改方法可以指定将节点（newNode）放置到指定位置（position）前面。

### （四）富文本编辑

富文本编辑本质是在页面中嵌入一个包含空HTML页面的iframe，通过设置designMode属性，使得HTML页面可编辑，而编辑对象则是该页面<body>元素的HTML代码。

#### 1、实现富文本编辑

**方法一：**designMode属性有两个值：off（默认值），on。在设置为“on”时，整个文档可编辑。

只有在页面完全加载完之后才能设置这个属性，因此，在包含页面需要使用onload事件设置designMode。

window.onload = function () {

//方法一

frames["myIframe"].document.designMode = "on";// myIframe是iframe的name值

//方法二

document.getElementById("myIframe").contentDocument.designMode = "on";

}

contentDocument属性代表加载在iframe中的html文档。

**方法二：**页面元素添加contenteditable属性，该属性有三个值：“true”表示打开，“false”表示关闭，“inherit”表示从父元素继承。各浏览器均支持。

<div class="iframeContains" contenteditable="true"> </div>

#### 2、操作富文本

可通过document.execCommand()方法，对富文本区域的内容进行样式编辑操作，类似在线编辑器。document.execCommand(执行命令名称，是否为当前命令提供用户界面的布尔值，执行命令必须的参数值（可为null）)，为确保跨浏览器的兼容性，第二个参数应始终设置为false。通过点击调用函数方式对富文本区进行操作。可操作加粗(bold),倾斜(italic),下划线(underline)等

<input type="button" onclick="showCss('BackColor,red')" value="背景色" />

//使用frames[name]方法获取

function showCss(obj) {

var isMore = obj.indexOf(",");

if (isMore > -1) {

var param = obj.substring(isMore + 1);

obj = obj.substring(0,isMore);

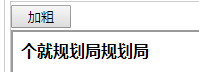
frames["myIframe"].document.execCommand(obj, false, param);

} else {

frames["myIframe"].document.execCommand(obj, false, null);

}

}



//contenteditable属性时使用当前窗口的document方法

function showCss2(obj) {

var isMore = obj.indexOf(",");

if (isMore > -1) {

var param = obj.substring(isMore + 1);

obj = obj.substring(0, isMore);

document.execCommand(obj, false, param);

} else {

document.execCommand(obj, false, null);

}

}

#### 3、表单与富文本

通过表单提交富文本的内容，需要将富文本的内容插入form表单中的隐藏字段，通过submit事件处理程序。

iframe中使用的方法：

EventUtil.addHandler(form, "submit", function (event) {

event = EventUtil.getEvent(event);//获得事件对象

var target = EventUtil.getTarget(event);//获得当前点击对象

var val = frames["myIframe"].document.body.innerHTML;

target.elements["Memo"].innerHTML = val;

});

contenteditable属性使用的方法：

EventUtil.addHandler(form, "submit", function (event) {

event = EventUtil.getEvent(event);//获得事件对象

var target = EventUtil.getTarget(event);//获得当前点击对象

var val = document.getElementById("myIframe2").innerHTML;

target.elements["Memo"].innerHTML = val;

});

## 30、使用Canvas绘图

HTMl5新增的canvas元素可用于绘图，需要先设置width和height属性，指定绘图区域的大小，通过getContext()方法获取2D上下文对象，可在canvas上编辑图像。

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext('2d');

可使用toDataURL()方法（canvas对象方法），导出canvas元素上绘制的图像，该方法接受一个参数，即图像的MIME类型格式。使用该方法时，图像不能来自其他域，否则会报错，如：www.example.com页面绘制图像来自www.orx.com网站，就会报错。

var imgUrl = canvas.toDataURL("image/png");

var img = document.createElement("img");

img.src = imgUrl;

document.body.appendChild(img);

### （一）2D上下文

#### 1、填充和描边

填充和描边由两个属性决定：fillStyle和strokeStyle。这两个属性的值可以是渐变对象，字符串或模式对象，默认值是“#000000”。设置颜色可使用颜色名，十六进制码，rgba，rgb，hsl和hsla。

//渐变对象

var trunkGradient = context.createLinearGradient(-5, -50, 5, -50); //线性渐变

trunkGradient.addColorStop(0, '#663300');

trunkGradient.addColorStop(0.4, '#996600');

trunkGradient.addColorStop(1, '#552200');

context.fillStyle = trunkGradient;

//模式对象

function getColor (intensity) {

var colors = ["#072933", "#2e4045", "#8c593b", "#b2814e", "#fac268", "#fad237"];

return colors[Math.floor(intensity / 2)];

}

context.fillStyle = getColor(radius);

//字符串

context.fillStyle = '#339900';

#### 2、绘制矩形

绘制矩形有关的方法包括fillRect()、strokeRect()、clearRect()，这三个方法接收4个参数：矩形X的坐标，y坐标，宽度，高度。

fillRect()方法绘制填充的矩形，通过fillStyle方法设置填充样式。

strokeRect()方法绘制描边的矩形，通过strokeRect方法设置描边样式。

clearRect()方法清除指定区域。

lineWidth属性控制描边线条宽度；lineCap属性控制线条末端形状是平头(butt)、圆头(round)、方头(square)；lineJoin属性控制线条相交的方式是圆交(round)、斜交(bevel)、斜接(miter)。

context.lineWidth = 4;

context.lineJoin = 'round';

context.strokeStyle = '#663300';

context.stroke();

#### 3、绘制路径

2D上下文绘制路径，必须先调用beginPath()方法，表示开始绘制新路径。绘制路径的方法有以下几种:

**（1）arc(x，y，radius，startAngle，endAngle，counterclockwise)**，表示绘制圆形路径，圆点是（x，y），以radius为半径，startAngle，endAngle分别表示起始和结束角度，一般设为0和Math.PI\*2，最后的参数表示是否按逆时针方向，false表示按顺时针方向。

context.beginPath();

context.arc(29, 400, 5, 0, Math.PI \* 2, true);

context.strokeStyle = "#000";

context.stroke();

context.closePath();

**（2）rect(x，y，width，height)**，表示绘制矩形路径，从(x，y)为起点绘制矩形，宽和高分别是width和height。该方法绘制的是矩形路径，不是strokeRect和fillRect所绘制的独立形状。

**（3）moveTo(x，y)**，表示移动到(x，y)点，不画线。

**（4）lineTo(x，y)**，表示从上一点绘制一条直线，到(x，y)点为止。

context.beginPath();//开始绘制

context.moveTo(33, 420); //移动到（33,420）这个起点

context.lineTo(41, 412); //从起点画线连接到(41,412)点

context.lineTo(48, 420);

context.closePath();//连接起点，闭合路径

以上绘制路径的方法没有设置样式，或者样式默认是#ffffff，所以在绘制完成后需要添加样式，方便能显示出来。使用stroke()或者fill()来闭合描边或填充样式。

#### 4、绘制文本

绘制文本主要有两个方法：fillText()和strokeText()。该方法可接收4个参数：要绘制的文本字符串、x坐标、y坐标和可选的最大像素宽度。

这两个方法都以下列3个属性为基础：

font：表示文本样式,字体及大小，用CSS中的格式编写，如"40px 微软雅黑"

textAlign：表示文本水平对齐方式，值有：start，end，left，right和center。建议使用start和end替换left和right，前者稳定性更稳妥，能同时适合从左至右和从右至左（阅读）的语言。

textBaseline：表示文本垂直对齐方式，即文本基线。值有：top，hanging，middle，alphabetic，ideographic和bottom。

context.font = "40px 微软雅黑";

context.fillStyle = '#000';

context.textAlign = 'center';

context.textBaseline= 'middle;

context.fillText('小纸盒的幻境森林！', 120, 50);

上面的文字会垂直居中显示，因为textAlign设置了“center”，textBaseline设置了“middle”。如果textAlign设置为“start”，则x坐标表示文本左端的位置（从左至右阅读）。如果为“end”，x坐标表示文本右端的位置（从左至右阅读）。

textBasiline设置了垂直对齐方式，top，y坐标表示文本顶端；bottom，y坐标表示文本底部，其他的值则说明y坐标分别指向字体的特定基线坐标。

context.textAlign = 'center';

context.textAlign = 'start';

context.textAlign = 'end';

#### 5、绘制图像

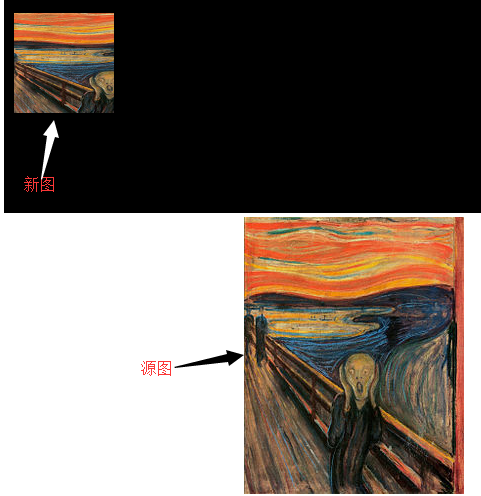
将图片绘制到画布上，可使用drawImage()方法。drawImage()方法可传入九个参数：drawImage(imgObj,x1,y1,w1,h1,x2,y2,w2,h2)。将imgObj对象进行绘制，绘制起点位置为源图（x1，y1），宽高（w1，h1），放到新的位置（x2，y2），新位置宽高（w2，h2）。

var clock = document.getElementById("clock");

var ctx = clock.getContext('2d');

var img = document.getElementById("scream");

ctx.drawImage(img,20,20,100,200,10,400,100,100);



drawImage也可以将<canvas>元素作为第一个参数进行绘制。

#### 6、阴影

2D上下班会为形状或路径等绘制阴影，通过以下几个参数设置：

shadowColor：用CSS颜色格式表示阴影颜色，默认黑色（#000）；

shadowOffsetX：X轴方向阴影偏移量，默认0；

shadowOffsetY：Y轴方向阴影偏移量，默认0；

shadowBlur：模糊像素数，默认0，数值越大越模糊。

context.shadowColor = 'rgba(0,0,0,0.2)';

context.shadowOffsetX = 15;

context.shadowOffsetY = -10;

//高斯模糊，值越大边缘越模糊

context.shadowBlur = 2;

//执行上述代码后，canvas渲染器会自动应用阴影效果，直到恢复canvas状态或重置阴影属性。故该处没有添加restore()方法

//context.save();不需单独保存，与文本阴影一起保存

context.font = "40px 微软雅黑";

context.fillStyle = '#960';

context.textAlign = 'center';

context.textBaseline = 'middle';

context.fillText('小纸盒的幻境森林！', 120, 50, 200);

context.restore();

#### 7、渐变

2D上下文中渐变有两种：createLinearGradient()线性渐变和createRadialGradient()径向渐变。

createLinearGradient()线性渐变传递的参数有四个：x1，y1，x2，y2。表示渐变的起点和终点的位置。

createRadialGradient()径向渐变主要是通过两个圆实现颜色渐变，传递的参数有六个：x1,y1,r1,x2,y2,r2。前三个参数表示其中一个圆的圆心坐标和半径，后三个参数表示另一个圆的圆心及半径。如要实现从同一点呈放射状渐变，需要将两个圆定义为同心圆。

创建渐变对象后，使用addColorStop()方法指定颜色值，该方法接收两个参数：色标位置，CSS颜色值。色标位置是一个从0到1之间的数字，表示渐变过程。

设置颜色值后，通过fillStyle或strokeStyle方法为该对象进行绘制。再绘制实体渲染对象，使用fillRect或strokeRect()绘制矩形或使用arc绘制圆形等。

#### 8、模式

模式就是重复的图像，可以用来填充或描边，使用createPattern()方法传递两个参数：image对象或HTML<img>元素和一个表示如何重复图像的字符串。第二个参数的值和CSS中background-repeat的属性值相同，包括“repeat”、"repeat-x”、“repeat-y”、“no-repeat”。

var img = new Image(); //创建Image对象，获取图片

img.src = "../../img/toTop.gif";

var line=new Image();

line.src="../../img/soil2.png";

var clock = document.getElementById("clock");

var ctx = clock.getContext('2d');

var pattern = ctx.createPattern(img, "repeat");//创建模式对象，设置属性值

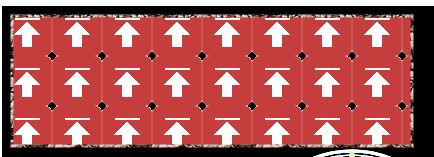
var pattern2 = ctx.createPattern(line, "repeat");

ctx.fillStyle = pattern;

ctx.fillRect(10,5, 400, 130);//绘制矩形框，用img填充矩形框

ctx.lineWidth = 3;

ctx.strokeStyle = pattern2;

 ctx.strokeRect(10,5,400,130);//设置描边线条大小，将图片作为背景线条，对矩形框进行描边

### （二）WebGL

WebGL是针对Canvas的3D上下文，它不是W3C制定的标准，是由Khronos Group制定的，浏览器中使用的WebGL就是基于OpenGL ES2.0制定的。

WebGL设计复杂的计算，需要提前知道数值的精度，而JavaScript数值无法满足需求，因此WebGL引入了一个概念，叫类型化数组。

#### 1、类型化数组

类型化数组也是数组，只是其元素被设置为特定类型的值。类型化数组的核心是ArrayBuffer的类型，每个ArrayBuffer对象表示的只是内存中指定的字节数，但不会指定保存什么类型的数据。通过ArrayBuffer只能是，为将来使用而分配一定数量的字节。

创建ArrayBuffer对象：

var arrayBuffer = new ArrayBuffer(20);//为arrayBuffer分配内存20B

通过该对象能获取的只有它包含的字节数，方法是访问其byteLength属性：

var blength = arrayBuffer.byteLength;//获取arrayBuffer的字节数

**（1）、视图**

使用ArrayBuffer（数组缓冲器类型）的一种特别的方式就是用它来创建数组缓冲视图，最常见的视图是DataView，通过它可以选择ArrayBuffer中一小段字节。

DataView可以传入三个参数：arrayBuffer对象，可选的字节偏移量（从该字节开始选择），可选的要选择的字节数。

var dataView = new DataView(arrayBuffer, 7, 10); //创建一个从字节7到字节16的新视图

实例化之后，DataView对象会把字节偏移量以及字节长度信息分别保存在byteOffset和byteLength属性中。通过读取属性获取值，dataView.byteOffset/dataView.byteLength。

通过getter和setter方法设置和获取内容，getInt8（byteOffset）/getInt16(byteOffset,littleEndian)/getInt32(byteOffset,littleEndian)/getFloat32(byteOffset,littleEndian)，有符号8/16/32位整数和32位浮点数。将Int改为Uint则是无符号N位整数和32位浮点数。

每个位数使用的字节数不一样，8位整数要用1B，16位整数要用2B，32位浮点数要用4B。需要明确知道数据所需字节，选择正确的读写方法。

16位或更大数值的方法都有一个可选参数littleEndian，该参数是布尔值，表示读写数值时是否采用小端字节序（即将数据的最低有效位保存在低内存地址中），默认大端字节序方式保存。

**（2）、类型化视图**

类型化视图也称为类型化数组，该视图分为好几种，都继承了DataView。

Int8Array/Int16Array/Int32Array/Float32Array/Float64Array，表示N位二补整数；Uint8Array/Uint16Array/Uint32Array，表示N位无符号整数。这些方法都可以接收3位参数，与DataView相似，只有第一个参数（ArrayBuffer）是必需的，第二个参数默认为0。

类型化数组是WebGL项目中执行各种操作的重要基础。

#### 2、WebGL

在浏览器中WebGL的名字叫“experimental-webgl”，引用时默认定义名称为gl。先验证浏览器是否支持getContext()方法，不支持会返回undefined，然后验证是否支持WebGL，不支持返回null。

var canvas = document.getElementById("WebGL");

alert(typeof canvas.getContext);//支持的话，返回function类型，不支持，返回undefined

if (canvas.getContext) {

var gl = canvas.getContext("experimental-webgl");

alert(typeof gl);//支持的话，返回Object类型，不支持，返回null

if (gl) {

//进行的操作

}

}

通过getContext()可传递第二个参数，为WebGL上下文设置一些选项，这个参数本身是一个对象，可包含下列属性：

（1）alpha：默认true，表示为上下文创建一个Alpha通道缓冲区；

（2）depth：默认true，表示可以使用16位深缓冲区；

（3）stencil：默认false，一般设为true，表示可使用8位模板缓冲区；

（4）antialias：默认true，表示使用默认机制执行抗锯齿操作；

（5）premultipliedAlpha：默认true，表示绘图缓冲区有预乘Alpha值；

（6）preserveDrawingBuffer：默认false，一般设为true，表示在绘图完成后保留绘图缓冲区。

var gl = canvas.getContext("experimental-webgl", {preserveDrawingBuffer:true});

##### （1）常量

在OpenGL中常量基本都带前缀“GL\_”，在WebGL中常量没有“GL\_”前缀，使用“gl.”替代，如：GL\_COLOR\_BUFFER\_BIT=>gl.COLOR\_BUFFER\_BIT。

##### （2）方法命名

OpenGL以及WebGL中很多方法都视图通过名字传达有关数据类型的信息，某些方法接收不同类型及数量的参数，可通过方法名后缀知道。方法名后缀会包含参数个数(1到4)和接收的数据类型(f表示浮点数，i表示整数)。也有接收的参数是数组参数，这些方法一般会包含字母v（vector，矢量）。如：gl.uniform4f()，gl.uniform3iv()。

##### （3）准备绘图

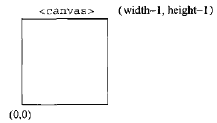
在实际操作WebGL上下文之前，会使用某种实色清除<canvas>，为绘图做好准备。

首先使用clearColor()方法指定要使用的颜色值，可接收4个参数：红，绿，蓝和透明度(即rgba)。每个参数必须是0到1之间的数值，表示每种分量在最终颜色中的强度。

其次调用clear()方法，传图参数gl.COLOR\_BUFFER\_BIT，清理缓冲区，使用之前定义的颜色来填充相应区域。

##### （4）视口与坐标

定义视口大小，默认可使用整个<canvas>元素，要改变视口大小，可调用viewport()方法，该方法可传入4个参数：(相对于<canvas>元素的)x坐标、y坐标、宽度、高度。视口坐标的原点（0,0）位于<canvas>元素的左下角，x轴和y轴正方向分别是向右和向上。

gl.viewport(0, 0, canvas.width, canvas.height);//设置视口宽高

##### （5）缓冲区

顶点信息保存在JavaScript的类型化数组中，使用前必须转换到WebGL的缓冲区，可调用gl.createBuffer()创建缓冲区，然后使用gl.bindBuffer()绑定到WebGL上下文，最后通过bufferData()方法将数据填充缓冲区。

bufferData()方法接收3个参数：buffer类型，保存数据的方法，使用缓冲区方式。传参时不明确buffer也没问题，一般使用Float32Array保存顶点信息，最后的参数用于指定使用缓冲区的方式。该方式有多种值：

gl.STATIC\_DRAW：数据只加载一次，在多次绘图中使用；

gl.STREAM\_DRAW：数据只加载一次，在几次绘图中使用；

gl.DYNAMIC\_DRAW：数据动态改变，在多次绘图中使用。

多数情况使用gl.STATIC\_DRAW即可。

var buffer = gl.createBuffer();//创建WebGL缓存区

gl.bindBuffer(gl.ARRAY\_BUFFER, buffer);//将缓存取绑定到WebGL上下文

gl.bufferData(gl.ARRAY\_BUFFER,new Float32Array([0,0.5,1]),gl.STATIC\_DRAW);

在包含缓冲区的页面重载之前，缓冲区始终保留在内存中，如果不想要某个缓存区，可直接调用gl.deleteBuffer(buffer)释放内存。

##### （6）错误

WebGL与JavaScript不同的是，有错误时WebGL不会报错，需要手动调用报错方法gl.getError()验证。gl.getError方法返回以下值：

gl.NO\_ERROR：表示没有错误（值为0）；

gl.INVALID\_ENUM：应传入WebGL常量，但却传错了参数；

gl.INVALID\_VALUE：在需要无符号数的地方传入了负值；

gl.INVALID\_OPERATION：在当前状态下不能完成操作；

gl.OUT\_OF\_MEMORY：内存溢出，无法完成操作；

gl.CONTEXT\_LOST\_WEBGL：因外部事件（如设备断电）导致丢失当前WebGL上下文。

每次调用该方法都会返回一个值，对于之后可能有多次的错误，可使用循环方法，直到返回NO\_ERROR。

var errorCode = gl.getError();//手动调用检测错误的方法

while (errorCode) {//循环判断，直到输出gl.NO\_ERROR

console.log("WebGL报错：" + errorCode);

errorCode = gl.getError();

}

##### （7）着色器

着色器（shader）是OpenGL中的另一个概念。WebGL有两种着色器：顶点着色器和片段（或像素）着色器。顶点着色器用于将3D顶点转换为需要渲染的2D点；片段着色器用于准确计算要绘制的每个像素的颜色。

着色器是由GLSL（OpenGL Shading Language，OpenGL着色语言）写的，GLSL是一种类C语言。每个着色器都有一个main()方法，该方法在绘图期间会重复执行。为着色器传递数据的方式有两种：Attribute和Uniform。

通过Attribute可以向顶点着色器中传入顶点信息，通过Uniform可以向任何着色器传入常量值。Attribute和Uniform在main()方法外部定义，分别使用关键字attribute和uniform，在这两个值类型关键字之后，是数据类型和变量名。

##### （8）绘图

WebGL只能绘制三种形状：点、线和三角。其他形状都是由这三种基本形状合成之后，再绘制到三维空间的。执行绘图操作要调用gl.drawArrays()或gl.drawElements()方法，前者用于数组缓冲区，后者用于元素数组缓冲区。

gl.drawArrays()或gl.drawElements()方法第一个参数都是一个常量，表示要绘制的形状。（JavaScript高级程序设计第3版476页）

### （三）、Three.js

Three.js是一个3D JavaScript库，里面封装了底层的图形接口，能支持所有WebGL支持的方法，除了WebGL外，还提供了基于Canvas、SVG标签的渲染器。

#### 1、使用Three.js

下载Three.js文件，由外部引用进项目中，如：<script src="../js/three.min.js"></script>，

一个典型的Three.js程序至少要包括渲染器（Renderer），场景（Scene），照相机（Camera），以及在场景中创建的物体。

#### 2、Three.js程序所需基本元素

**（1）渲染器与Canvas元素绑定：**

//绑定渲染器

var renderer = new THREE.WebGLRenderer({ canvas: document.getElementById("myCanvas") });

//设置背景色，用于清除画面的颜色

renderer.setClearColor(0x0000);

**（2）添加场景：**所有的物体都是添加到场景中的，所以它相当于一个大容器。

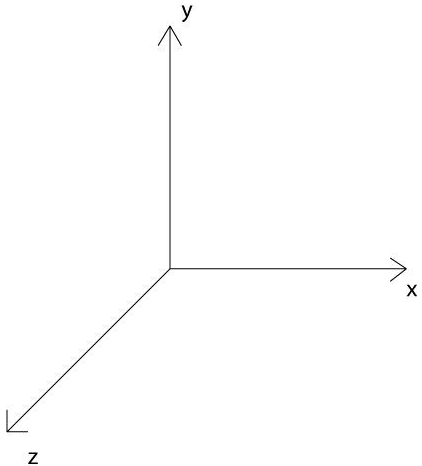
//添加场景

var scene = new THREE.Scene();

**（3）添加照相机：**

照相机投影实现三维空间到二维屏幕，投影氛围两种：正交投影和透视投影。设置完照相机之后都要添加至场景（scene）中，方便渲染（renderer）。

**（a）透视投影照相机**，故采用右手系坐标，确定照相机位置。

//添加照相机，采用右手系坐标，定义一个透视投影的照相机

var camera = new THREE.PerspectiveCamera(45, 4 / 3, 1, 1000);

//设置照相机的位置

camera.position.set(0, 0, 5);

//添加至场景中 scene.add(camera);

其中PerspectiveCamera(45, 4 / 3, 1, 1000) 4个参数，分别表示fov，aspect，near，far。

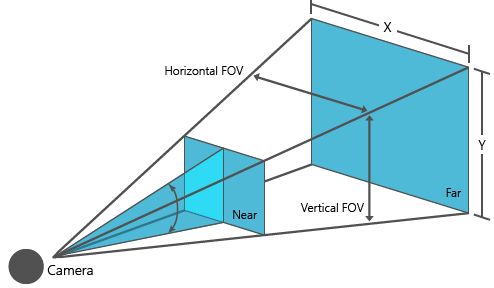
 this.fov = fov !== undefined ? fov : 50; //指明相机的可视角度,角度越大，视景体越小。可选参数,如果未指定,初始化为50

this.aspect = aspect !== undefined ? aspect : 1; //指明相机可视范围的长宽比,可选参数,如果未指定,初始化为1

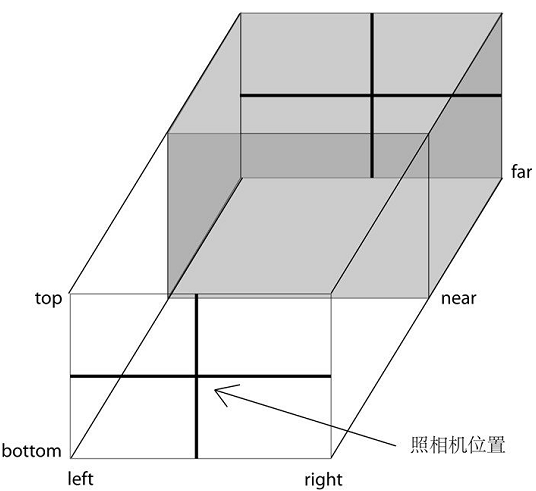
this.near = near !== undefined ? near : 0.1; //指明相对于深度剪切面的近的距离，必须为正数,可选参数,如果未指定,初始化为0.1

this.far = far !== undefined ? far : 2000; //指明相对于深度剪切面的近的距离，必须为正数,可选参数,如果未指定,初始化为2000

this.updateProjectionMatrix(); //调用updateProjectionMatrix方法,更新相机的投影矩阵.

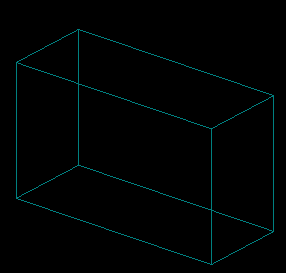
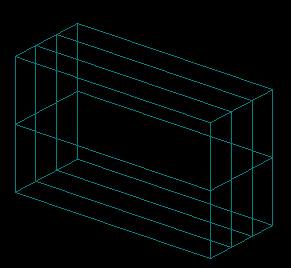
其中fov定义的是垂直视野，从底部到顶部，并以度为单位，水平视野是通过垂直视野计算出来的。相机位置点设置与正交投影照相机设置一样。

**（b）正交投影照相机**

通过var camera3 = new THREE.OrthographicCamera(-3, 2, 2, -2, 1, 10)方法，该方法设置六个参数（left，right，top，bottom，near，far），为保持照相机的横竖比例，需要保证(right - left)与(top - bottom)的比例与Canvas宽度与高度的比例一致。具体投影到屏幕显示的是下图灰色部分（称为视景体）。

camera2.position.set(4, 3, 5)：设置照相机的位置。将相机往右移动，物体（视景体）会往左移。如果将OrthographicCamera(left,right,top,bottom,near,far)方法中left往右移动（-3=》-1），物体（视景体）也会往左移。因为相机在平面上相对向右移动了。

camera2.lookAt(new THREE.Vector3(0, 0, 0))：指定照相机看向原点的方向，形成立体形状。

THREE.CubeGeometry(1, 1, 1)：绘制几何立方体，参数代表长，宽，高，还有其他3位参数表示长，宽，高分节的段数，可不写

THREE.CubeGeometry(2.5, 1.5, 1) THREE.CubeGeometry(2.5, 1.5, 1,1,2,3)

**（4）创建物体**：通过THREE.Mesh()方法创建物体（几何形状），Mesh对象,最终的网格对象,有高人把图形学建模比作是制作灯笼,先用Geometry创建灯笼的框架,然后将材质material贴在框架上,最后形成的总体灯笼,就是Mesh对象.下面看一下Mesh对象的用法和具体实现。

//创建几何物体

new THREE.Mesh(new THREE.SphereGeometry(1.9, 14, 18, Math.PI / 6, Math.PI), new THREE.MeshBasicMaterial({ color: 0xC0C0C0, wireframe: true }));

Geometry对象是需要创建的几何体，有立方体（THREE.CubeGeometry(width, height, depth, widthSegments, heightSegments, depthSegments)），球体（THREE.SphereGeometry(radius, segmentsWidth, segmentsHeight, phiStart, phiLength, thetaStart, thetaLength)），平面（THREE.PlaneGeometry(width, height, widthSegments, heightSegments)），圆形（THREE.CircleGeometry(radius, segments, thetaStart, thetaLength)），圆柱体（THREE.CylinderGeometry(radiusTop, radiusBottom, height, radiusSegments, heightSegments, openEnded)），圆环面（THREE.TorusGeometry(radius, tube, radialSegments, tubularSegments, arc)）。

#### 3、物体材质（Material）

材质（Material）是独立于物体顶点信息之外的与渲染效果相关的属性。通过设置材质可以改变物体的颜色、纹理贴图、光照模式等。

**（1）基本材质**

基本材质（BasicMaterial）的物体，渲染后物体的颜色始终为该材质的颜色，而不会由于光照产生明暗、阴影效果。

MeshBasicMaterial属性：

（a）wireframe：是否渲染线而非面，默认为false；

（b）color：十六进制RGB颜色，如红色表示为0xff0000；

（c）map：使用纹理贴图；

THREE.MeshBasicMaterial({color: 0xffff00, opacity: 0.75 })。

**（2）光照材质**

Lambert材质（MeshLambertMaterial）和Phong材质（MeshPhongMaterial）。光照材质需要设置光源，才能实现光照效果。

Lambert光照模型的主要特点是只考虑漫反射而不考虑镜面反射的效果，因而对于金属、镜子等需要镜面反射效果的物体就不适应，对于其他大部分物体的漫反射效果都是适用的。

//生成材质

var planeMesh = new THREE.MeshLambertMaterial({ color: 0x808080 });

color是用来表现材质对散射光的反射能力，还可以用ambient和emissive控制材质的颜色。

ambient表示对环境光的反射能力，只有当设置了AmbientLight后，该值才是有效的，材质对环境光的反射能力与环境光强相乘后得到材质实际表现的颜色。

emissive是材质的自发光颜色，可以用来表现光源的颜色。单独使用红色的自发光： new THREE.MeshLambertMaterial({     emissive: 0xff0000 })

Phong模型考虑了镜面反射的效果，因此对于金属、镜面的表现尤为适合。

//设置正方体

var cube = new THREE.CubeGeometry(1, 1, 1);

var cubeMaterial = new THREE.MeshLambertMaterial({ color: 0xFFB6C1 });

var cubeMesh = new THREE.Mesh(cube, cubeMaterial);

cubeMesh.castShadow = true;

cubeMesh.position.set(1, 0, 0);

scene.add(cubeMesh);

//设置长方体

var cube2 = new THREE.CubeGeometry(1, 2, 1);

var cube3 = new THREE.SphereGeometry(1, 16, 16);

var cubePhongMaterial = new THREE.MeshPhongMaterial({ color: 0xff0000, specular: 0xffff00, shininess: 1000 });

var cubeMeshPhong = new THREE.Mesh(cube2, cubePhongMaterial);

cubeMeshPhong.castShadow = true;

cubeMeshPhong.position.set(-3, -1, 0);

scene.add(cubeMeshPhong);

var cubeMeshPhong2 = new THREE.Mesh(cube3, cubePhongMaterial);

cubeMeshPhong2.castShadow = true;

cubeMeshPhong2.position.set(-5, -1, 0);

scene.add(cubeMeshPhong2);

**（3）法向材质**

法向材质可以将材质的颜色设置为其法向量的方向，有时候对于调试很有帮助。 法向材质的设定很简单，甚至不用设置任何参数： new THREE.MeshNormalMaterial()。材质的颜色与照相机与该物体的角度相关，只要改变照相机位置，可以观察两个角度的颜色变化。

var cube4 = new THREE.CubeGeometry(2, 2, 2);

var cubeNormalMaterial = new THREE.MeshNormalMaterial();

var cubeNormalMesh = new THREE.Mesh(cube4, cubeNormalMaterial);

cubeNormalMesh.castShadow = true;

cubeNormalMesh.position.set(1, -3, 0);

scene.add(cubeNormalMesh);

#### 4、网格（Mesh）

最常用的一种物体就是网格（Mesh），网格是由顶点、边、面等组成的物体；其他物体包括线段（Line）、骨骼（Bone）、粒子系统（ParticleSystem）等。

var material = new THREE.MeshLambertMaterial({ color: 0xffff00 });

var geometry = new THREE.CubeGeometry(1, 2, 3);

var mesh = new THREE.Mesh(geometry, material);

scene.add(mesh);

位置、缩放、旋转是物体三个常用属性。由于THREE.Mesh基础自THREE.Object3D，因此包含scale、rotation、position三个属性，可以单独设置某个方向轴的位置，如：mesh.position.z = 1;也可以同时设置：mesh.position.set(1.5, -0.5, 0)。其他两个属性设置方法和position一样。

#### 5、动画（Animation）

为了衡量画面切换速度，引入了每秒帧数FPS（Frames Per Second）的概念，是指每秒画面重绘的次数。FPS越大，则动画效果越平滑，当FPS小于20时，一般就能明显感受到画面的卡滞现象。

当FPS足够大（比如达到60），再增加帧数人眼也不会感受到明显的变化，反而相应地就要消耗更多资源。

**（1）setInterval（function，times）**：

setInterval()方法中times是以毫秒为单位，可设置时间，间隔重复调用绘图旋转方法。

//绘制圆环结

drawTorus();

var torus, renderer3, scene3, camera3,interval;

function initTorus() {

renderer3 = new THREE.WebGLRenderer({ canvas: document.getElementById("torusKnotCanvas") });

renderer3.setClearColor(0x000000);

scene3 = new THREE.Scene();

camera3 = new THREE.PerspectiveCamera(60, 1 / 1, 1, 100);

camera3.position.set(0, 0, 2);

scene3.add(camera3);

torus = new THREE.Mesh(new THREE.TorusKnotGeometry(0.5, 0.2, 100, 8), new THREE.MeshBasicMaterial({ color: 0x90EE90, wireframe: true }));

scene3.add(torus);

//间隔20毫秒调用torusAnimation方法

interval = setInterval(torusAnimation, 20);

}

function torusAnimation() {

torus.rotation.x += 0.02;

torus.rotation.y += 0.02;

renderer3.render(scene3,camera3);

}

function drawTorus() {

initTorus();

}

**（2）requestAnimationFrame(func)：**

requestAnimationFrame()方法可以告诉浏览器在合适的时候调用指定函数，通常可能达到60FPS，不需要指定调用方法的时间间隔。

由于requestAnimationFrame只请求一帧画面，因此，除了在init函数中需要调用，在被其调用的函数中需要再次调用requestAnimationFrame。

//绘制滚动球体

drawEarth();

var sphere, renderer2, scene2, camera2;

function drawEarth() {

initEarth();

animation();

}

function initEarth() {

renderer2 = new THREE.WebGLRenderer({ canvas: document.getElementById("sphereCanvas") });

//设置背景色，用于清除画面的颜色

renderer2.setClearColor(0x000000);

//添加场景

scene2 = new THREE.Scene();

//创建正交投影照相机

camera2 = new THREE.OrthographicCamera(-2, 2, -2, 2, 1, 10);

camera2.position.set(1.5, 4, 5);

camera2.lookAt(new THREE.Vector3(0, 0, 0));

scene2.add(camera2);

/\*THREE.SphereGeometry(radius(半径), segmentsWidth(经度上切片数), segmentsHeight（纬度上切片数）, phiStart（经度开始的弧度）, phiLength（经度夸过的弧度）, thetaStart, thetaLength)，设置了弧度，则前面设置的切片数是在弧度跨越的区域中进行切片\*/

sphere = new THREE.Mesh(new THREE.SphereGeometry(1.2, 14, 16), new THREE.MeshBasicMaterial({ color: 0x008080, wireframe: true }));

scene2.add(sphere);

renderer2.render(scene2, camera2);

}

function animation() {

requestAnimationFrame(animation);

//sphere.rotation.x += 0.02;

sphere.rotation.y += 0.03;

renderer2.render(scene2, camera2);

}

## 31、HTML5脚本编程

### （一）跨文档消息传递

跨文档消息传递(cross-document messaging)指的是来自不同域之间的消息传递（简称为XDM），如www.org.com与p2p. org.com两个页面通信。也有跨域的说法是要有两个不同的源。  
源 = 规则(http/https)+主机(localhost/其他域名)+端口号，只要有一个不一样，那么两个网页就是不同的源，在浏览器会实现javascript的同源策略，对于不同的源是不能通信的，但是HTML5实现了postMessage标准。

XDM的核心是postMessage()方法，目的是向另一个地方传递数据，对于XDM而言“另一个地方”指的就是包含在当前页面的<iframe>元素，或者当前页弹出的窗口。

postMessage()方法传递两个参数，第一个是“传递的消息”，第二个是“消息接收方来自哪个域”，第二个参数有效防止了信息泄露的问题，加强了信息安全。第二个参数也可以是“\*”，表示可以把消息发送给任何域的文档，不过不建议这样做。

获得frame或iframe的对象需要通过contentWindow属性，所有支持XDM的浏览器均支持contentWindow属性。contentWindow属性是指指定的frame或者iframe所在的window对象，当信息发送成功，需要验证发送信息地址的正确性。

var frame = document.getElementById("myIframe").contentWindow;

//发送信息“Hello World”到“http://www.baidu.com”

frame.postMessage("Hello World!", " http://www.org.com ");

//假设http://www.baidu.com接收到信息

EventUtil.addHandler(window, "message", function (event) {

//验证发送消息的域是已知的域

if (event.origin == " http://p2p. org.com ") {

//处理发送过来的数据

postMessage(event.data);

//发送回执(可选)

event.source.postMessage("Received", "http://p2p. org.com");

}

});

### （二）拖放事件

#### 1、设置被拖放元素

默认情况下图片，文本，链接都是可拖动的，不需要额外编写信息，部分元素不可拖动时，设置draggable属性为true，可实现拖动。这个拖动的是元素标签，不包含内容，所以需要通过dataTransfer.setData(format,data)来设置拖动的内容，在接收方设置dataTransfer.getData(format)接收内容。其中format表示data的数据类型，一般为字符串，可设为“text/plain”。接收方的getData(format)需要与setData(format)的一致。

var drag1 = document.getElementById("drag1");

var drag2 = document.getElementById("drag2");

var drag3 = document.getElementById("drag3");

var drag4 = document.getElementById("drag4");

var drag5 = document.getElementById("drag5");

EventUtil.addHandler(drag1, "dragstart", function (e) {

e.dataTransfer.setData('text/plain', 'drag1');

});

EventUtil.addHandler()为手动编写的添加监听事件的方法，可用window.addEventListener()代替。

setData()方法中的第二个参数在该方法中写的是被拖放元素的id，方便getData()时根据id获取标签元素。

当元素拖动时会触发dragstart，drag，dragend事件，当鼠标选中拖放元素并没有松开鼠标时，触发了dragstart事件，在该事件方法中设置拖放内容（setData）（e.dataTransfer.setData('text/plain', 'drag1')）。

#### 2、设置元素放置目标

放置元素到目标位置时会依次触发以下事件：dragenter，dragover，drop或者dragleave，所有元素都支持放置目标事件，但都默认不允许放置，所以需要重写dragenter，dragover事件的默认行为。

// 放置目标

var targetDiv = document.getElementById("container");

EventUtil.addHandler(targetDiv, "dragenter", function (e) {

EventUtil.preventDefault(e);

});

EventUtil.addHandler(targetDiv, 'dragover', function (e) {

EventUtil.preventDefault(e);

});

// 重写drop事件，处理获取的数据

EventUtil.addHandler(targetDiv, 'drop', function (e) {

var id = e.dataTransfer.getData('text/plain');

var src = document.getElementById(id);

var target = e.target;

target.appendChild(src);

});

## 32、JSON

JSON的语法可以表示以下三种类型的值：

**简单值**：字符串，数值，布尔值，null，JSON不支持JavaScript的特殊值undefined。

**对象：**对象是一种复杂数据类型，表示的是有一组有序的键值对，每个键值对的值可以是简单值，也可以是复杂数据类型的值。

**数组：**数组也是一种复杂数据类型，表示一组有序的值的列表，可通过数值索引访问其中的值，数组的值也可以是简单值、对象或数组。

### （一）三种类型值的介绍

#### 1、简单值

简单值是JSON数据形式最简单的，与JavaScript的表达形式差不多，但在字符串中的表达形式与JavaScript有区别：JSON字符串必须使用双引号（单引号会导致语法错误）。

#### 2、对象

JSON中的对象和JavaScript的对象字面量有些不同，JavaScript对象字面量：

var person = {

name: "Mark",

age:18 };

JSON对象字面量：

{

"name":"Mark",

"age":18

}

两者的差别是：JavaScript对象字面量中有声明变量，在末尾有添加分号（；）；JSON中没有声明变量，也没有在末尾添加分号（；），只是在属性名称上添加了双引号，注意必须是双引号，如果没写或者写成单引号也会报错。

#### 3、数组

在JavaScript和JSON中定义数组可以使用同样的语法：var values = [25,"hi",true]；JSON中也是没有变量，直接设置内容：[25,"hi",true]。

在JSON数组中同样没有变量和分号，把数组和对象结合起来可以构成更复杂的数据集合；

//创建js变量，以JSON格式赋值

var a = {

"name":"Mark",

"age": 18,

"family": ["father", "mother", "sister"],

"tel":13100000000

}

//判断obj是否为json对象

function isJson(obj) {

var isjson = typeof (obj) == "object" && Object.prototype.toString.call(obj).toLowerCase() == "[object object]" && !obj.length;

return isjson;

}

### （二）JSON对象方法

JSON对象有两个方法：stringify()和parse()，这两个方法分别用于把JavaScript对象序列化为JSON对象和把JSON对象字符串解析为原生JavaScript值。

如：JavaScript对象：

var a = {

"name":"Mark",

"age": 18,

"family": ["father", "mother", "sister"],

"tel":13100000000

}

//转换为JSON对象，并保存在jsonText变量里

var jsonText = JSON.stringify(a);

JSON.stringify()输出的JSON字符串不包含任何空格字符或缩进，基本如下：{ "name":"Mark","age": 18, "family": ["father", "mother", "sister"], "tel":13100000000 }

JSON.parse()方法将JSON数据转换为JavaScript数据类型，如：

//转换为JavaScript对象，并保存在jsText变量里

var jsText = JSON.parse(jsonText);

### （三）序列化选项

JSON.stringify()方法除了要序列化的JavaScript对象外，还可以接收另外两个参数，这两个参数可以指定使用不同的方式序列化JavaScript对象。第一个参数是个过滤器，可以是一个数组或函数；第二个参数是个选项，可以表示是否在JSON字符串中保留缩进。

#### 1、过滤结果

如果第二个参数是数组，则输出的结果将只包含数组中列出的属性。

例：//转换为JSON对象

var jsonText = JSON.stringify(a,["name","family"]);

结果：{"name":"Mark","family":["father","mother","sister"]}；

如果是函数，需要传入属性（键）名和属性值，根据属性（键）名显示属性值，其中属性名只能是字符串，而在值并非键值对结构的值时，键名可以是空字符串。最后，一定要提供default项，此时返回传入的值，如：var jsonText2 = JSON.stringify(a, function (key, value) {

switch (key) {

case "family":

return value.join(",");

break;

case "":

return 13022222222;

break;

default:

return value;

break;

}

},4);

结果：13022222222，不再是对象格式；

如果函数返回了undefined，那么相应属性会被忽略，在函数中未写键名的值也会输出，函数的设置可以改变相对应键名的值。

例：//创建js变量，以JSON格式赋值

var a = {

"name":"Mark",

"age": 18,

"family": ["father", "mother", "sister"],

"tel":13100000000

}

var jsonText2 = JSON.stringify(a, function (key, value) {

switch (key) {

case "family":

return value.join(",");

break;

case "age":

return 20;

break;

case "tel":

return undefined;

break;

default:

return value;

break;

} });

结果：{"name":"Mark","age":20,"family":"father,mother,sister"}//修改了age的值，由18改为20，tel值忽略了，未显示。

#### 2、字符缩进

JSON.stringify()除了对象之外的第二个参数（即整个方法的第三个参数）用于控制结果中的缩进和空白符。如果参数是一个数值代表每个级别缩进的空格数，如果是字符串，则表示用字符串代表缩进字符。不管是数值还是字符串，都有最长限度，数值时不超过10，大于10的自动转换为10；字符串最长不超过10个字符长，超过的话结果中将只出现前10个字符。

例：var jsonText2 = JSON.stringify(a, function (key, value) {

switch (key) {

case "family":

return value.join(",");

break;

case "age":

return 20;

break;

case "tel":

return undefined;

break;

default:

return value;

break;

}

},4);

结果：{

"name": "Mark",

"age": 20,

"family": "father,mother,sister"

} //方法中设置了空格4个字符长度

//设置字符串为空格符

var jsonText2 = JSON.stringify(a, function (key, value) {

switch (key) {

case "age":

return 20;

break;

case "tel":

return undefined;

break;

default:

return value;

break;

} },'--');

//方法中设置了字符串为空格符，且没有修改family的值为字符串，所以输出为数组格式

结果：{

--"name": "Mark",

--"age": 20,

--"family": [

----"father",

----"mother",

----"sister"

--]

}

#### 3、toJSON()方法

有时候JSON.stringify()不能满足对某些对象进行自定义序列化的需求，可以通过对象上调用toJSON()方法，返回其自身的JSON数据格式。使用toJSON方法后，这个对象也将被序列化为一个简单的字符串而非对象。

例：var b = {

"name": "Mark",

"age": 18,

"family": ["father", "mother", "sister"],

toJSON: function () {

return this.name;

},

"tel": 13100000000

}

结果："Mark"。//只返回了toJSON指定键名的值

### （四）解析选项

解析选项即将JSON格式数据转换为JavaScript格式数据，使用JSON.parse()方法除了添加JSON对象外，还可添加另一个参数，该参数是一个函数，将在每个键值对上调用，该函数称为还原函数。

该函数也接收两个参数，key和value，而且都需要返回一个值，如果还原函数返回undefined，表示从结果中删除相应的键，如果返回其他值，则将该值插入（替换）到结果中。

例：var a = {

"name":"Mark",

"age": 18,

"family": ["father", "mother", "sister"],

"tel":13100000000

}

//转换为JavaScript对象，并修改键名tel的值

var jsText2 = JSON.parse(jsonText3, function (key, value) {

if (key == "tel") {

return "13222222222";

}

else {

return value;

}

});

结果：数组格式：{name: "Mark", age: 18, family: Array(3), tel: "13222222222"}

## 33、Ajax与Comet

Ajax技术的核心是XMLHttpRequest对象（简称XHR）。

### （一）、XMLHttpRequest对象

#### 1、XHR的用法

在使用XHR对象时，调用的第一个方法是open，它接受3个参数：要发送的请求的类型（“get”、“post”等）、请求的URL和表示是否异步发送请求的布尔值，如：xhr.open("get","example.html",false)。

注意：只能向同一个域中使用相同端口和协议的URL发送请求，如果URL与启动请求的页面有任何差别，都会引发安全错误。

当发送了特定请求之后，必须调用send()方法，传递请求主体发送的数据，如果不需要通过请求主体发送数据，为兼容某些浏览器，则必须传入null。

在收到响应后，响应的数据会自动填充CHR 对象的属性，相关属性简介如下:

（1）responseText：作为响应主体被返回的文本。

（2）responseXML：如果响应的内容类型是“text/xml”或“application/xml”。这个属性中将保存包含着响应数据的XML DOM文档。

（3）status：响应的HTTP状态。

（4）statusText：HTTP状态说明。

在接收到响应后，第一部是检查status属性，以确定响应已经成功返回。当状态码为200时表示成功，为304时表示请求的资源没有被修改，可以直接使用浏览器中缓存的版本。

除了同步请求外还有异步请求，异步可以先检测readyState属性值，该属性表示请求/响应霍城的当前活动阶段，属性返回的值如下：

（1）0：未初始化。尚未调用open()方法；

（2）1：启动。已经调用open()方法，尚未调用send()方法；

（3）2：发送。已经调用send()方法，但尚未接收到响应

（4）3：接收。表示已经接收部分响应数据。

（5）4：完成。已经接收全部响应数据，且可以在客户端使用。

只要readyState属性的值由一个值变成另一个值，就会触发一次readystatechange事件。

var xhr = createXHR();

xhr.open("get","index.html",, true);

xhr.send(null);

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

//判断当前接收内容的响应状态

if ((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304) {

var text = xhr.responseText;

document.getElementById("name").innerHtml = text ;

alert("数据接收成功：" + xhr.responseText);

} else {

alert("数据接收失败：" + xhr.status);

}

}

};

//创建XMLHttpRequest对象方法

function createXHR() {

if (typeof XMLHttpRequest != "undefined") {

return new XMLHttpRequest();

}

else if (typeof ActiveXObject != "undefined") {

if (typeof arguments.callee.activeXString != "string") {

var versions = ["MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0", "MSXML2.XMLHttp"], i, len;

for (i = 0, len = versions.length; i < len; i++) {

try {

new ActiveXObject(versions[i]);

arguments.callee.activeXString = versions[i];

break;

}

catch (ex) {

console.log("创建XMLHttpRequest（）对象失败：" + ex.message);

}

}

}

return new ActiveXObject(arguments.callee.activeXString);

}

else {

throw new Error("该浏览器不支持XMLHttpReques对象！");

}

}

#### 2、HTTP头部信息

发送XHR请求时，默认情况是发送一下头部信息：

（1）Accept：浏览器能够处理的内容类型；

（2）Accept-Charset：浏览器能够显示的字符集；

（3）Accept-Encoding：浏览器能够处理的压缩编码；

（4）Accept-Language：浏览器当前设置的语言；

（5）Connection：浏览器与服务器枝江连接的类型；

（6）Cookie：当前页面设置的任何Cookie；

（7）Host：发出请求的页面所在的域；

（8）Referer：发出请求页面的URL；

（9）User-Agent：浏览器的用户代理字符串。

要成功发送请求头部信息，必须在调用open()方法之后且调用send()方法之前调用setRequestHeader();

如：var xhr = createXHR();

var b = "{'name':'jack','sex':'men','age':23}";

var url = "JSONHtml.html";

xhr.open("get", url, true);

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send("url=" + b);

#### 3、GET请求和POST请求

**Get方式:**

用get方式可传送简单数据，但大小一般限制在1KB下，数据追加到url中发送（http的header传送），也就是说，浏览器将各个表单字段元素及其数据按照URL参数的格式附加在请求行中的资源路径后面。另外最重要的一点是，它会被客户端的浏览器缓存起来，那么，别人就可以从浏览器的历史记录中，读取到此客户的数据，比如帐号和密码等。因此，在某些情况下，get方法会带来严重的安全性问题。

**Post方式:**

当使用POST方式时，浏览器把各表单字段元素及其数据作为HTTP消息的实体内容发送给Web服务器，而不是作为URL地址的参数进行传递，使用POST方式传递的数据量要比使用GET方式传送的数据量大的多。

总之，GET方式传送数据量小，处理效率高，安全性低，会被缓存，而POST反之。

**使用get方式需要注意：**  
      对于get请求（或凡涉及到url传递参数的），被传递的参数都要先经encodeURIComponent方法处理。例：var url = "update.php?username=" +encodeURIComponent(username) + "&content=" +encodeURIComponent(content)+"&id=1" ;

**使用Post方式需注意：**

1、设置header的Context-Type为application/x-www-form-urlencode确保服务器知道实体中有参数变量。通常使用XmlHttpRequest对象的SetRequestHeader("Context-Type","application/x-www-form-urlencoded;")。例：xmlHttp.setRequestHeader("Content-Type","application/x-www-  form-urlencoded");

2、参数是名/值一一对应的键值对,每对值用&号隔开.如 var name=abc&sex=man&age=18，注意var name=update.php?abc&sex=man&age=18以及var name=?abc&sex=man&age=18的写法都是错误的;

3、参数在Send(参数)方法中发送,例： xmlHttp.send(name); 如果是get方式，直接 xmlHttp.send(null);

4、服务器端请求参数区分Get与Post。如果是get方式则$username = $\_GET["username"]; 如果是post方式，则$username = $\_POST["username"];

**Post和Get 方法有如下区别:**

1、Post传输数据时，不需要在URL中显示出来，而Get方法要在URL中显示。

2、Post传输的数据量大，可以达到2M，而Get方法由于受到URL长度的限制,只能传递大约1024字节。

3、Post顾名思义,就是为了将数据传送到服务器段，Get就是为了从服务器段取得数据。而Get之所以也能传送数据,只是用来设计告诉服务器，你到底需要什么样的数据。Post的信息作为http请求的内容，而Get是在Http头部传输的。

get 方法用Request.QueryString["strName"]接收

post 方法用Request.Form["strName"] 接收

**注意：**

虽然两种提交方式可以统一用Request("strName")来获取提交数据，但是这样对程序效率有影响，不推荐使用。

一般来说，尽量避免使用Get方式提交表单，因为有可能会导致安全问题。

### （二）、进度事件

ProgressEvents定义了与客户端服务器通信有关的事件，主要有以下6个进度事件：

（1）loadstart：在接收到响应数据的第一个字节时触发；

（2）progress：在接收响应期间持续不断地触发；

（3）error：在请求发生错误时触发；

（4）abort：终止事件，在因为调用abort方法而终止连接时触发；

（5）load：在接收到完整的响应数据时触发。

（6）loadend：在通信完成或触发error、abort或load事件后触发。

#### 1、progress事件

onprogress事件处理程序会接收到一个event对象，其target属性是XHR对象，其中包含三个属性：lengthComputable、position、totalSize。其中lengthComputable是一个表示进度信息是否可用的布尔值，position表示已接收的字节数，totalSize表示根据Content-Length响应头部确定的预期字节数。以上是旧版属性，新版改为loaded表示已接收字节数，total表示接收总字节数，loaded/total表示进度的百分比。

progress事件返回进度信息可分成上传和下载两种情况，下载的progress事件属于XMLHttpRequest对象，上传的progress事件属于XMLHttpRequest.upload对象。

//下载进度实现

xhr.onprogress = downloadProgress;

function downloadProgress(event) {//未测试

if (event.lengthComputable) {

var percentComplete = event.loaded / event.total;

console.log("总计：" + t + "，当前进度百分比：" + percentComplete + "%");

}

}

### （三）、跨源资源共享

CORS(Cross-Origin Resource Sharing，跨源资源共享)，其背后的基本思想，就是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功或失败。

使用跨域的时候，如果需要使用跨域请求返回的内容，则需添加响应头 Access-Control-Allow-Origin:\* ，\* 代表了允许任何域使用响应的内容。

#### 1、IE对CORS的实现

微软在IE8中引入了XDR（XDomainRequest）类型，XDR对象的使用方法和XHR对象相似，也是创建一个XDomainResquest的实例，调用open，方法，再调用send方法，但是XDR的open方法只接收两个参数：请求类型，url。

所有的XDR都是异步执行的，不能用来创建同步请求。请求返回后，会触发load事件，响应的数据会保存在responseText属性中。在接收到响应后，你只能访问响应的原始文本，没法确定响应的状态代码。如果失败会触发error事件（包括响应中缺少Access-Control-Allow-Origin头部）。

其他浏览器对XMLHttpRequest对象实现了原生支持。

检测XHR是否支持CORS的最简单方式，就是检查是否存在withCredentials属性(携带证书)，再结合检测XDomainRequest对象是否存在，就可以兼顾所有浏览器。

### （四）、其他跨域技术

#### 1、图像Ping

图像Ping是与服务器进行简单、单向的跨域通信的一种方式。请求的数据是通过查询字符串形式发送的，而相应可以是任意内容，但通常是像素图或204响应（服务器处理了客户端的请求，但是没有返回任何内容）。

var img = new Image();

img.onload = img.onerror = function () {

alert("加载成功！");

}

img.src = "http://www.test.com/person?name=Mark";

图像Ping最常用于跟踪用户点击页面或动态广告曝光次数。图像Ping主要有两个缺点：一是只能发送get请求，二是无法访问服务器的响应文本。

#### 2、JSONP

JSONP是JSON with padding（填充式JSON或参数是JSON）的简写。JSONP看起来与JSON差不多，只不过是被包含在函数调用中的JSON，如：callback({"name":"Mark"});//callback函数返回name数据

JSON被js原生支持，所以JSONP可以运用script标签调用跨域资源，可以通过外部文件引用或者通过动态<script>元素，为src属性指定跨域URL。如：

var script = document.createElement("script");

script.src = "http://freegeoip.net/json/?callback=handleResponse";

document.body.insertBefore(script, document.body.firstChild);

或者：<script type="text/javascript" src="http://freegeoip.net/json/?callback=handleResponse"></script>

为了便于客户端使用数据，逐渐形成了一种非正式传输协议，人们把它称作JSONP，该协议的一个要点就是允许用户传递一个callback参数给服务端，然后服务端返回数据时会将这个callback参数作为函数名来包裹住JSON数据，这样客户端就可以随意定制自己的函数来自动处理返回数据了。

JSONP返回的数据都是JavaScript代码，且是自动生成，根据本地调用函数返回指定数据。如：

function handleResponse(response) {

alert("你的IP是："+response.ip+"，城市是："+response.city+",行政区（省份）："+response.region\_name);

//console.log(response);//输出所有返回数据

}

var script = document.createElement("script");

script.src = "http://freegeoip.net/json/?callback=handleResponse";

document.body.insertBefore(script, document.body.firstChild);

远程数据资源是http://freegeoip.net/json/，传递参数callback（默认参数名），调用本地函数是handleRequest()，返回的数据会包裹在函数的返回参数response中，若不知道对象的名称，如何调用显示指定数据时，可直接打印输出，根据远程js返回的数据，编写js代码，如上面的函数返回的全体数据是：handleResponse({ "ip": "113.118.226.147", "country\_code": "CN", "country\_name": "China", "region\_code": "44", "region\_name": "Guangdong", "city": "Shenzhen", "zip\_code": "", "time\_zone": "Asia/Shanghai", "latitude": 22.5333, "longitude": 114.1333, "metro\_code": 0 });

返回JSON数据对象，这样可以直接用response.ip来获取数据。

也可以用jQuery获取数据，引入jquery类库后，可直接用ajax方法获取，

jQuery(document).ready(function () {

$.ajax({

type: "get",

async: false,

url: "http://freegeoip.net/json/",

dataType: "jsonp",

jsonp: "callback",//传递给请求处理程序或页面的，用以获得jsonp回调函数名的参数名(一般默认为:callback)

jsonpCallback: "handleResponse",//自定义的jsonp回调函数名称，默认为jQuery自动生成的随机函数名，也可以不写，jQuery会自动为你处理数据

success: function (response) {

alert("你的IP是：" + response.ip + "，城市是：" + response.city + ",行政区（省份）：" + response.region\_name);

},

error: function () {

alert('fail');

}

});

});

## 34、高级技巧

### （一）、惰性载入函数

惰性实例化所要解决的问题是这样的：避免了在页面中js初始化执行的时候就实例化了类，如果在页面中没有使用到这个实例化的对象，那么这就造成了一定的内存的浪费和性能的消耗，那么如果将一些类的实例化推迟到需要使用它的时候才开始去实例化，那么这就避免了刚才说的问题，做到了“按需供应”。

有两种提供惰性载入的方式，

**1、在函数被调用时再处理函数。**

**原始函数：**

//创建XMLHttpRequest对象方法

function createXHR() {

if (typeof XMLHttpRequest != "undefined") {

return new XMLHttpRequest();

}

else if (typeof ActiveXObject != "undefined") {

if (typeof arguments.callee.activeXString != "string") {

var versions = ["MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0", "MSXML2.XMLHttp"], i, len;

for (i = 0, len = versions.length; i < len; i++) {

try {

new ActiveXObject(versions[i]);

arguments.callee.activeXString = versions[i];

break;

}

catch (ex) {

console.log("创建XMLHttpRequest（）对象失败：" + ex.message);

}

}

}

return new ActiveXObject(arguments.callee.activeXString);

}

else {

throw new Error("该浏览器不支持XMLHttpReques对象！");

} }

**惰性载入函数：**

//创建XMLHttpRequest对象方法

function createXHR() {

if (typeof XMLHttpRequest != "undefined") {

createXHR = function () {

return new XMLHttpRequest();

};

}

else if (typeof ActiveXObject != "undefined") {

createXHR = function () {

if (typeof arguments.callee.activeXString != "string") {

var versions = ["MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0", "MSXML2.XMLHttp"], i, len;

for (i = 0, len = versions.length; i < len; i++) {

try {

new ActiveXObject(versions[i]);

arguments.callee.activeXString = versions[i];

break;

}

catch (ex) {

console.log("创建XMLHttpRequest（）对象失败：" + ex.message);

}

}

}

return new ActiveXObject(arguments.callee.activeXString);

};

}

else {

createXHR = function () {

throw new Error("该浏览器不支持XMLHttpReques对象！");

};

}

return createXHR();

}

**2、声明函数时就指定适当函数。**

var createXHR = function () {

if (typeof XMLHttpRequest != "undefined") {

return function () {

return new XMLHttpRequest();

}

}

else if (typeof ActiveXObject != "undefined") {

return function () {

if (typeof arguments.callee.activeXString != "string") {

var versions = ["MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0", "MSXML2.XMLHttp"], i, len;

for (i = 0, len = versions.length; i < len; i++) {

try {

new ActiveXObject(versions[i]);

arguments.callee.activeXString = versions[i];

break;

}

catch (ex) {

console.log("创建XMLHttpRequest（）对象失败：" + ex.message);

}

}

}

return new ActiveXObject(arguments.callee.activeXString);

};

}

else {

return function () {

throw new Error("该浏览器不支持XMLHttpReques对象！");

};

} }