第一章 javascript简介

在早期JS主要用于一些输入验证工作，即不用将表单发送到服务器就可以验证表单的正确性，现在JS发展成为与浏览器窗口等进行复杂交互和计算的强大功能的语言，成为web的一部分。

JS作为一种客户端脚本语言出现，它的标准是ECMAScript，简称ES，以后各浏览器就将ES作为各自实现JS的标准。

JS由ES、BOM，DOM三部分组成，ES是核心，由ECMA-262定义

2008年，五大主流浏览器都实现了与ECMA-262兼容，即支持ES中的所有内容。

DOM文档对象模型

DOM是针对XML\HTML的应用程序接口（API），通过这个接口开发人员掌握了页面内容和结构的主动权，可以对页面进行修改。

BOM浏览器对象模型：提供与浏览器交互的方法。

第二章  在html中使用javascript

在页面中使用JS，既要保证与HTML页面共存，又要不影响页面的呈现效果。

script元素的几个属性：src、type、defer、async

1.<script type="text/javascript"  src="exam.js">

</script>包含外部文件

<script type="text/javascript" >

。。。。代码

</script>嵌入代码

text/javascript 是表示脚本语言的内容类型（MIME类型）。可以省略。

在浏览器对页面中的JS代码所有求值结束之前，页面中的其他内容都不会加载或显示。、

JS文件既可以直接嵌套在HTML中，也可以用src包含外部文件，甚至可以包含外部域的一个JS文件的URL。

如果有多个js文件，程序会根据js文件出现的先后顺序来对他们进行解析。除非用defer、async属性

async：表示应该立即下载脚本，但是不妨碍页面中的其他操作

defer：脚本可以延迟到文档完全被解析和显示之后进行

2. <script>标签的位置

一般放在head中，但是当程序比较多时，页面会出现延迟，一片空白，所以放在body内容的后面比较好。

延迟脚本：

加defer属性

先下载程序，然后在页面加载之后再按照顺序执行程序。

有defer属性之后，就可以将标签放在head中了，表示浏览器会立即下载脚本，但是延迟执行。只适用于外部脚本。defer

异步脚本：

加async属性

表示程序加载和执行和页面的显示同步进行，互不影响，先下载完的js程序先执行，会打乱顺序，先加载完的先执行。

3.嵌入代码与外部文件

推荐使用外部代码的原因：

可维护性，可缓存性（提升速度）、适应未来。

4.文档模式

加入<！doctype html>标准之后，告诉浏览器我在使用什么版本的HTML，以便浏览器能够按照标准来呈现CSS样式和解析JS程序，减少浏览器之间的不一致性。

5.<noscript>元素

当页面不支持js时，noscript中的内容可以代替js的内容在页面上显示。

第三章 基本概念

1.ECMAScript与javascript之间的关系

ECMAScript 是针对脚本语言指定的一个国际性的标准，javascript是针对这个标准进行的实现，还有其他的脚本语言也用这个标准。

ECMAScript 语言区分大小写，标识符以下划线、英文字母或美元符号开头，后面可以跟数字、字母、下划线和美元符号，起名字一般采用驼峰形式，即每个单词的首字母大写。

注释：

单行注释：//

多行注释：/\*

                 \*/

2.严格模式

“use strict”严格模式是ECMAScript 5引入的，用于指出代码中错误或者不安全的做法，提高效率和安全度，使得程序更加符合标准。告诉引擎切换到严格模式。

关键字和保留字不能用作标识符。

严格模式下不可以给未经声明的变量的赋值用作全局变量，m="hi";

八进制字面量在严格模式下无效

严格模式下不允许使用with语句

3 变量

变量是松散类型的，即一个变量可以存储任何一种类型的数据，而且可以改变变量存储值的类型。

4.一个数据的类型

数据类型：Undefined、Null、Boolean，Number和String。

typeof操作符可以用来检测给定变量的数据类型。

Undefined类型表示声明了变量但是没有初始化，只有一个值undefined

var m1;未初始化的变量m1==undefined;ture

var m2 未声明的变量

对于两者用typeof以后都显示undefined，所以最好初始化变量来区分未声明的和未初始化的变量。

引入该类型的目的是区分空对象引用和未初始化变量两者。

Null类型只有一个值null

表示一个空的对象指针，typeof返回object

如果想让一个变量来保存对象，就应该将一个变量显式初始化为null，这样可以体现将null作为空对象指针的惯例，而且可以区分null和undefined。

Boolean类型有两个值ture和false

是ECMAScript中用的最多的类型，其他类型的值都存在与true和false相等的值。

将其他类型的值转化为布尔值可以用Boolean函数，string类型中的空字符转化为false，其他转化为ture，number类型中的0或NaN转化为false，其他为ture。undefined转化为false，object类型中的null转化为false。

可以用在if语句中来控制流。

Number类型

浮点数值保存是整数保存空间的两倍，number.MIN\_VALUE和number.MAX\_VALUE保存了js中最大和最小的值。超过就会保存为+infinity和-infinity正无穷、负无穷。

isFinite()函数用来检验数值是否有效。

NaN是非数值，isNaN用来检测变量是不是数值或者可不可以被转换成数值。数字和布尔类型和数字形式的字符串都可以转换成数值。

数值转换：将非数值转化为数值，Number函数、parseInt函数和parseFloat函数。

Number（）函数如何处理布尔值、null值、undefined（NaN）值和字符串？

parseInt函数和parseFloat函数分别用来处理字符串中的整数和浮点数。

String类型

转义序列表示一个字符，字符串可以用双引号或者单引号。

表示字符串，可以用length属性取得其长度，但是只对由两个字节组成的字符有用，由四个字节组成的字符不能返回其长度。

字符串是无法改变的，即一旦创建了字符串变量想要改变的话必须先销毁原来的字符串，再用新的字符串填充这个变量。效率低

将其他数据类型转化为字符串形式，用age.toString()方法，一般不需要参数。将其他类型的值转换为用字符串表示的形式，不能用于null和undefined。

也可以用String函数，String（num），适用于所有类型，适用于null和undefined。

将一个值转化为字符串，也可以将该值用加号与字符串加在一起。

toString（）方法也可以传入参数表示输出数值的基制。

Object类型（可继承）

Object类型是一个抽象的对象，拥有自带的属性和方法，通过它创建的对象实例都拥有这些属性和方法并可以自己加属性和方法。

创建对象var o=new object();

Object类型的每个实例都有的属性：

1.constructor：构造函数，该属性指向实例对象的构造函数

2.hasOwnProperty(property name):用于检查给定的属性是否在该对象实例的属性中。

3.isPrototypeof(object)：用于检查传入的对象是否是当前对象的原型。

4.propertyIsEnumberable(property name):用于检查给定的属性是否可以用for-in语句来枚举。

5.toLocaleString():返回对象的字符串表示

6.toString()：返回对象的字符串表示

7.valueOf()：返回对象的字符串、数值或者布尔表示。

这些属性和方法是所有对象都有的。

5.操作符

操作符适用于很多值，例如字符串、布尔值、数字和对象，不过对对象进行操作时一般要调用其valueOf（）方法或者toString（）方法以便取得可操作的值。

一元操作符：前置++、--和后置++、--

不仅可以用到数值上，也可以用到其他数据类型，原则是先转换为数值类型再进行加减法。

一元+和一元-操作符：

num=+num;一元加操作符是将变量转化为数值类型

num=-num;一元减是将变量转化为数值类型再转化为负数。

位操作符

位操作是在二进制的基础上进行操作的，按位非（~），按位与（&）和按位或（|），左移（<<）、有符号的右移（>>）、无符号的右移（>>）。

布尔操作符

逻辑非（！）逻辑与（&&）、逻辑或（||）

相等操作符（==）和全等操作符（===）

前者先转换再比较，后者只比较不转换即“55”===55为false。

条件操作符（N1>N2）?N1：N2；

逗号操作符，返回最后一项，或者定义多个变量。

6.流控制语句

for、do while，while，if语句

if语句：ES会调用Boolean函数将if括号中的语句结果转换成布尔值，继而执行不同的分支。

for-in语句：用来枚举对象中的属性，保证对象不是null或者undefined

label语句：在代码中添加标签，可以与break和continue语句配合使用。

break语句：跳出整个循环

continue语句：跳出一次循环

with语句：将代码限制在一个对象中使用。

switch语句：可以在switch语句中使用任何值，字符串、对象都可以。

7.函数

函数参数：

函数不介意传递进来几个参数，也不介意传进来的是什么类型，因为传递进来的参数是用一个数组来保存的，并且这个数组可以用arguments对象来访问，arguments[0]等来代表传递进来的参数，所以函数中的参数不是必须的，只是为了提供便利。

arguments对象是一个特殊的对象，其实是当前函数的一个内置属性，类似于数组但是不是Array类型的实例。

通过访问arguments对象的length属性就可以获知有几个参数被传递进来。

arguments对象可以与命名参数一起使用，他的值永远与命名参数的值保持一致，arguments对象的长度与实参而不是形参有关。没有被赋值的形参将自动被赋值undefined。

函数无法实现重载，但是可以根据传入函数中参数数量的不同做出不同的反映，弥补不能重载的缺陷。

 函数重载要通过参数的类型和数量不同来区分，但是ES关于参数是一个数组，无法区分。

第四章 变量、作用域和内存问题

4.1.基本类型和引用类型

基本类型表示五种基本的数据类型，是按值进行访问的。var a=3;变量就是一个值

引用类型是指由多个值组成的对象，是按引用进行访问的，变量名o实际上是一个指针，指向保存对象的内存空间。var o=new Object;

javascript不允许直接访问内存中的位置,所以是在操作引用而不是对象本身。

访问对象时其实是在访问对象的引用而不是对象本身。

1）动态的属性：

我们可以给引用类型的值添加属性和方法，还可以删除其属性和方法，但是不可以对基本类型的值这样做。

2）复制变量值

基本类型复制变量时，两个变量是独立的，互不影响。

引用类型复制变量时，两个变量指向同一个对象，改变其中一个，另一个也会改变。

3）传递参数

所有的参数都是按值传递的，访问变量有按值和引用两种方法，但是传递参数却只有按值传递一种。

在向参数传递引用类型的值时，会将这个值在内存中的地址复制给一个局部变量，所以局部变量的变化会反应在函数的外部。

说明：将对象作为参数传递时，也是按值传递的，传递的是其指向的对象的内存地址，使得函数内外的两个变量都指向该对象，所以函数内部修改了参数的值也会反映在外部变量。

假如是按引用传递的话，就是将原本指向对象的实参现在指向形参。

4）检测类型

检测基本数据类型时用typeof可以。

检测引用数据类型时只会显示是对象，但是不会显示是什么类型的对象，用instanceof操作符来检测是什么类型的对象。

person instanceof Object

person instanceof Array

如果满足就会输出ture，不满足输出false

4.2.执行环境及作用域

执行环境也称环境，定义了变量或者函数有权访问的其他数据，每一个环境都有一个与之关联的对象，环境中定义的变量和函数都包含在这个对象中，最外层的环境称为全局环境，对应的对象是window对象，所有的全局变量和函数都是作为window对象的属性和方法创建的。

说明：每一个环境都有对应的变量对象，JS中的全局环境和局部环境是由函数来划分的。

作用域链：作用域链是一层层嵌套的执行环境，作用域链中的第一个对象是函数所在的活动对象，下一个变量对象来自其包含环境，直到全局执行环境。

内部环境可以通过作用域链访问所有的外部环境，而外部环境却不可以访问内部环境所有的变量和函数。

延长作用域链

当遇到with语句或者try-catch语句时，所包含的变量对象就被添加到作用域链的前端，就可以延长作用域链。

没有块级作用域

JS没有块级作用域，即在for循环中定义的变量，在循环结束时仍然可以调用，因为还在一个执行环境中。循环不是一个执行环境。

在一个函数中定义的变量称为局部变量，在函数调用结束后就不能调用。

块级作用域就是函数。

4.3.垃圾收集

JS会周期性的找出不再使用的变量，然后释放其内存，实现内存的自动管理。

垃圾收集器会跟踪所有的变量分辨其是否有用，对于没用的变量做上标记，以便于以后清除它。

标记变量的两种方式：标记清除和引用计数

标记清除：

垃圾收集器会将内存中的变量全部加上标记（可以用任何方式标记），然后去掉环境中的变量的标记和被环境中的变量引用的变量的标记，还有标记的变量就可以进行清除工作了。

引用计数：

跟踪记录每个值被引用的次数，当引用数减到0，就可以清除内存。但是当遇到循环引用时就会出现错误，因为引用值不可能为0

4.4.性能问题

垃圾收集的周期也是一个重要的问题。也可以即时进行垃圾收集。

管理内存：

JS分配给web浏览器的内存一般少于分配给桌面应用程序的，目的是为了防止JS的网页耗尽内存导致系统崩溃。内存限制问题会影响给变量分配内存以及一个线程能包含的语句数量，所以及时的释放内存很重要。

解除引用：用于释放全局变量，局部变量会自动清除。将全局变量设置为null。

第五章 引用类型（与基本数据类型相对应）

所有基本类型的值都不能调用方法，当读取基本类型的值时后台会创建一个对象，就可以用方法了。

对象是某个特定原生引用类型的实例，比如说Object引用类型和Array引用类型，已经定义好了基础的属性和方法，也可以自己添加。

1.Object类型（相当于结构）

创建Object实例的方法：

var o=new Object;   new操作符和Object构造函数

o.name="dsakd";

var o={

    name:"nisdjka",

    age:20

};对象字面量表示法

var o={};创建一个只包含默认属性和方法的对象

o.name="dsakd";

一般常用对象字面量法，而且该方法适用于向函数传递大量的可选参数，直接传递花括号的内容。

访问对象属性的两种方法：

点表示法和方括号表示法

o.name和o["name"]  方括号使用法可以用在属性名有空格、靠变量来访问属性等特殊情况下，否则一般用点表示法。

2.Array类型（数组类型）

ES数组的特点：数组中的每一项可以保存任何类型的值，且数组可以动态调整大小。

创建数组两种方法：

var p=new Array();   var p=new Array(3);  var p=new Array("hdsja","fas","fafa");

var p=["dsah","das","dasda"];  使用数组对象字面量创建对象时，不会调用构造函数

调用数组：方括号中的索引p[0]

length属性作用

不是只读的，是可以改变的，可以用来删除元素或者添加元素，每次数组变化之后，length属性都会更新，等于最大索引值加一。

var p=["dsah","das","dasda"];

p.length=2;

alert(p[2]);undefined ,删除元素

var p=["dsah","das","dasda"];

p.length=4;

alert(p[3]);undefined ,添加新项

var p=["dsah","das","dasda"];

p[p.length]="gyhgh";长度等于3

p[p.length]="dasdf"; 长度等于4，添加新项

检测数组：检测一个对象是不是数组？

instanceof 适用于只有一个全局环境的时候，当网页包含多个框架时，不同的框架有不同的Array构造函数，有可能显示false

Array.isArray(value)则适用于所有情况，适用于存在两个以上不同的全局执行环境。

转换方法

数组继承的valueOf方法返回数组本身的字符串形式

toString()方法返回每一个数组项的字符串形式组成的字符串，实际上是调用每一项的toString（）方法

toLocaleString()方法在默认的情况下都会返回以逗号分隔的字符串的形式，调用每一项的toLocaleString（）方法

而直接将数组传递给alert，会调用对象的toString（）方法。

而join方法则会返回以不同分隔符隔开的字符串的形式，p.join("||").

注：Array类型的valueOf（）方法、toString()方法和join（）方法都返回字符串形式的数组项。

栈方法（后进先出）

可以让数组的行为表现的像栈一样，推入（push方法）和移除（pop方法）是数组默认的方法，也可以与数组其他方法一起用。

p.push（"red","blue"）推入方法可以接受任意数量的参数，将他们逐个添加到末尾，并返回数组的长度。

p.pop（）移除方法，从数组末尾移除一项，减少length的值，并返回删除的一项的值。

队列方法（先进先出）

可以让数组的行为表现的就像队列一样，推入（push方法）和前端移除（shift方法）是数组默认的方法。unshift和pop也可以。

p.shift();表示将数组中的首项移除并返回该值，将length减一。

重排序方法

reverse（）方法会反转数组的值，sort（）方法是以升序的方法来排列数组，但是它对每一项都调用toSrting（）方法，所以是按照字符串的方法进行比较的（即使是数字数组），所以数字值之间的比较会出现问题。

解决方法是sort()方法可以接受一个比较函数compare()作为参数，以便指示哪个值大哪个值小。values.sort(compare);

数组项操作方法

concat（）方法：基于当前数组创建一个新数组，并返回该新数组，当传递给该方法的是一个或多个参数，就将这些参数都加在原数组末尾构成新数组，当传递给该方法一个或多个数组时，就将该数组的每一项加到原数组中，构成新数组，但是原数组不变。

slice（）方法：基于当前数组的一个或多个项来构成一个新数组，相当于切片。该方法可以接受一个或两个参数，一个参数（1）表示取出从指定下标（1）到末尾的值构成新数组。有两个参数时（1,4）表示取出下标1到3之间的数值构成新数组。负数要加数组数量

splice（）方法：向数组的中部插入项，有插入、删除和替换三种操作。

删除：p.splice（0,1），需要两个参数，可以删除任意数量的项，第一个参数表示起始位置，第二个参数表示要删除的项数。

插入：p.splice（2,0,"red","blue"）;需要三个参数，起始位置、要删除的项数、要插入的项。

替换：p.splice（2,1,"red","blue"）;需要三个参数，起始位置、要删除的项数、要插入的项。

该方法永远返回由删除的项组成的数组，若没有删除返回空数组。

位置方法：查找某一项在数组中的位置

indexOf（）方法：从数组的首项开始查找，需要两个参数，要查找的项和开始检索的位置（可选）

lastIndexOf（）方法：从数组的末项开始查找，需要两个参数，要查找的项和开始检索的位置（可选）

找到返回位置，找不到返回-1，使用严格的全等的操作符

迭代方法（可以对每一项运行同一个函数）

ES5为数组定义了5种迭代方法，每个方法接收两个参数，要在每一项上运行的函数和运行该函数的作用域对象（可选）

函数接收三个参数，item数组项值index数组项位置，array数组对象本身

every（）方法：对数组中的每一项运行给定的函数，若都返回ture，则整个返回ture

some（）方法： 对数组中的每一项运行给定的函数，若其中一个返回ture，则整个返回ture

filter（）方法：对数组中的每一项运行给定的函数，返回ture所在的项组成的数组。查询某些符合条件的数组项

forEach（）方法：对数组中的每一项运行给定的函数，没有返回值

map（）方法：对数组中的每一项运行给定的函数，返回每次函数调用的结果组成的数组。

var numbers = [1,2,3,4,5,4,3,2,1];

var everyResult = numbers.every(function(item, index, array){

return (item > 2);

});

alert(everyResult); //false

var someResult = numbers.some(function(item, index, array){

return (item > 2);

});

alert(someResult); //true

归并方法（对数组每一项的数值执行函数并进行归并）

reduce（）方法和reduceright（）方法，会迭代数组的所有项，并返回一个值。前一种是从第一项开始向后遍历，另一个是向前遍历。

函数接收四个参数：prev前一个值cur当前值index数组项位置，array数组对象本身

var values = [1,2,3,4,5];

var sum = values.reduce(function(prev, cur, index, array){

return prev + cur;

});

alert(sum);//15

3.Date类型

创建一个日期对象

var now=new Date();不传递参数的情况下，自动获得当前时间。要想根据特定的时间创建对象，必须传入从1970年1月1日到现在为止的毫秒数。

为了简化该过程，提出两种方法：Date.parse()方法和Date.UTC()方法

var now=new Date(Date.parse("May 25,2004")); 或者

var now=new Date("May 25,2004");

接受一个表示日期的字符串参数，并返回相应日期的毫秒数

Date.UTC()方法传入的参数是不同的，传入的参数是年份，基于0的月份，月中的那一天，小时数等等。

var now =new Date(Date.UTC(2000，0))方法 或者

var now=new Date(2000，0);

Date.now()方法返回表示调用这个方法时的日期和时间的毫秒数。

继承的方法

Date类型也重写了toLocaleString()，toString()，和valueOf方法，前两种方法会返回不同形式的时间信息，valueof直接返回日期的毫秒数，可以用来比较日期。

日期格式化方法

专门将日期转化为字符串的方法，呈现方法因浏览器而异

4.RegExp类型

ES用RegExp类型来支持正则表达式

字面量形式定义正则表达式：

创建一个正则表达式：var expression=/ pattern / flags;

pattern部分可以是任意一种简单或者复杂的正则表达式，可以带有一种或者多种标志（flags）

g:表示全局模式，将全部遍历所有字符串，不是找到一个就停止

i:不区分大小写

m:多行模式

var pattern1=/ at / g;匹配所有字符串中所有at的示例

var pattern1=/ [bc]at / i; 匹配第一个bat或者cat的实例，不区分大小写

var pattern1=/ .at / gi;匹配所有以at结尾的三个字符的组合实例

如果想要对元字符进行匹配，必须进行转义。比如要在字符串中加“？”号，要将问号进行转义，加/号即可。

RegExp构造函数来定义

var pattern1=new RegExp("at","i");元字符需要双重转义。

pattern1.test("atjdkadad");返回ture

RegExp 实例属性

每个实例都具有下列属性：

global：布尔值，表示是否设置了g标志

ignoreCase：布尔值，表示是否设置了i标志

multiline：布尔值， 表示是否设置了m标志

lastIndex:整数，表示搜索下一个匹配项的字符位置，从0开始

source：正则表达式的字符串形式，即返回字面量形式的正则表达式。

RegExp 实例方法

RegExp对象的主要方法是exec（），该方法是专门为捕获组设计的，捕获组就是正则表达式的子表达式匹配的结果依次进行保存，用一对括号来表示一组,该方法返回Array的实例，且有两个额外的属性：index和input属性

var text="mom an dad and baby";

var pattern=/mom( and dad( and baby)?) ?/gi;

var match=pattern.exec(text);输入参数是要应用的字符串，返回包含匹配项信息的数组。第一项是与整个表达式匹配的字符串，第二项是与捕获组表达式匹配的字符串。

alert（match.index）;表示匹配项在text字符串中的位置

alert（match.input）; 表示应用正则表达式的字符串

对于该方法而言，每次只会返回一个匹配项，若加上全局符号g，会一直接着查找，不设置全局符号，会一直返回第一个符合项。

RegExp对象的第二个方法就是test（）

接收一个字符串参数，当模式与字符串参数匹配的情况下，返回ture，经常用在if语句中

继承的toLocalString（）方法和toString（）方法都返回正则表达式的字面量形式，valueOf方法返回正则表达式本身

RegExp 构造函数属性

var text = "this has been a short summer";

var pattern = /(.)hort/g;

/\*

\* Note: Opera doesn't support input, lastMatch, lastParen, or multiline.

\* Internet Explorer doesn't support multiline.

\*/

if (pattern.test(text)){

alert(RegExp.input); //this has been a short summer

alert(RegExp.leftContext); //this has been a

alert(RegExp.rightContext); // summer

alert(RegExp.lastMatch); //short

alert(RegExp.lastParen); //s

alert(RegExp.multiline); //false

}

5.Function类型

ES中函数是对象，每个函数都是Function类型的实例，而且都与其他引用类型一样具有属性和方法，函数名是指向函数对象的指针。

定义函数的三种方法：

function sum(num1,num2)  函数声明法

{

return num1+num2;

}

var sum=function(num1,num2){  函数表达式

ruturn num1+num2;

};

没有函数名，通过变量就可以引用，末尾有一个分号，像声明其他变量一样

var sum=new Function("num1","num2","return num1+num2");构造函数法

function sum(num1, num2){

return num1 + num2;

}

alert(sum(10,10)); //20

var anotherSum = sum;

alert(anotherSum(10,10)); //20

函数名表示指向对象的指针，所以一个函数可以有多个名字，即多个函数名指向原函数,函数是对象。函数名是指针。

sum = null;

alert(anotherSum(10,10)); //20

没有重载：函数名相同时，只是把指向原函数的指针又指向了其他函数。

函数声明与函数表达式（函数声明提升）

函数声明在解析器执行程序之前就已经添加到执行环境中了，所以函数声明在函数调用之后也可以正常运行。但是函数表达式不可以。

作为值的函数

函数名本身就是变量，所以函数也可以作为值来使用，可以将函数像参数一样传递，也可以将函数作为另一个函数的返回值返回。

函数内部属性

函数内部有两个特殊的对象：arguments和this

arguments是保存函数参数的数组对象，他有一个属性callee，是一个指针，指向拥有这个arguments对象的函数。（用在递归中）

function factorial(num){

if (num <= 1) {

return 1;

} else {

return num \* arguments.callee(num-1)

}

}

var trueFactorial = factorial;

factorial = function(){

return 0;

};

alert(trueFactorial(5)); //120

alert(factorial(5)); //0

this对象引用的是当前环境的对象，当时全局环境时，引用的就是window对象。

函数对象的属性：caller

调用当前函数的函数的引用

函数的属性和方法

两个属性：length：表示函数希望接收的参数数量（形参）

                  prototype：重要

两个非继承而来的方法：apply（）和call（）用来在特定的作用域中调用函数

apply（）方法是将一个函数对象的内容应用到另一个对象中，其实就是改变this指向的函数，callSum1函数可以使用sum函数中的内容，将callSum1的this和参数都传入sum函数。

apply和call函数都是劫持另外一个函数的属性和方法，用在自己的函数域内。

function sum(num1, num2){

return num1 + num2;

}

function callSum1(num1, num2){

return sum.apply(this, arguments);

}

function callSum2(num1, num2){

return sum.apply(this, [num1, num2]);

}

alert(callSum1(10,10)); //20

alert(callSum2(10,10)); //20

call（）方法与apply差不多，唯一不同的是传入call的是列出来的参数

function sum(num1, num2){

return num1 + num2;

}

function callSum(num1, num2){

return sum.call(this, num1, num2);

}

alert(callSum(10,10)); //20

apply和call方法作用：扩充函数的作用域

window.color = "red";

var o = { color: "blue" };

function sayColor(){

alert(this.color);

}

sayColor(); //red

sayColor.call(this); //red，在全局作用域中调用该函数，并将全局作用域this值传入函数

sayColor.call(window); //red

sayColor.call(o); //blue

显式的调用函数，对象不需要与方法产生任何耦合关系，对象中不需要包含这个方法。

bind（）方法：创建一个函数的新实例，this值会被绑定到传给bind方法的值上。

window.color = "red";

var o = { color: "blue" };

function sayColor(){

alert(this.color);

}

var objectSayColor = sayColor.bind(o);

objectSayColor(); //blue

函数继承的toLocaleString()，toString()，和valueOf方法只返回函数的代码。

6.基本包装类型（是与基本类型相对应的引用类型）

当读取基本类型值并调用其方法时就说明是基本包装类型，而不是基本类型值。

为了便于操作基本类型值，ES还提供了三种特殊的引用类型：Boolean、Number、String，称为基本包装类型，有各自的属性和方法。

每当读取一个基本类型值的时候，后台就会自动创建一个对应的基本包装类型的对象，从而让我们能调用一些方法来操作这些数据。

引用类型和基本的包装类型区别：

生存期：使用new创建的引用类型对象，离开当前作用域后还一直存在于内存中，而自动创建的基本类型对象只存在于一行代码，然后立即被销毁 ，所以不能为基本类型对象添加属性和方法。

var s1="my dog";

var s2=s1.substring(2);调用方法1，在读取时创建基本类型对象

调用方法2：也可以显式利用构造函数来调用

var s1=new String（“my dog”）; 不推荐，分不清基本包装类型和引用类型

调用方法3：

var s1=new Object（“my dog”）;根据传入的值类型创建相应的基本包装类型实例

alert（s1 instanceof String）；ture

注：对基本包装类型的实例使用typeof会返回Object，所有基本包装类型的对象在转换为布尔类型时值都是ture

Boolean类型

var booleanObject=new Boolean（ture）；创建Boolean对象

valueOf()方法返回ture或者false，toString（）方法返回字符串形式的“ture”和“false”

用处不大，经常会造成人们的误解。

基本类型的布尔值和引用类型的布尔值区别：

1.typeof返回Object和Boolean

2.instanceof返回ture和false

Number类型

创建Number对象：var num=new Number(10);

Number类型继承并重写了toLocaleString()，toString()，和valueOf方法 ，valueof返回对象表示的基本类型数值，其他返回字符串形式的数值。

将数值格式化为字符串的方法：

toFixed（）方法会按照指定的小数位返回数值的字符串表示，var num=10;alert(num.toFixed(2));//“10.00”

toExponential（）方法会返回按照指数e的形式的字符串表示  num=10;alert(num.toExponential(2));//“1.00e1”

toPrecision（）方法会根据给出的数值来决定调用toFixed还是toExponential方法，可以找出表示一个数的最佳形式，根据传入的表示数的位数

String类型

var num=new String( “hello world”）；

toLocaleString()，toString()，和valueOf方法 都返回对象表示的字符串值。

属性：length

字符方法：访问字符串中特定字符

charAt（1）：返回位置1处的字符

charCodeAt（1）：返回位置1处的字符编码

s1[1];

字符串操作方法： concat（）方法，用于将一个或多个字符串连接起来，返回拼接后的新字符串，可以添加多个参数。加号更常用！

                              slice（）方法，取出一段字符串组成新的字符串，一个或两个参数,负值时加长度

                              substring（）方法，与slice一样，但是将负值全部转换为0

                              substr（）方法，第二个参数是返回字符数，而不是最后一个字符的位置，将第一个位置的负值加长度，第二个变0

字符串位置方法：

indexOf和lastIndexOf，与数组一样，找子字符串在整个字符串中的位置，找不到返回-1

trim（）方法修剪的意思

创建一个字符串的副本，删除前置和后置的空格，返回结果。

字符串大小写转换方法：toLowerCase（）、toLocaleLowereCase（）等四种

字符串的模式匹配方法

match（）方法：与RegExp对象的主要方法是exec（）一样（捕获组），区别是在字符串上调用还是在正则表达式上调用

var text = "cat, bat, sat, fat";

var pattern = /.at/;

var matches = text.match(pattern);

alert(matches.index); //0

alert(matches[0]); //"cat"

alert(pattern.lastIndex); //0

search（）方法：与match参数的方法相同，返回的是匹配项的索引。

var pos = text.search(/at/);

alert(pos); //1

replace（）方法替换子字符串，若第一个参数为字符串，则只替换第一个子字符串，若为正则表达式，且指定全局，会替换所有

var result = text.replace("at", "ond");

alert(result); //"cond, bat, sat, fat"

result = text.replace(/at/g, "ond");

alert(result); //"cond, bond, sond, fond"

split（）方法基于指定的分隔符将字符串分割成多个子字符串，第二个参数表示数组大小

var colorText = "red,blue,green,yellow";

var colors1 = colorText.split(","); //["red", "blue", "green", "yellow"]

var colors2 = colorText.split(",", 2); //["red", "blue"]

var colors3 = colorText.split(/[^\,]+/); //["", ",", ",", ",", ""]

字符串比较方法：LocaleCompare（）方法

var stringValue = "yellow";

alert(stringValue.localeCompare("brick")); //1

alert(stringValue.localeCompare("yellow")); //0

alert(stringValue.localeCompare("zoo")); //-1

这种方法取决于实现，所以最好用下面这种方法。

function determineOrder(value) {

var result = stringValue.localeCompare(value);

if (result < 0){

alert("The string 'yellow' comes before the string '" + value + "'.");

} else if (result > 0) {

alert("The string 'yellow' comes after the string '" + value + "'.");

} else {

alert("The string 'yellow' is equal to the string '" + value + "'.");

}

}

determineOrder("brick");

determineOrder("yellow");

determineOrder("zoo");

fromCharCode（）方法：接收一个或多个字符编码，将它们变成一个字符串，是构造函数String的一个静态方法

alert(String.fromCharCode(104, 101, 108, 108, 111)); //"hello"

7.单体内置对象

由ES实现提供的、不依赖于宿主环境的对象，这些对象在程序运行之前就存在了，比如Object、Array等

Global对象：

全局对象：在全局作用域上定义的属性和方法都是Global对象的属性和方法，比如isNaN()、isFinite()等

还有如下方法

URI编码方法：

encodeURI（）：对整个URI进行编码以便于发给浏览器，只对空格进行编码

encodeURIComponent（）：对URI的一部分进行编码，对任何非标准字符进行编码（除了数字字母）

decodeURI（） 解码

decodeURIComponent（）

eval（）方法：将字符串转换为可执行的代码

window对象：Global全局对象是window对象的一部分，可以通过window对象访问其属性和方法。

不能直接访问Global对象，可以通过window对象和this指针来访问。

Math对象：

属性：Math.E---e指数

          Math.LN10---10的自然对数

           。。。。

最大最小方法：min（）和max（）

找数组最大值：Math.max.apply(Math,values);将Math值作为this的值

将小数舍入为整数方法：ceil（）、floor（）、round（）分别是向上舍入、向下舍入和标准舍入

random（）方法：返回0到1之间的一个随机数

selectFrom（2,10）方法：得到一个介于2到10 之间的随机数

 其他方法：Math.sin、abs等等

第六章 面向对象的程序设计

对象是无序属性的集合，属性可以包括基本值、对象和函数。每个对象都是基于一个引用类型创建的，可以是原生类型或者是自定义。

6.1.理解对象

属性类型：数据属性和访问器属性，是内部才用的属性，用两对方括号表示，JS中不能直接访问，是为了实现JS引擎用的。

数据属性：包含一个数据值的位置，在这个位置可以读取和写入。

有四个描述行为的特性：

[[Configurable]]：是否可以删除属性重新定义，是否可以修改属性的特性

[[Enumerable]]：是否可以通过for-in循环返回属性，可枚举

[[Writable]]：是否可以修改属性的值，false为只读

[[Value]]：保存属性的值，读取和写入都是从这里定义

修改属性 默认的值用到Object.defineProperty（）方法，参数是三个：属性所在的对象、属性名和一个描述符对象

var person = {};

Object.defineProperty(person, "name", {

writable: false,

value: "Nicholas"

});

alert(person.name);

person.name = "Michael";

alert(person.name);

name属性的值是不可修改的，修改属性默认的特性需要用到defineProperty

var person = {};

Object.defineProperty(person, "name", {

configurable: false,//不能从对象中删除属性，不能修改属性的特性

value: "Nicholas"

});

alert(person.name);

delete person.name;

alert(person.name);

将Configurable设置成false之后就不允许改变属性的特性了，不能在变成ture

var person = {};

Object.defineProperty(person, "name", {

configurable: false,

value: "Nicholas"

});

//throws error

Object.defineProperty(person, "name", {

configurable: true,

value: "Nicholas"

});

这些特性是为了实现javascript引擎用的，在JS中不能直接访问，只是为了理解对象。

访问器属性：不包含数据值，只包含一对getter和setter函数，

[[Configurable]]：是否可以删除属性重新定义，是否可以修改属性的特性

[[Enumerable]]：是否可以通过for-in循环返回属性，可枚举

[[set]]:写入属性时调用的函数

[[get]]:读取属性时调用的函数

var book = {

\_year: 2004,

edition: 1

};

Object.defineProperty(book, "year", {

get: function(){

return this.\_year;

},

set: function(newValue){

if (newValue > 2004) {

this.\_year = newValue;

this.edition += newValue - 2004;

}

book.year;2004--读取

book.year = 2005;写入

alert(book.edition); //2

year是访问器属性，包含两个函数，写入该属性值调用set，读取调用get函数，改变一个属性的值，另外的也会改变。

不鞥直接操作数据，可以保护数据，有时候只可以读取不可以写入

定义多个属性：defineProperties可以同时定义数据属性和访问器属性

读取属性的特性：getOwnPropertyDescriptor（）方法，取得给定属性的描述符，参数为对象和对象属性，返回一个对象，如果为数据属性，返回的对象有对应的四种属性，访问器属性也是如此。

var book = {};

Object.defineProperties(book, {

\_year: {

value: 2004

},

edition: {

value: 1

},

year: {

get: function(){

return this.\_year;

},

set: function(newValue){

if (newValue > 2004) {

this.\_year = newValue;

this.edition += newValue - 2004;

}

}

}

});

var descriptor = Object.getOwnPropertyDescriptor(book, "\_year");

alert(descriptor.value); //2004

alert(descriptor.configurable); //false

alert(typeof descriptor.get); //"undefined"

var descriptor = Object.getOwnPropertyDescriptor(book, "year");

alert(descriptor.value); //undefined

alert(descriptor.enumerable); //false

alert(typeof descriptor.get); //"function"

6.2 创建对象

工厂模式可以用于创建相似的object类型，而构造函数和原型模式可以用来创建自定义对象类型。

使用同一个接口创建很多对象，会产生大量重复的代码。

工厂模式：在函数中创建对象并返回对象。由于ES中没有类，就发明了一种函数，用函数来封装以特定接口创建对象的细节，防止用同一个接口创建多个重复的对象

function createPerson(name, age, job){

var o = new Object();

o.name = name;

o.age = age;

o.job = job;

o.sayName = function(){

alert(this.name);

};

return o;

}

var person1 = createPerson("Nicholas", 29, "Software Engineer");

var person2 = createPerson("Greg", 27, "Doctor");

优点：解决了创建多个相似对象的问题。

缺点：不知道对象是什么类型的？

构造函数模式

可以用来创建特定类型的对象，如Array，也可以自己创建自定义构造函数。

function Person(name, age, job){

this.name = name;

this.age = age;

this.job = job;

this.sayName = function(){

alert(this.name);

};

}

var person1 = new Person("Nicholas", 29, "Software Engineer");

var person2 = new Person("Greg", 27, "Doctor");

构造函数就是函数，只是调用方式不一样，要用new字符调用

创建自定义的构造函数表示将来可以将它的实例标识为一种特定的类型。

Person("Greg", 27, "Doctor"); //直接调用。不使用new操作符时，将属性和方法都添加给window对象

window.sayName(); //"Greg"

var o = new Object();

Person.call(o, "Kristen", 25, "Nurse");

o.sayName(); //"Kristen"

call函数将对象Person在对象o中调用，o中就有了所有属性和方法

构造函数首字母大写，与工厂模式的区别是没有显式的创建对象，直接赋给this对象，没有return语句。

person1和person2对象都有一个constructor构造函数属性，都指向Person

现在person1和person2既是Object对象的实例，也是Person对象的实例

构造函数的问题：每个方法都要在每个实例上重新创建一遍，函数是对象，也就是实例化了一个对象。

解决方法：将方法定义在构造函数外部，实例共享一个方法，但是这样的话就没有封装性可言了，而且方法比较多时要定义多个全局函数。

原型模式

让所有对象实例都共享原型的属性和方法，不必通过构造函数传递参数。

function Person(){ //构造函数Person

}

Person.prototype.name = "Nicholas"; //通过prototype属性为原型对象添加属性

Person.prototype.age = 29;

Person.prototype.job = "Software Engineer";

Person.prototype.sayName = function(){

alert(this.name);

};

var person1 = new Person();//实例

person1.sayName(); //"Nicholas"

var person2 = new Person();

person2.sayName(); //"Nicholas"

alert(person1.sayName == person2.sayName); //true

将所有方法和属性都放进了Person的prototype属性中，构造函数是空的，但是依然可以通过调用构造函数来创建新对象，但是所有对象的属性和方法都一样。

创建的每一个函数（构造函数）都有一个prototype原型属性，这个属性是一个指针，指向它的原型对象。

所有原型对象都有一个默认属性，即constructor（构造函数）属性，这个属性指向构造函数

调用构造函数创建一个实例person1,时，person1中有一个内部指针【【prototype】】，指向原型对象。

对象实例与构造函数无关，只与原型函数有关

[[prototype]] 属性无法访问，但是通过isPrototypeOf（）方法可以。

Person.prototype.isPrototypeOf(Person1);  ture

Person.prototype.isPrototypeOf(Person2);  ture

Object.getPrototypeOf(person1)==person.prototype;//取得对象的原型

读取对象属性：先在实例中查找属性，没有的话在原型中查找。

可以通过在实例中重写属性值来创建具有不同属性值的属性，叫做实例属性和原型属性，也可以通过delete删除属性。

要判断一个属性是存在于实例中还是原型中，可以用hasOwnProperty方法，该方法只有属性存在于实例中时才返回ture，继承自object类型。

原型与in操作符（in单独使用或者放在for-in循环中）

in操作符会在通过对象能访问到的属性时返回ture，不管是实例还是原型，与hasOwnProperty方法一起使用就可以分清属性在实例中还是原型中。

for-in循环中返回的是所有能通过对象访问的可枚举的属性，包括实例和原型，也包括屏蔽了原型中不可枚举属性的实例属性。

不可枚举属性包括：hasOwnProperty,propertyIsEnumberable,toString,toLocaleString,valueOf，constructor等属性。

取得对象上可枚举的实例和原型属性用Object.keys（）方法，用原型引用返回原型中的可枚举属性，用实例引用返回实例中定义的（不包含原型属性）属性。

function Person(){

}

Person.prototype.name = "Nicholas";

Person.prototype.age = 29;

Person.prototype.job = "Software Engineer";

Person.prototype.sayName = function(){

alert(this.name);

};

var keys = Object.keys(Person.prototype);

alert(keys); //"name,age,job,sayName"

想得到所有的实例属性，无论他是否可枚举，用Object.getOwnPropertyNames

function Person(){

}

Person.prototype.name = "Nicholas";

Person.prototype.age = 29;

Person.prototype.job = "Software Engineer";

Person.prototype.sayName = function(){

alert(this.name);

};

var keys = Object.getOwnPropertyNames(Person.prototype);

alert(keys); //"constructor,name,age,job,sayName"

 注：取得原型或实例中的属性，除了for-in循环列举可枚举的属性，还可以用object.keys和object.getOwnPropertyNames等方法。

更加简单的原型语法，用一个包含所有属性和方法的对象字面量来重写整个原型对象，避免一直输入Person.prototype

function Person(){

}

Person.prototype = {

name : "Nicholas",

age : 29,

job: "Software Engineer",

sayName : function () {

alert(this.name);

}

};

现在Person.prototype是一个基于对象字面量创建的新对象，constructor已经不再指向Person。而是指向Object

改正：

function Person(){

}

Person.prototype = {

constructor：Person，

name : "Nicholas",

age : 29,

job: "Software Engineer",

sayName : function () {

alert(this.name);

}

};

现在constructor属性变成可枚举的了。可以设置其enumberable属性来将其constructor属性设置成不可枚举的。

原型的动态性

当我们对原型做出任何修改时，都会在实例中表现出来。

但是当我们完全重写原型对象时，实例与原来的原型是连接的，而不是新的原型对象。

原生对象的原型（Array、Object等）

原生对象的原型也是用原型模式来创建的，因此可以为原生对象的原型添加自定义方法，但是不推荐修改原生对象的标准。

 所有原生对象的公有属性和方法都可以在原生对象的原型中找到。

原型对象的缺点：对于包含引用类型值的属性，问题突出，是由其共享属性的本质所导致的。

function Person(){

}

Person.prototype = {

constructor: Person,

name : "Nicholas",

age : 29,

job : "Software Engineer",

friends : ["Shelby", "Court"],

sayName : function () {

alert(this.name);

}

};

var person1 = new Person();

var person2 = new Person();

person1.friends.push("Van");

alert(person1.friends); //"Shelby,Court,Van"

alert(person2.friends); //"Shelby,Court,Van"

alert(person1.friends === person2.friends); //true

friends数组存在于Person.prototype中，而不是person1中，所以会改变其原型数组值。

组合使用构造函数和原型模式

创建自定义类型的最常见的方式就是组合使用构造函数和原型模式，构造函数用于定义实例属性和引用类型属性，原型模式用于定义方法和共享的属性。所以每个实例都拥有自己的一份属性，又共享对方法的引用。还支持向构造函数传递参数。

function Person(name, age, job){

this.name = name;

this.age = age;

this.job = job;

this.friends = ["Shelby", "Court"];

}

Person.prototype = {

constructor: Person,

sayName : function () {

alert(this.name);

}

};

var person1 = new Person("Nicholas", 29, "Software Engineer");

var person2 = new Person("Greg", 27, "Doctor");

person1.friends.push("Van");

alert(person1.friends); //"Shelby,Court,Van"

alert(person2.friends); //"Shelby,Court"

alert(person1.friends === person2.friends); //false

alert(person1.sayName === person2.sayName); //true

动态原型模式

组合使用构造函数和原型模式唯一缺点是没有实现构造函数的封装性，原型和构造函数是独立的，所以提出动态原型模式。

function Person(name, age, job){

//properties

this.name = name;

this.age = age;

this.job = job;

//methods

if (typeof this.sayName != "function"){

Person.prototype.sayName = function(){

alert(this.name);

};

}

}

var friend = new Person("Nicholas", 29, "Software Engineer");

friend.sayName();

当sayname不存在时才进行原型初始化，也就是说原型初始化只进行一次，将所有的信息都封装在构造函数中，弥补了构造函数的缺点。

寄生构造函数模式

function Person(name, age, job){

var o = new Object();

o.name = name;

o.age = age;

o.job = job;

o.sayName = function(){

alert(this.name);

};

return o;

}

var friend = new Person("Nicholas", 29, "Software Engineer");

friend.sayName(); //"Nicholas"

该模式与工厂模式只差一个new字符，可以在特殊的情况下为对象创建构造函数。

稳妥构造函数模式

稳妥对象：没有公共属性，不使用new操作符创建实例，方法不引用this的对象，适合在一些安全的环境中使用。

function Person(name, age, job){

var o = new Object();

o.sayName = function(){

alert(name);

};

return o;

}

var friend = Person("Nicholas", 29, "Software Engineer");

friend.sayName(); //"Nicholas"

除了调用sayname方法没有比的方法访问其数据成员，很适合安全的执行环境。

稳妥构造函数与寄生构造函数创建的对象与构造函数之间没有任何关系，不可以使用instanceOf操作符。

6.3 继承

ES只支持实现继承，而且主要根据原型链来实现实现继承。

原型链

基本思想是利用原型让一个引用类型继承另一个引用类型的属性和方法，让一个引用类型实例等于另一个引用类型的原型，则这个原型指向另一个原型（继承了这个类型），这样原型指向组成的链叫做原型链。

function SuperType(){

this.property = true;

}

SuperType.prototype.getSuperValue = function(){

return this.property;

};

function SubType(){

this.subproperty = false;

}

//inherit from SuperType

SubType.prototype = new SuperType();

 重写了subtype的原型，代之以另一个原型的实例。

SubType.prototype.getSubValue = function (){

return this.subproperty;

};

var instance = new SubType();

alert(instance.getSuperValue()); //true

subtype继承了supertype，通过创建supertype的实例。存在于supertype的实例中的属性和方法，在SubType.prototype中也有。后来又添加了一个新的方法在subtype原型中。

我们没有用subtype默认的原型对象，而是给它换了个新原型，也就是supertype的实例，新原型具有实例的所有属性和方法，而且还指向subtype的原型，instance指向原型subtype，subtype原型指向supertype原型

优点：扩展了原型搜索机制，可以一直搜索到顶端。

别忘记默认的原型

所有的引用类型默认都继承了Object，也是通过原型链继承的，所以函数的默认原型都是Object的实例，这也是为什么所有的自定义类型都会继承toString（）等方法的原因

确定原型与实例的关系

alert(instance instanceof Object); //true

alert(instance instanceof SuperType); //true

alert(instance instanceof SubType); //true

由于原型链的关系，instance可以说成是 Object、SuperType、SubType任何一个类型的实例。

alert(Object.prototype.isPrototypeOf(instance)); //true

alert(SuperType.prototype.isPrototypeOf(instance)); //true

alert(SubType.prototype.isPrototypeOf(instance)); //true

 instance的原型是 Object、SuperType、SubType任何一个。

谨慎的定义方法：

function SuperType(){

this.property = true;

}

SuperType.prototype.getSuperValue = function(){

return this.property;

};

function SubType(){

this.subproperty = false;

}

//inherit from SuperType

SubType.prototype = new SuperType();

//new method

SubType.prototype.getSubValue = function (){

return this.subproperty;

};

//override existing method

SubType.prototype.getSuperValue = function (){

return false;

};

var instance = new SubType();

alert(instance.getSuperValue()); //false

var instance1 = new SuperType();

alert(instance1.getSuperValue()); //ture

想要覆盖超类型中的方法，必须在实例替换原型之后再定义新的方法。

当通过subtype实例调用getSuperValue方法时，调用的是新定义的方法，当用supertype实例来调用getSuperValue方法时是调用旧的方法。

当通过原型链进行继承时，不能通过对象字面量创建原型方法，因为这样会重写原型。

//inherit from SuperType

SubType.prototype = new SuperType();

//try to add new methods ?this nullifies the previous line

SubType.prototype = {

getSubValue : function (){

return this.subproperty;

},

someOtherMethod : function (){

return false;

}

};

var instance = new SubType();

alert(instance.getSuperValue()); //error!

原型链的问题

1.原型中的引用类型属性会被所有实例所共享，所以我们用构造函数来定义属性，而不是原型，但是通过继承，实例中的属性会被原型所继承，所以该原型所对应的实例就会共享该引用类型属性。通过实例向原型数组中添加元素时，由于数组存在于原型中，所以会改变原来原型中的数组。

2.创建子类型的实例时，不能向超类型的构造函数传递参数。

借用构造函数

优点1：优点解决原型中包含引用类型值的问题。

原理：在子类型构造函数的内部调用超类型构造函数，这样每一个子类型实例都会拥有自己的color属性的副本。

function SuperType(){

this.colors = ["red", "blue", "green"];

}

function SubType(){

//inherit from SuperType

SuperType.call(this);

}

var instance1 = new SubType();

instance1.colors.push("black");

alert(instance1.colors); //"red,blue,green,black"

var instance2 = new SubType();

alert(instance2.colors); //"red,blue,green"

每次调用SubType构造函数都会调用SuperType构造函数初始化colors数组。

优点2：可以在子类型构造函数的内部向超类型构造函数传递参数。

function SuperType(name){

this.name = name;

}

function SubType(){

//inherit from SuperType passing in an argument

SuperType.call(this, "Nicholas");

//instance property

this.age = 29;

}

var instance = new SubType();

alert(instance.name); //"Nicholas";

alert(instance.age); //29

给超类型的构造函数传递参数，然后定义自己的属性。

缺点：借用构造函数（将属性和方法都放在构造函数中），每次调用构造函数都会实例化方法，函数复用无从谈起。

组合继承

使用原型链实现对原型属性和方法的继承，通过借用构造函数来实现对实例属性（特别是引用类型）的继承。

function SuperType(name){

this.name = name;

this.colors = ["red", "blue", "green"];

}

SuperType.prototype.sayName = function(){

alert(this.name);

};

function SubType(name, age){

SuperType.call(this, name);

this.age = age;

}

借用构造函数

SubType.prototype = new SuperType();

SubType.prototype.sayAge = function(){

alert(this.age);

};

var instance1 = new SubType("Nicholas", 29);

instance1.colors.push("black");

alert(instance1.colors); //"red,blue,green,black"

instance1.sayName(); //"Nicholas";

instance1.sayAge(); //29

var instance2 = new SubType("Greg", 27);

alert(instance2.colors); //"red,blue,green"

instance2.sayName(); //"Greg";

instance2.sayAge(); //27

让不同的subtype实例拥有自己的属性，又可以共享相同的方法，成为JS中最常用的继承模式。

原型式继承

寄生式继承

寄生组合式继承

第七章 函数表达式

定义函数两种方式：函数声明和函数表达式

函数声明提升：在执行代码之前会先读取函数声明，所以可以先调用函数后函数声明。

sayHi();

function sayHi(){

alert("Hi!");

}

函数表达式：

var functionName=function (a,b,s){

//函数体

}；匿名函数，与其他表达式一样，使用前必须先赋值。下面的做法错误！

sayHi();

var sayhi=function（）{

alert("Hi!");

}；

不可以这样做！！

//never do this!

if(condition){

function sayHi(){

alert("Hi!");

}

} else {

function sayHi(){

alert("Yo!");

}

}

可以这样做！！

if(condition){

var sayhi=function(){

alert("Hi!");

};

} else {

var sayhi=function (){

alert("Yo!");

};

}

匿名函数可以当做值来做函数的返回值。

7.1 递归

使用arguments.callee 或者命名函数表达式来解决。

function factorial(num){

if (num <= 1){

return 1;

} else {

return num \* arguments.callee(num-1);

}

}

var anotherFactorial = factorial;

factorial = null;

alert(anotherFactorial(4)); //24   严格模式下，不能访问arguments.callee属性

命名函数表达式

 var factorial=（function  f( num){

if (num <= 1){

return 1;

} else {

return num \* f(num-1);

}

}）；

在严格模式和非严格模式下都行得通。

7.2 闭包

函数可以使用函数外部的所有全局变量值，但是函数内部的值外部却不可以知道，这个时候就需要函数闭包，连接函数内部与外部，闭包就是能够访问函数作用域的变量的函数，即函数中的函数。

正常情况下，函数中的局部活动对象在函数执行之后就会销毁，内存中只保存全局作用域，但是闭包不一样。

闭包缺点1：外部函数在引用结束后活动对象仍然没有销毁，因为闭包在引用他的活动对象，所以造成内存占用过多。

闭包与变量

function createFunctions(){

var result = new Array();

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

result[i] = function(){

return i;

};

}

return result;

}

闭包缺点2：闭包只能取得包含函数中任何变量的最后一个值10

关于this对象：匿名函数的执行环境具有全局性，this对象通常指向window

var name = "The Window";

var object = {

name : "My Object",

getNameFunc : function(){

return function(){

return this.name;

};

}

};

alert(object.getNameFunc()()); //"The Window"

解决方法：将外部作用域中的this对象放在闭包能够访问到的变量里，就可以让闭包访问对象了。

var name = "The Window";

var object = {

name : "My Object",

getNameFunc : function(){

var that = this;this指向object

return function(){

return that.name;

};

}

};

alert(object.getNameFunc()()); //"MyObject"

 内存泄漏

7.3 模仿块级作用域（私有作用域）

JS中没有块级作用域，即循环中定义的变量循环结束后仍然可以访问。

匿名函数可以用来模仿块级作用域

（function （）{

//块级作用域

}）（）；

将函数声明放在一个圆括号中表明它是一个函数表达式，而后面所跟的圆括号是立即调用该匿名函数。这样在该区域内定义的变量只会在for循环中存在，构成一个私有作用域，但是同时又是一个闭包。所以可以访问外界函数变量。

function outputNumbers(count){

(function () {

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

alert(i);

}

})();

alert(i); //causes an error

}

 这种技术经常在全局作用域中被用作函数外部，从而可以限制向全局作用域中添加过多的变量和函数，可以减少闭包占用内存过多的问题。

 7.4 私有变量

私有变量包括函数的参数、局部变量和在函数内部定义的其他函数。

可以访问私有变量和私有函数的公有方法：特权方法。

特权方法1：在构造函数中定义特权方法（闭包）

特权方法作为闭包可以访问私有变量和函数

使用构造函数来定义特权方法缺点就是每个实例都要创建一组相同的方法。

特权方法2：在私有作用域中定义私有变量

在私有作用域中定义私有变量和私有函数，同样可以创建特权方法。

私有变量有利于保护变量，只有 闭包才能访问它。

模块模式：为单例创建私有变量和特权方法，单例就是只有一个实例的对象

增强的模块模式

第八章 BOM

BOM：浏览器对象模型，主要用于控制窗口、框架和弹出窗口。BOM提供了很多对象用于访问浏览器的功能，与网页内容无关，DOM与网页有关。

8.1window对象

window对象是JS访问浏览器的接口，连接JS和浏览器，又是ES规定的Global对象。

window对象连接BOM和JS，同时又连接DOM与JS。

全局作用域：在全局作用域中声明的变量和函数都会成为window对象的属性和方法。

定义全局变量和在window对象上直接定义属性区别：

全局变量不能被delete删除，而window对象上的属性可以。因为通过var设置的变量属性有一个[[configurable]]，默认设置为false

窗口关系及对象

框架：就是一个浏览器窗口显示不止一个页面

如果页面中包含框架，则每个框架都有自己的window对象，并且保存在frames数组中。可以通过数值索引来访问相应的window对象。

<html>

<head>

<title>Frameset Example</title>

</head>

<frameset rows="160,\*">

<frame src="frame.htm" name="topFrame">

<frameset cols="50%,50%">

<frame src="anotherframe.htm" name="leftFrame">

<frame src="yetanotherframe.htm" name="rightFrame">

</frameset>

</frameset>

</html>

创建了一个框架集，一个框架居上，两个框架居下。可以通过window.frames[0]或者window.frames[“topFrame”]来访问上方的框架。

top对象指向浏览器窗口，也可以通过top.frames[0]来访问上方的框架。

patent是另一个对象，始终指向当前框架的直接上层框架

rightFrame 框架中包含了一个另一个框架redframe和blueframe，则redframe和blueframe 的parent对象指向rightframe，而topframe的parent对象指向的就是top对象。

self对象始终指向window对象，所以self对象和window对象可以互换使用。

存在框架时，每个框架的window对象包含的属性和方法都不一样，包含的原生类型的（Array）的构造函数也不一样。

窗口位置：用来确定和修改窗口的位置

无法在跨浏览器的情况下取得窗口左边和上边的精确值，但是可以通过moveTo（）方法和moveBy（）方法来将窗口精确的移动到某个位置。这两种方法可能会被浏览器禁用，并且只适用于最外层的window对象。

窗口大小

resizeTo（）方法和resizeBy（）方法来调整浏览器的大小，这两种方法可能会被浏览器禁用，并且只适用于最外层的window对象。

导航和打开窗口

使用window.open()方法可以导航到一个特定的URL，也可以打开一个新的浏览器窗口。

接收四个参数：URL、窗口目标（希望URL在此打开）、一个特性字符串、打开页面是否取代当前页面的布尔值。

//等同于<a href="[http://](NULL) ...... " target="topframe">

window.open("[http://](NULL) ......  ","topframe") ;如果有名为topframe的窗口或者框架，就在该窗口或框架打开，若没有，就新建一个窗口名为topframe。第二个参数也可以是：\_self,\_parent,\_blank,\_top等

若第二个参数不是一个已经存在的窗口或者框架，就会根据第三个参数传入的字符串来创建新的窗口。在不打开新窗口的情况下，会忽略第三个参数。

window.open("[http://](NULL) ......  ","topframe"，“height=400,width=400,top=10,resizable=yes”) ;弹出窗口特性设定

window.open()方法会返回一个指向新的窗口的引用，浏览器不允许我们改变主浏览器窗口的位置和大小。但是我们却可以调整新打开的浏览器窗口的大小和位置。

var win=window.open("[http://](NULL) ......  ","topframe"，“height=400,width=400,top=10,resizable=yes”) ;

win.resizeTo(500,500);

win.moveTo(100,100);

win.close();关闭新窗口

新创建的window对象有一个opener属性，指向打开他的原始窗口对象，但是原始窗口没有指向它打开窗口的指针。

第10章 DOM

DOM（文档对象模型）是针对HTML和XML文档的一个API，它描绘了HTML中一个层次化的节点树，允许开发人员添加、移除和修改页面的一部分，本章主要讨论与HTML相关的DOM1级的内容。

IE中的DOM对象都是以COM对象的形式实现的，使得IE行为不同。

10.1节点层次

DOM可以将任何HTML和XML文档描绘成一个多层节点组成的结构，节点分为几种不同的类型，每种节点都有自己的属性和方法，各自之间又都有联系。

一共有12种节点类型，文档节点是每个文档的根节点Document。

Node类型：除了IE任何浏览器都可以访问Node类型

DOM1定义了一个Node接口，由所有的节点类型实现，在JS中这个接口是作为Node类型实现的。所有的节点类型都继承自Node类型，所以有相同的属性和方法。

Node.ELEMENT\_NODE:元素类型，可以用数字1表示

Node.TEXT\_NODE:文本节点，用数字3表示

Node.DOCUMENT\_NODE:文档类型，用数字9表示

每个节点都有一个nodeType属性，表明节点的类型

if(someNode.nodeType==1)

alert("node is a element");

nodeName属性和nodeValue属性：对于元素来说，nodeName取的是元素的标签名，nodeValue是null。

节点关系

节点间的关系可以用家族关系来描述，子元素、父元素和同胞元素，每个节点都有childNodes属性，其中保存着一个NodeList对象（类似Array的动态对象），保存着它的子节点，可以通过方括号的形式或者item的形式访问其中的元素。

var firstchild=somenode.childNodes[0];

var secondchild=somenode.childNodes.item(1);

var count=somenode.childNodes.length;

parent属性，previousSibling和nextSibling可以取得同胞节点，firstChild和lastChild分别指向第一个和最后一个子节点。

hasChildNodes（）方法用来检查是否有子节点。

ownerDocument属性指向文档节点Document

操作节点

appendChild（）方法用来向childNodes的列表末尾添加一个节点，并返回这个添加的节点。

insertBefore（）方法用来向childNodes列表中间任意位置添加节点，参数为要插入的节点和参考节点，返回插入的节点。

replaceChild（）方法用来代替原来的节点，参数为要插入的节点和要代替的节点，返回已经代替的节点

removeChild（）方法用来移除原来的节点，参数为要移除的节点，返回移除的节点。

上述方法只能用于有子节点的节点类型，有些类型没有子节点会产生错误。

其他方法

cloneNode（）方法用于复制一份调用该方法的节点的副本，并接受布尔值来决定是否深复制，它是一个孤儿，除非插入到文档中。

该方法不会复制节点的事件处理程序，但是IE会复制，所以在复制前最好移除事件处理程序。

normalize（）方法用来文档树中的文本节点，当出现空白的文本节点删除，连续的文本则合并，在调用方法的节点的后代中查找。

Document类型

Document类型表示文档，document对象表示整个HTML页面，它是HTMLDocument对象的一个实例，可以取得与页面有关的信息。常用

document.documentElement或者document.childNodes[0]来取得对html元素的引用。还有document.body取得对body的引用。

Document节点的子节点有可能是html元素、doctype或者注释，文档类型是只读的，而且只能有一个子节点。

document.doctype取得对<!doctype>元素的引用，可能是文档节点的子节点，浏览器对它支持度不一样，很少用。

document.title可以读取title中的文本信息，甚至可以改变其title中的文本。

URL、domain和referrer：表示整个URL和取其中的域名，和前一个页面的URL。只有domain可以改变。

如果页面是来自不同的子域，可以通过将他们的domain设置成一样的使得这两个域的对象互相通信。

查找元素

document.getElementById();通过ID来查找元素，传输字符串，找到则返回该元素，找不到返回null。区分大小写，若多个元素的id相同只返回第一个

document.getElementsByTagName();通过标签名img、p查找元素，返回一个HTMLCollection对象，可以通过方括号或者item来取得其中的元素。

                                                          或者通过namedItem（“name1”）属性来取得对象中的特定的项。namedItem是HTMLCollection专有的方法

取得文档中所有的元素：document.getElementByTagName(“\*”);IE将注释实现为元素

document.getElementsByName():通过取得name相同的元素来查找，常用来取得单选按钮。

特殊集合，都是HTMLCollection对象

document.anchors：取得所有的带name特性的a元素

document.forms:取得所有的form元素

document.images:取得所有的img元素

document.links:包含文档中所有带href特性的a元素

DOM一致性检测

要检测浏览器中实现的DOM功能与版本，用document.implementation.hasFeature(“XML”，“1.0”)；

文档写入

在页面加载时向网页中动态的写入文本，write直接写入，writeIn在末尾加入换行符。比如动态写入当前日期

还可以写入外部资源，比如JS文件，要用转义字符。若在最后调用write则会重写整个页面。

open和close用来打开和关闭网页的输出流。

Element类型

用来表现HTML元素，提供了对元素标签名、子节点和特性的访问。

nodeName或者tagName取得元素的标签名

HTML元素由HTMLElement类型或者更加具体的子类型来表示，继承自Element类型，并有自己的属性和方法。

HTML元素的属性：id、title、className取得其相应的特性值。只有公认的属性才可以用这种方法

取得元素的属性值可以用的方法：getAttribute（），传入id、class、title，用于取得原有以及自定义属性。

setAttribute（）方法用来设置属性及值，参数为属性名和值，可以操作原有的属性和自定义属性。

removeAttribute（）方法用来移除某个属性。

IE会将自定义的属性当做元素的特性，其他浏览器不会。

attributes属性：只有Element类型使用该属性，

创建元素：ducument.createElement("div");创建元素的同时制定了其ownerDocument属性

并且可以操作该元素的属性、添加子节点或者其他操作，div.id="nsahd";

创建元素之后，要将元素添加到文档中才可以显示，document.body.appendChild(div);将元素添加到body中。在IE浏览器中可以将整个元素标签传入来创建。IE中使用将元素完整标签传入createElement中以避开IE中许多问题。

如果想取得元素的子节点或者后代节点，也可以使用getElementById等方法，只是在元素的后代中查找。

Text类型

表示文本节点，nodeName为"#text",nodeType为3，nodeValue为文本内容，有父节点为元素。无子节点。

通过nodeValue和data属性来访问节点中的文本。

操作文本内容：

appendData（text）:文本结尾添加text文本

deleteData（offset，count）；从指定的位置删除字符

insertData（offset，text）；从指定的位置插入字符

replaceData（offset，count，text）；用text替换从offset开始的字符

splitText（offset）；将文本分为两个文本节点

substringData（offset，count）；提取从指定位置开始的字符串

length属性表示文本中字符数量

每个可以包含内容的元素最多有一个文本，而且必须有内容，可以为空格。

<div>hello world</div>

通过div.firstChild或者div.childNodes[0]来访问文本内容。div.firstChild.nodeValue来修改它的内容。

创建文本节点

document.createTextNode();创建文本节点，参数为创建的文本内容，创建之后要将节点添加到结构树中。

var element = document.createElement("div");

element.className = "message";

var textNode = document.createTextNode("Hello world!");

element.appendChild(textNode);

document.body.appendChild(element);

如果一个节点包含两个文本节点，这两个文本节点就会连接起来，中间不会有空格。

规范化文本节点

父元素调用normalize方法可以将文本节点合并为一个。

分割文本节点

splitText（）方法

comment类型

用来表示注释，nodeType为8，没有子节点，拥有除了splitText以外Text类型所有的属性和方法。通过nodeValue或者data来访问其内容，通过父节点访问

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

var comment = div.firstChild;

alert(comment.data);

document.createComment也可以创建注释节点，但是很少用。

CDATASection类型

只针对于XML文档，表示CDATA区域，通过父节点来访问，并且可以通过createCDATASection来创建节点。

DocumentType类型：只有火狐、opera和Safari支持

包含与文档的doctype有关的所有信息，nodetype为10，nodename为doctype的名称，nodeValue值为null，parentNode为Document，没有子节点

不能动态创建，将DocumentType对象放在document.doctype中。

document.doctype.name表示文档类型的名称，HTML或者XHTML。

DocumentFragment类型

表示文档片段，没有相应的标记，文档片段虽然不能添加到文档中，但是可以用来保存将来可能会添加到文档中的节点，作为一个仓库使用。

document.createDocumentFragment()创建文档片段，将节点添加到文档片段中就是在文档树中删除了整个节点，也可以从文档片段中取其子节点添加到文档树中，但是文档片段不会被添加到文档树中。

var fragment = document.createDocumentFragment();

var ul = document.getElementById("myList");

var li = null;

for (var i=0; i < 3; i++){

li = document.createElement("li");

li.appendChild(document.createTextNode("Item " + (i+1)));

fragment.appendChild(li);

}

ul.appendChild(fragment);

Attr类型

用来表示元素的特性节点，但不建议使用。有三个属性：name,value和specified，特性名称与特性值和是否明确定义或默认

使用document.createAttribute()方法创建特性节点

10.2 DOM操作技术

用JS代码来代替HTML代码不麻烦，但是有时候要比处理其他部分困难。

动态脚本：动态脚本是指页面加载时不存在，但是后来某一时刻要通过修改DOM动态添加的脚本。

插入外部文件和直接插入JS代码

用JS来实现HTML中的内容

HTML：<script type="text/javascript" src="client.js"></script>

DOM:

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

script.type="text/javascript";

script.src="client.js";

document.body.appendChild(script);

可以用函数封装，并在需要的时候调用。

直接插入代码：

HTML:<script type="text/javascript">

     function sayhi(){

    alert("hello");

}

DOM:

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

script.type="text/javascript";

script.appendChild(document.createTextNode("function sayhi(){slert(     )}"));

document.body.appendChild(script);

IE视script元素是一个特殊的元素，不允许DOM访问它的子节点，但是可以设置script.text="function sayhi(){slert(     )}";

动态样式

动态样式：页面记载时不存在，将来动态添加到页面上

将CSS包含在HTML中的元素有两个link元素和style元素

HTML:<link rel="stylesheet" type="text/css" href="styles.css">

DOM:

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

link.rel="stylesheet";

link.type="text/css";

link.href="styles.css";

var head=document.getElementByTagname("head")[0];

head.appendChild(link);

HTML:<style type="text.css">

body{

         background-color:red;

}

</style>

DOM:

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

style.type="text/css";

style.appendChild(document.createTextNode("body  background-color:red;}"));

var head=document.getElementByTagname("head")[0];

head.appendChild(link);

IE将style元素视为一个特别的元素，不能访问它的子节点，可以通过style.styleSheet.cssText来添加文本

操作表格

用js代码来表示HTML中的表格太过麻烦，DOM为table元素等添加了很多属性和方法。

使用NodeList、NameNodeMap和HTMLCollection这三个集合都是动态更新的，每次访问都会运行一次基于文档的查询，最好将取得的值缓存起来。

第十一章 DOM扩展

将一些DOM扩展标准化一直是众望所归，主要扩展包括选择符API和HTML5。

11.1选择符API

根据CSS选择符选择与某个模式匹配的DOM元素，致力于让浏览器支持CSS查询。

querySelector（）方法，接受一个CSS选择符，返回一个与该模式匹配的第一个元素，找不到返回null。可以通过document或者Element类型来调用

//get the body element

var body = document.querySelector("body");

//get the element with the ID "myDiv"

var myDiv = document.querySelector("#myDiv");

//get first element with a class of "selected"

var selected = document.querySelector(".selected");

//get first image with class of "button"

var img = document.body.querySelector("img.button");

querySelectorAll（）方法，接受一个CSS选择符，返回所有匹配的元素Nodelist，找不到返回null

var ems = document.getElementById("myDiv").querySelectorAll("em");

//get all elements with class of "selected"

var selecteds = document.querySelectorAll(".selected");

//get all <strong> elements inside of <p> elements

var strongs = document.querySelectorAll("p strong");

可以用方括号和item来取得其中的一个或几个元素。

matchesSelector（）方法是Element类型新增的方法，检验调用该方法的元素是否与选择符匹配。不同的浏览器支持的方法名称不一样，可以编写包装函数。

11.2 元素遍历

对于元素间的空格，IE9之前不会返回文本节点，而其他浏览器都会返回，为了弥补这一个差异，为DOM元素新增了5个属性。

childElementCount：返回子元素的个数（不包括文本节点和注释）

firstElementChild：指向第一个元素，firstChild的元素版

lastElementChild：最后一个元素，lastChild的元素版

previousElementSibling

nextElementSibling

利用这些属性进行元素遍历，不用担心空白文本节点，用于遍历元素下的子元素。

11.3 HTML5

HTML5定义了大量的JSAPI,定义了浏览器应该有的DOM扩展。下面是HTML5中与DOM有关的内容。

与类相关的扩充：简化CSS类的用法

getElementByClassName（）方法：接收一个或多个类名，返回带有指定类名的所有元素。可以通过document或者元素调用。

操作类名时需要使用className属性来添加、修改和删除类名，但是当类名有多个时，修改其中一个的值太过繁琐。

为元素新增了classList属性：可以操作类名，是一个集合，通过下标来访问

<div class="bd user disabled">

div.classList.remove("disabled");删除某个类

div.classList.add("disabled"); 添加

div.classList.contains("disabled");  是否包含

div.classList.toggle("disabled");  切换

除非需要全部删除类名或者重写全部类名，否则可以不用className属性了。

焦点管理

document.activeElement这个属性始终会引用当前获得焦点的元素，元素获得焦点的方法有：页面加载、用户输入和调用focus方法。

文档刚刚加载完成，document.activeElement指向body元素

document.hasFocus()方法用来确认文档是否获得了焦点，可以知道用户是否正在和页面进行交互。

HTMLDocument的变化

readyState属性：document.readyState有两个可能的值，complete和loading表示文档已经加载完成和正在加载，以进行往下的操作。

document. compatMode用来区分浏览器用来渲染页面的模式是标准的还是混杂的，值为CSS1Compat时为标准的，为BackCompat时为混杂的。

document.head属性取得对head元素的引用

字符集属性

document.charset属性表示文档中实际使用的字符集，也可以为它赋于新的值。

document.defaultcharset 属性用来表示浏览器默认的字符集该是什么

自定义数据属性

可以为元素添加非标准的属性，前提是加前缀data-，添加以后可以通过元素的dataset属性来访问自定义属性的值。

<div id="myDiv" data-appId="12345"></div>

每个data-name形式的值在dataset属性上都有一个属性映射，去掉data-，比如div.dataset.appId="123";可以访问和修改自定义属性。

插入标记：将一段HTML代码作为字符串传入，即可取得其中的HTML标记

当需要插入大量的标记时，使用DOM非常麻烦，而且要将顺序连接起来。

innerHTML属性：在读模式下，innerHTML属性返回调用元素起始和结束标记内的HTML。

在写模式下，会创建新的DOM树来代替原来的元素子节点，将传入的内容相应的转换为元素。

为innerHTML设置新的字符串以后，浏览器会将这个字符串解析为新的DOM树，与原来的字符串不一样，是经过HTML序列化之后的结果。

限制：插入script元素不会执行，因为他是一个没有作用域的元素，即不显示在页面中，浏览器将把他删除，解决办法要在它之前加入有作用域的元素。

并不是所有元素都支持innerHTML元素，style元素就不支持，因为它没有作用域。

window.toStaticHTML()方法用来在将字符串插入之前对其进行处理，去掉事件处理程序和脚本节点，只有IE8支持。

outerHTML属性：与innerHTML不同的是读模式下返回全部标记，包括调用元素本身，写模式下完全替代原来的元素。

insertAdjacentHTML()方法：参数为两个插入位置和要插入的HTML文本。

第一个参数有四个值：

beforebegin：在当前元素之前插入一个紧邻的同辈元素

afterbegin:作为当前元素的第一个子元素插入

beforeend：作为当前元素的最后一个子元素插入

afterend：作为当前元素之后插入一个紧邻的同辈元素

内存与性能问题：在删除DOM时，它所含的事件并没有删除，会导致内存增加，所以要手动删除事件处理程序和JS属性，以防内存占用过多。

在设置innerHTML和outerHTML时会创建一个HTML解析器，解析器使用过多也会带来性能损失，所以尽量将使用innerHTML的次数控制的很少。

页面滚动问题：scrollIntoView（）方法

调用该方法的元素就可以出现在视口中，设置焦点也可以获得同样的功效。

11.4 专有扩展

第十二章 DOM2和DOM3

DOM1级主要描述了HTML\XML文档的底层结构，DOM2和DOM3级在此基础上引入了更多的交互能力，支持更高级的XML。

DOM2DOM3分为许多不同的模块，分别描述不同的子集。

DOM2级核心在DOM1级的基础上没有引入新类型，只是增加了新的属性和方法，增强了新类型。DOM3级核心增加了新的类型

12.1DOM的变化

针对XML命名空间的变化

将不同的XML元素混合在一起，共同构成格式良好的文档，而不用担心命名冲突。

DocumentType的变化：新增三个属性

publicId、systemId表示文档声明类型中的两个信息段，通过document.doctype.publicId、document.doctype.systemId访问

internalSubset用于访问在文档声明中的额外定义。

Document类型的变化

importNode（）方法：用于从一个文档中取得一个节点转移到另一个文档中，成为文档结构的一部分。

每个节点都有一个ownerDocument属性，当时用appendChild时传入不同文档的节点将会产生错误，但是使用importNode（）方法传入一个节点时该节点的访问权将会归该文档所有。

参数：要复制的节点和是否深复制的布尔值，返回节点的副本，然后插入到现有的文档中。

DOM2级视图新增了一个defaultView属性用来指向拥有给定文档的归属窗口，IE中相同属性为parentWindow。

DOM2级核心为document.implementation提供了两个新方法：createDocumentType（）和createDocument（）方法

前者用于创建一个新的DocumentType节点，接收三个参数，文档类型名称、publicId和systemId。创建新的文档时使用

后者用于创建新文档，参数为命名空间的URI，文档元素的标签名和文档类型。

DOM2级HTML增加document.implementation.createHTMLDocument属性用于创建新的HTML，参数为新创建的HTML的标题。

Node类型的变化

isSupported（）方法：用于确定当前节点具有什么能力，参数为特性名和特性版本号。最好结合能力检测

DOM3级引入了两个辅助比较节点的方法：isSameNode（）和isEqualNode（）方法

接收一个节点参数，前者在引用节点和传入节点相同时返回ture，相同是指两个节点引用同一个对象

                                 后者在引用节点和传入节点相等时返回ture，相等是指两个节点的类型属性等都相等。

为DOM节点添加和获取额外的数据setUserData（）方法和getUserData（）方法

前者是将数据指定给节点，参数为设置的键，实际的数据和处理函数。后者是得到该数据，传入设置的键即可。

12.2 样式

1.访问元素的样式

在HTML中定义样式有三种方式：link元素包含外部样式表，style元素定义嵌入式样式，style特性针对特定元素的样式

对于支持style特性的元素在JS中都有一个对应的style属性，所有在style特性中定义的属性都可以通过该元素的style属性来访问。对于CSS属性，可以通过style属性将其转化为JS能访问的属性，对于有下划线的属性要去掉下划线并且转换为驼峰的形式，只要取得元素的引用，就可以为它设置样式和读取

style对象的属性和方法

style.cssText:读取模式下可以访问style特性中的CSS代码，写入模式下会重写整个style特性的值，为元素应用多项变化。

style.length:应用给元素的CSS属性的数量，可以配合方括号或者item（）方法使用法来取得其中的属性名。

style.getPropertyValue（属性名）:可以取得属性的字符串值

style.getPropertyCSSValue:返回一个包含两个属性的对象--cssText和cssValueType表示值的类型

style.removeProperty:将某元素的样式从属性中移除，这样就可以看到默认的属性值。

计算的样式

style对象可以提供支持style特性的任何元素的信息，但是不包含那些从其他样式表层叠而来的影响当前元素的样式信息，document.defaultView.getComputedStyle（元素，伪元素字符串）方法可以取得当前元素的计算后的样式。IE不支持该属性，可以使用元素的currentStyle属性。

所有计算的样式都是只读的。

2.操作样式表

CSSStyleSheet类型表示的是样式表，包括link元素包含的外部样式表和style元素中定义的样式表。它是一套只读的接口，不允许修改样式表属性。它继承自styleSheet，StyleSheet可以定义非CSS样式表。

从StyleSheet继承而来的属性

disabled：表示样式表是否被禁用的布尔值，可读写，ture表示样式表被禁用。

href：若样式表是通过link元素包含的，则是样式表的URL。

ownerNode:指向当前样式表的父节点，即link元素或者style元素

type：表示样式表类型的指针。css/text

CSSStyleSheet类型独有的属性和方法

cssRules：样式表中包含的样式规则的集合，IE中使用rules属性

deleteRule（index）：删除指定位置的规则，IE支持removeRule（）方法

insertRule（rule,index）:在指定位置插入指定的规则，IE支持addRule（）方法

应用于文档的所有样式表都是通过document.styleSheets集合来表示的，可以通过length属性来获知样式表的数量以及通过方括号来取得样式表。

也可以通过sheet属性或者IE的styleSheet属性来取得样式表对象。

CSSRule对象是样式表中的每一条规则，是一个基类，而CSSStyleRule是继承于它

CSS规则：CSSStyleRule类型表示样式信息，其中三个重要的属性是cssText、selectorText和style。

cssText与style.cssText不一样，前者将返回选择符文本以及花括号，后者只会返回样式信息。前者只读，后者可以被修改。

selectorText：返回当前规则的选择符文本

function getStyleInfo(){

var sheet = document.styleSheets[0];

var rules = sheet.cssRules || sheet.rules;

var rule = rules[0];

alert(rule.selectorText);%当前规则的选择符文本

alert(rule.style.cssText);%完整得CSS代码

alert(rule.style.backgroundColor);

alert(rule.style.width);

alert(rule.style.height);

}

function changeStyleInfo(){

var sheet = document.styleSheets[0];

var rules = sheet.cssRules || sheet.rules;

var rule = rules[0];

rule.style.backgroundColor = "red";

样式表中也可以通过style属性来读取和修改样式表中的内容。

创建规则

向现有样式表中添加规则需要使用insertRule（）方法，参数为规则文本和在哪里插入的索引。IE中支持addRule（）方法。

function insertRule(sheet, selectorText, cssText, position){

if (sheet.insertRule){

sheet.insertRule(selectorText + "{" + cssText + "}", position);

} else if (sheet.addRule){

sheet.addRule(selectorText, cssText, position);

}

}

添加规则太多的情况下不建议使用，太繁琐，可以使用动态加载样式表。

删除规则

从样式表中删除规则的方法是deleteRule（）方法，参数为要删除的规则的位置。IE支持removeRule（）方法。

3.元素大小

偏移量：包括所有的内边距、滚动条和边框大小

offsetHeight：元素在垂直方向上占用的空间大小

offsetWidth：元素在水平空间上占用的空间大小

offsetLeft：元素的左外边框到包含元素的左内边框的距离

offsetTop：元素的上外边框到包含元素的上内边框的距离

包含元素的引用保存在offsetParent属性中。所有的偏移量属性都是只读的，每次读取都要重新计算。

客户区大小：元素内容及其内边距占据的空间大小（去除边框）clientHeight和clientWidth，可以用来确定浏览器视口的大小，通过document.documentElement.clientWidth调用或者document.body.clientWidth调用，指的是html元素或者body元素的大小 。

所有的客户区属性都是只读的，每次读取都要重新计算。

滚动大小：包含滚动内容的元素的大小？？？？？？

scrollHeight

scrollWidth

scrollLeft

scrollTop

确定元素大小

每个元素都有一个getBoundingClientRect（）方法，返回一个矩形对象，包含四个属性：left,top,right,bottom。给出了元素相对于视口的位置。

12.3 遍历

定义了两个辅助完成顺序遍历的DOM结构的类型：NodeIterator和TreeWalker，能基于给定的起点对DOM结构执行深度优先的遍历操作。IE不支持遍历

NodeIterator

可以使用document.createNodeIterator（）方法创建实例，参数为四个

root:想要作为搜索起点的树中的节点

whatToShow：表示要访问哪些节点的数字代码，通过一个或多个滤波器来确定访问哪些节点，这个参数的值以常量的形式在NodeFilter类型中定义。

                       NodeFilter.SHOW\_ALL

                       NodeFilter.SHOW\_ELEMENT

                       。。。。。

                       可以使用或操作符来组合多个选项

filter：该参数用来自定义一个功能类似节点过滤器的函数，定义一个NodeFilter对象，该对象只有一个方法acceptNode（），若是应该访问的节点就返回NodeFilter.FILTER\_ACCEPT,否则返回NodeFilter.FILTER\_SKIP。

entityReferenceExpansion：表示是否要扩展实体引用

filter参数

可以用来指定自定义的NodeFilter对象，每个对象只有一个方法。即acceptNode（）

NodeIterator 两个主要方法是nextNode（）和previousNode（）方法，用于正向遍历和反向遍历

var div = document.getElementById("div1");

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

var output = document.getElementById("text1");

var node = iterator.nextNode();

while (node !== null) {

output.value += node.tagName + "\n";

node = iterator.nextNode();

}

TreeWalker是更高级版本的NodeIterator,可以实现不同方向上的遍历，也拥有nextNode（）和previousNode（）方法，但也有

parentNode

firstChild

lastChild

nextSibling

previousSibling

创建TreeWalker对象要使用document.createTreeWalker()方法，参数与上一个方法相同。

优点是可以在DOM结构中随着任意方向移动。

currentNode属性表示当前正在遍历的节点，可以修改遍历继续进行的起点。

12.4 范围

通过范围可以选择文档中的一个区域，而不必考虑节点的界限，选择在后台进行，用户看不到的。

DOM中的范围

通过document.createRange（）方法来创建DOM范围，每一个范围由Range类型的实例表示，这个实例拥有很多属性和方法。

startContainer：选区中第一个节点的父节点

endContainer：选区中最后一个节点的父节点

startOffset：给定节点在其父节点中的子节点中的索引，当是文本节点时，就是前面字符的数量

endOffset：范围选择了几个节点

当将范围放在文档中时。这几个属性就会被赋值。

选择范围：selectNode（）和selectNodeContents（）方法，接收一个节点做参数，前一个方法选择整个节点及其子节点，后者只选择子节点。

实现复杂选择

setStart（）方法：参数为一个参照节点和偏移量，参照节点变成startContainer，偏移量变成startOffset

setEnd（）方法：参数为一个参照节点和偏移量，参照节点变成endContainer，偏移量变成endOffset

优点是：可以选择节点的一部分

var p1 = document.getElementById("p1"),

helloNode = p1.firstChild.firstChild,

worldNode = p1.lastChild,

range = document.createRange();

range.setStart(helloNode, 2);

range.setEnd(worldNode, 3);

操作DOM范围中的内容

选中的范围可以创建一个文档片段，为了创建文档 片段需要正确的格式，以创建正确的DOM树。范围知道自己缺少什么标签，会自动添加。

deleteContents（）方法可以将范围从文档中删除，删除后剩下的任然保持良好的格式，返回剩下的DOM树

extractContents（）方法也可以移除该范围，但是返回该范围，以后可以将该范围插入到别的地方

cloneContents（）方法可以创建范围的副本，插入到别的地方。

在调用这些方法之前，范围的格式还是不完整的。

插入DOM范围中的内容

使用insertNode（）方法向范围选区的开始处插入一个节点。但是不会补充缺少的标签

surroundContents（）方法可以环绕范围插入内容，参数为环绕范围内容的节点，

折叠DOM范围

折叠范围是指范围中未选择文档的任何部分，可能折叠到选区的开始位置或者结束位置。

collapse（）方法来选择折叠范围，接收一个布尔值，表示折叠到开始还是最后，折叠结束后检查collapsed特性。

比较DOM范围

compareBoundaryPoints（）方法来确定这些范围是否有公共的边界。参数为表示比较方是的常量值和比较的范围。

复制DOM范围

cloneRange（）方法产生范围的副本

清理DOM范围

detach（）从创建的文档中分离该范围

第十三章 事件

JS与HTML之间的交互是通过事件来完成的，事件是指文档或者浏览器窗口发生的一些特定的交互瞬间，可以用侦听器来预订事件，以便于事件发生时执行相应的代码。

13.1事件流

事件流是指从页面中接收事件的顺序，分为事件冒泡流和事件捕获流。

事件冒泡流表示事件是从具体元素开始，逐一向上传播，在每一级节点上都会发生。

事件捕获流表示事件是从上往下流。

为什么事件要传播？

因为事件源本身可能无法处理事件，比如按钮本身不能处理按按钮事件，就要传播到文档的其他部分来处理该事件。

DOM事件流

DOM2级事件流规定的事件流包括三个阶段：事件捕获阶段、处于目标阶段和事件冒泡阶段。

事件捕获为截获事件提供了机会，处于目标是目标接收到事件，冒泡阶段是可以在这个阶段对事件进行反应。

13.2事件处理程序

事件是指用户或者浏览器的某种行为，例如click、load等都是事件的名字。而事件处理程序就是为了响应某个事件的函数，加on开头。处理click事件的程序叫做onclick。

HTML事件处理程序

某个元素支持的每种事件都可以，都可以使用一个与事件处理程序相同名字的HTML特性来指定。例如要在单击按钮时执行一些JS代码。

<input type="button" value="Click Me" onclick="alert('Clicked')" />

<input type="button" value="Click Me" onclick="alert(&quot;Clicked&quot;)" />

<script type="text/javascript">

function showMessage(){

alert("Hello world!");

}

</script>

<input type="button" value="Click Me" onclick="showMessage()" />

在HTML中定义的事件处理程序可以包含要执行的具体动作，也可以调用在页面的其他地方定义的脚本。

事件处理程序中有一个局部变量event，通过event变量可以直接访问事件对象，this的值是指事件的目标元素。

扩展作用域：在函数内部可以像访问局部变量一样访问document和元素本身的属性，这样就可以访问元素自己的属性。

在HTML中指定事件处理程序的缺点：

1.时差问题：用户可能会在HTML元素一出现就触发相应的事件，如果函数定义在事件后面，还没来得及加载完成，就会报错。

2.HTML与JS代码紧密耦合，修改时要改两个地方

DOM0级事件处理程序

通过JS指定事件处理程序的传统方法就是将一个函数赋值给一个事件处理程序属性。每个元素都有自己的事件处理程序属性，通常小写如onclick，将这种属性的值设置为一个函数，就可以为元素指定事件处理程序。

var btn = document.getElementById("myBtn");

btn.onclick = function(){

alert(this.id);

};

这些代码运行之前不会指定事件处理程序，但是如果按钮在此程序之前就会怎么按按钮都没有反应，因为还没加载完成。

DOM0级方法指定的事件处理程序被认为是元素的方法，this指元素，可以在事件处理程序中引用元素的任何属性和方法。

var removeBtn = document.getElementById("myRemoveBtn");

removeBtn.onclick = function(){

btn.onclick = null;%删除事件处理程序

};

将事件处理程序添加到冒泡阶段

DOM2级事件处理程序

DOM2级事件都定义了两个方法来处理指定和删除事件处理程序的操作。

addEventListener（）方法：

removeEventListener（）方法

参数为要处理的事件名、事件处理程序的函数和一个布尔值，ture表示在捕获阶段调用处理程序，false表示在冒泡阶段调用。

var btn = document.getElementById("myBtn");

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

alert(this.id);

}, false);

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

alert("Hello world!");

}, false);

函数的作用域也是元素。

DOM2级事件处理程序主要优点就是可以添加多个事件处理程序。只能通过removeEventListener来移除，而且参数是一模一样的，匿名函数无法移除

最好是将事件处理程序出发在事件流的冒泡阶段，以便于最大限度的兼容各种浏览器。

如果需要在事件到达目标元素之前捕获它，就将事件处理程序定义在事件捕获阶段。

IE事件处理程序

IE实现了与DOM中相似的两个方法attachEvent（）方法和detachEvent（）方法，参数为事件处理程序名称和事件处理函数，触发在冒泡阶段。

var btn = document.getElementById("myBtn");

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

alert("Clicked");

});

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

alert("Hello world!");

});

IE与DOM0级方法的主要区别是事件处理程序的作用域，IE的事件处理程序的作用域是全局作用域，this表示window。

也可以为它添加多个事件，但是是以相反的顺序进行，也可以移除，通过相同的参数。

使用IE事件处理程序的浏览器是IE和opera浏览器。

跨浏览器的事件处理程序

addHandler（）方法：按照不同的情况来使用DOM0级方法、DOM2级方法和IE方法来添加事件。它属于EventUtil对象，接收参数：要操作的元素、事件名称和事件处理程序函数。

removeHandler（）方法

13.3事件对象

在触发DOM上的某个事件时，会产生一个事件对象event，包含所有与事件相关的信息。所有浏览器都支持该对象，但是支持程度不一样。

DOM中的事件对象

兼容DOM的浏览器都会将一个event对象传入到事件处理程序中

event有很多属性和方法：currentTarget和target，this 始终指向currentTarget，target指向事件最终的目标。

var btn = document.getElementById("myBtn");

btn.onclick = function(event){

alert(event.currentTarget === this);

alert(event.target === this);

将事件处理程序指定给了目标元素，这三个值是相同的

document.body.onclick = function(event){

alert(event.currentTarget === document.body); //true

alert(this === document.body); //true

alert(event.target === document.getElementById("myBtn")); //true

将事件处理程序定义在body元素上（按钮元素的父元素），这三个值不一样，事件是在冒泡阶段触发。

需要一个函数处理多个事件，可以使用type属性。type表示被触发的事件的类型

var btn = document.getElementById("myBtn");

var handler = function(event){

switch(event.type){

case "click":

alert("Clicked");

break;

case "mouseover":

event.target.style.backgroundColor = "red";

break;

case "mouseout":

event.target.style.backgroundColor = "";

break;

}

};

btn.onclick = handler;

btn.onmouseover = handler;

btn.onmouseout = handler;

要阻止特定事件的默认行为，可以使用preventDefault（）方法

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

link.onclick = function(event){

event.preventDefault();

};

stopPropagation（)方法用于立即停止事件在DOM层次中的传播。

eventPhase（）方法用于确定事件当前位于事件流的哪个阶段。1表示捕获阶段，2 表示处于目标。3表示冒泡阶段

IE中的事件对象

要访问IE中的event对象的几种方法：看事件处理程序的方法。

使用DOM0级方法添加事件处理程序时，通过window.event来调用event对象。

使用attachEvent来定义事件处理程序时，也会有一个event参数传入函数

event的属性和方法：

cancelBubble用于停止冒泡和事件捕获 ，布尔值为ture时取消捕获和冒泡。

returnValue（）方法用于取消事件的默认行为 ，布尔值为false时就可以取消事件的默认行为。

event.srcElement用于指定事件的真正目标，与target相同。

type属性

跨浏览器的事件对象

13.4 事件类型

DOM3级事件规定的时间类型：

UI事件：用户界面事件，当用户与页面发生交互时发生

焦点事件：当元素获得或者失去焦点

鼠标事件：通过鼠标在页面上执行操作时触发

滚轮事件：使用鼠标滚轮时触发

文本事件：在文档中输入文本

键盘事件：通过键盘执行操作

合成事件：当为输入法编辑器输入字符时触发

变动事件：当底层DOM结构发生变化时

UI事件

UI事件包括的load事件和unload事件还有resize事件和scroll事件都是HTML事件。

load事件：当页面完全加载后，就会触发WINDOW上面的load事件或者所有框架都加载完毕在框架集上触发、所有图像都加载完毕在img元素上触发，有两种定义onload事件处理程序的方式。

页面完全加载包括所有图像和JS文件和CSS等外部资源。

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

alert("Loaded!");

});

这是一种用JS来指定事件处理程序的方法，跨浏览器

<body onload="alert('Loaded!')">

<p>Load event example.</p>

</body>

另一种是在body元素中通过相应的特性来指定，在window对象上面发生的任何事情都可以在body元素上定义相应的属性来实现。

图像也可以触发load事件，无论是DOM中的图像还是HTML中的图像。

HTML：

<img src="smile.gif" onload="alert('Image loaded.')" />

在img元素中为任何图像添加onload事件处理程序，当图像加载完毕时就可以显示提示信息。

DOM：

var image = document.getElementById("myImage");

EventUtil.addHandler(image, "load", function(event){

event = EventUtil.getEvent(event);

alert(EventUtil.getTarget(event).src);

可以通过src属性访问并显示该信息。

创建新的img元素时，可以为其制定一个事件处理程序，以便图像加载完毕后给出提示，重要的是在指定src属性之前先指定事件。因为图像是在设置了其src属性后开始下载，而非添加到文档中后才开始下载。

script元素也可以触发load事件，以便开发人员确定动态加载的JS文件是否加载完毕，这个元素以非标准的方式来支持load事件。

script元素是在设置了src属性并且添加到文档后才开始下载。

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

EventUtil.addHandler(script, "load", function(event){

alert("Loaded");

});

script.src = "example.js";

document.body.appendChild(script);

在IE和opera浏览器中，link元素也可以触发load事件，确定样式表是否加载完毕。在指定了link元素的href属性和添加到文档中后才可以下载样式表。

unload事件

当文档被完全卸载之后就会触发，当从一个页面切换到另一个页面时就会发生，利用这个事件最多的情况就是清除引用，避免内存泄漏。

也是包括两种方式来指定事件处理程序，此时调用元素的属性和方法就会出现错误。

resize事件

当浏览器窗口被调整到一个新的高度或者宽度时，就会在window对象上触发resize事件，也可以通过JS或者body中的onresize属性来定义。

在兼容DOM得浏览器上，传入的event对象只有一个属性target，值为document。

关于什么时候会触发resize事件不同的浏览器的行为不同，有的浏览器会在每一次调整触发，有的则在最后一次调整时触发，所以不要在事件处理程序上加入大计算量的代码，以防不断触发影响浏览器的速度。

scroll事件

发生在window对象上，可以通过body元素的scrollLeft和scrollTop来监测这一变化（在混杂模式下）或者通过html元素来检测变化（标准模式下）。

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

if (document.compatMode == "CSS1Compat"){

alert(document.documentElement.scrollTop);

} else {

alert(document.body.scrollTop);

}

输出页面的垂直滚动位置，该事件会在滚动时不断地被触发，所以程序尽量简单。

焦点事件

焦点事件会在页面元素获得或者失去焦点时触发，利用这些元素与document.hasfocus()方法以及document.activeElement属性就可以知晓用户在页面上的行踪。

主要的两个事件是focus和blur

focus：在元素获得焦点时触发，不会冒泡

blur：在元素失去焦点时触发，不会冒泡

focusIn：在元素获得焦点时触发，会冒泡

focusOut：在元素失去焦点时触发，会冒泡，被DOM3采纳为标准方式

鼠标与滚轮事件

DOM3级定义了9个鼠标事件

click：在用户单击鼠标按钮（左）或者按下回车键时触发。

dblclick：在用户双击鼠标按钮（左）时触发

mousedown：在用户按下任意鼠标按钮时触发，不能通过键盘触发该事件

mouseup：在用户释放鼠标按钮时触发，不能通过键盘来触发

mouseenter：在用户从元素外部首次进入元素内部时触发，不会冒泡

mouseleave：在位于元素上方的鼠标光标首次移动到元素范围之外时触发，不会冒泡

mousemove：当鼠标指针在元素内部移动时反复触发

mouseout：当鼠标从一个元素移到另一个元素时触发

mouseover：从一个元素内部移入另一个元素内部

页面上的所有元素都支持鼠标事件，而且都可以被取消。

mousewheel：鼠标滚轮事件，跟踪鼠标滚轮

客户区坐标位置

鼠标事件都是在浏览器视口中的特定位置上发生的，位置信息保存在事件对象的clientX和clientY属性中。表示事件发生时鼠标在视口中的位置。

页面坐标位置

告诉我们事件是在页面的哪个位置发生的，通过事件对象的pageX属性和pageY属性来保存。

屏幕坐标位置

screenX和screenY属性表示事件发生时鼠标相对于整个屏幕的位置

修改键

虽然鼠标事件都是通过鼠标来触发的，但是键盘上的某些键也会影响鼠标的行为，比如shift、ctul、alt和meta键。他们经常被用来修改鼠标的行为。

DOM为此规定了4个属性：shiftKey、stulKey、altKey和metaKey，布尔值，ture表示按下了该键

相关元素

mouseout事件和mouseover事件都有目标元素和相关元素，团购event.relatedTarget属性来访问

鼠标按钮

DOM的event对象中存在一个button属性，0表示主鼠标按钮，1 表示滚轮，2 表示鼠标右键，对于mousedown和mouseup事件来说

鼠标滚轮事件

当用户通过鼠标滚轮与页面进行交互时，就会触发mousewheel事件。可以在任何元素上触发，最终冒泡到document或者window元素上。

event对象包含一个wheelDelta属性，当用户向前滚动时，该属性值是120的倍数，向后时是-120的倍数。

firefox浏览器有一个名为DOMMouseScroll事件，event.detail属性保存关于滚轮的信息，是3和-3 的倍数。

键盘与文本事件

3个键盘事件：

keydown：用户按下键盘上的任意键时触发，按住不放会重复触发

keypress：用户按下字符键时触发

keyup：用户释放键盘上的键时触发

1个文本事件

textInput：在文本插入文本框时会触发

当用户按下键盘上一个字符键，先触发keydown事件，后触发keypress事件，这两个事件都是在文本框发生之前触发的，而后触发keyup事件，发生在文本框发生变化之后。

键码

在发生keydowm和keyup事件时，event对象有一个keyCode属性保存按下的键的代码。

字符编码

event对象中有一个charCode属性，只有在发生keypress事件时才有，表示按下的键的ASCII码

textInput事件

当用户在可编辑区域中输入字符时就会触发该事件，在可编辑区域才能触发该事件。

复合事件

用于处理IME（输入法编辑器），IME通常需要同时按住多个键，最终只输出一个字符。

三个事件：

compositionstart：在IME的文本复合系统打开时触发，表示要开始输入了。

compositionend：在IME的文本复合系统关闭时触发，表示恢复正常键盘输入状态。

compositionupdate：在向输入字段输入字符时触发

比文本事件多一个data属性，可能为以下的值

如果在compositionstart 事件发生时访问，data的值是正在编辑的文本

如果在compositionupdate事件发生时访问 ，包含正在插入的新字符

如果在 compositionend事件发生时访问 ，包含输入会话中所有插入的文本

变动事件

变动事件是在DOM结构发生变化时触发，有如下事件

DOMSubtreeModified：在DOM结构发生任何变化时就会触发，在其他事件触发后都会触发这个事件

DOMNodeInserted：在一个节点作为另一个节点的子节点插入时触发

DOMNodeRemoved：在节点从其父节点中被移除后触发

DOMNodeInsertedIntoDocument：在一个节点被插入文档中时触发，在DOMNodeInserted后触发

DOMNodeRemovedFromDocument ：一个节点被从文档中删除触发，在DOMNodeRemoved 后触发

DOMAttrModified ：当特性被修改之后触发

DOMcharacterDataModified：当文本节点的值被改变

删除节点

当删除一个节点时，首先在该节点触发 DOMNodeRemoved事件，relatedNode属性等于其父节点，会冒泡

在该元素上触发 DOMNodeRemovedFromDocument事件

然后依次在该元素的子元素上触发 DOMNodeRemovedFromDocument事件

在该元素的父元素上触发DOMSubtreeModified事件

插入节点

当插入一个节点时，首先在该节点触发 DOMNodeInserted事件，relatedNode属性等于其父节点，会冒泡

在该元素上触发 DOMNodeInsertedIntoDocument事件

在该元素的父元素上触发DOMSubtreeModified事件

HTML5事件

contextmenu事件

该事件用来表示何时应该显示上下文菜单，以便开发人员取消默认的上下文菜单而提供自定义的菜单。单击鼠标右键可以调出上下文菜单。

首先在该事件中取消默认的菜单，然后将自定义的菜单放在某个位置。

beforeunload事件

这个事件是在页面卸载之前触发，用来阻止页面卸载操作。将控制权交给用户，让用户来决定是否卸载该页面。

DOMcontentLoaded事件

当页面中的DOM树加载完毕后就触发该事件，而不管JS文件、CSS文件和图像是否加载完毕，为了今早的让页面与用户进行交互。

该事件的目标元素是document元素。该事件一定会在load事件之前触发

readystatechange事件

提供与文档或者元素加载状态有关的信息，支持该事件的每个元素都有一个readystate属性，可能包含下列值

uninitialized：对象尚未初始化

loading：正在加载数据

loaded：加载完成

interactive：可以操作对象了，但是还没有完全加载

complete：对象已经加载完成

pageshow和pagehide事件

pageshow事件在页面显示时触发，若页面第一次显示，则在load触发后触发，若不是第一次显示则在完全恢复后触发。在window对象上触发

pagehide事件在页面卸载时触发，在unload之前触发

hashchange事件

当URL参数列表发生变化时通知开发人员，将该事件处理程序加载在window上，此时的event有两个属性oldURL和newURL存放新的和旧的参数列表。

设备事件

内存和性能

事件委托

解决事件处理程序过多的解决方案就是事件委托，利用事件冒泡，只指定一个事件处理程序，就可以管理某一个类型的所有事件。

例如onclick事件，因为可以冒泡到document层次，所以可以只为页面指定一个该事件，而不必给每个可点击的元素分别添加。

尽量在DOM树中层次最高的元素上加事件处理程序，这样在其子元素上单击就可以冒泡到父节点上进行事件处理。

移除事件处理程序

当一个元素被移除时，它所带的事件处理程序无法被垃圾回收机制回收，造成内存泄漏，最好手工移除事件处理程序。

如果页面被卸载之前没有清理干净事件处理程序，程序就会滞留在内存中，最好在页面卸载之前听过onunload事件移除所有事件处理程序。

模拟事件

可以使用JS来触发各种事件，用来模拟触发事件，是测试web应用程序的一种方法。

DOM中的事件模拟

1.创建event对象：在document上使用createEvent方法来创建event对象，参数表示要创建事件类型的字符串:UIEvent、MouseEvent、mutationEvent和HTMLEvent

2.初始化：每种类型的event对象初始化的方法不一样，传入适当的参数就可以初始化

3.触发事件：dispatchEvent方法来触发事件

模拟鼠标事件

EventUtil.addHandler(btn2, "click", function(event){

//create event object

var event = document.createEvent("MouseEvents");传入参数创建鼠标event对象

//initialize the event object

event.initMouseEvent("click", true, true, document.defaultView, 0, 100, 0, 0, 0, false,传入参数初始化，15个参数

false, false, false, 0, btn2);

//fire the event

btn.dispatchEvent(event);触发事件

});

模拟键盘事件

IE中的事件模拟

EventUtil.addHandler(btn2, "click", function(event){

//create event object

var event = document.createEventObject();

//initialize the event object

event.view = window;

event.detail = 0;

event.screenX = 100;

event.screenY = 0;

event.clientX = 0;

event.clientY = 0;

event.ctrlKey = false;

event.altKey = false;

event.metaKey = false;

event.shiftKey = false;

event.button = 0;

event.relatedTarget = null;

//fire the event

btn.fireEvent("onclick", event);

});

采用不同的方法触发模拟事件。

第十四章 表单脚本

14.1 表单的基础知识

在HTML中，使用type特性为“submit”的input和button按钮就可以，表单使用form元素表示，但是在JS中表单是用HTMLFormElement类型表示，继承了HTMLElement的属性和方法，但也有自己的属性和方法：

submit（）方法

reset（）方法

target属性

name属性

取得form元素引用的方法有两种：

1.为form元素提供id，根据getElementById（）方法找到它

2.用document.forms来提取所有的表单，并根据下标来找具体的表单。

提交表单

HTML方法

通用提交按钮

<input type="submit" value="submit form">

自定义提交按钮

<button type="submit">submit form</button>

图像按钮

<input type="img" src="1.gif">

用户单击提交按钮或者图像按钮就会提交表单，或者在表单拥有焦点的情况下按回车键提交表单，这种方式会在数据提交之前触发submit事件，以有机会验证表单数据，并决定表单是否可以提交或者阻止。

JS方法

用JS也可以提交表单，用submit（）方法，无需表单包含按钮，但是不会触发submit事件，所以要提前验证表单数据。

提交表单最大的问题就是重复提交表单，可以在第一次提交表单之后就禁用提交按钮，或者利用onsubmit事件处理程序取消后续的表单提交操作。

重置表单

在HTML中，使用type特性为“reset”的input和button按钮就可以 ，触发reset事件,所以可以在必要的时候取消重置表单

<input type="reset" value="reset room">

<button type="reset">reset form</button>

也可以通过JS来重置表单，使用reset（）方法，触发reset事件

form.reset()

表单字段

每个表单都有一个elements属性，表示表单中所有表单元素的有序集合，可以通过下标或者name来访问它。

公有的表单字段属性

disabled：布尔值，表示当前字段是否被禁用

name：当前字段的名称

type：当前字段的类型

readonly：只读

form：指向当前字段所属表单的指针

除了form属性，我们可以动态的修改表单的属性，操作表单，例如可以按下提交按钮之后就禁用提交按钮，即在submit事件发生后就设置disabled属性。

input和button的type可以修改，select不可以

共有的表单字段方法

每个表单字段都有链两个方法：focus和blur方法，focus用于将浏览器的焦点聚集在某表单元素上，即激活表单元素，使其可以响应键盘事件。

可以在load事件即页面加载完毕后就在表单的第一个表单元素上调用focus方法。

HTML5为表单字段新增了一个属性autofocus，是指这个属性就可以将焦点自动移动到相应的字段。

<input type=“text” autofocus>布尔值

blur将焦点从该元素上移走。

共有的表单字段事件

blur事件：在表单元素失去焦点时触发

focus事件：得到焦点时触发

change事件：input和textarea元素失去焦点且value值发生改变时触发，或者select元素选项改变时触发

当用户改变了当前字段的焦点，或者我们调用了blur和focus方法时都会触发blur事件和focus事件，这两个事件在所有表单元素上都是一样的。

但是change事件在不同的表单元素上是不一样的。

通常，可以使用focus事件和blur事件来以某种方式改变用户界面以提醒用户该元素获得焦点或失去焦点。

change事件则用来验证用户输入的数据形式。

14.2 文本框脚本

在HTML中，有两种方法来表示文本框，一种是input表示的单行文本框，另一种是textarea表示的多行文本框。

<input type="text" size="25" maxlength="50" value="initial value">

<textarea rows="25" cols="5">initial value</textarea>

都可以通过value的值来访问其中的值或者设置其中的值。最好使用JS代码来设置value属性的值。

选择文本

select（）方法用于选择文本框中的所有文本，在调用这个方法时，表单就会获得焦点并且选择所有的文本，这样不用一一删除每个默认值字符。

EventUtil.addHandler(textbox, "focus", function(event){

event = EventUtil.getEvent(event);

var target = EventUtil.getTarget(event);

target.select();

});

在文本框获得焦点时选择所有的文本，是一种常见的做法，通过表单的易用性。

select事件，在调用了select（）方法时就会引发select事件或者用户选择了文本时就会触发该事件。

取得选择的文本

添加了selectionStart和selectionEnd属性，可以取得所选的文本，这两个属性都是基于0的偏移量的值。

选择部分文本

setSelectionRange（）方法，参数为第一个文本的索引和最后一个文本的索引，每个元素都有该方法。

过滤输入

当不允许输入字符，只允许输入数字时，就可以检测keypress事件，当事件发生后检测输入的字符是否为数字，不是的话就屏蔽该输入。

将输入的字符与标准字符相比较，若不符合的话就屏蔽该输入。

操作粘贴板

6个粘贴板事件

beforecopy事件

copy事件

beforecut事件

cut事件

beforePaste事件

paste事件

在实际的事件发生之前，通过before事件可以在向剪贴板发送数据和取得数据之前修改事件。

要访问剪贴板中的数据，可以使用clipboardData对象，在IE中，这个对象是window对象的属性，其他浏览器中是event对象的属性，有三个方法：

getData（）：参数为数据格式，text或者URL

setData（）：参数为数据类型和要放在剪贴板中的文本

clearData（）

自动切换焦点

档用户填写完当前字段时，自动切换焦点到下一个字段。必须知道用户已经输入了已知长度的字符。

HTML5约束验证API

为了将表单提交之前进行验证，HTML5提供了一些新的功能。就是要在HTML标记中为特定的字段提供一些约束。

1.必填字段

第一种情况是在表单字段中指定required属性，在提交表单时该表单字段不能空着。这个属性适用于input、textarea和select字段。

<input type="text" required>

2.其他输入类型

HTML5为input元素的type属性新增了几个值，email和url。要求输入的值必须符合相应的形式。

3.数值范围

input元素的type值还可以是number。

4.输入模式

HTML5为文本字段新增加了pattern属性，是一个正则表达式，用来匹配文本框中的值。

5.检测有效性

使用checkValidity（）方法可以检测该字段是否有效，有效的判断方法是是否符合前面的约束。

validity告诉你什么字段有效

6.禁用验证

设置novalidate属性可以告诉表单不用进行验证。

14.3 选择框脚本

选择框是通过select元素和option元素创建的。HTMLSelectionElement类型有以下属性和方法

add（）：向控件中插入新的option元素，其位置在相关项之前

multiple：布尔值，表示是否支持多相选择

options：控件中所有option元素的集合

remove（）：移除给定位置的项

selectedIndex：基于0的选中项的索引

size：选择框中可见的行数

选择框的type属性不是select-one就是select-multiple，取决于是否有multiple属性

value的值：

若没有选中的项，选择框的value属性保存空字符串

若选中其中一项，value等于该项的value属性的值

若选中一项，没有value特性，则存放其文本

若选中多个项，则value保存第一项的value的值

每个option元素都有一个HTMLOptionElement对象表示，该对象有如下属性和方法

index：当前选项在options集合中的suoyin

label：当前选项的标签

selected：布尔值，表示当前选项是否被选中

text：选项的文本

value：选项的值

选择选项

对于只允许选择一项的选择框来说，访问选中项的最简单的方式就是使用选择框的selectedIndex属性

另一种选择选项的方式就是取得对某一项的引用，然后将其selected属性设置为ture。

添加选项

第一种方法是使用DOM方法

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

newOption.appendChild(document.createTextNode(textTextbox.value));

newOption.setAttribute("value", valueTextbox.value);

selectbox.appendChild(newOption);

第二种方法是使用option构造函数

第三种方法是add（）方法

selectbox.add(newOption, undefined);

移除选项

第一种方法可以使用DOM的removeChild（）方法

第二种是使用选择框的remove方法

第三种是将相应的项设置成null

移动和重排选项

14.4 表单序列化

14.5富文本编辑

第十五章 使用canvas绘图

canvas元素负责在页面上设定一个区域，就可以利用JS在这个区域上绘制图形

15.1 基本用法

<canvas id="drawing" width="200" height="200">a drawing of something</canvas>

该元素对应的DOM元素对象也有width和height属性，可以改变大小，如果不添加任何样式或者不绘制任何图形，看不到这个元素。

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

var context=drawing.getContext("2d");%取得2D上下文对象

使用toDataURL（）方法，可以导出在canvas元素上绘制的元素的URL。

//get data URI of the image

var imgURI = drawing.toDataURL(“image/png”);

//display the image

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

image.src = imgURI;

document.body.appendChild(image);

15.2 2D上下文

填充和描边

属性fillStyle和描边strokeStyle

context.fillStyle="red";

context.strokeStyle="#0000ff";

绘制图形

矩形是唯一一个能直接在2D上下文中绘制的图形。与矩形有关的方法是fillRect（）、strokeRect（）、clearRect（）方法。

这四个方法接收四个参数：矩形x坐标、y坐标、宽度和高度。

fillRect（） 是在画布上绘制的矩形填充指定的颜色，与fillStyle结合使用。

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

//make sure <canvas> is completely supported

if (drawing.getContext){

var context = drawing.getContext("2d");

//draw a red rectangle

context.fillStyle = "#ff0000";

context.fillRect(10, 10, 50, 50);从10,10处开始画矩形

//draw a blue rectangle that's semi-transparent

context.fillStyle = "rgba(0,0,255,0.5)";半透明蓝色

context.fillRect(30, 30, 50, 50);

strokeRect（） 方法用指定描边的颜色来画矩形

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

//make sure <canvas> is completely supported

if (drawing.getContext){

var context = drawing.getContext("2d");

//draw a red rectangle

context.strokeStyle = "#ff0000";

context.strokeRect(10, 10, 50, 50);

//draw a blue rectangle that's semi-transparent

context.strokeStyle = "rgba(0,0,255,0.5)";

context.strokeRect(30, 30, 50, 50);

clearRect（）方法 用来清除在画布上的矩形，可以将画布上的任意地方的矩形区域变成透明的。

绘制路径

开始绘制路径，要调用beginPath（）方法，表示要开始绘制路径了。

arc（x,y,半径，开始角度，结束角度，是否逆时针）方法：画圆

moveto（x,y）方法：将绘图游标移到（x,y）处，不划线

lineto（x,y）方法：从上一点绘制一条直线到（x,y）

rect（x,y,width,height）：绘制矩形路径

路径完成，你想用fillStyle填充，可以使用fill（）方法，想用strokeStyle（）方法描边，可以调用stroke（）方法

isPointInPath（x,y）方法：在画布关闭之前确定画布上一点是否在路径上。

绘制文本

fillText（）方法：参数为绘制的文本字符串，x坐标、y坐标和可选的最大像素宽度

以以下属性为基础：

font：表示文本样式、大小以及字体

textAlign：表示文本对齐方式center、start（在文本左侧）和end（在文本右侧）

textBaseLine：文本的基线，文本的垂直对齐方式

context.font = "bold 14px Arial";

context.textAlign = "center";

context.textBaseline = "middle";

context.fillText("12", 100, 20);

要将文本控制在某个区域，用到measureText()方法，参数为要输入的文本，返回一个对象，该对象有一个width属性。表示文本宽度

变换

translate（x,y）方法:将坐标原点移到（x,y）

rotate（角度）方法：围绕原点旋转一定的角度

scale（x,y）：缩放图像，x方向上乘x，与方向上乘y

save（）方法：保存当前的上下文设置

restore（）方法：将设置返回一级，返回原来的状态。

绘制图像

如果你想将一幅图像绘制到画布上，可以使用drawImage（）方法。传入img元素

var image = document.getElementById("smiley");

//draw regular size

context.drawImage(image, 10, 10);绘图起点

//draw smaller

context.drawImage(image, 50, 10, 20, 30);绘图起点和你期望的高度和宽度

//draw just part of the image

context.drawImage(image, 0, 10, 50, 50, 0, 100, 40, 60);绘图起点和你期望的高度和宽度和图像的一部分（x,y,width,height）;

另一种方法是将另一个canvas元素当做第一个参数，将一个画布上的内容绘制到当前的画布上。

图像不能来自其他的域，否则会出现错误。

阴影

shadowColor：表示的阴影颜色

shadowOffsetX：x轴方向的阴影偏移量，默认为0

shadowOffsetY： y轴方向的阴影偏移量，默认为0

shadowBlur：模糊的像素数，默认为0，即不模糊

context.shadowOffsetX = 5;

context.shadowOffsetY = 5;

context.shadowBlur = 4;

context.shadowColor = "rgba(0, 0, 0, 0.5)";

渐变

createLinearGradient（x,y,x,y）方法：参数为起点坐标和终点坐标，创建一个指定大小的渐变，返回实例的引用

addColorStop（）方法：指定色标，参数为色标位置和颜色，色标位置是一个0到1的数字

var context = drawing.getContext("2d"),

gradient = context.createLinearGradient(30, 30, 70, 70);

gradient.addColorStop(0, "white");

gradient.addColorStop(1, "black");

然后就可以将fillStyle 或者strokeStyle的值设置为这个，用渐变来填充或者描边。

模式

模式其实就是重复的图像。

createPattern（）方法：创建模式，参数为img元素和如何重复的字符串。

var context = drawing.getContext("2d"),

image = document.images[0],

pattern = context.createPattern(image, "repeat");

context.fillStyle = pattern;

context.fillRect(10, 10, 150, 150);

第二个参数为repeat、repeat-x,repeat-y和no-repeat模式，图像只在指定的区域上重复图像。

使用图像数据

getImageData（x,y,width,height)方法：取得原始数据数据，返回ImageData对象的实例，具有以下三个属性。

width,height和data，data是一个数组，保存着图像上每一个像素的数据，每一个像素用四个元素来保存，表示红、绿、蓝和透明度值。

合成

globalAlpha：用于指定全局透明度，介于0到1 之间

15.3 WebGL

是针对canvas元素的3D上下文，是基于openGL指定的。

类型化数组

WebGL涉及的复杂计算需要提前知道数值的精度，因此引入类型化数组。核心是ArrayBuffer对象，每个对象只表示内存中指定的字节数。

var buffer=new ArrayBuffer(20);

byteLength属性用来表示其中包含的字节数。

视图

可以用ArrayBuffer类型来创建数组缓冲器视图。最常见的视图是DataView，通过它可以选择ArrayBuffer中的一小段字节

var view=new DataView(buffer);创建新的视图，基于整个缓冲器

var view=new DataView(buffer，9);创建一个开始于字节9的新视图

var view=new DataView(buffer，9，10); 创建一个从字节9到字节18的新视图

view.byteOffset和view.byteLength来访问字节偏移量和字节长度信息

读取和写入DataVierw时，要根据实际操作的数据类型，选择相应的getter和setter方法，不同的方法的第一个参数都是偏移量，表示从哪里开始读取和写入。但是我们要自己记住不同的类型需要占用的字节数，以及将数据保存到哪里。带来很多工作量。

DataView视图能让我们在字节级别上读写数组缓冲器上的数据。

类型化视图

类型化数组也是一种数组，而且元素必须是某种特定的数据类型，除此之外与DataView无差别。

Int8Array:表示8位二补整数

Unit8Array:表示8位无符号整数，8 位数据保存要用1B，16 位要用2B，32位要用4B

var view=new Int8Array(buffer);创建新的视图，基于整个缓冲器

var view=new Int16Array(buffer，9);创建一个开始于字节9的新视图

var view=new Unit16Array (buffer，9，10); 创建一个从字节9到字节18的新视图

意味着可以在缓冲器中保存不同类型的数据。

每一个视图构造函数都有一个属性BYTES\_PER\_ELEMENT，表示每个元素需要多少字节，可以利用该属性来初始化缓冲器。

优点2：可以不用创建ArrayBuffer对象，只要传入希望数组保存的元素数，相应的构造函数就会自动创建一个ArrayBuffer对象。

var view=new Int8Array(20);

优点3：可以将常规数组转化为类型化数组，只需要将数组传入构造函数即可

var view=new Int8Array([10,20,30,40,50]);

可以通过方括号方法和下标法来访问数组中的每一个元素，也可以为它赋值，数据类型不匹配不会抛出错误。

subarray（）方法用来基于底层数组创建一个新的视图，参数为开始索引和结束索引。

WebGL上下文

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

//是否有canvas元素

if (drawing.getContext){

var gl;

try {

gl = drawing.getContext("experimental-webgl");是否有WebGL，取得上下文

} catch (ex) {

//noop

}

if (gl){

一般都把WebGL上下文对象命名为gl。

通过给getcontext方法传入第二个参数，可以为上下文设置一些选项。

常量

WebGL支持大多数openGL中的常量。

方法命名

很多方法都试图给出有关数据类型得信息，gl.uniform3i()表示接收3个整数参数，i表示整数，f表示小数，后面加v表示数组

准备绘图

准备绘图之前都要清除缓冲区

gl.clearColor(0,0,0,1);

gl.clear(gl.COLOR\_BUFFER\_BIT);

视口与坐标

开始绘图之前要先定义WebGL的视口，使用viewport()方法，参数为相对于canvas元素的x，y坐标以及宽度和高度

视口坐标的原点在canvas元素的左下角，x,y轴是分别向上和向右。这样就可以在canvas元素的一部分上绘图。

视口内部的坐标与外部的坐标也不一样，视口的中心是原点。

缓冲区

创建上下文的缓冲区

var buffer=gl.createBuffer();创建一个缓冲区

gl.bindBuffer(gl.ARRAY\_BUFFER,buffer);将该缓冲区设置为上下文的当前缓冲区

gl.bufferData( gl.ARRAY\_BUFFER ,new Float32Array([0 0.5 1]),gl.STATIC\_DRAW);使用数据初始化缓冲区

释放缓冲区

gl.deleteBuffer（buffer）；

错误

JS与WebGL的最主要的区别在于后者一般不会抛出错误，所以必须在调用某个可能错误的方法后调用gl.getError（）方法，返回错误类型的常量。

着色器

分为两种着色器：顶点着色器和片段着色器

绘图

只能绘制三种形状：点、线和三角。gl.drawArrays()和gl.drawElements()用于数组缓冲区和元素数组缓冲区

第一个参数为常量，表示要绘制的图形

第十六章 HTML5脚本编程

16.1跨文档消息传递

简称XDM，表示不同域之间传递消息，核心是postMessage（）方法，向另一个地方传递消息，另一个地方指的是页面中的iframe元素或者由当前页面弹出的窗口。

接收两个参数：一条消息和表示消息接收方来自哪个域的字符串

var iframeWindow=document.getElementById("myframe").contentWindow;

iframeWindow .postMessage("a secret","http://www.ffs.com");

接收到XDM消息后，会触发window对象的message事件，触发后，向onmessgae事件处理程序中传入三个参数：

data：传入的信息

origin：接收消息的域

source：发送消息的文档的window对象的代理，不是实际的window对象，只用于调用postMessage方法

有了XDM，可以保证页面不受恶意内容的侵扰，因为它只通过XDM与内嵌的框架进行通信。

16.2 原生拖放

拖放事件

拖动某些元素时，将依次触发下列事件

dragstart

drag

dragend

按下鼠标并移动鼠标时，会触发dragstart事件，然后触发drag事件，最后触发dragend事件。三个事件的目标都是被拖动的元素。

当某个元素被移到有效的放置目标上，就会依次触发dragenter、dragover和dragleave或drop，三个事件的目标都是作为放置目标的元素。

自定义放置目标

当拖动元素被放到某些无效的放置目标时，不会触发drop事件，但是可以将这些元素变成有效的放置元素，即重写dragenter和dragover的目标行为。

dataTransfer对象

只有拖放而没有数据变化时没有用的，为了在拖放操作时实现数据交换，引入dataTransfer对象，用于在拖放事件处理程序中向放置目标传递字符串。

该对象的两个主要方法getData（）和setData（）方法

两个方法的第一个参数都是传输的数据类型，text或者URL

event.dataTransfer.setData("text","some text");

var text=event.dataTransfer.getData("text");

event.dataTransfer.setData("URL","http://dasd.com");

var text=event.dataTransfer.getData("URL");

当拖动文本框中的文本时，浏览器会调用setData（）方法，并将文本以text的格式保存在对象中，当拖动图像时，会保存URL。

dropEffect和effectAllowed

dataTransfer对象的两个属性：dropEffect和effectAllowed，表示被拖动的元素和作为放置目标的元素能够接收什么操作。

可拖动

默认情况下，图像、连接和文本时可以拖动的，文本在选中的情况下可拖动，其他任何时候都可以拖动。想让元素可拖动，可以设置draggable属性为ture

其他成员

dataTransfer还有下列属性和方法：

addElement（）；为拖动操作添加元素

clearData（）：清除以特定格式保存的数据

setDragImage（）：

types：当前保存的数据类型

16.3 媒体元素

在网页中嵌入相关的视频和音频，添加了两个标签：audio和video。表示音频和视频

16.4 历史状态管理

第十七章 错误处理与调试

17.1 浏览器报告的错误

浏览器都具有某种向用户报告JS错误的机制，在编写脚本时别忘了打开JS的报告功能。

IE

IE是唯一一个在窗口界面显示JS错误信息的浏览器，左下角会出现一个黄色图标。

17.2 错误处理

try-catch语句

作为处理异常的一种标准方式

try {

window.someNonexistentFunction();

} catch (error){

alert(error.name);

alert(error.message);

}

finally语句

该子句一经使用，它的代码就一定会执行。return语句也会被忽略。

错误类型

Error

EvalError：使用eval函数异常时抛出

RangeError：数值超出范围时抛出

ReferenceError：找不到对象的情况下抛出

SyntaxError：将语法错误的JS字符串传入eval函数时抛出

TypeError：在变量中保存意外的类型时抛出

URIError：URL格式错误抛出

抛出错误

第十八章 javascript 与XML

也是一种标记语言，用于处理数据，将数据进行标记，使其具有结构性。

18.1 浏览器对XML的支持

创建一个XML文档

var xmldom = document.implementation.createDocument("", "root", null);

alert(xmldom.documentElement.tagName); //"root"

var child = xmldom.createElement("child");

xmldom.documentElement.appendChild(child);

DOMparser类型

目的：将XML解析为DOM文档

var parser = new DOMParser();创建一个DOMparser实例

var xmldom = parser.parseFromString("<root><child/></root>", "text/xml");调用parseFromString方法。第一个参数为要解析的XML字符串，第二个为参数类型

alert(xmldom.documentElement.tagName); //"root"root即为其文档元素，child为其子元素，就可以使用DOM方法对文档进行操作了

alert(xmldom.documentElement.firstChild.tagName); //"child"

var anotherChild = xmldom.createElement("child");

xmldom.documentElement.appendChild(anotherChild);

var children = xmldom.getElementsByTagName("child");

alert(children.length); //2

XMLSerializer类型

将DOM文档序列化为XML字符串

var parser = new DOMParser();

var xmldom = parser.parseFromString("<root><child/></root>", "text/xml");

//convert back into XML

var serializer = new XMLSerializer();

var xml = serializer.serializeToString(xmldom);

alert(xml);

18.1.4IE8及之前的XML

18.2浏览器对Xpath的支持

XPath是用来在文档中查找节点的一种手段，DOM3中对XPath模块做出了标准。

18.2 DOM3

第二十二章 高级技巧

22.1 高级函数

安全的类型检测

typeof（）操作符对正则表达式经常显示函数，而instanceof操作符则在多个全局作用域中会出问题

在任何值上调用object的原生toString（）方法会显示[object nativeconstructorname],即原生构造函数名

作用域安全的构造函数

当使用new调用构造函数时，this对象会指向新创建的对象实例，但是如果忽略new操作符，this对象可能会指向window对象，给全局对象赋值。

解决问题的方法就是创建一个作用域安全的构造函数，先检测一下this的指向，在决定怎么调用构造函数。

惰性载入函数

为了避免代码多次运行，解决问题是使用惰性载入方法。

第一种是在函数被调用时再处理函数，第二种是在声明函数时就指定适当的函数。保证函数的分支只执行一次。

函数绑定

函数绑定可以使得在将函数作为参数传递的同时保留代码执行环境。

使用bind（）方法，将函数绑定到特定的环境中。

函数柯里化

22.2防篡改对象

开发人员可以将对象定义为防篡改对象，而且一旦定义了就无法取消。

不可扩展对象

默认情况下，任何对象都是可以扩展的，即可以增加他的属性和方法。

使用Object.preventExtensions()方法可以防止对象被扩展。

var person = { name: "Nicholas" };

Object.preventExtensions(person);

person.age = 29;

alert(person.age);undefined

虽然不可以扩展对象，但是已有的属性和方法却可以删除和修改，使用Object.isExtensible（）方法可以确定该对象是否可以扩展

密封的对象

密封对象不可扩展，而且不可删除属性和方法，但是可以修改，所有成员[[configurable]]属性都设置成false。

使用Object.seal（）方法，seal是密封的意思

var person = { name: "Nicholas" };

Object.seal(person);

person.age = 29;

alert(person.age); //undefined

delete person.name;

alert(person.name); //"Nicholas"

使用isSealed（）方法可以确定对象是否被密封了。

冻结的对象

这是最严格的房篡改对象级别，不可扩展，是密封的，而且不可修改，每个属性和方法的[[writable]]属性都被设置为false。

用Object.freeze()方法来冻结对象

var person = { name: "Nicholas" };

Object.freeze(person);

person.age = 29;

alert(person.age); //undefined

delete person.name;

alert(person.name); //"Nicholas"

person.name = "Greg";

alert(person.name); //"Nicholas"

Object.isFrozen()方法来确定是否被冻结

冻结或者密封对象可以防止重要的对象被篡改，有利于保护对象。

22.3 高级定时器

定时器对队列的工作方式是当特定的时间过去后，将代码添加到队列中，等空闲时执行，而不是立刻执行。

重复的定时器

setInterval重复定时器的问题是定时器代码可能在代码再次被插入到队列之前还没有完成执行，导致定时器代码一直执行，没有停顿。

出现的问题：某些间隔会被跳过，多个定时器的代码执行间隔会比预期的小。

解决：使用链式setTimeOut（）调用

Yieldind Processes

JS占用的内存大小和处理器时间呗严格控制，其中一个控制是长时间运行脚本的限制，产出一定时间就会提醒是否继续运行脚本，定时器可以跳过此限制

若循环占用太长的时间，可以用定时器将其分成几部分，定时进行处理，就不会出现脚本长时间运行的错误。

函数节流

当函数第一次执行时，创建一个定时器，在间隔后执行代码，当第二次调用时，若上一次的定时器还没有被执行，就替换为一个新的定时器。

22.4 自定义事件

22.5 拖放

拖放功能也流行到了web上，使得某些元素可以随着鼠标移动。

第二十三章 离线应用与客户端存储

支持web离线应用是HTML5另一个重点，即设备在没有网的情况下也能进行的应用，

开发离线应用的几个步骤：

首先确定设备是否能连上网，以便执行下一次操作

还必须能访问一定的资源，才能正常工作

最后，必须有一定的空间来保存数据，无论是否上网都不影响读写

23.1 离线检测

确定设备是否能连上网，HTML5定义了一个navigator.onLine属性，为ture表示能上网，为false表示不能上网。

除了该属性，为了更好地确定网络是否可用，HTML5定义了两个事件：online和offline事件，当网络状态变化时就会触发这两个事件。

在页面加载后，先通过navigator.onLine属性来取得初始状态，然后通过上述两个事件来确定网络连接状态是否变化。

23.2 应用缓存

HTML5的应用缓存是专门为开发离线web应用而设计的，就是从浏览器的缓存区中分出来一部分缓存区。要想在这个缓存区中保存数据，可以使用描述文件，列出要下载和缓存的资源。

#CACHE MANIFEST

#Comment

file.js

file.css

要将页面与描述文件相关联，可以在html元素中的manifest属性中指定这个文件的路径。

有一个applicationCache对象，该对象有一个status属性，属性的值是一个常量。表示应用缓存的当前状态。

23.3 数据存储