Javascript 权威指南

颜色约定:

- 紫色：表示一个类型或者一种定义的名称
- 蓝色：一句话里面的重点
- 绿色：注释
- 深红：很有用的技巧
- 鲜红：需要注意的地方

第二章:语法结构

1. Javascript是大小写敏感的,这点在HTML编程中很重要.
2. 写Javascript时不要省略语句结束时的分号! 一条语句不要分成两行写. 因为Javascript会自动在未加分号的语句后面加上分号,这有时会误解自己的意思.

第三章:数据类型和值

1. Javascript支持三种数据类型: 数字,字符串,布尔值. 此外还有空(null)和未定义(undefined).
2. 一个对象表示的是值的集合(也可以是复合值,如其它的对象) Js的对象有两种,有序的(数组)和无序的及函数. 感觉在Js中什么都是对象.
3. Js中不区分整形与浮点型,所有的数字都是浮点的.采用的浮点是64位的,在有时的运算时(如位运算)是32位的.
4. 16进制数以"0x"开头,8进制数以"0"开头.
5. 当大于无穷在或小于无穷小时,Js中表示为Infinity和-Infinity.
6. 当用0/0(据说有些其它的运算也会)时,会返回NaN.这是一个特殊的非数字的数字. NaN与任何数都不相等,包括它自己(这点与C#中相同). 有isNaN()检测这个值是否为NaN, isFinite()检测一个数字是否为NaN,无穷大或无穷小.
7. Js中没有char类型,表示char类型时就使用长度为1的字符串. 由单引号定界的字符串中可以包含双引号,由双引号定界的字符串中可以包含单引号. 字符串必须写在一行,写成两行可能会被截断.
8. 数字会在需要的时候自动转换成字符串, eg: s=100+""; 可以用String()或toString()显式的转换为字符串. toString(8)表示以8进制来进行转换.

   3个数字到字符串的方法: var n=123456.789;

   n.toFixed(0); // "123457"

n.toFixed(2); // "123456.79"
n.toExponential(1); // "1.2e+5"
n.toExponential(3); // "1.235e+5"
n.toPrecision(4); // "1.235e+5"
n.toPrecision(7); // "123456.8"

9. Tip: 把字符串转换成数字 var number="23"-0; //使用-号而不是+号,+会引发字符串的连接
10. parseInt()及parseFloat() 它们从字符串的开始处转换和返回任何的数字,忽略或舍去非数字部分. 如果它们不能返回数字,就会返回NaN
    eg: parseInt("3 asdfasdf"); //return 3
11. 布尔值对应数字(true->1;false->0). 其它类型转换为布尔类型:
    - 数字:0或NaN转换为false,其它的数字都为true
    - 字符串: 空字符串为false,其它均为true
    - 对象: 任何非空的对象,函数,数组均为true,空值及未定义的对象为false
12. Tip: var abc=!!x; //两次!运算转换为布尔值
13. Js中的函数也是真正的数值,它可以存储在变量,数组,对象中,这点与其它的语言有很大的不同.
14. Js中的对象可以作为关联数组来使用,因为它们能够将任意的数据值和任意的字符串关联起来.
    eg: abc.length 等价于 abc["length"] //这样就可以用拼接字符串的方式调用属性或者方法.
15. 对象直接量: var point={ x:2.3, y:1.2 };
16. 当一个对象用于字符串环境时,Js调用对象的toString()方法,并使用该函数返回的字符串的值.
    当一个对象用于数字环境时,Js先调用对象的valueOf()方法,如果返回的是基本类型的值,那么就使用. 否则就调用toString(),然后再把返回的字符串转换为数字.
17. undefined和null并不相同,但是==运算符把这两个看作相等, 区分undefined和null可以使用===运算符或typeof
18. Error类表示运行时的错误,每个Error对象都有一个message属性用于存放特定的消息. 预定义的错误对象有Error,EvalError,RangeError,ReferenceError,SyntaxError,TypeError,URIError
19. 包装的原理: Js的基本数据类型都有一个对应的对象,这些类是那些基本类型的包装.
    当我们使用一个字符串时,Js会为这个字符串内部创建一个String对象,String对象代替了原始的字符串.
20. 字符串类型在Js中比较特殊,理论是字符串应该是引用类型,但是在Js中的字符串是表现为值类型的.

第四章:变量

1. 在Js中,变量在使用前是必须声明的,如果没有显式的声明,Js会隐式的声明它,隐式声明的变量总是被创建为全局变量,即使该变量只在一个函数体内使用,所以在使用一个变量前最好显式的声明它.
2. 由var声明的变量是永久的.使用delete删除这些变量会引发错误
3. Js的函数是可以嵌套的,这点像Pascal
4. Js没有块作用域,函数中声明的所有变量,无论是在那个块中声明的,在整个函数中它们都是有定义的.所在,将所有的声明放在函数的开头是个好习惯.
5. 未定义的变量与未赋值的变量是不同的,试图读取未定义的变量会引发一个错误,读取未赋值的变量会返回undefined.
6. 值类型保存的是实际的值,引用类型保存的是对值的引用,这点与C#相同. Js中也有与C#类似的垃圾收集机制.
7. Js中,变量就是属性,全局变量就是一个特殊的全局对象的属性. 局部变量就是调用对象的属性.
8. Js中变量作用域中很重要的就是作用域键,Js执行环境有一个作用域键,当函数引用一个变量时,首先检查的是调用对象(局部作用域),其次再检查全局对象是否有这个属性.当作用域键有很多对象时,以此类推.

第五章:表达式和运算符

1. 赋值操作的优先级非常低,几乎总是最后才被执行的.
2. 在Js中,由于所有的数字都是浮点型,所以除法的结果也是浮点型的. 如5/2的结果是2.5,而不是2. 除数为0时的结果是正无穷或者负无穷. (这点与C#不同)
   特殊的0/0的结果是NaN
3. 对于非数字的对数,"+"运算符还有将对数转换成数字的功能.如果参数不能转换成数字,将返回NaN var a = +"1000"; //return 1000
4. ===运算符是等同运算符,它采用严格的同一性定义检测两个运算数是否完全相同.
   ==运算符是相等运算符,它采用宽松的同一性定义(允许进行类型转换)检测两个运算数是否相等.
   eg: var a=("23"===23); //return false;
   var a=("'23"==23); //return true;
   null == undefined结果是true, 但是 null === undefined结果是false
5. in 运算符检测左边的字符串是否为右边的对象的属性 var a="toString" in Date; //return true;
   in 可以这样写 if(o.x==undefined) return true;
   instanceof 运算符检测左边的对象是否为右边所描述的类型 (相当于C#中的is运算符)
   var a=new Date();
   var b=a instanceof Date; //return true;

6. 一个容易搞错的地方 (所以,不要吝啬括号,不确定的地方就用括号来确定)
   var s=1+2+" bind mice"; //return "3 blind mice"
   var s="blind mice:"+1+2; //return "blind mice: 12"
7. 如果位运算符对超过32位的数字使用,将返回NaN. 因为位运算时把变量当作整数来处理的.
8. 使用delete使用的运算数是一个不存在的属性,它将返回true(可以这样理解:当返回时,对象是没有那个属性的,不管是不是一直就没有,反正是成功的使对象没有那个属性了) delete并不是将属性的值设置为undefined,而是将它们彻底的删除,它们将不复存在.
9. void运算符总是返回undefined.
10. 逗号运算符从左到右计算表达式,最后返回最右边的表达式的值.
11. for (p in o) 相当于C#中的foreach语句. 这个语句遍历对象o的所有属性.
    for/in语句并不一定能枚举出所有的属性,有一些标记为只读的,永久的属性是不可枚举的.
    for/in循环列出的属性没有特定的顺序,而且虽然它可以枚举出所有的用户定义的属性,但是却不能枚举出某些预定义的属性和方法(内建类型的原型对象的属性不能用一个for/in循环来枚举的.
12. ()其实是一个运算符,这个运算符用来调用函数.

第六章:语句

1. Js中的switch语句是采用===来匹配的,所以表达式必须在没有类型转换的情况下进行匹配.
2. 用含有副作用的case语句来作表达式是不好的习惯,因为每次执行switch语句时,并不会计算所有case表达式. 有时会很难预测程序的行为,所以应该把case表达式限制在常量表达式的范围.
3. 当break和标签一起使用时,它将跳到这个带有标签的语句的尾部,或者终止这个语句.该语句可以是用任何括号括起来的语句,它不一定是循环或者switch
   continue只能使用在while,do/while,for,for/in中.
   while循环与for循环中使用continue语句并不相同. while中,它直接跳到循环处,在for中要先计算increment表达式,再跳转到循环条件处.
4. 函数定义通常出现在Js代码的顶层.它们可以嵌套在其它函数中,但是也只能在那些函数的顶层.函数不能出现在if,while循环或者其它任何语句中(这点和Pascal不同的)
5. 语句是在运行时执行的,而函数则是在实际运行之前,当Js代码被解析或者被编译时定义的.(函数定义在解析时发生,而不是在运行时发生的)
   alert(f(4)); //显示16,函数在解析时已经定义了
   var f=0; //重新覆盖了f的定义
   function f(x) { return x*x; }
   alert(f); //显示0
6. with语句用于暂时修改作用域键 with语句的效率很低,不应该使用.

7. Js中允许合法的使用空语句. 所以 for(int i;i<10;i++); //有时这是一种很难被检查出来的错误 Js会把这种当作一个空循环来处理

第七章:对象和数组

1. 对象是已命名的值的一个集合.而数组是一种特殊的对象,可以把数组当作是有序的值的集合.
2. 对象直接量: var circle={x:"abc", y:12, z:true};
3. Object()构造函数创建一个空的对象. 可以这样简写 var a={}; 类似的空数组可以这样 var a=[];
4. 通用的Object属性和方法:
   - constructor属性: 这个属性引用了类的构造函数
   - hasOwnProperty()方法返回属性是否为对象的非继承的属性 var o={};
     o.hasOwnProperty("toString"); //return true;
5. 数组应该是大于等于0的整数,如果使用了负数,浮点数或者其它的.Js会把它转换成字符串,并作为属性的名字.
   eg: var a[-12.2]="asdf"; //其实相当于a.-12.2="asdf";
6. 如果给length设置了一个比它的当前值小的值,那么数组将被截断,这个长度之外的元素都会被抛弃.
7. Js不支持多维数组,当因为Js的数组其实是对象,所以要实现多维数组很简单的.
8. 数组的方法:
   - join()方法: 把一个数组的所有元素都转换成字符串 var a=[1,2,3]; var s=a.join(); //s="1,2,3"
   - reverse(): 返回顺序颠倒了的数组
   - sort(): 返回排好序的数组,可以给sort()传递一个方法用以排序 var s=a.sort(function (a,b) { return a-b; });
   - concat(): 将新数组或数值连接到数组上 (注:只能展开一级)
     a.concat(4,5) //return [1,2,3,4,5]
     a.concat([4,5],[6,7]) //return [1,2,3,4,5,6,7]
     a.concat(4,[5,[6,7]]) //return [1,2,3,4,5,[6,7]]
   - slice(): 返回数组的一个片段 a.slice(1,2); //return [2,3]
   - splice(): 插入和删除数组元素的通用方法. 需要时查看参考手册
   - pop(),push(): 把数组当作栈来使用,把新元素加到尾部或从尾部删除
   - unshift(),shift(): 与pop,push相同,不同的是操作头部

第八章:函数

1. 如果return语句没跟表达式,就返回undefined.没有return的情况也一样
2. 函数可以有可变个数的参数. 如果传递的参数超过要求的个数,多余的会被忽略掉.如果个数少于要求的个数,忽略的参数会赋于undefined. 可选的参数应该放在参数列表的末尾处以使它们可以被忽略掉.

   函数传递的全部参数可以使用它的arguements数组来引用.(arguments只能在函数体内部使用)

   arguments还定义了属性callee来引用正在执行的函数

   函数要求的参数个数可以从它的原型的arguments数组中获得
3. 定义的同时调用一个函数 (function(x){return x*x;})(10);　　//匿名函数
4. 函数命名习惯: 以动词开始,以小写字母开始 如 setTitle()
5. 在Js中,函数也是数据,所以可以像数值一相进行赋值,传递操作.(这点有些变态,但是很好用) 甚至为函数定义新的属性..
6. 在方法体是.this关键字指向调用方法的对象.在全局总数中,this指向全局变量
7. 函数自身的length属性返回的是函数所需要的实际参数的数目.也就是函数的形式参数列表中声明的形式参数的数目

   if(arguments.length!=argument.callee.length) throw new Error(" Argument Exception");　//用于检查传递的参数数目是否正确的技巧
8. 所有的函数都有两个方法:
   - call(): 临时以某个对象的身份来调用此函数,这很有用

     f.call(o,1,2); 等价于以下代码

     o.m=f;

     o.m(1,2);

     delete o.m
   - apply(): 与call()类似,只不过要传递给函数的参数是由数组来指定的.　f.apply(o,[1,2]);
9. Js中的函数是通过词法来划分作用域的,而不是动态划分.意味着,函数在定义他们的作用域中运行,而不是执行它们的作用域.
10. 如果把对嵌套的函数的引用保存在一个全局变量中,就构成了一个闭包.

    我对闭包的理解是: 由于垃圾收集机制是采用如果一个变量已经没有任何链接可以指向它,就标记它已经失效,可以执行垃圾收集了.所以把一个嵌套的函数的引用保存在一个全局变量中,这个函数就总能够被链接到,那么它就永远不会被执行垃圾收集.它就是永久有效的局部变量.
11. Js的函数是将要执行的代码以及这些代码的作用域构成的一个综合体
12. Function构造函数很不好用,几乎很少使用. 而且Function构造出来的函数不遵循词法作用域,它总是顶层的函数. 所以很不好用.

第九章:类,构造函数和原型

1. 构造函数通常是没有返回值的,这样的话构造函数初始化作为this的值来传递对象.如果有返回值时,构造函数将返回定义的返回值,而不是新创建的对象
2. 每个Js对象都有一个原型并从原型那里继承属性. 一个对象的原型就是它的构造函数的prototype属性的值.

   所以,通常这样定义对象的方法:

   ```
   function Rectangle(w,h){
     this.width=w;
     this.heigth=h;
   }
   Rectangle.prototype.area=function(){ return this.width*this.height;};    //原型是定义不变的方法及属性的理想地方
   ```

   使用这种在原型上创建方法的好处在于 : 这个新创建的方法会成为实例方法,而不是类方法(静态方法)
3. 每个类都有一个带有一组属性的原型对象. Js必须在读取和写入属性值时,执行一种基本的不对称. 这是Js从原型中继承的原理:

   当读取o的属性p时,Js先检查o是否有一个名为p的属性,如果没有,检查它的原型是否有一个名为p的属性,这样就能够保证继承的机制的生效. 但是当写入o的属性p时,Js不使用原型对象,直接给o加一个p属性然后写入(这样才能保证类的实例不会覆盖原型的属性).
4. 扩展内建的类型: String.prototype.endsWith=function (c) { return (c==this.charAt(this.length-1)); }

   除非理由充足,否则还是不要扩展内建的对象. 而且,绝对不应该为Object扩展属性或方法.

   但有时候,扩建内建类型是一个解决浏览器兼容性的办法,虽然比较危险
5. 在Js中模拟类:
   - 实例属性: 普通的由构造函数创建和初始化的属性
   - 实例方法: Rectangle.prototype.area=function(){ return this.width*this.height;}; (在实例方法中可以使用this引用调用时所基于的对象)
   - 类属性(静态属性): Rectangle.Unit=new Rectangle(1,1); //直接在构造函数上定义
   - 类方法(静态方法): 与定义类属性类似 (注:在静态方法中this引用构造函数自身)
6. Js中没有定义私有成员的方法,但是可以用闭包实现私有成员.
7. 有时,在将对象转换成字符串时,ValueOf会比toString更优先使用,所以在需要转换成字符串时可以显式的调用toString来确保正确的执行
8. 如果尝试使用Js的关系运算符,Js首先调用对象的ValueOf()方法,如果这个方法返回一个基本类型值,就比较它们.

9. Js中继承的原理：只要确保子类的原型对象本身是父类的一个实例,这样它就会继承了父类的所有属性
    - 首先在子类的构造函数中调用父类的构造函数. ParentClass.call(this,x,y,z);
    - 必须显式的把子类的原型对象设置为父类的一个实例 ChildClass.prototype=new ParentClass();
    - 显式的设置子类的原型对象的constructor属性为父类的构造函数 ChildClass.prototype.constructor=ParentClass;(因为已经将子类的原型对象设置为父类的一个实例了,所以构造函数当然应该是ParentClass)
    - 删除父类构造函数在原型对象中创建的所有属性,因为重要的是原型对象从它的原型那里继承的那些属性.
      delete ParentClass.prototype.x;
      delete ParentClass.prototype.y;

    在子类中调用父类中已经被覆盖了的方法 ParentClass.prototype.toString.apply(this);

10. typeof(null)=object, typeof(undefined)=undefined,另外,任何数组的类型都是object,因为数组是对象.但是,任何函数的类型都是function,尽管函数也是对象
11. 对象是它自己的类的一个实现,也是它的任何父类的一个实例.
12. 查看对象类型的绝佳方法：Object.prototype.toString.apply(o);

第十章:模块和名字空间

1. Js模块的第一条规则：一个模块不应该为全局变量添加多于一条的标记
2. 在测试一个名字空间是否存在之前,应该先用var声明一个全局标记,但是不要去赋值. 因为,读取一个未声明的全局标记会抛出异常,而试图读取一个未定义的标记,只是返回一个未定义的值.
   var com;
   if(!com||!com.davidflanagna||!com.davidflanagna.class) throw new Error("Not Load Namespace");
3. 模块的初始化代码应该被放到一个匿名的函数中去,在定义后直接调用 (function() { //do some })();
4. 模块的私有属性都应该加上"_"下划线做前缀
5. 为了区分公有变量和私有变量,可以使用两个名字空间,一个名字空间存储公有变量,一个存储私有变量.这样就能够很好的区分出来.

第十一章:正则表达式的模式匹配

1. 可以通过RegExp()构造函数来创建一个RegExp对象(var a=new RegExp("\\d{5}","g");),不过通常使用直接量,即:
   var regEmail=/s[abcd]/;　　//用"/"包括起来的将做为正则表达式
2. 标志: 它说明高级模式匹配的规则. 标志是在"/"符号之外说明的,即它们不出现在两个斜杠之前,而是位于第二个斜杠之后. eg: var a=/\bjava\b/gi;
   - i: 执行不区分大小写的匹配
   - g: 执行一个全局的匹配.(找到所有匹配,而不是只找第一个)
   - m: 多行模式(^匹配一行的开头,$匹配一行的结尾)
3. 用于模式匹配的String方法:
   - search(): 以正则表达式为参数,返回第一个与之匹配的子串的开始字符的位置,如果没有匹配,返回-1 eg: "Javascript".search(/script/i);
   - replace(): 执行检索与替换操作,第一个参数是正则表达式,第二个参数是要替换的字符串. 如果正则表达式中有标记"g",就替换所有的匹配,否则只替换第一次匹配.(第二个参数可以是函数,用以动态的计算字符串)
   - match(): 如果正则表达式是全局的,就返回所有匹配结果的数组.
     否则,返回一个这样的数组:第一个元素为完整的匹配,其余的为与表达式匹配的子串(也就是replace中的$n 对组的引用) 返回的数组还包含两个属性,index和input: index为匹配出现在位置,input为原字符串的副本
   - split(): 分隔字符串,返回一个数组
4. RegExp的类方法(静态方法):
   - exec(): exec总返回一个匹配,而且提供该匹配的完整的信息. 当正则表达式有"g"标记时,它将把该对象的lastIndex设为匹配字符串的位置,下次调用时会从这个位置开始匹配,这样就可以遍历所有的匹配.(如果没有发现匹配,会把lastIndex置0)

     ```
     var pattern=/Java/g;
     var text="Javascript is more fun";
     var result;
     while ((result=pattern.exec(text))!=null){
        alert("Matched '"+result[0]+"'"+"at postion "+result.index+"; next search begin at "+result.lastIndex);
     }
     ```
   - test(): 等价于调用exec(),如果exec()返回的不是null,它就返回true,否则返回false.

第十三章:客户端Javascript

1. 在客户端Javascript中,表示HTML文档的是Document对象,Window对象代表显式该文档的窗口(或帧).Window对象是窗户端编程中的全局对象.
2. 在编写使用多窗口(帧)的Javascript应用程序时,应用程序中出现的每个窗口都对应一个Window对象,而且都为窗户端面Javascript代码定义了一个唯一的执行环境.
3. 具有src属性的<script>标记的行为像指定的Javascript文件的内容直接出现在标记之间一样的.
4. 同源安全策略不允许来自一个域的文档中的Javascript和来自另一个域的内容交互.(脚本本身的来源则没有什么关系,只是关系到脚本被嵌入的文档的来源)
5. 如果编写了一个并不产生任何文档输出的脚本,例如定义了一个函数但并不调用document.write()的脚本,可以使用<script>标记中的defer属性来提示浏览器这样做是安全的:继续解析HTML文档并延迟脚本的执行,直到遇到一个无法延迟的脚本.
   defer属性只在IE中有效,且在IE中,脚本被延迟到了文档的结束时才被载入!(千万要注意)
6. 在Js中如果出现"</script>",HTML解析器会把它当作Js脚本的结束,所以这时就应该拆分为"</"+"script>"两个字符串.(如果脚本包含在CDATA中,就没有这个问题)
7. 当浏览器载入一个Javascript URL时，它会执行URL中所包含的Javascript代码,并且使用最后一个Js语句或表达式的值,转换为一个字符串,作为新载入的文档的内容显示. 如果没有返回值,那么将执行Js代码而不改变当前显示文档.
   如果需要确保Js中最后没有返回值,可以使用一个void(0)函数作为最后一条语句.
8. 一个Js代码可以作为一个<from>的标记的action属性,这样当用户提交这个表单时,URL中的Js代码就会执行.
9. 脚本是按照它们出现的顺序来执行的.
10. 如果一个脚本只是定义了稍后使用的函数和变量,并且没有调用document.write()函数或尝试修改文档内容,惯例规定它应该出现在文档的<head>而不是<body>中.
11. onload事件是在文档完全解析之后调用的(所以onload中可以访问所以元素及脚本定义的所有函数及声明的所有变量),它们必须不调用document.write,任何这样的调用都会重新打开一个新的文档并覆盖掉当前的文档,而不是在当前文档后面添加,用户甚至没有机会看到文档.
    onunload事件给于页面是的Js最后一次调用的机会,它会在文档关闭时执行.(点关闭按钮时也会执行)
12. Js中单线程的,如果Js需要较长时间的运行,应该给出一个进度指示器来告诉用户浏览器没有崩溃.
13. 在onload事件已经触发后操作文档是安全的,并且也应该这么做,使用onload句柄触发所有的文档修改.
14. 如果页面中包含一些较大的图片,如果希望在图片载入之前就操作文档,可以把代码放在文档的最后来达到这个目的.
    IE中:可以把代码放在一个有defer属性的<script>标记中(这利用了IE解析defer的一个错误)

FF中:DOMContentLoaded事件中执行代码.这个事件在文档被解析而外部对象还没载入时执行.

15. 功能测试是解决不兼容性的一种强大的技术 (利用的是Js中对不存在的属性及方法应用在布尔值环境时返回的是false)

```
if(element.addEventListener){}          //支持W3C的标准语法
else if(element.attachEvent){}          //IE支持的方法
else {}                                 //提示浏览器不支持
```

16. Js的恰当角色是增加信息的表现力,而不是负责信息的表现.Js可访问性的一条重要原则是:即使Js解释器被关闭,使用它的页面也能(或者至少以某种形式)发挥作用.
17. Js的一些限制(这些限制是安全所必须的)
    - Js可以打开及关闭窗口.现代的浏览器一般要求只有用户的点击才能打开或关闭一个窗口.(现在一般也不能改变状态栏的值了)
    - 无法打开一个太小,或者太大的窗口. 无法创建一个没有标题栏或者状态行的浏览器窗口.
    - HTML的FileUpload元素的value属性无法设置.(这样可以防止脚本偷取用户的文件)
    - 脚本不能读取从不同的服务器上载入的文档的内容,除非这个文档就是包含该脚本的文档.类似,脚本不能在来自不同的服务器的文档上注册事件监听器.
18. 同源策略: 一个脚本只能够读取和包含这一脚本的文档来源相同的窗口和文档的属性
    (XMLHttpRequest也受同源策略的限制)
    脚本本身的来源和同源策略并不相关,相关的是脚本所嵌入的文档的来源,理解这一点很重要! eg:一个来自域A的脚本包含到一个域B的Web页中,这个脚本可以完整地访问包含它的文档的内容.如果脚本打开一个新的窗口并载入来自域B的另一个文档,脚本也对这个文档具有完全访问权. 但是,如果打开并载入一个来自域A(或域C)的文档,同源策略就会限制脚本访问这个文档.
    协议同源:使用Http协议载入的一个文档和另一个使用Https协议载入的文档具有不同的来源,即使它们来自于同一个服务器.
19. 如果两个窗口(或帧)含有的脚本把Document.domain属性设置成了相同的值,那么这两个窗口就不再受到同源策略的约束,这们可以互相读取对方的属性.
    domain值中至少要有一个点号,不能把它设置为"com"或其它顶级域名.
20. 跨站脚本攻击(XXS):

    假如一个页面会将某个URL参数显示出来,如果会把URL中name参数显示出来,那么可以输入".html?name=%3Cscript src=siteB/evil.js %3E%3C/script%3E.其中"%3Cscript src=siteB/evil.js %3E%3C/script%3E"中"<script src=siteB/evil.js></script>"的URL编码,这样SiteB中的脚本就可以访问利用其它站点来访问用户的数据或欺骗用户了.

    通常,防止XSS攻击的方式是在使用任何不可信的数据来动态创建文档内容前,从其中移除HTML标记.

    eg: name=name.replace(/</g,"<".replace(/>/g,">");

第十四章:脚本化浏览器窗口

1. setTimeout()返回一个不确定的值(这是一个整数值),这个值可以传递给clearTimeout()来取消规划的函数的执行.(setInterval()和clearInterval()类似)
2. 使用setTimeout()有一个技巧,就是注册一个函数在0微秒之后执行,这段代码会在浏览器完成当前的挂起事件后尽可能快的执行.有时候必须使用这种技术来延迟代码的执行.
3. Location对象的toString()方法返回href属性的值,可以使用location代替location.href. Location对象的search属性返回问号之后的那部分URL,这部分通常是某种类型的查询字符串
4. 当调用replace()时,指定的URL就会替换浏览器历史列表中的当前URL,而不是在历史列表中创建一个新条目.因此,如果使用方法replace()使一个新文档覆盖当前文档,Back按钮就不能返回最初文档.
5. window对象的location与Document对象的location对象不同,前者是一个对象,后者只是一个字符串.document.location=document.URL
6. 在大多数情况下,document.location和location.href相同,但是如果服务器存在重定向, document.location包含的是已经装载URL,而location.href包含的则是原始请求的文档的URL
7. window.screenX; window.screenY; (window.screenLeft; window.screenTop)
   //浏览器相对于屏幕左上角的位置
   window.innerWidth; window.innerHeight; (document.documentElement.clientWidth;)
   //浏览器显示正文的宽和高
   window.pageXOffset; window.pageYOffset; (document.documentElement.scrollLeft;)
   //滚动条的位置
8. Screen对象的availWidth,availHeight指定的是可实际使用的显示大小,它排除了像桌面任务栏的特性所占用的空间
9. open()方法: 第二个参数是新窗口的名字,可以在<a>或<from>的Target属性中使用.第四个参数指定是否应该替换掉窗口浏览历史的当前项,默认是false.
10. 即使一个窗口关闭了,代表它的window对象仍然存在,但除是检测它的closed属性之外,就不应该再使用它人任何其他属性及方法了.
11. confirm()与prompt()方法会产生阻塞,在大多数浏览器中,alert()方法也会产生阻塞.
12. Window对象的onerror属性比较特殊,如果给这个属性赋一个函数,那么只要这个窗口中发生了Js错误,该函数就会被调用,即它成了窗口的错误处理句柄.(不可以简单的定义一个名为onerror的函数来实现这个效果,因为IE不支持,所以应该给Window对象的onerror属性赋于函数)
13. 现在已经强烈反对使用帧(但是推荐使用iframes),对Js来说,<iframes>与<frameset>创建的帧是一样的.

第十五章:脚本化文档

1. Document属性:
    - bgColor: 文档的背景色
    - cookie: 允许Js程序读写HTTP cookie
    - domain: 读取设置文档的域
    - lastModified: 文档的最后修改日期
    - location: 造价于URL,已经废弃
    - referrer: 包含把浏览器带到当前文档的链接(如果存在)
    - title:<title></title>之间的文本
    - URL: 装载文档的URL
2. 每个from对象都有一个名为elements[]的集合属性,其中包含了代表表单中所包含表单元素的对象,在表单提交之前,Form对象触发一个onsubmit事件句柄,这个句柄可以执行客户端的表单验证,如果它返回false,浏览器将不会提交表单
3. 如果两个文档元素具有相同的name属性,那么docmuent.myName属性就变成了保存了两个元素的引用的一个数组.
4. Js的所有事件句柄属性都必须使用小写, eg:document.myfrom.onsubmit=validateform; // 此处只是赋值一个函数,不能包括(),()是调用的意思
5. 在DOM中,HTML表示为一棵树.
   根结点是Document,每个结点都可以引用子节点(children),父结点(parent),兄弟节点(sibling),后代(descendant)及祖先(ancestor).
6. Node对象的childNodes属性返回节点的孩子的一个列表,并且firstChild,lastChild,nextSibling,previousSibling和parentNode属性提供了遍历节点的树的方法.
   appendChild(),removeChild(),replaceChild()和insertBefore()方法能够向一个文档树中添加或者从一个文档树中移除子点.
7. 常见的节点类型:

接口	nodeType常量	nodeType值
Element	Node.ELEMENT_NODE	1
Text	Node.TEXT_NODE	3
Document	Node.DOCUMENT_NODE	9
Comment	Node.COMMENT_NODE	8

DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11
Attr	Node.ATTRIBUTE_NODE	2

29. DOM树根部的Node是一个Document对象,这个对象的documentElement属性引用了一个Element对象,它代表了文档的根元素.对HTML文档来说,就是<html>标记. 使用document.body引用<body>元素.
30. 在一个DOM树中只有一个Document对象,DOM树的大部分节点是表示标记的Element对象和表示文本串的Text对象.(有些解释器保留了Comment对象)
31. Element接口的getAttribute(),setAttribute(),removeAttribute()可以用来查询,设置,删除一个元素的属性.
    还可以使用getAttributeNode()方法来返回一个表示属性和它的值的Attr对象.它定义了specified属性用来指示文档中是否直接指定了该属性,或判断它的值是否为默认值.(Attr对象不出现在元素的childNodes[]数组中,不像Element和Text节点那样直接中文档树的一部分)
32. HTMLElement接口定义了id,style,title,lang,dir,className属性. DOM标准为HTML属性定义属性是为了方便编写脚本,查询和设置树属性值的一般方法是Element对象的getAttribute(),setAttribute(),当使用非HTML标准的部分的属性时,需要用到这些方法.
33. 使用HTML专有的DOM标准时,注意命名规则:一般是小写字母开关,后面的单词大写字母开关(eg:maxlength=>maxLength),当属性名与Js关键字冲突时,加上html前缀,特例是class属性,转换成className属性
34. NodeList元素并不是一个真正的数组,只是一个类似于数组的对象.(例如,它没有sort()方法)
35. 由于DOM标准定义了接口,而不是类,所以它没有定义任何构造函数方法.
    例如:创建一个新的Text对象的方法是document.createTextNode("haha")而不是var t=new TextNode("haha");
36. DocumentFragment是一种特殊类型的节点,它自身不出现在文档中,只作为连续节点集合的临时容器,并允许将这些节点作为一个对象来进行操作.
37. innerHTML属性:
    - 当查询一个HTML元素的这一属性值时,所得到的是表示该元素的孩子的一个HTML文本字符串.
    - 使用+=运算符来为innerHTML属性附加一些文本通常效率不高,因为需要序列化.
    - innerHTML已经被所有浏览器支持. 这是操作DOM的很好的快捷方式.
38. 选定的文本: window.getSelection().toString(); document.getSelection();
    在FF中,可以用e.value.substring(e.selectionStart,e.selectionEnd)得到文本框中选取的内容.

第十六章:层叠样式表和动态HTML

1. CSS概述:
   - body { } //设置<body>标记的样式 h1,h2 { } 同时设置两个
   - table b { } //指定<table>标记中的<b>标记的样式(注意,中间没有逗号)
   - .title { } //指定所有class为title的标记的样式
   - p.attention {} //指定在<p>标记有class="attention"时的样式
   - #p1 {} //id为"p1"的标记的样式
2. 在FF可以指定一个备用的样式表,来允许用户选择不同的风格,在查看->页面风格中可以选择
   <link rel="alternate stylesheet" href="largetype.css" type="text/css" title="Large Type" >
3. 用户样式表覆盖默认的浏览器样式表,作者样式表覆盖用户样式表,内联样式表覆盖所有样式表.
   在一个样式表中,如果一个元素上应用了多条样式规则,最详细的规则定义的样式将覆盖不太详细的规则定义的发生冲突的样式.
4. position属性:
   - static：指定根据文档内容的正常流动元素定位.这是默认的设置
   - absolute: 绝对定位的元素可以相对于文档的<body>标记进行定位.如果它嵌套在另一个绝对定位的元素中,则相对于那个元素定位
   - fixed: 具有fixed定位的元素总是可见的,并且不随文档其余的元素滚动.(可惜IE不支持)
   - relative: 将根据常规布置元素,然后相对于它在常规流中的位置进行调整.在常规文档流中分配给元素的空间仍然分配给它,它两边的元素不会向它靠近来填充那个空间,但它们也不会从元素的新位置被挤走.
5. 查询元素的位置和大小: 一个元素的offsetLeft,offsetTop属性返回这个元素的X和Y坐标.类似地,offsetWidth,offsetHeight属性返回元素的宽度和高度.(这些属性指定了一个元素相对于其它元素的X坐标和Y坐标,而这个元素就是offsetParent属性)
   对于被定位的元素来说,offsetParent通常是<body>标记或者<html>标记,或者是被定位的元素的一个定位的祖先,可以使用一个循环来得到一个元素的相对于窗口的定位

```
function getX(e){
    while(e){
        // ...
        e=e.offsetParent;
    }
}
```

6. z-index指定元素的第三维.这们的绘制顺序是从最低的z-index到最高的z-index. 这只适用于兄弟元素,对于不是兄弟元素的重叠,并不能指定那个元素在上面.

7. visibility和display样式属性的区别在于它们对非动态定位的元素的影响. display:none的元素不会为它们在文档中分配位置,而visibility:hidden的元素虽然不可见,但还是为它们分配了元素位置的.
8. margin和padding属性都指定了一个元素周围的空白区域.不同之处在于,margini指定了边框外部的空间,即边框和相邻元素之间的部分;而padding属性指定了边框内部的空间,即边框和元素内容之间的部分.
9. 如果没有为一个元素指定一个背景颜色或者背景图片,那么这个元素的背景通常是透明的.
10. 设置一个元素半透明的可兼容的方法为: style="opacity:.5; filter:alpha(opacity=50);"
11. clip属性的值指定了元素的剪切区域,eg:style="clip: rect(0px 100px 100px 0px);" 括号中的四个值都是长度值,必须有单位,而不允许用百分比.(负数表示指定延伸到为元素指定的边框之外的剪切区域)
12. 用元素的style属性获取的CSS只能设置元素的内联样式,不能用CSS2Properties对象的属性获取应用到元素的样式表的样式信息.设置的也是内联属性(具有最高的优先级)
13. 如果一个CSS属性名含有一个或多个连字符,那么CSS2Properties属性名删除了连字符,且原来紧接在连字符后的字母改成大写开头(eg:font-family=>fontFamily)
14. 所有的定位属性都要求有单位.而且必需设置成字符串. eg:e.style.left="300"是错误的,应该为e.style.left="300px"
15. CSS属性返回的是字符串而不是数字,所以如果想让两个高度相加时:
    e1.style.height+e2.style.height会造成字符串连接,应该都加上parseInt()
    parseInt(e1.style.height)+parseInt(e2.style.height)
16. getComputedStyle()的返回值是一个CSS2Properties对象,它代表了应用于指定的元素或伪元素的所有样式.(getComputedStyle()返回的对象是只读的)
    IE提供了一个替代的方法: 每个HTML元素都有一个保存其计算样式的currentStyle属性.
17. 脚本化样式表本身:
    - <link>和<script>元素都定义了一个disabled属性来启用或者关闭相应的设置.可以在Js中动态的打开或者关闭
    - 文档的样式表存储在文档对象的styleSheets[]数组中,这个数组的元素是CSSStyleSheet对象,可以这样引用:
      document.styleSheets[0].cssRules[0](IE中是document.styleSheets[0].rules[])
    - insetRule(),deleteRule() (IE中是addRule(),removeRule()) 可以动态的添加或删除样式表的规则

第十七章:事件和事件处理

1. <img>有两个比较特殊的事件 onabort:图像装载被中断时引发; onerror:图像加载过程中发生了

错误引发

2. 事件句柄中的Js代码运行在不同于全局Js代码的作用域中. 这点很重要!
3. 显式地调用事件句柄并不是完全模拟事件发生时的真正状况.例如,如果调用一个Link对象的onclick方法,它并不能使浏览器根据那个链接把新文档装载进来,而只能执行定义为那个属性值的函数.
4. 通常,如果浏览器执行某种默认动作来响应一个事件,那么可以返回false阻止浏览器执行那个动作
   事件句柄从来不要求显式地返回值,如果不返回值,就会发生默认的动作.
5. 在把一个HTML属性的值设置为Js代码串,以便定义事件句柄时,隐式地定义了一个函数.以这种方式定义的事件处理函数的作用域与用常规方法定义的全局Js函数不同.
   事件句柄的作用域链的第一个是调用对象,这点与其它函数相同.但是,它的下一个对象却并非全局对象,而是触发事件句柄的对象.
   onclick="alert(b4.value)" 等价于:

```
b3.onclick=function(){
 with(document){
  with(this.form){
   with(this){
      alert(b4.value);
   }}}}        //作用域链是从Document延伸到引发事件的元素的
```

   解决这个问题的理想方法就是让它们只调用在别的地方定义的全局函数,并返回结果.(因为函数在定义它们的作用域内执行,所以不会产生干扰)
6. 2级DOM中的高级事件处理 (IE不支持)
   - 在事件的传播分发阶段,目标的每个祖先元素都有机会处理事件.
   - 虽然所有的事件都受事件传播的捕捉阶段的支配,但并非所有的类型的事件都起泡的. 一般来说,原始输入事件起泡,而高级语义事件不起泡.
   - 默认的动作只在事件传播的三个阶段都完成之后才会执行.
   - stopPropagation()停止事件的传播,preventDefault()阻止默认动作的发生.
   - 2级DOM中可以为一个事件注册任意多个事件.使用
     addEventListener(onmousedown,functionForThisEvent,false)
     第三个参数是一个布尔值,如果值为true,则指定的事件句柄将在事件传播的捕捉阶段用于捕捉事件,如果为false,事件句柄就是常规的,当事件直接发生在对象上,或发生在元素的子女上,又向上起泡到该元素时引发.
   - 使用addEventListener注册的多个事件将在事件发生时执行,但是并不保证它们的执行顺序,不要认为它们是按顺序执行的.
   - removeEventListener可以删除事件.
   - 当句柄被调用时,this关键字所引用的对象正是在其上注册了这个句柄的对象.

- Event接口:
  - type:事件的类型
  - target 发生事件的节点
  - currentTarget 发生当前正在处理事件的节点(在事件处理函数中,应该用这个属性而不是this关键字)
  - eventPhase 所处事件传播过程的阶段
  - timeStamp 事件何时发生
  - bubbles 该事件是否在文档树中起泡
  - cancelable 事件能否用preventDefault()取消
- UIEvent接口
  - view:发生事件的windw对象
  - detail: 一个数字,提供额外的信息
- MouseEvent接口
  - button: 一个数字,表示那个鼠标键改变了状态 0:左键; 1:中键; 2:右键;
  - altKey,ctrlKey,metaKey,shiftKey: 指示鼠标事件发生时,这些键是否被按下
  - clientX,clentY: 鼠标指针相对于客户区浏览器窗口的坐标(不考虑滚动)
  - screenX,screenY: 鼠标指针相对于显示器的坐标
  - relatedTarget: 引用与事件的目标节点相关的节点.

7. IE事件模型
   - IE Event对象最重要的属性如下
     - type: 一个字符串,声明发生的事件的类型
     - srcElement: 发生事件的文档元素(DOM: Target)
     - button: 一个整数,声明被按下的鼠标键. 1:左;2:右;4:中. 如果同时按下多个键,就把这些值加在一起.
     - clientX,clientY: 两个整数,声明事件发生时鼠标的坐标,其值是相对于包含窗口的左上角生成的
     - offsetX,offsetY: 两个整数,声明鼠标相对于源元素的位置(eg:确定点击了image的那个像素)
     - altKey,ctrlKey,shiftKey: 布尔值,鼠标事件时这些键是否被按下
     - keyCode: 键的代码. String.fromCharCode()可把字符代码转换成字符串.
     - fromElement,toElement: fromElement是mouseover中鼠标移动过的文档元素, toElement是鼠标移动到的文档元素
     - cancelBubble: 布尔值,设为True可阻止当前事件进一步起泡.

        - returnValue: 布尔值,设为True可阻止浏览器执行默认动作
    - IE不把Event对象作为参数传递给事件句柄,而通过全局的Window对象的event属性访问Event对象.
    - attachEvent(),detachEvent()与addEventListener(),removeEventListener()类似,除了以下几点:
        - 由于IE事件模型不支持事件捕捉,所以attachEvent,detachEvent只有两个参数,事件类型和句柄函数
        - 传递给IE方法的事件句柄应该包括一个"on"前缀,标准DOM不能有这个前缀.
        - attachEvent()注册的函数将被当作全局函数调用,而不是作为发生事件的文档元素的方法.(关键字this引用的是window对象,而不是事件的目标元素)
        - attachEvent()允许同一事件句柄注册同一个函数多次,而且注册多少次就会调用多少次.
    - setCapture(),releaseCapture()是所有HTML元素的方法,当在一个元素上调用setCapture()的时候,所有后续的鼠标事件都被引导到这个元素,并且这个元素的句柄可以在这些事件起泡之前处理它们. (注意:这只对鼠标事件有效,并包括所有的鼠标事件)
      调用setCapture()时,鼠标事件专门分派,直到调用releaseCapture()或者捕获中断,当Web浏览器失去焦点,鼠标事件可能中断,这时setCapture()调用所基于的元素会接收到一个onlosecaputre事件,通知它们不再会接收到鼠标事件.
8. 当一个鼠标事件发生时,事件对象的clienX和clienY属性保存了鼠标指标的位置,这个位置在窗口坐标中,它相对于浏览器的"视口"的左上角,并且没有考虑到文档的滚动
9. 按键事件: 有3种按键事件onkeydown,onkeypress,onkeyup. 其中keypress事件最为友好的,和它相关的事件对象包含了所产生的实际字符的编码.keydown,keyup是较底层的,它们的按键事件包含一个和键盘所生成的硬件相关的"虚拟按键码".
    - 在IE中,只有当按键有一个ASCII码时,也就是是一个可打印字符或一个控制字符时,keypress事件才会发生.
    - keydown对功能按键来说最有用,而keypress对可打印按键来说最有用.
    - 按键事件的细节:
        - 如果Alt,Ctrl,Shift和一个按键一起按下,则可以通过事件对象的altKey,ctrlKey,shiftKey属性来表示,这在IE和FF是兼容的
        - FF中定义了两个属性,keyCode存储了一个按键地较低层次的虚拟按键码,并且和keydown事件一起发送. charCode存储了按下一个键时所产生的可打印的字符的编码,并且和keypress事件一起发送.
          IE中只有一个keyCode属性,并且它的解释取决于事件的类型,对于keydown来说,keyCode是一个虚拟按键码,对于keypress来说,keyCode是一个字符码.

- 字符码可以使用静态函数String.fromCharCode()转换为字符.
- 注意,如果用户使用鼠标向字段粘贴文本的话,onkeyup事件句柄不会被触发,为处理这种情况,可以注册一个onchange句柄

10. 在0级事件模型中,不需要合成事件,因为事件句柄通过各种事件句柄属性可以获得.然而在高级事件模型中,没有办法查询使用addEventListener或attachEvent注册的句柄的集合.

第十八章:表单和表单元素

1. 只有真正点击提交按钮才会触发onsubmit事件,调用表单的submit()方法则不会触发它.
2. onchange事件并不是用户每次在文本框中敲击一个键都会触发,它只在用户改变了一个元素的值,并把输入焦点移到了其它表单元素时才会触发.
3. <button>标记比<intpu type="button" />更灵活一些,因为它不是显示由value属性指定的纯文本,而是显示出现在<button></button>之间的HTML内容

第十九章:cookie和客户端持久性

1. Document的cookie属性是一个字符串属性,可以用它对当前网页的cookie进行读操作,创建操作,修改操作和删除操作.
2. 当对cookie进行读操作时,可以得到一个字符串,这个字符串包含了应用到当前文档的所有cookie的名字和值.
3. 除了名字和值外,每个cookie都有四个可选的属性,分别控制它的生存期,可见性和安全性
   - expires：指定了cookie的生存期. 这个属性已经被max-age属性所取代了,max-age用秒来设置cookie的生命期.
   - path：默认情况下,cookie会和创建它的网页以及与这个网页处于同一个目录下的网页和处于该目录的子目录下的网页关联. 可以通过设置path属性来改变cookie有有效路径.(eg:设置cookie的path为"/"就能够能服务器上的所有网页可见.
   - domain：与path类似,但是是用于设置cookie的域. 注意,不能将一个cookie的域设置为服务器所在的域之外的域.
   - secure：指定了在网络上如何传输cookie的值.
4. cookie的操作:
   - cookie的存储: 只需要将cookie属性设置为以下形式的值即可:
     docmuent.cookie="version="+encodeURICompoent(document.lastModified);
     cookie的值不能有分号,逗号,或空白符.因此就使用encodeURICompoent进行编码,解码使用decodeURICompoent.(escape,unescape已经废弃)

```
document.cookie="version="+docment.lastModified+                    //
name/value键值对
                    "; max-age="+(60*60*24)+                    //
max-age以秒为单位
                    "; expires="+nextYear.toGMTString()+                    // 与
max-age二选一,需要设置为时间的toGMTString()方式
                    "; path='/'; domain='www.123.com'; secure='LLS'"
```

- 要改变一个cookie的值,使用同一个name,path和domain以及新的值再设置一次cookie的值即可.
- 要删除一个cookie,再次使用相同的name,path和domain并将max-age属性设置为0,或者expries设置为已经过了的一个时间即可.

5. cookie主要用于少量数据的不经常存储,一般来说一个站点的cookie要小于20个,每个cookie的数据量要小于4KB
6. 在Js中使用document.cookie返回的是一个字符串,这个字符串存放的是当前文档应用的所有cookie.
   在对cookie属性进行读操作时,得到的字符串不包含任何有关各种cookie属性的信息.虽然cookie属性允许设置这些值,但是却不允许读取它们的值.

第二十章:脚本化HTTP

1. 使用<img><iframe><script>可以实现一些脚本化HTTP. 但是实现起来比较麻烦
2. 创建XMLHttpRequest对象
   - 在大多数浏览器中可以通过一个简单的构造函数来实现 var request = new XMLHttpRequest();
   - 在IE7之前,可以构造一个ActiveXObject对象来实现 var request = new ActiveXObject("Msxml2.XMLHTTP");
3. 大多数HTTP请求都是用GET方法完成的,该方法只是下载该URL的内容,另一个有用的方法是POST,这是大多数HTML表单所使用的方法:它允许指定的变量的值作为请求的一部分.HEAD是另一个有用的HTTP方法:它要求服务器只是返回和该URL关联的头部.
4. 在创建了XMLHttpRequest对象之后就可以调用open()方法来打开一个异步的XMLHttpRequest.
   request.open("GET",url,true).第3个参数指示使用同步还是异步的方法来响应,通常都是采用异步的方法.
   open()并不实际地向Web服务器发送请求,它只保存自己的参数,等到稍后实际发送请求的时候再使用.
5. 在发送请求之前,必须设置所有所需的请求头部.

request.setRequestHeader("Accept-Language","en");

6. 把请求发送给服务器 request.send(null); send()函数的参数是请求体,对于HTTP GET请求,参数总是null,然而对于POST请求,它包含要发送给服务器的表单数据. (注意:这个null是必须的,并不允许省略)
7. send()并不返回一个状态代码,一旦它返回,可以使用请求对象的status属性来检查服务器所返回的HTTP状态代码,这个代码的可能值是在HTTP协议中定义了的.状态200意味着请求成功,并且这个响应可以获得.
8. 异步响应: 对XMLHttpRequest来说, 异步响应在onreadystatechange句柄中控制.任何时候,只要readyState变了,事件句柄就被调用.readyState是一个HTTP请求的状态的整数值. 0:open()还未调用; 1:open()已经调用,但send()还没调用; 2:send()已经调用,服务器还未响应; 3:正在接收数据; 4:响应完成;
   由于兼容性的原因,应该忽略掉除状态4以外的其它所有状态值.
9. onreadystatechang事件句柄中,必须检查readyState以确定事件为什么被调用.
10. onreadystatechang事件句柄中,XMLHttpRequest对象并不会被传递,所以要确保onreadystatechang函数的作用域能访问XMLHttpRequest对象.
11. 使一个请求过期的方法,一个异步的请求应该有一个时间的限制,可以在发送请求时调用一个setTimtOut()来调用XMLHttpRequest.abort().然后当响应到达时就调用clearTimtOut()清除这个调用 .

第二十一章:JavaScript和XML

第二十二章:脚本化客户端图形

1. 为了让一幅图像能够缓存,首先要用Image()构造函数创建一个Image对象,把这个对象的src属性设置为想要的URL,从而把一幅图像载入到对象中,浏览器就会缓存它
2. 每个Image对象有一个complete属性,当图像正在载入时,这个属性为false,只有当图像已经载入或者一旦浏览器停止尝试载入它的时候,这个属性才变为true.
3. Js设置一个元素的style属性时:
   - 可以设置style中的一个属性,如a.style.width="100px";
   - 而不可以设置整个style属性, 如a.style="width:100px; height:100px;" // 这是错误的
   - 但是可以使用setAttribute来设置整个style属性 如 a.setAttribute("style","width:100px; height:100px;"); //这是允许的