Part1_Ch01-05_JS 基础

Ch01_了解 JavaScript

1.1Web 脚本编程基础

1.2 服务器端与客户端编程

客户端脚本：脚本和页面内容一同发送给用户（非服务器运行脚本），本地操作

1.3JavaScript 简介

1.4JavaScript 起源

1.5<script>标签

JS 代码不仅仅能出现在文档的 body 部分，也能出现在其他位置

1.6DOM 简介

1.6.2window 和 document 对象

顶端：浏览器 window 对象

window 的子对象 document，包含全部 HTML 内容及其他构成页面显示的资源

1.6.3 对象表示法

我们用句点方式表示树形结构里的对象：

parent.child

document 对象是 window 对象的子对象，所以在 DOM 里表示为：

window.document

HTML 页面的 body 部分在 DOM 里是 document 对象的一个子对象，表示为：

window.document.body

这种表示法的最后一个部分除了可以是对象外，还可以是属性或方法：

object1.object2.property

object1.object2.method

举例来说，若想访问当前文档的 title 属性，即 HTML 标签<title>和</title>：

window.document.title

注意：window 对象永远包含当前浏览器窗口，可以简化表示，但是，如果打开了多个窗口，或者使用框架集，那么每个窗口或框架斗殴单独的 window 和 document 对象，为了访问其中的某一个文档，需要使用相应的窗口名称和文档名称。

1.7 与用户交互

1.7.1 window.alert()

在用户点击“确定”按钮前，页面上不能进行其他任何操作。具有这种行为模式的对话框被称为“模态”对话框。

Ch02_创建简单的脚本

2.1 在 Web 页面里添加 JavaScript

有两种方法可把 JavaScript 代码集成到 HTML 页面：

[1] 用<script></script>标签直接把 JavaScript 语句包含在 HTML 文件里；
[2] 把 JavaScript 代码保存到单独的文件，然后利用<script>标签的 src（源）属性把这个文件包含到页面里。
    1) <script src='mycode.js'></script>
    2) 如果 JavaScript 文件与调用脚本不在同一个文件夹，就需要使用相对或绝对路径：
        a)<script src='/path/to/mycode.js'></script>
        b) 或
        c)<script scr='http://ww.example.com/path/to/mycose.js'></script>
    3) 按照惯例，JavaScript 代码文件的后缀是.js。但实际可以使用任何后缀，浏览器都会把其中的内容当作 JS 来解释。

把 JavaScript 代码保存到单独的文件有不少好处：

[1] 当 JavaScript 代码有更新时，这些更新可以立即作用于使用这个 JavaScript 文件的页面。这对于 JavaScript 库是尤为重要的（后面会介绍）；
[2] HTML 页面的代码可以保持简洁，从而提高易读性和可维护性；
[3] 可以稍微改善一点性能。浏览器会把包含文件进行缓存，当页面或其他页面再次需要使用这个文件时，就可以直接从内存读取了。

注意：外部文件里不能使用<script>和</script>标签，也不能使用任何 HTML 标签，只能是纯粹的 JavaScript 代码。

JavaScript 可以放置到 HTML 页面的<head>或<body>区域里，但一般情况下，我们把 JavaScript 代码放到<head>区域，从而让文档的其他部分能够调用其中的函数。

当 JavaScript 代码位于文档的 body 区域时，在页面被呈现，遇到这些代码就会解释和执行，然后继续完成页面的其他内容。

说明：有时在<script>标签里可以看到 HTML 风格的注释标签<!--和-->，比如：

```
<script>
    <!--
    …JS 语句…
    -->
</script>
```

这是为了兼容不能识别<script>标签的老版本浏览器。这种“注释”方式可以防止老版本浏览器把 JavaScript 源代码当作页面内容显示出来。除非我们有特别明确的需求要支持老版本的浏览器，否则是不需要使用这种技术的。

注意：如果利用<script>元素的 src 属性包含了外部的 JS 文件，就不能在<script>和</script>之间包含 JS 语句了，而是必须为空。

2.2JavaScript 语句

可以单行分割或使用分号（可同行）

“注释”：//或/**/，或同 HTML

2.3 变量

JS 的变量名区分大小写，一般使用驼峰法命名

JS 的变量可以不在赋值之前声明，但最好声明（var 变量名），声明时可以同时赋值

字符串可以使用双引号或单引号

2.4 操作符

2.4.1 算术操作符

基本同 C，注意 JS 的除法应该直接就是浮点数的

2.4.3 对字符串使用操作符“+”

[1] 两个字符串连接；

[2] 字符串变量+数值变量：数值变量将转换为字符串，之后连接

2.5 捕获鼠标事件

2.5.1onClick 事件处理器

使用方式之一是给 HTML 元素添加一个属性。

HTML 不区分大小写，而 XHTML 区分大小写，且要求全部的 HTML 元素及属性名称都使用小写字母。

2.5.2onMouseOver 和 onMouseOut 事件处理器

某一元素占据的区域内生效。

关键词 this 指 HTML 元素本身。

图像变化的经典方式：

```
<img src='image1.jpg' alt='image 1' onmouseover='this.src="image2.jpg";'
onmouseout="this.src='image1.jpg';"/>
```

现在已经被更高效的“层叠样式表”(CSS)取代。

Ch03_使用函数

3.1 基本语法

关键词 function 函数名(){}

可以在一个<script>元素里创建多个函数

JS 的函数名称区分大小写，以字母或下划线开头，另可包含数字

3.2 调用函数

页面加载时，包含在函数定义区域内的代码不会被执行，而是在被“调用”时执行。

如 document.write()或 window.alert()等“方法”实际上是属于特定类的函数。

JS 支持函数的嵌套调用

3.3 参数

多个参数，使用逗号分隔，参数名（无类型）

函数中变量的作用域基本类似 C。

注意：如果函数定义里的某个参数没有接收到值，JavaScript 可能会报告错误，或是函数执行结果不正确。如果调用函数时传递了过多的参数，JS 会忽略多出来的参数。

3.4 从函数返回值

return 语句

可返回任何 JS 支持的数据类型

3.5 变量作用域

JavaScript 返回的错误提示取决于浏览器的设置

基本同 C 中的变量作用域

函数内可以直接使用外部定义的全局变量

Ch04_DOM 对象和内置对象

4.1 与用户交互

4.1.1alert()

模态对话框 alert()，以字符串为参数。

“模态”意味着脚本暂时停止运行，页面与用户的交互也被暂停，直到用户关闭对话框为止。

4.1.2confirm()

confirm()也弹出一个模态对话框，但有两个按钮可以返回不同的布尔值，返回值可以赋值给变量。

4.1.3prompt()

prompt()打开的模态对话框允许用户输入信息，另 prompt()方法还有第二个可选参数，表示默认的输入内容。

prompt()对话框的返回值：

[1] 输入信息，点击“确定”或回车，返回用户输入的字符串；
[2] 用户没有输入信息，直接“确定”或回车，返回第二个可选参数的值（如果有的话，否则可能是“undefined”）；
[3] 若简单关闭对话框（“取消”或 Esc），返回“null”（测试的结果是直接是此字符串）。

说明：JS 在多种情况下使用 null 表示空值。作为数值使用时，它代表 0；作为字符串使用时，它代表空字符串“”；作为布尔值时，它代表“假”。

4.2 根据 id 选择元素

除了此方法外，还可使用 document 对象的多种方法遍历 DOM 对象（见 Part3）。

[1] getElementById()，得到页面特定的元素，能够访问它的全部属性和方法。
    a. 注意：为了让范例代码得到期望的结果，这个页面元素一定要设置 id 属性。HTML 页面元素的 id 属性要求是唯一的，所以这个方法能够返回与 id 匹配的唯一元素。
[2] innerHTML 属性
    a. 与 getElementById()组合，可以访问该元素的 HTML 内容，并可以重新设定选定元素的内容（覆盖原有内容）

4.3 访问浏览器历史纪录

window.history：使我们可以使用访问过的 URL 列表，但是不能直接地修改这些 URL。

对象 window.history 只用一个属性 length，表示用户访问过的页面的数量。

history 对象有三个方法：

[1] forward()：相当于点击浏览器的“前进”，得到历史列表里的下一个页面；
[2] backward()：相当于点击浏览器的“后退”，得到历史列表里的前一个页面；
    a. history.next();
[3] go()：有一个正的或负的整数作为参数，可以跳到历史记录列表里的相对位置，负数回退，正数前进。另外，这个方法也可以接收字符串作为参数，找到历史记录列表里第一个匹配的 URL

4.4 使用 location 对象

location 对象包含当前加载页面的 URL 信息。

页面的 URL：[协议]//[主机名]:[端口]/[路径][搜索][hash]

location 对象的属性

属性	内容
location.href	完整 URL

location.protocol	协议，’http:’
location.host	[主机名]:[端口]
location.hostname	[主机名]
location.port	[端口]
location.pathname	‘/tools/display.php’
location.search	‘?section=435’
location.hash	‘#list’

4.4.1 使用 location 对象导航

location 对象有两种方式可以帮助用户导航至新页面:

[1] 直接设置对象的 href 属性：location.href=”www.newpage.com”;
   a. 使用此方法把用户转移到新页面时，原始页面还保留在浏览器的历史记录里，用户可以利用浏览器的“后退”按钮方便地返回到以前的页面。
[2] location.replace()方法，把当前页面从历史记录列表里删除（新 URL 替换它）

4.4.2 刷新页面

location.reload()

提示：如果使用没有参数的 reload()方法，当浏览器的缓存里保存了当前页面时，就会加载缓存的内容。为了避免这种情况，确保从服务器获得页面数据，可以在调用 reload()方法时添加参数 true：document.reload(true);

4.5 浏览器信息：navigator 对象

navigator 对象包含了浏览器程序本身的数据。

当我们需要了解用户浏览器的功能时，使用 navigator 不是好选择：不是所有浏览器都支持 navigator 全部的属性，且未必正确。而“功能检测”是一种更精确的跨浏览器手段来检测用户浏览器的功能，从而决定如何进行相应的操作。

4.6 日期和时间

Date 对象用于处理日期和时间，但 DOM 里没有现成的 Date 对象，而是要我们在需要时创建自己的 Date 对象，每个 Date 对象都代表不同的日期和时间。

4.6.1 创建具有当前日期和时间的 Date 对象

var myDate=new Date();

myDate 为一个 Date 对象，具有某些方法 getXXX()

4.6.2 创建具有指定日期和时间的 Date 对象

new Date(参数)：
milliseconds/dateString/(year,mouth,day,hours,minutes,seconds,milliseconds)

[1] 时间戳
[2] 日期字符串
[3] 在使用分散的各部分参数时，位置靠后的参数是可选的，不明确指定的参数值是 0；

4.6.3 设置和编辑日期与时间

设置：setDate()

```
myDate.setDate(myDate.getDate()+33);
```

格式转换：toDateString()，toTimeString()

4.7 利用 Math 对象简化运算

与 Date 对象不同，Math 对象不需创建就可使用，只需调用它的方法。

Math 的一些常用方法

方法	描述
ceil(n)	返回 n 向上取整到最近的整数
floor(n)	返回 n 向下取证到最近的整数
max(a,b,c,…)	返回最大值
min(a,b,c,…)	返回最小值
round(n)	返回 n 四舍五入到最近的整数
random()	返回一个 0 到 1 之间的随机数

注意：这些方法属于 Math，而不属于对象，对象放在括号里

4.7.3 随机数

```
function myRand(range){
    return Math.round(Math.random()*range)
}
```

4.7.4 数学常数

Math 的属性：E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2

4.7.5 关键字 with

任何对象都可以使用关键字 with，但 Math 对象是最适合用来示范的。with 可以减少键盘输入

关键词 with 以对象作为参数，然后是一对花括号，其中包含着代码块。代码块里的语句在调用特定对象的方法时可以不明确指定这个对象，因为 JavaScript 会假定这些方法是属于作为参数的那个对象。

范例：

```
with(Math){
    var myRand=random();
    var biggest=max(3,4,5);
    var height=round(76.35);
}//调用这些方法的代码块与 Math 对象实现了关联。
```

JS 对月份的计数是从 0 开始，到 11 结束。

Date 函数具有处理时区的方法，支持 UTC，getUTCDate()，setUTCMonth()。利用 getTimeZoneOffset()方法可以获得本地时间与 UTC 时间的时差。

Date()的方法 getFullYear()和 setFullYear()指 4 位数的年份，而无 Full 则为 2 位，已因千禧年跨越停用。

Ch05_数据类型

JS 属于“宽松类型”的编程语言，意味着 JS 变量在不同的场合可以被解释为不同的类型。

JS 中不必事先声明变量的数据类型就可使用变量，这时 JS 解释程序会根据情况做出它认为正确的判断。如果我们现在变量里保存了一个字符串，稍后又想把它当作数值使用，这在 JS 里完全可行，前提是字符串里的确包含“像”数值的内容。

5.1.2 浮点数

指数表示方法中，e 表示“10 的幂”，如 35.4e5

JS 可以处理十六进制数值，前缀 0x

5.1.3 非数值（NaN）

当脚本尝试把一些非数值数据当作数值处理，却无法得到数值时，其返回值就是 NaN。

比如，若尝试用一个整数乘一个字符串，得到的结果就是非数值。

利用 isNaN()函数可检测非数值，返回 True/False

5.1.4 使用 parseFloat()和 parseInt()

此二为可以把字符串强制转换为数值格式的函数。

[1] parseFloat()返回浮点数或 NaN
   a. 如果被解析的字符串的首字符是数字，函数会一直解析到数值结束
   b. 应该是只接受一个小数点

[2] parseInt()函数返回整数或 NaN
   a. 另有第二个可选参数，用于指定数值的基，从而返回二进制、八进制或其他进制数值所对应的十进制数值。
   b. ？二进制小数怎么办？没办法，遇到小数点就停了。
   c. 此函数应该是去尾（firebug）
   d. 开头要是数字，否则 NaN

5.1.5 无穷大

正负 2 的 53 次，Infinity，-Infinity

isFinity()用于判断一个数值是否是无穷大，NaN 也返回 false

5.2 字符串

5.2.1 转义序列

常用转义序列

转义序列	代表的字符
\t	制表符
\n	换行
\”	双引号
\’	单引号
\\	反斜线
\x99	ASCII 字符的值，以 2 位十六进制数值表示
\u9999	统一编码字符的值，以 4 位十六进制数值表示

5.2.2 字符串方法

string 对象的一些常用方法

方法	描述
concat	连接字符串，返回结果字符串的一个拷贝
indexOf	返回指定值在字符串里出现的第一个位置；若没有找到，返回-1
lastIndexOf	返回指定值在字符串里出现的最后一个位置
replace	在一个字符串里搜索制定的子字符串，并用新的子串进行替换//似乎直接只能替换第一个-->使用正则表达式或用/g 全局 return unescape(crumb.substring(nameEquals.length,crumb.length)).replace(/#/g," ");
split	把字符串分解为一系列子串，保存到数组里；返回一个新数组
substr	从制定的开始位置，提取指定数量的字符组成字符串； 有一个或两个参数，从第一个参数制定的索引位置开始提取字符，返回一个新字符串。第二个参数制订了要提取的字符数量，是可选的；如果没有制定，提取到字符串结束为止。
toLowerCase/ toUpperCase	把字符串转换为小写/大写字符

字符串级联：操作符“+”

5.3 布尔值

说明：JS 的关键字 null（空）和 undefined（未定义）

[1] 当我们想让变量具有有效值，却又不是任何具体值时，就把 null 赋给变量。对数值来说，null 相当于 0；对字符串来说，null 相当于空字符串””；对布尔变量来说，null 表示 false。

[2] undefined 不是关键字，而是预定义的全局变量。当某个变量已经在语句里使用了，但却没有被赋予任何值时，它的值不是 0 或 null，而是 undefined，表示 JS 不能识别它。

5.4 数组

5.4.1 创建新数组

var myArray=new Array();或 var myArray=[];

5.4.2 初始化数组

在创建数组时，可以同时加载数据：

var myArray=['hello','hi'];

或在数组创建之后，添加元素数据：

var myArray=[];

myArray[0]='hello';

注意：可以跳过前面的下标，比如直接指定下标 4 对应的值(firebug)，前面的值会保持 undefined。

array.length：数组有一个 length 属性，表示数组包含了多少项，自动更新，但实际为最大的索引加 1，即 undefined 也被计入了。

5.4.3 数组的方法

数组的常见方法

方法	描述
concat	返回值为合并的多个数组，但应该不改变原数组
join	把多个数组元素合并为一个字符串。可选参数为连接字符/分隔符，类似于 Python
toString	以字符串形式返回数组（逗号分隔）
indexOf	在数组中搜索指定元素
lastIndexOf	返回与搜索规则匹配的最后一个元素
slice	根据指定的索引和长度返回一个新数组。参数一为开始的索引值，参数二为要提取的元素数量。
sort	根据字母顺序或提供的函数对数组进行排序
splice 注意：splice()方法会改变原数组。如果代码的其他部分仍要使用该数组，应在操作前拷贝到新变量里。	在数组指定索引添加或删除一个或多个元素。 array.splice(index,howmany,[new elements]) 参数一指定在数组的什么位置进行操作，参数二说明要删除多少个元素（若

	为 0 则不删除元素），参数三可选，是要插入的新元素列表。 返回值为被删除的元素。

注意：数组与字符串的一些方法具有相同的名称，甚至是几乎类似的功能，使用时请注意这些方法所处理的数据类型，否则可能得不到预想的结果。

JavaScript 规范没有规定字符串的最大长度，这应该是由浏览器和操作系统决定的。对于特定的运行环境来说，它应该是由可用内存决定的一个数值。

JavaScript 不直接支持联合数组，但可以利用对象来实现这种行为，后面会介绍。

Part2_Ch06-10_JS 进阶

Ch06_功能更强大的脚本

6.1 条件语句

6.1.1if()语句

if(条件为真) 执行操作;

条件可为布尔变量或表达式

6.1.2 比较操作符

特别的“===”（严格相等），表示值和类型都要相等。比如字符串“2”会被解释为 2，如果是双等号，就会返回 true

其他见附录 B。

优先级不明

6.1.3 测试相等

6.1.4if 进阶

[1] 多行语句，同 C 语言：

```
if(){
}else{
}
```

[2] 问号冒号表达式，同 C 语言：

```
(条件为真)?[条件为真执行的语句]:[条件为假执行的语句];
```

6.1.5 测试多个条件（同 C 语言）

```
if(){
}else if{
}else{
}
```

6.1.6switch 语句

同 C 语言

6.1.7 逻辑操作符

应当也类似于 C 语言

6.2 循环和控制结构

6.2.1while

同 C 语言

6.2.2do…while

同 C 语言

```
do{

}while(this condition is true);
```

6.2.3for

同 C 语言（C99）

同 C++或 C11 标准，可以在 for 的小括号里用关键词 var 声明计数变量。

6.2.4 使用 break 跳出循环，使用 continue

break 语句中断循环，把程序导向右花括号后面的第一条语句。

使用 continue 命令可以停止当前循环，直接进入下一次循环。不是停止循环并转到循环体后面的语句，而是只中断当前循环，然后进入下一次循环。

6.2.5 利用 for…in 在对象集里循环

类似于 Python，循环会对集合中每个对象执行一次，然后结束。

```
for (i in set){

}
```

说明：在 JS 里，数组是一种对象。利用 for…in 循环可以操作任何对象的属性，无论是 DOM 对象、JS 内置对象，还是我们创建的对象。

6.3 调试代码

在 Firefox 中可以用 Ctrl+Shift+J 组合键打开 Firefox 的错误控制台 Error Console

6.3.1 循环标题/标题变换

[1] 使用事件处理器：window 对象的 onLoad 方法。

   a. <body onload=”somefunction()”>
   b. 此事件处理器一旦页面加载完成就运行我们的代码。

[2] JS 的 setInterval()函数。它能够重复运行 JS 函数，每次运行有一定间隔。

   a. setInterval()函数有两个参数，参数一是要重复运行的函数，参数二是每次执行的时间间隔(单位为毫秒)
   b. setInterval(myFunc,5000)

设置断点进行调试，注意观察变量的值

各浏览器调试方法：

[1] Chrome/Chromium：
    a. Ctrl+Shift+I；
    b. http://code.google.com/chrome/devtools/docs/scripts-breakpoints.html
    c. https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints?utm_source=dcc&utm_medium=redirect&utm_campaign=2016q3
[2] Firefox-Firebug：
    a. http://getfirebug.com/javascript
[3] IE9：
    a. F12
    b. http://msdn.microsoft.com/en-us/library/ie/dd565622(v=vs.85).aspx
[4] Opera：
    a. Dragonly
    b. http://www.opera.com/dragonly/documentation

Ch07_面向对象编程

7.1 什么是面向对象编程（OOP）

[1] 面向过程：
    a. 特点：把数据保存到变量里，然后由一系列指令操作变量。每个指令（或一系列指令，比如函数）都能创建、删除或修改数据，显得数据与程序代码在某种程度上是“分离”的。
[2] 面向对象：
    a. 程序与其操作的数据密切关联——OOP 把程序的数据包含在被称为“对象”的独立体里，每个对象都有自己的属性（数据）和方法（指令）。
    b. 通用功能的对象和方法&实例
        i. 说明：对象的实例是对象“模块”的特定实现，是基于特定数据的能够工作的对象。在面向对象编程中，这种对象模板一般被称为“类”。但此书中不适用“类”这个术语，因为 JS 实际上并不使用类。但它的“构造函数”概念与之类似，见后。
    c. 优点：
        i. 代码复用。面向对象编程能够以多种方式复用代码。利用普通的函数也能实现代码复用，但跟踪全部需要传递的变量、它们的作用域和含义很困难。而若用 OOP 来实现，只需要标明每个对象的属性和方法，保证它们遵守规则，其他程序或程序员就可轻松使用这些对象。
        ii. 封装。通过仔细设置属性和方法对于程序其他部分的“可见性”，我们可以定义对象如何与脚本的其他部分相互作用。对象的“内部”内容对外是隐蔽的，外部代码若要访问对象的数据，只能通过对象标明的接口。
        iii. 继承。遇到如下情况：需要编写的代码与已经编写过的代码几乎是相同的，但不完全相同。利用“继承”，可以基于已经定义的对象来创建新对象。新对象会“继承”老对象的属性和方法，还可根据需要添加或调整属性和方法。

[3] JS 不属于面向对象语言，但提供了足够的支持。

7.2 创建对象

7.2.1 创建直接实例（声明对象的直接实例）

JS 有一个内置对象 Object，利用它可以创建一个空白的对象：

myNewObject=new Object();//得到一个新对象，此时还没有任何属性和方法。

添加属性：myNewObject.info='I am a shiny new object';//对象具有了一个文本字符串类型的属性。

添加方法：首先定义一个函数，然后把它附加到 myNewObject 上作为方法。

```
function myFunc(){
        alert(this.info);
};
myNewObject.showInfo=myFunc;
```

使用句点形式可调用这个方法：myNewObject.showInfo();

注意：在把函数关联到对象时，只使用函数名称，不能包含括号。因为我们是要把函数 myFunc()的定义赋予 myNewObject.showInfo 方法。如果使用像下面这样的代码：myNewObject.showInfo=myFunc();。**其作用是让 JS 执行函数 myFunc()，然后把它的返回值赋予 myNewObject.showInfo。**

//有点像函数指针

7.2.2 使用关键字 this

[1] 在内嵌的事件处理器中使用的 this 是指 HTML 元素本事，如中 this 指的是 img 元素。

[2] 在函数（或方法）里使用 this 时，指向函数的"父对象"。

a. 在函数最初声明时，它的父对象是全局对象 window。Window 对象并没有名为 info 的属性，如果直接调用 myFunc()函数，会报错。

b. 现在，将 myFunc()赋予 myNewObject 对象的方法 showInfo，则对于 showInfo()方法来说，它的父对象是 myNewObject，所以 this 是 myNewObject。

7.2.3 匿名函数

非创建一个单独的函数，不需要给函数命名。

```
myNewObject.showInfo=function(){
        alert(this.info);
}
```

使用类似的语句，可给实例化的对象添加任意多的属性和方法。

提示：JS 还可使用 JSON（JS 对象标签）技术直接创建对象的实例，见 Ch08。

7.2.4 使用构造函数（用于创建同一对象的多个实例：避免重复地创建对象、添加属性、定义方法等）

“单例”：是有一个全局实例的对象，有适用的场合，如程序的用户应该只有一个相关的 userProfile 对象，其中包含他的姓名、最后访问的页面等类似的属性。

对象构造函数会创建某种模板，方便实现多次实例化。

关键是在声明的函数的定义中使用 this 添加属性和方法：

```
function myObjectType(){
        this.info='I am a shiny new obj.';
        this.showInfo=function(){alert(this.info);}
        this.setInfo=function(newInfo){this.info=newInfo;}
}//注意这里的 setInfo 只是一个方法，使用它的时候才有用。
```

7.2.5 对象实例化

创建对象实例被称为“实例化”一个对象。

```
var myNewObject=new myObjectType()
//感觉上还是很像 C++的
```

7.2.6 构造函数参数

把对象实例化时，还可通过向构造函数传参来定制对象。//也类似于 C++

区别是，声明 function 函数名(参数表)时，参数表不需要写类型 var。

7.3 使用 prototype 扩展和继承对象

基于已有对象来修改对象，使其拥有新的方法或属性，甚至可创建完全崭新的对象。

7.3.1 扩展对象

当一个对象已被实例化之后，若想使其具有新的方法和属性，则可使用关键字 prototype 添加属性和方法，并可以用于对象的全部实例。不影响已有方法。

对象名.prototype.新方法名=function(){}，如
Person.prototype.sayHi=function(){alert(“Hi, ”+this.name);}

7.3.2 继承

继承：从一种对象类型创建另一种对象类型，新对象类型继承老对象类型的属性和方法，还可添加自己的属性和方法。通用的对象类型—细化—特定的类型。

JS 使用 prototype 模拟实现继承：因为 object.prototype 可以添加新方法和属性，所以可以用它把已有的构造函数里的全部方法和属性都添加给新的对象。

先创建一个新对象，然后用 prototype 将老对象的属性和方法继承。

```
function Pet(){
        this.animal="";
```

```
        this.name="";
        this.setAnimal=function(newAnimal){this.animal=newAnimal;}
        this.setName=function(newName){this.name=newName;}
}
function Dog(){
        this.breed="";
        this.setBreed=function(newBreed){this.breed=newBreed;}
}
Dog.prototype=new Pet();
```

7.3.3 扩展 JS 内置的对象

比如可以添加返回字符串逆序结果的方法：

```
String.prototype.backwards=function(){}
```

7.4 封装

封装：把数据和指令隐藏到对象内部，具体实现方法在不同的语言里有所区别。在 JS 中的实现为，在构造函数内部声明的变量和函数只能在对象内部使用，对于外部不可见。

被封装的操作在构造函数内部进行。

如果没有使用 this 把变量和函数"注册"为属性和方法，它们就不能从函数外部调用，即"私有的"。

7.5 使用功能检测

浏览器的检测很复杂，navigator 对象包含的信息可能是有偏差的，且新浏览器或新版本可能包含新的功能和特性，因此已有的浏览器检测代码可能崩溃。

与检测浏览器相比，更好的做法是让 JS 查看浏览器是否支持代码所需的功能。方式是检测特定对象、方法或属性是否可用，一般为尝试使用对象、方法或属性，然后检测 JS 返回的值。

[1] 用 if 检测方法是否可用：

```
if(document.getElementById){myElement=document.getElementById(id);}else{}
```

[2] 用 typeof 操作符检测某个 JS 函数是否存在：

a. if(typeof document.getElementById=="function"){}else{}
b. typeof 的返回值："number"，"string"，"boolean"，"object"，null，undefined
c. 不仅可检测 DOM 和内置对象、方法和属性的存在，还可以检测脚本中创建的对象、方法和属性。

7.9 练习（见文件）

Ch08_JSON(JS 对象标签)简介

8.1JSON 是什么

JSON 是 JS 对象的一种简单紧凑的标签，是一种轻量级的数据交换语法，并可以用来创建 JS 对象的实例。 使用 JSON 表达方式时，对象可以方便地转换为字符串来进行存储和转换（比如在不同程序或网络之间）。

然而，JSON 的真正优雅之处在于对象在 JSON 里是以普通 JS 代码表示的，因此可以利用 JS 的“自动”解析功能，让 JS 把 JSON 字符串的内容解释为代码，而不需要其他的解析程序或转换器。

JSON 官网：http://json.org/

官方的 JSON 规范文档：http://www.ietf.org/rfc/rfc4627

如何判断浏览器是否支持 JSON？

利用 typeof 操作符判断是否存在 JSON 对象：

if(typeof JSON==’object’){}else{}

//在使用这段代码时，要确保脚本里没有自己定义的名为 JSON 的对象。

8.1.1JSON 语法

JSON 语法的表示方式是一系列成对的参数与值，参数与值由冒号分隔，每对之间以逗号分隔：”param1”:”value1”,””param2”:”value2”,”param3”:”value3”

最终这些序列用花括号包装起来，构成表示数据的 JSON 对象：

```
var jsonObject={
        ”param1”:”value1”,
        ”param2”:”value2”,
        ”param3”:”value3”
}
```

对象 jsonObject 的定义使用标准 JS 标签的子集，是标准的 JS 代码。

使用 JSON 标签编写的对象也具有属性和方法，能够利用句点标签进行访问，如：alert(jsonObject.param1);

更好的是，JSON 是一种以字符串格式实现数据交换的通用语法。不仅仅是对象，任何能够以系”参数”:”值”对表示的数据都能够用 JSON 标签表示。使用前面提到的“序列化”过程可以方便地把 JSON 对象转换为字符串，而序列化的数据便于在网络环境中进行存储和传输。如何序列化 JSON 见 8.3。

说明：作为一种通用的数据交换语法，JSON 的用途有些类似于 XML，但它更易于阅读和理解。另外，大型 XML 文件的解析过程较慢，而 JSON 提供的是 JS 对象，随时可以使用。

Flickr 如何支持 JSON：http://www.flickr.com/services/api/response.json.html

8.2 访问 JSON 数据

字符串的“去序列化”：为还原以 JSON 字符串编码的数据，需要把字符串转换为 JS 代码。

8.2.1 使用 eval()将 JSON 字符串转换为 JS 对象（以前浏览器不直接支持 JSON）

eval()函数使用 JS 解释程序解析 JSON 文本来生成 JS 对象：

```
var myObject=eval('(' + jsonObjectString + ')');
```

然后就可以在脚本里使用这个 JS 对象：

```
var user='{"username":"philb1234","location":"Spain","height":1.80}';

var myObject=eval('('+user+')');

alert(myObject.username);
```

说明：JS 的 eval 函数会计算或运行作为参数传递的内容。如果参数是表达式，eval()会计算它的值，比如：

```
var x=eval(4*3);
```

如果参数包含一个或多个 JS 语句，eval()执行这些语句：

```
eval("a=1;b=2;document.write(a+b);");
```

注意：字符串必须像这样包含在括号里，避免造成含义不明确的 JS 语法。

8.2.2 使用浏览器对 JSON 直接支持

浏览器会创建一个名为 JSON 的 JS 对象来管理 JSON 的编码和解码。这个对象有两个方法：stringify()和 parse()。

JSON.parse()方法用于解释 JSON 字符串。它接收一个字符串作为参数，解析它，创建一个对象，并且根据字符串里发现的"参数":"值"对设置对象的参数：

```
var Mary='{"height":1.9,"age":36,"eyeColor":"brown"}';

var myObject=JSON.parse(Mary);

var out="";

for(i in myObject){

        out+=i+" = "+myObject[i]+'\n';

}

alert(out);
```

8.3JSON 的数据序列化

在数据存储和转换时，“序列化”是指把数据转换为便于通过网络进行存储和传输的形式，稍后再恢复为原始的格式。

JSON 选择字符串（切记输入的是字符串，要是完整的 JSON 格式，带上大括号、引号等）作为序列化数据的格式。在直接支持 JSON 的浏览器中，可使用 JSON。stringify()方法。

JSON.stringify()

利用 JSON.stringify()方法可以创建对象的 JSON 编码字符串。

先创建一个简单的对象，添加一些属性。然后使用 JSON.stringify()方法序列化这个对象。

<body onload=”函数名()”></body>//当页面加载完成之后，页面<body>元素附加的 window 对象的 onload 时间处理器会调用等号后的函数。

8.4JSON 数据类型

“参数”:”值”：

[1] 参数部分不能是 JS 保留的关键字，不能以数字开头，除了下划线和$外，不能包含任何特殊字符。
[2] 值可以是数值、字符串、布尔值、数组、对象、null(空)
[3] JS 有一些数据类型不属于 JSON 标准，包括 Date、Error、Math 和 Function。这些数据必须用其他数据格式表示，使用遵循相同编码和解码规则的其他编解码程序进行处理。

8.5 模拟关联数组

由 Ch05，JS 数组里的元素具有唯一的数值索引，在部分其他语言可以使用文本形式的索引（如 Python 的字典是文本形式的索引），从而让代码的描述性更强。

JS 不直接支持“关联”数组，但利用对象可以模拟这种行为，比如 JSON 标签。

提示：在 JS 中，object[“property”]和 object.property 是相同的语法。

注意：JSON 模拟的并不是真正的关联数组。？？？P95？？？

8.6 使用 JSON 创建对象

JSON 能以类似以方括号表示数组的方式定义 JS 对象。

提示：虽然 JSON 是为描述 JavaScript 对象而开发的，但它是独立于任何编程语言和平台的，Java、PHP、C 等都包含 JSON 库和工具。

8.6.1 属性

提示：JS 语句 var myObj=new Object();会创建对象的一个“空”实例，没有任何方法和属性，其对应的 JSON 标签表示方法是 var myObject={};

8.6.2 方法

利用匿名函数可以给对象添加方法。

var user={

“username”:”ABC”;//参数、值对列出对象属性，可用 user.username 访问对象的属性。

“setName”:function(newName){this.username=newName;}

}

注意：在 JS 环境中使用这种方式添加方法是可以的，但当 JSON 作为通用数据交换格式时，不能这样使用。在直接支持 JSON 解析的浏览器里，以这种方式声明的函数会解析错误，但 eval()函数仍然可以用。如果实例化的脚本只是在自己的脚本里使用，还是可以这样使用的。JSON 安全性见 8.7。

8.6.3 数组

属性值可以是数组（放在值的位置上），可以索引访问该数组的元素（从 0 起）

8.6.4 对象

JSON 对象还可包含其他对象。通过把数组元素本身设置为 JSON 编码的对象，则可利用句点标签访问它们。//有点像指针或结构体的一层层嵌套。

处理多层次 JSON 对象。

8.7JSON 安全性

使用 JS 的 eval()函数能够执行任何 JS 命令，可能导致安全问题，特别是处理来源不明的 JSON 数据时。

更安全的方法是使用内置 JSON 解析器的浏览器把 JSON 字符串转换为 JS 对象，它**只识别** JSON 文本，而不会执行脚本命令。同时，内置的 JSON 解析器的速度也比 eval()快一些。

较新的浏览器都内置了 JSON 解析器，ECMAScript（JS）标准也明确了它的规范。

8.11 练习（见文件）

Ch09 响应事件

9.1 理解事件处理器

事件触发：为特定事件触发后执行编写的代码，需要从特定 HTML 元素的特定事件处理程序里调用它。

[1] 探测用户操作行为；
[2] 非用户操作触发，如 window.onload 事件是在页面加载完成时触发的。

9.1.1 事件范例

常见事件处理器

事件处理器	响应的事件
onBlur	用户离开字段
onChange	用户修改了值，正要离开

onClick	用户点击
OnDblClick	用户双击鼠标
onFocus	用户进入字段（点击它或跳转到它）
onKeydown	在元素激活时，一个按键被按下
onKeyup	在元素激活时，一个按键被释放
onKeypress	在元素激活时，一个按键被按下，然后被释放
onLoad	对象已经加载 window.onload 事件在浏览器完整加载页面时触发。
onMousedown	鼠标按钮在一个对象上被按下
onMouseup	鼠标按钮被释放
onMouseover	鼠标移动到对象上
onMousemove	鼠标在对象上方移动
onMouseout	鼠标离开对象
onReset	用户重置表单
onSelect	用户选择了对象的一些内容
onSubmit	用户提交表单
onUnload	用户关闭浏览器窗口

9.1.2 添加事件处理器

[1] 与 HTML 混合在一起的内联事件处理器：

a. <input type=”button” onclick=”myFunction()”/>
b. 将 HTML 和 JS 代码分离通常是好事情，见 Ch13。

[2] 注册事件处理器：

a. 假设有个函数 buttonAlert()，现要在 HTML 页面上一个按钮的 onClick 事件处理器中调用它。
在 JS 代码里利用 document.getElementById()方法访问按钮元素，再把函数赋予它的 onclick 方法：
document.getElementById(“myButton”).onclick=buttonAlert;//类似函数指针，注意若有括号则变成赋返回值
此方法的优势：
灵活性，比如可以根据条件停止或启动事件处理器，或是把它应用于其他页面元素，或是修改它的操作。而在以内联方式设置事件处理器时，实现上述操作很麻烦。
b. 另外，如果函数 buttonAlert()不需要在其他部分使用，则可以利用匿名函数的方式来添加事件处理器：
document.getElementById(“myButton”).onclick=function(){//若干行代码}

9.1.3 删除事件处理器

要删除事件处理器，只需赋值为 null：

document.getElementById(“myButton”).onclick=null;

null 会覆盖以前赋予的任何内容，从而有效删除事件处理器。

9.2 默认操作

一般情况下，特定 HTML 元素的事件处理器实在元素默认操作之前执行的。如给一个链接的 onClick 事件处理器赋予一个函数：

```
<a href="target.html" id=”myLink”>Link text</a>
<script>
        document.getElementById("myLink").onclick=function(){
//要执行的代码
        this.href="http://www.google.com";
}
</script>
```

当用户点击这个链接时，其默认操作是打开链接指向的 target.html。由于 onClick 事件处理器在默认操作之前执行，则可把元素的默认操作修改为其他动作。//广告？如果只执行一次？之后转为正常？？？

由于 onClick 事件处理器的代码首先执行，当链接跳转时，其 href 属性已经被修改为 www.google.com 了。

9.2.1 禁止默认操作

利用事件处理器优先执行的特点，可以禁止 HTML 元素的默认操作。

如果事件处理器给 HTML 元素返回一个 false，元素默认操作就不会执行。

```
<a href="target.html" id=”myLink”>Link text</a>
<script>
        document.getElementById("myLink").onclick=function(){
//要执行的代码
        this.href="http://www.google.com";
        return false;
}
</script>
```

再稍微调整一个代码，则可让时间处理器自己决定是否允许默认行为发生。

```
<a href="target.html" id=”myLink”>Link text</a>
<script>
        document.getElementById("myLink").onclick=function(){
//要执行的代码
        this.href="http://www.google.com";
```

return confirm("I’m going to send you to Google instead. OK?");//函数返回值将有 confirm()对话框响应决定，确定/true/跳转到新的 URL，取消/false/默认操作不进行且不跳转

}

</script>

【注意】9.2.1 完全无效，打开的仍是 target.html，ie11/Firefox。（见文件）

9.2.1.1 禁止 onSubmit 事件的默认操作（见文件）[表单验证]

每个 HTML 表单元素都有一个 onSubmit 事件处理器，在表单为提交时触发。我们可以给这个事件处理器赋予一个函数，让函数返回 false，从而阻止表单元素的提交。

9.3event 对象（事件详情，如键盘上哪个键被按下了，事件触发时鼠标在什么位置）

event 对象由浏览器自动生成，包含的属性涉及被触发事件的方方面面。

浏览器兼容性问题对于如何处理事件的影响超过对于 JS 其他部分的影响。为掌握如何编写优秀的跨浏览器代码，既要理解 W3C 兼容浏览器采取的方式，又要理解微软 IE 类型浏览器的方式。

9.3.1W3C 方式

在 Firefox 等更严格遵循 W3C 规范的浏览器中，当事件被触发时，会自动把 event 对象作为参数传递给事件处理器。为了访问 event 对象的属性，需要给它设置一个名称，即给事件处理器声明一个变量，如：

myElement=document.getElementById(“someID”);

myElement.onclidk=function(e){…}

在这个反例中，访问变量 e 的属性就可以访问 event 对象的属性：

myElement.onclidk=function(e){alert(e.type);}

提示：使用“e“作为名称是个惯例，但不是必须的，可任意有效命名。

9.3.2 微软方式

微软给 window 对象设计了一个 event 属性，它包含最近一次被触发时间的细节：

myElement=document.getElementById(“someID”);

myElement.onclick=function(e){alert(window.event.type);}

9.4 跨浏览器的事件处理器

技巧：在事件处理函数里检测 event 对象的存在性。如果有 event 对象，就说嘛是 W3C 兼容的浏览器；如果不存在，就可能是微软类型的，这是需获得 window.event 属性，把它赋予对象 e：

myElement.onclick=function(e){

if(!e){var e=window.event;}

…

}//此方法属于浏览器功能检测

但是，W3C 和微软规范对于 event 对象包含什么属性的标准不同。

W3C 和微软通用事件属性

属性	描述
Type	事件类型
altKey	Alt 键是否按下（布尔值）
clientX, client	相对于浏览器窗口的事件坐标
ctrlKey	Ctrl 键是否按下（布尔值）
keycode	键盘字符编码
screenX, screen	相对于屏幕的事件坐标
shiftKey	Shift 键是否按下（布尔值）

一些不同的事件属性

微软	W3C	描述
fromElement	relatedTarget	mouseover 或 mouseout 事件从哪个对象来
toElement	relatedTarget	mouseover 或 mouseout 事情到哪个对象去
offsetX, offsetY	n/a	事件在元素里的水平和垂直坐标
n/a	pageX, pageY	事件在文档里的水平和垂直坐标
x, y	layerX, layerY	事件在<body>元素里的水平和垂直坐标
srcElement	Target	接收事件的对象

为实现跨浏览器的代码，需要在脚本中检测 event 对象是否具有我们需要的方法和属性，如：

if(!e) var e=window.event;

var element=(e.target)?e.target:e.srcElement

列出某事件的所有属性，for(i in e)

9.5 事件处理器高级注册方式

添加事件处理器的方式：

[1] 内联方式：
   a. 以 HTML 标签属性的形式添加，并且直接添加函数
   b. element.onclick=myFunction
   c. HTML 和 JS 代码混合在一起，影响了代码的灵活性和可维护性

[2] 利用 document.getElementById()方法访问元素，再把函数赋予它的方法
   a. 缺点：每个属性只能添加一个事件处理函数，再次赋值会覆盖。
   b. 可以包装多个函数为一个函数，但没有灵活性

9.5.1W3C 方式

可以根据需要添加或删除任意数量的事件处理器：

[1] element.addEventListerner('click',myFunction,false);
[2] element.addEventListerner('click',myOtherFunction,false);
[3] element.removeEventListerner('click',myFunction,false);
[4] 参数一知名要捕获的事件；参数二指明事件要执行的函数；参数 3 是布尔值，表示当多个嵌套元素捕获同一个事件时事件处理的顺序，比如当用户点击 元素里的<p>元素时，哪个元素应该限制性自己的 onClick 事件处理器，一般情况下设为 false。
[5] 提示：当两个或多个嵌套元素捕获到同一个事件时，浏览器厂家和 W3C 采取两种方式之一进行处理。一种称为"捕获"，也就是外部的元素首先运行事件处理器，然后是嵌套的元素，逐渐向内层深入。另一种是"冒泡"方式，最内部的事件处理器首先执行，由内向外逐渐执行。详见：www.quirksmode.org/js/events_order.html
    a. //感觉可以用来写广告，双向广告，中间是正常跳转

9.5.2 微软方式

微软提供两种类似的方法：

[1] element.attachEvent('onclick',myFunction);
[2] element.detachEvent('onclick',myFunction);
[3] 没有参数三，且事件名称使用前缀 on（对应 W3C）

9.5.3 跨浏览器的实现方式

仍为使用特性检测来判断可以使用哪种事件处理方式，用 if 检查有无某方法。

9.5.3.1 添加和删除事件处理器（见文件，未得到应有效果）

9.7 问答

一段需要检测鼠标点击的脚本在 IE 中运行不正常是为什么？

- W3C 和微软的 event 对象都有 button 属性，但返回的整数值是不一样的。
- 在 W3C 浏览器里，左键 0，中键 1，右键 2
- 在微软浏览器里，左键 1，右键 2，中键 4
- 它们都是 2 的幂，所以返回的数值能够表示出按钮同时按下的状态（和/位）

9.9 练习（见文件）

Ch10JavaScript 和 cookie

前面的 JS 技术还不能把信息从一个页面传递到另一个页面，而 cookie 能够在用户的计算机上保存少量数据并且远程获得它们，从而让网站可以保存一些细节信息，比如用户的习惯设置或上一次访问网站的时间。

10.1 什么是 Cookie

HTTP 协议是一种"无状态"协议，页面从服务器发送给浏览器后，事务就完成了，不保存任何信息。这给在浏览器会话期间或是会话之间维持某种连续性造成困难。

cookie 可以记录用户的最后一次访问，保存用户的偏好设置的列表，或是购物车商品。站点不能完全依赖 cookie，因为很多用户不允许站点在自己的计算机上保存 cookie（广告）。

10.1.1cookie 的局限

浏览器对于能够保存的 cookie 数量有限制，通常为几百个。一般情况下，每个域名 20 个 cookie 是允许的，而每个域最多能保存 4KB 的 cookie。

除了尺寸限制可能导致的问题，有很多原因都可能导致硬盘上的 cookie 消失，比如到达有效期限，或是用户清理 cookie 信息，或是换用其他浏览器。因此，永远都不应该使用 cookie 保存重要数据，而且在编写代码时一定要考虑到不能获取所期望 cookie 时的情况。

10.2document.cookie 属性

JS 使用 document 对象的 cookie 属性存储和获取 cookie。

每个 cookie 基本上就是一个由成对的名称和值组成的字符串，如：username=sam

当页面加载到浏览器里时，浏览器会收集与页面相关的全部 cookie，放到“类似字符串（但不是真正的字符串）”的 document.cookie 属性里。在这个属性里，每个 cookie 以分号分隔：

username=sam;location=USA;status=fullmember;

10.2.1 数据的编码和解码

某些字符不能在 cookie 里使用，包括分号、逗号及空白符号（比如空格和制表符）。在把数据存储到 cookie 之前，需要对数据进行编码，以便实现正确的存储。

在存储信息之前，使用 JS 的 escape()函数进行编码，而获得原始的 cookie 数据时就使用相应的 unescape()函数进行编码。

escape()函数把字符串里任何非 ASCII 字符都转换为相应的 2 位或 4 位十六进制格式，比如空格转换为%20，&转换为%26。

举例来说，下面的代码会输出变量 str 里保存的原始字符串及 escape()编码以后的结果：

```
var str='Here is a (short) piece of text.';

document.write(str+'<br/>'+escape(str));
```

左括号%28，右括号%29

除了*、@、-、_+、.、/之外的特殊符号都会被编码。

10.3cookie 组成

document.cookie 里的信息看上去就像是由成对的名称和值组成的字符串，每一对数据的形式是：name=value;

但实际上 cookie 还包含其他一些相关信息。

说明：2011 年发布的 RFC6265 是 cookie 的正是规范，参见 http://tools.ietf.org/html/rfc6265。

10.3.1cookieName 和 cookieValue

cookieName 和 cookieValue 就是在 cookie 字符串里看到的 name=value 里的名称与值。

10.3.2domain

domain 属性向浏览器指明 cookie 属于哪个域。这个属性是可选的，在没有指定时，默认值是设置 cookie 的页面所在的域。

这个属性的作用在于控制子域对 cookie 的操作。举例来说，如果其设置为 www.example.com，那么子域 code.example.com 里的页面就不能读取这个 cookie。但如果 domain 属性设置为 example.com，那么 code.example.com 里的页面就能访问这个 cookie。

但是，不能把 domain 属性设置为页面所在域之外的域。

10.3.3path

path 属性制定可以使用 cookie 的目录。如果只想让目录 documents 里的页面设置 cookie 的值，就把 path 设置为/documents。这个属性是可选的，常用的默认路径是/，表示 cookie 可以在整个域里使用。

10.3.4secure

secure 属性是可选的，而且几乎很少使用。它表示浏览器在把 cookie 发送给服务器时，是否应该使用 SSL 安全标准。

10.3.5expires

每个 cookie 都有个失效日期，过期就自动删除了。expires 属性要以 UTC 时间表示。如果没有设置这个属性，cookie 的生命期就和当前浏览器会话一样长，会在浏览器关闭时自动删除。

10.4 编写 cookie

要编写新的 cookie，只要把包含所属属性的值赋予 document.cookie 就可以了：

```
document.cookie="username=sam;expires=15/06/2013 00:00:00";
```

使用 JS 的 Date 对象可以避免手工输入日期和时间格式：

```
var cookieDate=new Date(2013,05,15);
document.cookie="username=sam;expires="+cookieDate.toUTCString();
```

在实际编写代码时，应该用 escape()函数来确保在给 cookie 赋值时不会有非法字符：

```
var cookieDate=new Date(2013,05,15);
var user=”Sam Jones”;
document.cookie=="username=”+escape(user)+”;expires="+cookieDate.toUTCString();
```

10.5 编写 cookie 的函数（见文件）

注意：浏览器本身的安全机制可能会阻止编写 cookie 的函数，则需要把文件上传到互联网或局域网的 Web 服务器。（因为这时是使用 file://协议查看计算机上的文件，可以本地 localhost）

注意：IE11 的 cookie 在 network-单条项目

10.6 读取 cookie

读取 cookie 值的过程很大程度上依赖于 split()函数，以指定字符为分隔符，把分解的结果保存到数组里：array[index]

var crumbs=document.cookie.split(';');

要获得特定名称的 cookie，接下来要对数组 crumb 进行搜索，得到特定的部分。

然后使用 indexOf()和 substring()返回 cookie 值的部分，再通过 unescape()函数进行解码，得到 cookie 值。

10.7 删除 cookie（见文件，在 EditPlus 中有效，IE 和 Firefox 失败：Warning: Unknown: failed to open stream: No such file or directory in Unknown on line 0//应当和 Apache 和 PHP 有关）

要想删除一个 cookie，只需要把它的失效日期设置为今天以前的日期，浏览器就会认为它已经失效了，从而删除它。

注意：即使在脚本里删除了 cookie，某些浏览器的有些版本也会把 cookie 维持到重新启动浏览器。如果 cookie 是否被删除是程序运行的条件，就应该使用 getCookie 来测试被删除的 cookie，确保它的确不存在了。

10.8 在一个 cookie 里设置多个值

方法是把需要的值组成一个字符串，让它成为要保存在 cookie 里的值。为了以后分解其中给你的信息，要在这个字符串里放置特殊字符（所谓的"定界符"）来分隔不同的数据，如"|"。之后用 split()分割。

有些浏览器要求 cookie 的数量不能超过 20，如果用一个 cookie 保存多个数值，可以在一定程度上打破该限制，但信息总体仍不能超过 4KB.

注意：在使用一个 cookie 保存多个值时，不能使用可能出现在编码数据里的字符（除非那个字符也当作定界符），也不能使用等号(=)或分号(;)，因为它们用于组成"name=value"和分隔多对数据。另外，cookie 一般不能包含空白和逗号，所以它们也不能当作分界符。

10.12 练习（见文件）

注意：replace()方法的"全局"配置项/g，或者使用正则表达式。

Part3_Ch11-15 文档对象模型（DOM）

Ch11 遍历 DOM