**目录**

[13.3 数组操作 2](#_Toc21931)

[第14章 函数 6](#_Toc31847)

[14.1 函数 6](#_Toc8089)

[14.2 本地变量与全局变量 9](#_Toc8702)

[14.3 递归 11](#_Toc77)

[第15章 事件 17](#_Toc30875)

[15.1 事件的概念 17](#_Toc19191)

[15.2 鼠标单击事件 18](#_Toc831)

[15.3 鼠标经过/移开事件 19](#_Toc26817)

[15.4 光标聚焦/失焦事件 21](#_Toc30681)

[15.5 内容选中/改变事件 23](#_Toc2980)

[15.6 加载/卸载事件 25](#_Toc19772)

[第16章 对象 26](#_Toc4347)

[16.1 对象的概念 26](#_Toc28127)

[16.2 Date对象 27](#_Toc22973)

[16.3 String对象 29](#_Toc23990)

[16.4 Math对象 33](#_Toc19845)

[第17章 浏览器对象模型BOM 35](#_Toc21399)

[17.1 window对象 35](#_Toc9935)

[17.2 计时器 37](#_Toc2830)

[17.3 Screen对象 41](#_Toc23052)

[第18章 文档对象模型DOM 43](#_Toc31119)

[18.1 DOM的概念 43](#_Toc13956)

[18.2 获取结点对象 44](#_Toc10844)

[18.3 结点属性 47](#_Toc5918)

[18.4 结点操作 49](#_Toc24233)

**13.3 数组操作**

**数组操作**

数组的常见操作主要包括：

1. **计算数组长度**：引用数组的**length**属性获取数组长度，需要注意的是，JS数组的length属性是**可变**的。

|  |  |
| --- | --- |
| **范例：计算数组长度** | |
| *var* arr = [0, 1, 2, 3, 4];  console.log(arr.length);  arr.length = 10;  console.log(arr.length); | |
| **运行结果** | 5  10 |

2. **增加元素**：

* 使用下一个**未使用**的索引，任何时刻可以不断向数组增加新元素。

|  |  |
| --- | --- |
| **范例：增加元素** | |
| *var* arr = [0, 1, 2, 3, 4];  arr[5] = 5;  console.log(arr); | |
| **运行结果** | [0, 1, 2, 3, 4, 5] |

* 使用**unshift()**方法可以向数组第一个元素前面添加一个元素，返回值为数组长度。

|  |  |
| --- | --- |
| **范例：头部增加元素** | |
| *var* arr = [0, 1, 2, 3, 4];  arr.unshift(5);  console.log(arr); | |
| **运行结果** | [5, 0, 1, 2, 3, 4] |

* **push()**方法可以向数组最后一个元素后面添加一个元素，返回值为数组长度。

|  |  |
| --- | --- |
| **范例：尾部增加元素** | |
| *var* arr = [0, 1, 2, 3, 4];  arr.push(5);  console.log(arr); | |
| **运行结果** | [0, 1, 2, 3, 4, 5] |

3. **删除元素**：

* **shift()**方法可以删除数组的第一个元素，返回值为被删除元素。

|  |  |
| --- | --- |
| **范例：删除头部元素** | |
| *var* arr = [0, 1, 2, 3, 4];  arr.shift();  console.log(arr); | |
| **运行结果** | [1, 2, 3, 4] |

* **pop()**方法可以删除数组的最后一个元素，返回值为被删除元素。

|  |  |
| --- | --- |
| **范例：删除尾部元素** | |
| *var* arr = [0, 1, 2, 3, 4];  arr.pop();  console.log(arr); | |
| **运行结果** | [0, 1, 2, 3] |

4. **合并数组**：使用**concat()**方法可以合并两个或多个数组，该方法不会改变原有数组，而是返回一个新的合并完的数组。

|  |  |
| --- | --- |
| **范例：合并数组** | |
| *var* arr1 = [1, 2, 3, 4, 5];  *var* arr2 = [6, 7, 8, 9, 10];  console.log(arr1.concat(arr2));  console.log(arr1);  console.log(arr2); | |
| **运行结果** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  [1, 2, 3, 4, 5]  [6, 7, 8, 9, 10] |

5. **数组转字符串**：使用**join(separator)**方法可以将数组转换为字符串，其中separator参数可选，用于指定要使用的分隔符，如果该参数省略，则使用逗号作为分隔符。

|  |  |
| --- | --- |
| **范例：数组转字符串** | |
| *var* arr = [0, 1, 2, 3, 4];  console.log(arr.join()); | |
| **运行结果** | 0,1,2,3,4 |

6. **字符串转数组**：使用**split(separator, n)**方法可以将字符串转换为数组，其中参数separator必选，用于指定将字符串按某个字符切割成若干个子字符串，并以数组的形式返回。参数n可选，用于指定返回的数组的最大长度，如果设置了该参数，返回的子串数量不会多于n；如果没有设置该参数，整个字符串都会被分隔。

|  |  |
| --- | --- |
| **范例：字符串转数组** | |
| *var* str = "hello HTML hello CSS hello JavaScript";  *var* arr = str.split(' ');  console.log(arr); | |
| **运行结果** | ["hello", "HTML", "hello", "CSS", "hello", "JavaScript"] |

7. **翻转数组**：使用**reverse()**方法可以颠倒数组中元素的顺序。

|  |  |
| --- | --- |
| **范例：翻转数组** | |
| *var* arr = [1, 2, 3, 4, 5];  console.log(arr.reverse()); | |
| **运行结果** | [5, 4, 3, 2, 1] |

8. **数组排序**：使用**sort(sortfun)**方法可以将数组进行排序，其中参数sortfun可选，用于指定排序规则，而且必须是函数，该参数省略则按照字符编码顺序排序。

|  |  |
| --- | --- |
| **范例：数组排序** | |
| *var* arr = [98, 1, 21, 8, 12, 2, 10, 25];  arr.sort(*function*(*a*, *b*) {      return *a* > *b* ? 1 : -1;  });  console.log(arr); | |
| **运行结果** | [1, 2, 8, 10, 12, 21, 25, 98] |

9. **数组切分**：使用**slice(start, end)**方法可以返回数组中被选定的元素，不包含下标为end的元素。其中参数start必选，用于指定开始位置，如果是负数则从数组尾部开始算起。参数end可选，用于指定结束位置，没有该参数省略，则切分到数组结束为止，如果是负数则从数组尾部开始算起。

|  |  |
| --- | --- |
| **范例：数组切分** | |
| *var* arr = [0, 1, 2, 3, 4, 5, 6];  console.log(arr.slice(2, 5)); | |
| **运行结果** | [2, 3, 4] |

10. **查找元素**：使用**indexOf(item, start)**方法可以查找指定元素，如果查找成功则返回该元素的下标，如果查找失败者返回-1。其中参数item必选，用于指定需要查找的元素，参数start可选，用于指定在数组中开始检索的位置，如省略则从第一个元素开始检索。

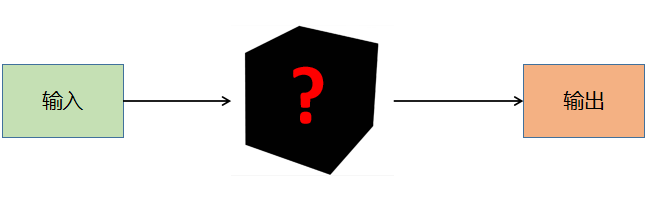
|  |  |
| --- | --- |
| **范例：查找元素** | |
| *var* arr = [0, 1, 2, 3, 4, 5, 6];  console.log(arr.indexOf(4)); | |
| **运行结果** | 4 |

**第14章 函数**

**14.1 函数**

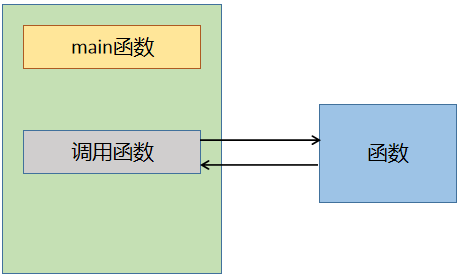
**函数(Function)**

函数执行一个**特定**的任务，JS提供了大量**内置函数**，例如alert()用来显示警告对话框、parseInt()用来将字符串转换为整型等。



**函数调用**

当**调用**函数时，程序**控制权**会转移给**被调用**的函数，当函数执行**结束**后，函数会把程序序控制权交还给其**调用者**。



**函数定义**

函数的定义需要使用关键字**function**，语法格式如下：

|  |
| --- |
| *function* funcName(*parameterList*) {      //code  } |

**为什么要使用函数？**

为什么不把所有的代码全部按顺序写，还需要自定义函数呢？使用函数有以下好处：

1. 避免“**代码复制**”：“代码复制”是程序质量不良的表现

2. 便于代码的**维护**

3. 避免重复制造“**轮子**”，提高开发效率

**函数的设计方法**

在设计函数的时候需要考虑以下的几点要素：

1. 确定函数的**功能**

2. 确定函数的**参数**（是否需要参数、参数的个数、参数的类型）

3. 确定函数的**返回值**（是否需要返回值、返回值的类型）

|  |  |
| --- | --- |
| **范例：函数实现返回最大值** | |
| *function* max(*num1*, *num2*) {      // if(num1 > num2) {      //     return num1;      // } else {      //     return num2;      // }      return *num1* > *num2* ? *num1* : *num2*;  }  console.log(max(4, 12));  console.log(max(54, 33));  console.log(max(0, -12));  console.log(max(-999, -774)); | |
| **运行结果** | 12  54  0  -774 |

|  |  |
| --- | --- |
| **范例：函数实现累加和** | |
| *function* sum(*start*, *end*) {  *var* total = 0;      for(*var* i = *start*; i <= *end*; i++) {          total += i;      }      return total;  }  console.log("1-100的累加和 = " + sum(1, 100));  console.log("1024-2048的累加和 = " + sum(1024, 2048)); | |
| **运行结果** | 1-100的累加和 = 5050  1024-2048的累加和 = 1574400 |

|  |  |
| --- | --- |
| **范例：函数实现输出i行j列由自定义字符组成的图案** | |
| *function* print\_chars(*row*, *col*, *c*) {  *var* str = "";      for(*var* i = 0; i < *row*; i++) {          for(*var* j = 0; j < *col*; j++) {              str += *c*;          }          str += "\n";      }      console.log(str);  }  print\_chars(5, 10, '?'); | |
| **运行结果** | ??????????  ??????????  ??????????  ??????????  ?????????? |

**14.2 本地变量与全局变量**

**本地变量(Local Variable)**

JS的本地变量是在**函数**里面被声明的，这些变量的**作用域**在**本地**，也就是说这些变量只能在函数内部可用。本地变量在函数**调用时被创造**，在函数**结束时被销毁**。

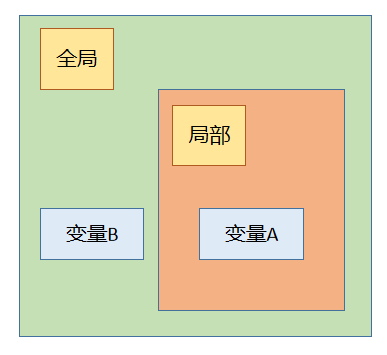
在函数中，函数的每次调用就会产生一个**独立**的空间，在这个空间中的变量，是函数的这次运行所独有的，函数的参数也是本地变量。

|  |  |
| --- | --- |
| **范例：本地变量** | |
| *function* test(*a*) {  *a* = 2;      console.log("a = " + *a*);  }  *var* a = 1;  console.log("a = " + a);  test(a);  console.log("a = " + a); | |
| **运行结果** | a = 1  a = 2  a = 1 |

**全局变量(Global Variable)**

JS的全局变量就是在**函数外**被声明的变量，作用域为**全局**，**所有**的在页面上的**脚本**和**函数**都可以获取这些变量。全局变量在其被声明时创建，在**页面被关闭时被销毁**。

全局变量的优先级**低于**[局部变量](https://baike.baidu.com/item/%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F/9844788" \t "https://baike.baidu.com/item/%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F/_blank)，当全局变量与局部变量重名的时候，起作用的是局部变量，全局变量会被暂时**屏蔽**掉。



|  |  |
| --- | --- |
| **范例：全局变量** | |
| *var* a = 1;          // 全局变量  *function* test() {  *var* a = 2;      // 本地变量      console.log("a = " + a);  }  test(); | |
| **运行结果** | a = 2 |

**14.3 递归**

**递归(Recursive)**

要理解递归，先得理解递归（见14.3章节）。

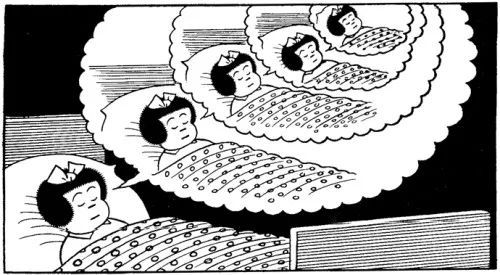
在函数的内部，直接或者间接的**调用自己**的过程就叫作递归。对于一些问题，使用递归可以简洁易懂的解决问题，但是递归的缺点是**性能低**，占用大量系统**栈空间**。

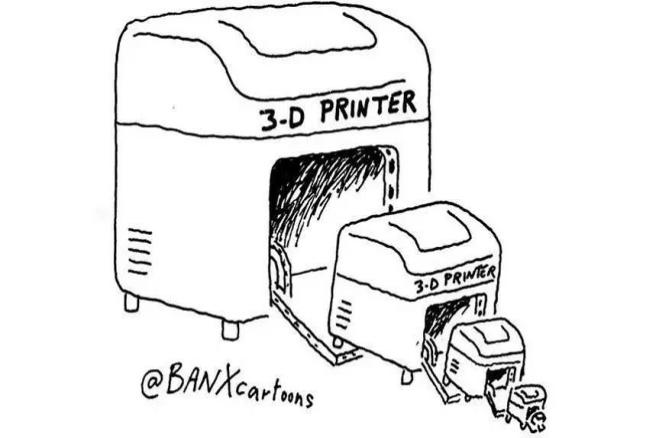
递归算法很多时候可以处理一些特别复杂、难以直接解决的问题，比如：

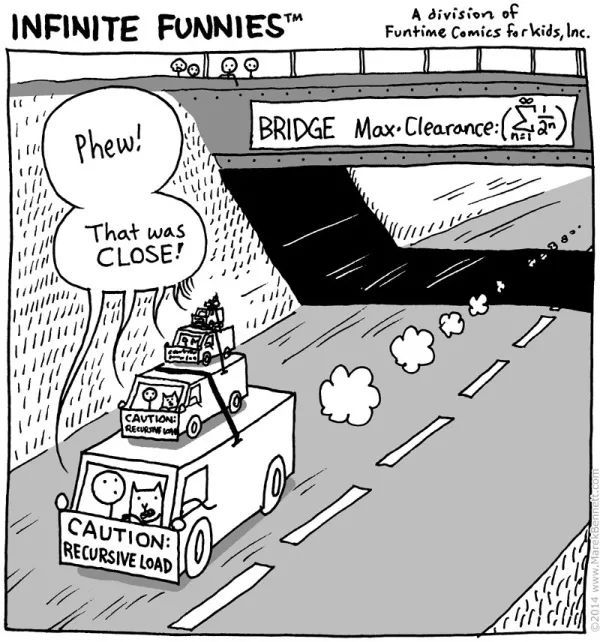
1. 迷宫问题

2. 汉诺塔问题

在定义递归函数时，一定要确定一个“**结束条件**”，否则会造成无限递归的情况，最终会导致**栈溢出**。



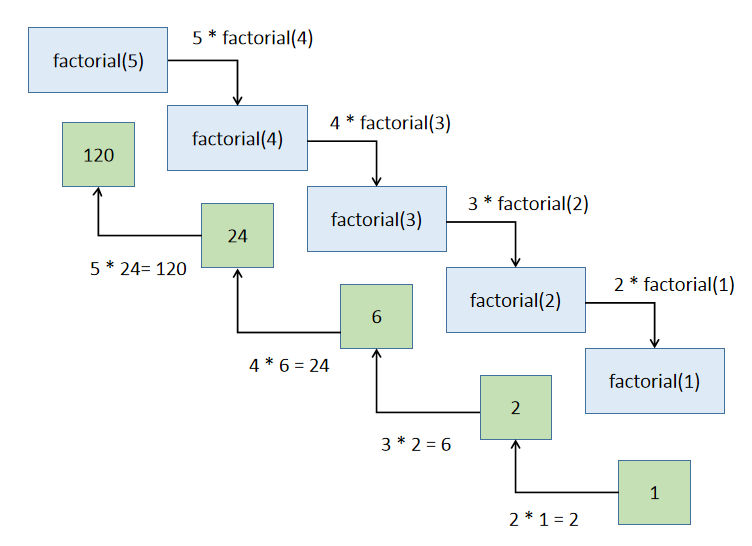




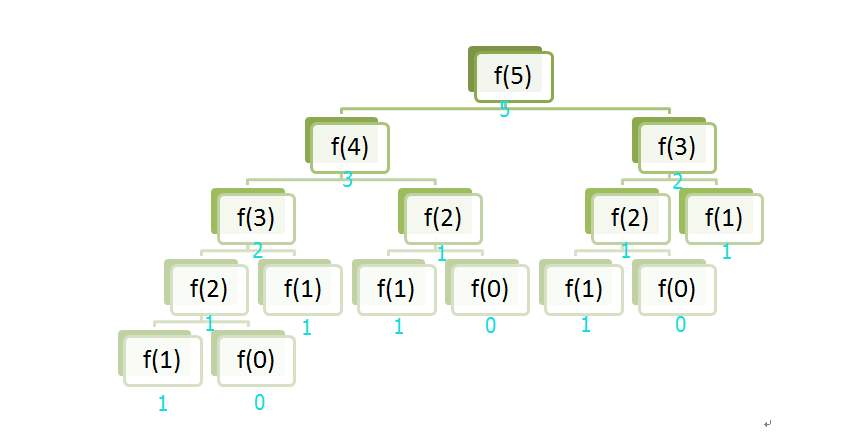
|  |  |
| --- | --- |
| **范例：无限递归** | |
| *function* tell\_story() {      console.log("从前有座山");      console.log("山里有座庙");      console.log("庙里有个老和尚和小和尚");      console.log("老和尚对小和尚在讲故事");      console.log("他讲的故事是：");      tell\_story();  }  tell\_story(); | |
| **运行结果** | 从前有座山  山里有座庙  庙里有个老和尚和小和尚  老和尚对小和尚在讲故事  他讲的故事是：  从前有座山  山里有座庙  庙里有个老和尚和小和尚  老和尚对小和尚在讲故事  他讲的故事是：  ... |

递归函数一般需要定义递归的**出口**，即“**结束条件**”，确保递归能够在适合的地方退出。

|  |  |
| --- | --- |
| **范例：阶乘** | |
| *function* factorial(*n*) {      if(*n* == 0 || *n* == 1) {          return 1;      }      return *n* \* factorial(*n* - 1);  }  console.log("5! = " + factorial(5)); | |
| **运行结果** | 5! = 120 |

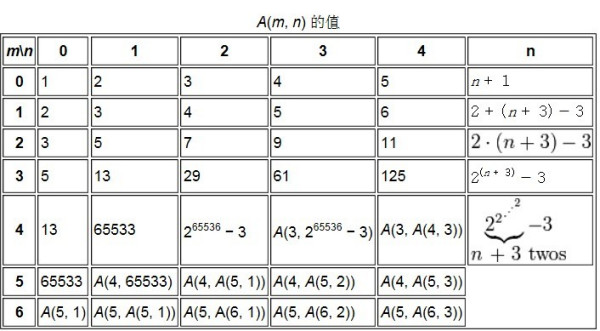


|  |  |
| --- | --- |
| **范例：斐波那契数列（递归）** | |
| *function* fibonacci(*n*) {      if(*n* == 1 || *n* == 2) {          return 1;      }      return fibonacci(*n* - 2) + fibonacci(*n* - 1);  }  n = 7;  console.log("斐波那契数列第" + n + "位：" + fibonacci(7)); | |
| **运行结果** | 斐波那契数列第7位：13 |



|  |  |
| --- | --- |
| **范例：斐波那契数列（迭代）** | |
| *function* fibonacci(*n*) {  *var* f = new *Array*(*n* + 1);      f[1] = f[2] = 1;      for(*var* i = 3; i <= *n*; i++) {          f[i] = f[i - 2] + f[i - 1];      }      return f[*n*];  }  n = 7;  console.log("斐波那契数列第" + n + "位：" + fibonacci(7)); | |
| **运行结果** | 斐波那契数列第7位：13 |

|  |  |
| --- | --- |
| **1165-1范例：阿克曼函数** | |
| *function* A(*m*, *n*) {      if(*m* == 0) {          return *n* + 1;      } else if(*m* > 0 && *n* == 0) {          return A(*m*-1, 1);      } else if(*m* > 0 && *n* > 0) {          return A(*m*-1, A(*m*, *n*-1));      }  }  console.log(A(3, 4)); | |
| **运行结果** | 125 |





|  |  |
| --- | --- |
| **timg (1)范例：汉诺塔** | |
| *var* move = 0;       // 移动次数  /\*\*   \* *@brief*  汉诺塔算法   \* *@note*   把 n 个盘子从 src 借助 mid 移到 dst   \* *@param*  n: 层数   \* *@param*  src: 起点柱子   \* *@param*  mid: 临时柱子   \* *@param*  dst: 目标柱子   \*/  *function* hanoi(*n*, *src*, *mid*, *dst*) {      if(*n* == 1) {          console.log(*n* + "号盘：" + *src* + " -> " + *dst*);          move++;      } else {          // 把前 n-1 个盘子从 src 借助 dst 移到 mid          hanoi(*n*-1, *src*, *dst*, *mid*);          // 移动第 n 个盘子          console.log(*n* + "号盘：" + *src* + " -> " + *dst*);          move++;          // 把刚才的 n-1 个盘子从 mid 借助 src 移到 dst          hanoi(*n*-1, *mid*, *src*, *dst*);      }  }  hanoi(4, 'A', 'B', 'C');  console.log("步数 ==> " + move); | |
| **运行结果** | 1号盘：A -> B  2号盘：A -> C  1号盘：B -> C  3号盘：A -> B  1号盘：C -> A  2号盘：C -> B  1号盘：A -> B  4号盘：A -> C  1号盘：B -> C  2号盘：B -> A  1号盘：C -> A  3号盘：B -> C  1号盘：A -> B  2号盘：A -> C  1号盘：B -> C  步数 ==> 15 |

**第15章 事件**

**15.1 事件的概念**

**事件**

JS创建**动态页面**时，事件是可以被JS**侦测**到的**行为**。网页中的每个元素都可以产生某些可以出发JS函数或程序的事件。例如当用户单击或者提交表单数据时，就发生一个鼠标单击事件，需要**浏览器做出处理**，返回给用户一个结果。

JS主要的事件包括：

|  |  |
| --- | --- |
| **事件** | **说明** |
| onclick | 鼠标单击事件 |
| onmouseover | 鼠标经过事件 |
| onmouseout | 鼠标移开事件 |
| onchange | 文本框内容改变事件 |
| onselect | 文本框内容被选中事件 |
| onfocus | 光标聚集 |
| onblur | 光标离开 |
| onload | 网页导入 |
| onunload | 关闭网页 |

**15.2 鼠标单击事件**

**鼠标单击事件onclick**

当在网页上**单击鼠标**时，就会发生该事件，同时**onclick**事件调用的程序块就会被执行，onclick通常与**按钮**一起使用。

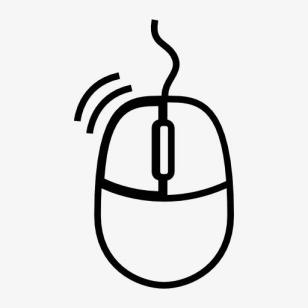


|  |
| --- |
| **范例：鼠标单击事件onclick** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>鼠标单击事件onclick</title>      <script type="text/JavaScript">  *var* cnt = 0;  *function* feedback() {              cnt++;              console.log("我被点击了"+ cnt + "次");          }      </script>  </head>  <body>      <form action="get/post">          <input type="button" value="点击" onclick="feedback();">      </form>  </body>  </html> |

**15.3 鼠标经过/移开事件**

**鼠标经过事件onmouseover**

当鼠标**移到一个对象上**时，该对象就会触发**onmouseover**事件，并执行onmouseover事件调用的程序。



|  |
| --- |
| **范例：鼠标经过事件onmouseover** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>鼠标经过事件onmouseover</title>      <script type="text/JavaScript">  *function* feedback() {              alert("卸载之前再想想吧...");          }      </script>  </head>  <body>      <form action="get/post">          <input type="button" value="卸载"              onmouseover="feedback();">      </form>  </body>  </html> |

**鼠标移开事件onmouseout**

当鼠标**移开当前对象**时，执行**onmouseout**事件调用的程序。

|  |
| --- |
| **范例：鼠标移开事件onmouseout** |
| <!DOCTYPE html>  <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>鼠标移开事件onmouseout</title>      <script type="text/JavaScript">  *function* feedback() {              alert("不要离开！只要输入密码，再点击登录就OK啦！");          }      </script>  </head>  <body>      <form action="get/post">          密码：<input type="password"                  onmouseout="feedback();">          <input type="button" value="登录">      </form>  </body>  </html> |

**15.4 光标聚焦/失焦事件**

**光标聚焦事件onfocus**

当网页中的对象**获得聚点**时，执行**onfocus**事件调用的程序。

|  |
| --- |
| **范例：光标聚焦事件onfocus** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>光标聚焦事件onfocus</title>      <script type="text/JavaScript">  *var* flag = true;  *function* feedback() {              if(flag) {                  alert("不要填错啦！");                  flag = false;              }          }      </script>  </head>  <body>      <form action="get/post">          密码：<input type="password"                  onfocus="feedback();">          <input type="button" value="登录">      </form>  </body>  </html> |

**失焦事件onblur**

onblur事件与onfocus事件是**相对事件**，当**光标离开**当前获得聚焦对象的时候，就会触发**onblur**事件，同时执行被调用的程序。

|  |
| --- |
| **范例：失焦事件onblur** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>失焦事件onblur</title>      <script type="text/JavaScript">  *function* feedback() {              alert("确定输对了再点登录哟！");          }      </script>  </head>  <body>      <form action="get/post">          密码：<input type="password"                  onblur="feedback();">          <input type="button" value="登录">      </form>  </body>  </html> |

**15.5 内容选中/改变事件**

**内容选中事件onselect**

当文本框或者文本域中的文本**被选中**时，触发**onselect**事件，同时调用的程序就会被执行。

|  |
| --- |
| **范例：内容选中事件onselect** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>内容选中事件onselect</title>      <script type="text/JavaScript">  *function* feedback() {              console.log("文本内容被选中");          }      </script>  </head>  <body>      <form action="get/post">          <textarea rows="10" cols="30" onselect="feedback();">填写个人信息，不少于20个字。</textarea>      </form>  </body>  </html> |

**内容改变事件onchange**

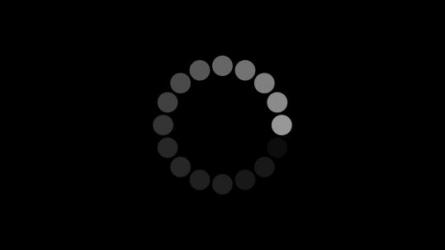
通过**改变文本框的内容**可以触发**onchange**事件，同时执行被调用的程序。

|  |
| --- |
| **范例：内容改变事件onchange** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>内容改变事件onchange</title>      <script type="text/JavaScript">  *function* feedback() {              console.log("文本内容被修改");          }      </script>  </head>  <body>      <form action="get/post">          <textarea rows="10" cols="30" onchange="feedback();">填写个人信息，不少于20个字。</textarea>      </form>  </body>  </html> |

**15.6 加载/卸载事件**

**加载事件onload**

**加载事件会在页面加载完成后立即发生**，同时执行被调用的程序。注意，**加载事件需要写在<body>标签内**。



|  |
| --- |
| **范例：加载事件onload** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>加载事件onload</title>      <script type="text/JavaScript">  *function* feedback() {              alert("页面加载完成");          }      </script>  </head>  <body onload="feedback();">      <p>Hello World!</p>  </body>  </html> |

**卸载事件onunload**

当用户**退出页面**时（页面**关闭**、页面**刷新**等），就会触发**onunload**事件，同时执行被调用的程序。注意，**不同浏览器对onunload事件的支持不同**。

**第16章 对象**

**16.1 对象的概念**

**对象(Object)**

JS中的所有事物都是**对象**，如字符串、数值、数组、函数等，每个对象带有**属性**和**方法**。对象的**属性**反映了该对象的某些**特定性质**，如字符串的长度、图像的长宽等，对象的**方法**指的是能够在对象上执行的**动作**，如表单的提交、时间的获取等。

JS提供了多个**内建对象**，如String、Date、Array等，使用对象前需要先使用**new**关键字进行**定义**。

|  |  |
| --- | --- |
| **范例：访问对象的属性** | |
| *var* arr = new *Array*(1, 2, 3, 4, 5);  console.log(arr.length); | |
| **运行结果** | 5 |

|  |  |
| --- | --- |
| **范例：访问对象的方法** | |
| *var* str = "Hello World";  console.log(str.toUpperCase()); | |
| **运行结果** | HELLO WORLD |

**16.2 Date对象**

**Date对象**

**Date对象**可以存储任意一个**日期**，并且可以精确到毫秒数（1/1000秒）。使用**默认构造函数**创建的日期对象有初始值，为**当前电脑系统时间**。



|  |  |
| --- | --- |
| **范例：定义Date对象** | |
| *var* date1 = new *Date*();  console.log(date1);  *var* date2 = new *Date*(2021, 3, 19);      //此处月份从0开始  console.log(date2); | |
| **运行结果** | 2021-03-19T05:09:47.713Z  2021-04-18T16:00:00.000Z |

Date对象中处理时间和日期的常用方法包括：

|  |  |
| --- | --- |
| **方法名称** | **功能描述** |
| getDate()/setDate() | 返回/设置日期 |
| getFullYear()/setFullYear() | 返回/设置年份，用四位数表示 |
| getYear()/setYear() | 返回/设置年份 |
| getMonth()/setMonth() | 返回/设置月份，月份从0开始 |
| getHours()/setHours() | 返回/设置小时，24小时制 |
| getMinutes()/setMinutes() | 返回/设置分钟数 |
| getSeconds()/setSeconds() | 返回/设置秒钟数 |
| getTime()/setTime() | 返回/设置时间（毫秒为单位） |
| getDay() | 返回0-6的数字表示星期，0表示星期天 |

|  |  |
| --- | --- |
| **范例：获取今日星期** | |
| *var* date = new *Date*();  *var* weekday = [      "星期天",      "星期一",      "星期二",      "星期三",      "星期四",      "星期五",      "星期六"  ];  console.log("今天是" + weekday[date.getDay()]); | |
| **运行结果** | 今天是星期五 |

**16.3 String对象**

**String对象**

定义**String对象**后就可以访问它的属性和方法。

|  |  |
| --- | --- |
| **范例：计算字符串长度** | |
| *var* str = "Hello World!"  console.log(str.length); | |
| **运行结果** | 12 |

**大小写转换**

使用String对象的**toUpperCase()**和**toLowerCase()**方法可以将字符串进行**大小写字母转换**。

|  |  |
| --- | --- |
| **范例：大小写字母转换** | |
| *var* str = "Hello World!"  console.log(str.toUpperCase());  console.log(str.toLowerCase()); | |
| **运行结果** | HELLO WORLD!  hello world! |



**返回指定的字符**

使用**charAt()**方法可**返回指定位置的字符**，返回的字符是长度为1的字符串。语法如下：

|  |
| --- |
| stringObj.charAt(index); |

字符串第一个字符的下标是0，最后一个字符的下标为length-1，如果参数index不在0与length-1之间，该方法将返回一个空字符串。

|  |  |
| --- | --- |
| **范例：返回指定位置的字符** | |
| *var* str = "Hello World!"  console.log(str.charAt(6)); | |
| **运行结果** | W |

**返回指定的字符串首次出现的位置**

**indexOf()**方法可以**返回某个指定的字符串值在字符串中首次出现的位置**。语法如下：

|  |
| --- |
| stringObj.indexOf(substring, startPos); |

该方法将从头到尾地检索字符串，检查是否含有需检索的子串。参数startPos为可选参数，用于规定开始查找的位置，如果没有设置此参数将从头开始查找。如果找到了子串，在返回子串的第一次出现位置。如果要检索的字符串值没有出现，则该方法返回-1。

|  |  |
| --- | --- |
| **范例：查找子串** | |
| *var* str = "Hello World!";  console.log(str.indexOf("Hell"));  console.log(str.indexOf("o", 6));  console.log(str.indexOf("JS")); | |
| **运行结果** | 0  7  -1 |

**字符串分隔**

**split()**方法可以**将字符串分割为字符串数组**，并返回此数组。语法如下：

|  |
| --- |
| stringObj.split(separator, limit); |

其中参数separator必选，用于指定将字符串按某个字符切割成若干个子字符串，并以数组的形式返回。参数limit可选，用于指定分隔的次数。

如果把空字符串作为separator，那么字符串的每个字符之间都会被分隔。

|  |  |
| --- | --- |
| **范例：字符串分隔** | |
| *var* str = "hello HTML hello CSS hello JavaScript";  console.log(str.split(" "));  console.log(str.split(" ", 4)); | |
| **运行结果** | ["hello", "HTML", "hello", "CSS", "hello", "JavaScript"]  ["hello", "HTML", "hello", "CSS"] |

**提取字符串**

**substring()**方法用于**提取字符串中介于两个指定下标之间的字符**。语法如下：

|  |
| --- |
| stringObj.substring(startPos, stopPos); |

该方法返回的内容是从startPos开始到stopPos-1处的所有内容，其长度为stopPos-startPos。如果startPos和stopPos相等，那么返回的就是一个空串（长度为0的字符串）。如果startPos比stopPos大，那么该方法在提取子串之间会先交换这两个参数。

|  |  |
| --- | --- |
| **范例：提取字符串** | |
| *var* str = "HelloWorld";  console.log(str.substring(2, 8));  console.log(str.substring(3, 3));  console.log(str.substring(7, 3)); | |
| **运行结果** | lloWor  loWo |

**提取指定数目的字符**

**substr()**方法用于**从字符串中提取从startPos位置开始的指定数目的字符串**。语法如下：

|  |
| --- |
| stringObj.substr(startPos, length); |

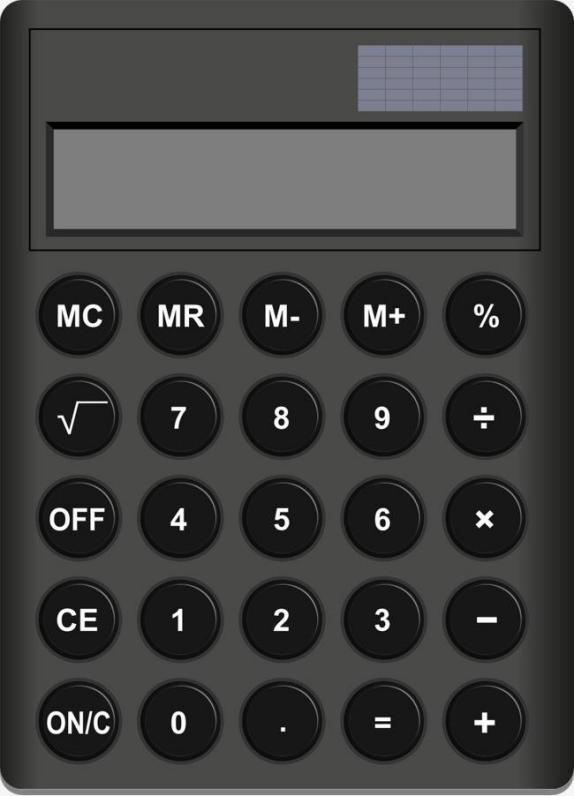
如果参数startPos是负数，则从字符串的尾部开始算起，如果startPos为负数且绝对值大于字符串长度，则startPos会被视为0。

|  |  |
| --- | --- |
| **范例：提取指定数目的字符** | |
| *var* str = "HelloWorld";  console.log(str.substr(2, 3));  console.log(str.substr(-5, 4)); | |
| **运行结果** | llo  Worl |

**16.4 Math对象**

**Math对象**

**Math对象**提供对数据的**数学计算**。需要注意的是，Math对象是一个**固有**的对象，**无需创建**它，直接把Math作为对象使用就可以调用其所有属性和方法，这是它与其它对象的区别。



Math对象的属性：

|  |  |
| --- | --- |
| **属性** | **说明** |
| E | 返回算术常量e，即自然对数的底数（约等于2.718） |
| LN2 | 返回2的自然对数（约等于0.693） |
| LN10 | 返回10的自然对数（约等于2.302） |
| LOG2E | 返回以2为底e的对数（约等于1.442） |
| LOG10E | 返回以10为底e的对数（约等于0.434） |
| PI | 返回圆周率（约等于3.14159） |
| SQRT1\_2 | 返回2的平方根的倒数（约等于0.707） |
| SQRT2 | 返回2的平方根（约等于1.414） |

Math对象的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| sin(x) | 返回x的正弦 |
| cos(x) | 返回x的余弦 |
| tan(x) | 返回x的正切 |
| acos(x) | 返回x的反余弦值 |
| asin(x) | 返回x的反正弦值 |
| atan(x) | 返回x的反正切值 |
| **ceil(x)** | **对x进行上取整** |
| **floor(x)** | **对x进行下取整** |
| **abs(x)** | **返回x的绝对数** |
| **exp(x)** | **返回e的x次幂** |
| log(x) | 返回x的自然对数（底为e） |
| **pow(x, y)** | **返回x的y次幂** |
| **max(x, y)** | **返回x和y中的最大值** |
| **min(x, y)** | **返回x和y中的最小值** |
| **round(x)** | **返回x的四舍五入最接近的整数** |
| **sqrt(x)** | **返回x的平方根** |
| **random()** | **返回0-1之间的随机数** |

**第17章 浏览器对象模型BOM**

**17.1 window对象**

**window对象**

**window**对象是**浏览器对象模型BOM(Browser Object Model)**的核心。

window对象方法有：

|  |  |
| --- | --- |
| **方法** | **描述** |
| alert() | 显示带有一段消息和一个确认按钮的警告框 |
| prompt() | 显示可提示用户输入的对话框 |
| confirm() | 显示带有一段消息以及确认按钮和取消按钮的对话框 |
| open() | 打开一个新的浏览器窗口或查找一个已命名的窗口 |
| close() | 关闭浏览器窗口 |
| print() | 打印当前窗口的内容 |
| focus() | 把键盘焦点给与一个窗口 |
| blur() | 把键盘焦点从顶层窗口移开 |
| moveBy() | 可相对窗口的当前坐标把它移动指定的像素 |
| moveTo() | 把窗口的左上角移动到一个指定的坐标 |
| resizeBy() | 按照指定的像素调整窗口的大小 |
| resizeTo() | 把窗口的大小调整到指定的宽度和高度 |
| scrollBy() | 按照指定的像素值来滚动内容 |
| scrollTo() | 把内容滚动到指定的坐标 |
| setInterval() | 每隔指定的时间执行代码 |
| setTimeout() | 在指定的延迟时间之后来执行代码 |
| clearInterval() | 取消setInterval()的设置 |
| clearTimeout() | 取消setTimeout()的设置 |

**17.2 计时器**

**计时器**

在JS中，可以在设定的时间间隔之后执行代码，而不是在函数被调用后立即执行。

计时器的类型分为2种：

1. **一次性计时器：仅在指定的延迟时间之后触发一次**。

2. **间隔性触发计时器：每隔一定的时间间隔就触发一次**。

计时器的方法有：

|  |  |
| --- | --- |
| **方法** | **描述** |
| **setTimeout()** | **在指定的延迟时间之后来执行代码** |
| **clearTimeout()** | **取消setTimeout()的设置** |
| **setInterval()** | **每隔指定的时间执行代码** |
| **clearInterval()** | **取消setInterval()的设置** |

**setTimeout()**

**setTimeout()**计时器，在**载入后延迟指定时间**后，去执行一次表达式，**仅执行一次**。语法如下：

|  |
| --- |
| setTimeout(代码, 延迟时间); |

其中，第一个参数为要调用的函数或要执行的代码串，第二个参数表示在执行代码前需等待的时间，以**毫秒**为单位（**1s = 1000ms**）。

|  |
| --- |
| **范例：setTimeout()计时器** |
| //网页打开2秒后弹出提示框  setTimeout("alert('Welcome')", 2000); |

clearTimeout()和setTimeout()一起使用，用于**停止计时器**。语法如下：

|  |
| --- |
| clearTimeout(id\_of\_setTimeout); |

**参数id\_of\_setTimeout是setTimeout()返回的ID值**，该值标识要取消的延迟执行代码块。

|  |
| --- |
| **范例：计数器** |
| **counter.html** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>计数器</title>      <script src="counter.js"></script>  </head>  <body>      <form action="get/post">          <input type="text" id="num">          <input type="button" value="stop" onclick="stopCount();">      </form>  </body>  </html> |
| **counter.js** |
| *var* cnt = 0;        // 计数  *var* counter;        // 计数器  /\*\*   \* 每隔1000毫秒计数加1   \*/  *function* count() {      document.getElementById('num').value = cnt;      cnt++;      counter = setTimeout(count, 1000);  }  /\*\*   \* 停止计数器   \*/  *function* stopCount() {      clearTimeout(counter);  }  setTimeout(count, 1000);        // 启动计数器 |

**setInterval()**

**setInterval()**在执行时，**从载入页面后每隔指定的时间执行代码**。语法如下：

|  |
| --- |
| setInterval(代码, 交互时间); |

其中，第一个参数表示要代用的函数或要执行的代码串，第二个参数表示周期性执行或调用表达式之间的时间间隔，以**毫秒**为单位（**1s = 1000ms**）。

**clearInterval()**方法可**取消由setInterval()设置的交互时间**。语法如下：

|  |
| --- |
| clearInterval(id\_of\_setInterval); |

**参数id\_of\_setInterval是setInterval()返回的ID值**。

|  |
| --- |
| **范例：实时显示当前时间** |
| **current\_time.html** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>显示当前时间</title>      <script src="current\_time.js"></script>  </head>  <body>      <form action="get/post">          <input type="text" id="time" size="50">      </form>  </body>  </html> |
| **current\_time.js** |
| // function clock() {  //     var date = new Date();  //     document.getElementById("time").value = date;  // }  // setInterval(clock, 1000);  // 箭头函数  setInterval(() *=>* {  *var* date = new *Date*();      document.getElementById("time").value = date;  }, 1000); |

**17.3 Screen对象**

**Screen对象**

Screen对象用于获取用户的**屏幕信息**。window.screen对象在编写时**可以不使用**window前缀。语法如下：

|  |
| --- |
| window.screen.属性; |

|  |  |
| --- | --- |
| **属性** | **描述** |
| availHeight | 窗口可以使用的屏幕高度，单位像素 |
| availWidth | 窗口可以使用的屏幕宽度，单位像素 |
| colorDepth | 用户浏览器表示的颜色位数，通常为32位（每像素的位数） |
| pixelDepth | 用户浏览器表示的颜色位数，通常为32位（每像素的位数），IE不支持此属性 |
| height | 屏幕的高度，单位像素 |
| width | 屏幕的宽度，单位像素 |

**screen.availWidth**和**screen.availHeight**属性返回访问者屏幕的宽度和高度，单位为像素，**减去界面特性**，比如**任务栏**等。不同系统的任务栏**默认高度**不一样，及任务栏的位置可在屏幕上下左右任何位置，所以有可能可用宽度和高度不一样。

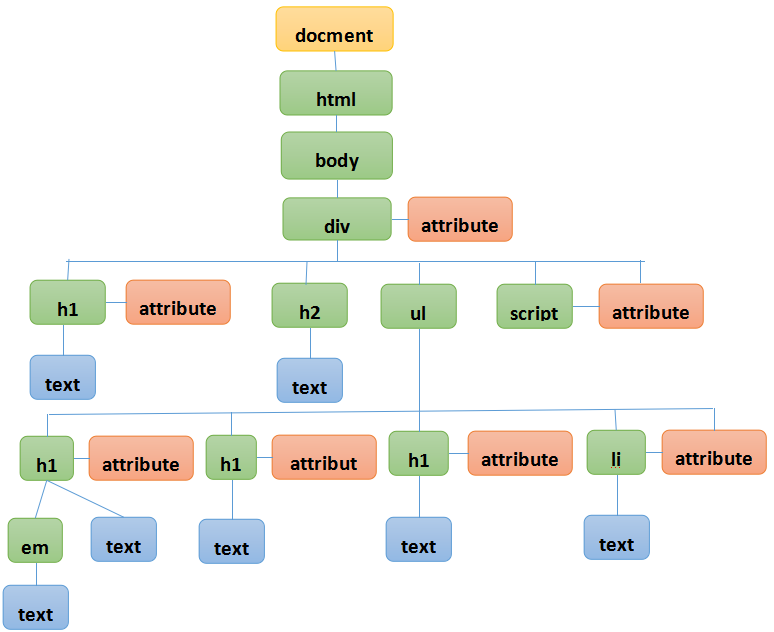
|  |  |
| --- | --- |
| **范例：屏幕信息** | |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>屏幕信息</title>      <script type="text/JavaScript">          console.log("屏幕分辨率：" + screen.width + "\*" + screen.height)          console.log("屏幕可用宽高：" + screen.availWidth + "\*" + screen.availHeight);      </script>  </head>  <body>    </body>  </html> | |
| **运行结果** | 屏幕分辨率：1536\*864  屏幕可用宽高：1536\*824 |

**第18章 文档对象模型DOM**

**18.1 DOM的概念**

**DOM (Document Object Model)**

**文档对象模型DOM**定义访问和处理HTML文档的标准方法，DOM将HTML文档呈现为带有元素、属性和文本的**树结构**（结点树）。



**18.2 获取结点对象**

**getElementById()**

**getElementById()**方法可返回拥有**指定ID的第一个对象的引用**。语法如下：

|  |
| --- |
| document.getElementById(elementID); |

如果没有指定ID的元素则返回null，如果存在多个指定ID的元素则返回第一个。

|  |
| --- |
| **范例：定时变换颜色** |
| **random\_color.html** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>随机颜色</title>      <script src="random\_color.js"></script>  </head>  <body>      <div id="square" style="width: 100px; height: 100px"></div>  </body>  </html> |
| **random\_color.js** |
| /\*\*   \* 随机生成RGB颜色代码   \* *@returns* rgb颜色   \*/  *function* randomRGB() {  *var* r = Math.floor(Math.random() \* 256);  *var* g = Math.floor(Math.random() \* 256);  *var* b = Math.floor(Math.random() \* 256);      return "rgb(" + r + ", " + g + ", " + b + ")";  }  /\*\*   \* 获取元素结点，设置背景颜色   \*/  *function* changeColor() {  *var* obj = document.getElementById("square");      obj.style.background = randomRGB();  }  // 每隔300ms改变颜色  setInterval(*function* () {      changeColor();  }, 300); |

**getElementsByClassName()**

**getElementByClassName()**方法返回文档中**所有指定类名**的元素集合，作为NodeList对象。NodeList对象代表一个有顺序的结点列表，可以通过索引来访问列表中的结点。使用NodeList的length属性可以确定指定类名的元素个数，并循环各个元素来获取某个元素。语法如下：

|  |
| --- |
| document.getElementsByClassName(className); |

**getElementByName()**

**getElementByName()**方法返回带有**指定名称**的结点对象的集合。语法如下：

|  |
| --- |
| document.getElementsByName(name); |

与getElementById()方法不同的是，通过元素的name属性查询元素，而不是通过id属性。文档中的name属性可能不唯一，所以getElementByName()方法返回的是元素的数组，而不是一个元素。

**getElementsByTagName()**

**getElementsByTagName()**方法返回带有**指定标签名**的结点对象的集合，返回元素的顺序是它们在文档中的顺序。语法如下：

|  |
| --- |
| document.getElementsByTagName(tagName); |

**18.3 结点属性**

**结点属性**

**getAttribute()**方法可以通过元素结点的属性名称获取**属性的值**。语法如下：

|  |
| --- |
| elementNode.getAttribute(name); |

其中，elementNode可以使用getElementById()、getElementsByTagName()等方法获取到元素结点，参数name为需要查询的元素结点的属性名称。

**setAttribute()**方法可以**增加**一个指定名称和值的新属性，或者把一个**现有**的属性设定为指定的值。语法如下：

|  |
| --- |
| elementNode.setAttribute(name, value); |

其中，name表示要设置的属性名，value表示要设置的属性值。

|  |  |
| --- | --- |
| **范例：设置结点属性值** | |
| **getAttribute.html** | |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>设置结点属性</title>      <script src="getAttribute.js"></script>  </head>  <body>      <a class="link" href="https://www.baidu.com">百度</a>      <a class="link" href="https://www.bilibili.com">哔哩哔哩</a>  </body>  </html> | |
| **getAttribute.js** | |
| window.onload = *function*() {  *var* links = document.getElementsByClassName("link");      for(*var* i = 0; i < links.length; i++) {          console.log(links[i].getAttribute("href"));          links[i].setAttribute("target", "\_blank");      }  }; | |
| **运行结果** | https://www.baidu.com  https://www.bilibili.com |

**18.4 结点操作**

**创建结点createElement()**

**createElement()**方法可以**创建**结点元素，此方法可返回一个Element对象。语法如下：

|  |
| --- |
| document.createElement(tagName); |

其中参数tagName为字符串值，用来指明创建元素的类型。createElement()方法要与appendChild()或insertBefore()方法联合使用，将元素显示在页面中。

**插入结点appendChild()**

**appendChild()**方法可以在指定结点的最后一个子结点列表之后**添加**一个新的子结点。语法如下：

|  |
| --- |
| elementNode.appendChild(newNode); |

**插入结点insertBefore()**

**insertBefore()**方法可以在已有的子结点前**插入**一个新的子结点。语法如下：

|  |
| --- |
| elementNode.insertBefore(newNode, node); |

**删除结点removeChild()**

**removeChild()**方法可以从子结点列表中**删除**某个结点，如删除成功，此方法可返回被删除的结点，如失败则返回null。语法如下：

|  |
| --- |
| elementNode.removeChild(node); |

**替换结点replaceChild()**

**replaceChild()**方法可以实现子结点的**替换**，返回被替换对象的引用。语法如下：

|  |
| --- |
| elementNode.replaceChild(newNode, oldNode); |

当oldNode被替换时，所有与之相关的属性内容都将被移出。