使用**NodeList**

理解 NodeList 及其“近亲”NamedNodeMap 和HTMLCollection，是从整体上透彻理解DOM的

关键所在。这三个集合都是“动态的”；换句话说，每当文档结构发生变化时，它们都会得到更新。因

此，它们始终都会保存着最新、最准确的信息。从本质上说，所有NodeList 对象都是在访问DOM文

档时实时运行的查询。例如，下列代码会导致无限循环：

var divs = document.getElementsByTagName("div"),

i,

div;

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

div = document.createElement("div");

document.body.appendChild(div);

}

第一行代码会取得文档中所有<div>元素的HTMLCollection。由于这个集合是“动态的”，因此

只要有新<div>元素被添加到页面中，这个元素也会被添加到该集合中。浏览器不会将创建的所有集

合都保存在一个列表中，而是在下一次访问集合时再更新集合。结果，在遇到上例中所示的循环代码

时，就会导致一个有趣的问题。每次循环都要对条件i < divs.length 求值，意味着会运行取得所

有<div>元素的查询。考虑到循环体每次都会创建一个新<div>元素并将其添加到文档中，因此

divs.length 的值在每次循环后都会递增。既然i 和divs.length 每次都会同时递增，结果它们的

值永远也不会相等。

如果想要迭代一个NodeList，最好是使用length 属性初始化第二个变量，然后将迭代器与该变

量进行比较，如下面的例子所示：

var divs = document.getElementsByTagName("div"),

i,

len,

div;

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

div = document.createElement("div");

document.body.appendChild(div);

}

这个例子中初始化了第二个变量len。由于len 中保存着对divs.length 在循环开始时的一个快

照，因此就会避免上一个例子中出现的无限循环问题。在本章演示迭代NodeList 对象的例子中，使用

的都是这种更为保险的方式。

一般来说，应该尽量减少访问NodeList 的次数。因为每次访问NodeList，都会运行一次基于文

档的查询。所以，可以考虑将从NodeList 中取得的值缓存起来。