第 4 章 JavaScript 基于对象编程

JavaScript 脚本是基于对象（Object-based）的编程语言，通过对象的组织层次来访问并给对象施以相应的操作方法，可大大简化 JavaScript 程序的设计，并提供直观、模块化的方式进行脚本程序开发。本章主要介绍 JavaScript 的基于对象编程、DOM 的模型层次以及有关对象的基本概念等，并引导读者创建和使用自定义的对象。

4.1 面向对象编程与基于对象编程

在软件编程术语中，存在两个类似的概念：面对对象编程（Object Oriented Programming：OPP）和基于对象编程（Object-based Programming），它们在对象创建、对象组织层次、代码封装和复用等方面存在较大的差异。

在了解它们之间差异之前，先来了解对象的概念。

4.1.1 什么是对象

对象是客观世界存在的人、事和物体等实体。现实生活中存在很多的对象，比如猫、自行车等。不难发现它们有两个共同特征：都有状态和行为。比如猫有自己的状态（名字、颜色、饥饿与否等）和行为（爬树、抓老鼠等）。自行车也有自己的状态（档位、速度等）和行为（刹车、加速、减速、改变档位等）。若以自然人为例，构造一个对象，可以用图 4.1 来表示，其中 Attribute 表示对象状态，Method 表示对象行为。

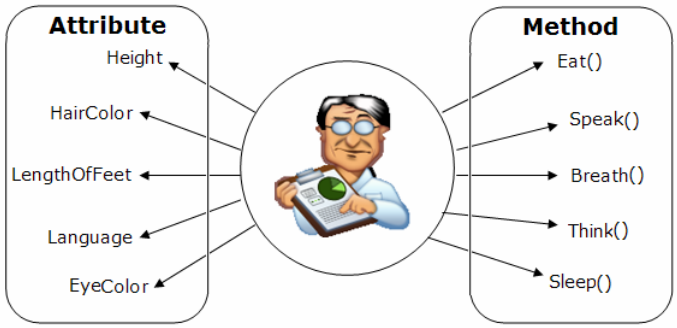

图 4.1 以自然人构造的对象

在软件世界也存在对象，可定义为相关变量和方法的软件集。主要由两部分组成：

- 一组包含各种类型数据的属性
- 允许对属性中的数据进行操作且有相关方法

以 HTML 文档中的 document 作为一个对象，如图 4.2 所示。

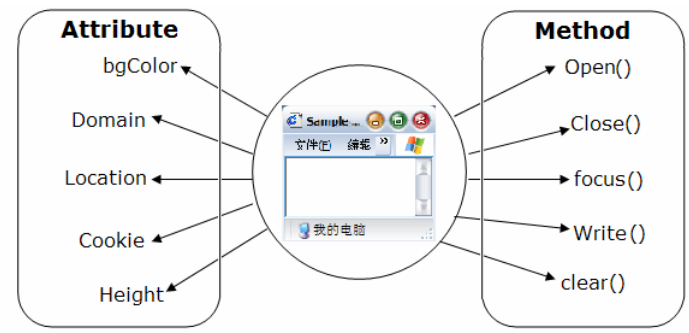

图 4.2 以 HTML 文档中的 document 构造的对象

综上所述，凡是能够提取一定度量数据并能通过某种途径对度量数据实施操作的客观存在都可以构成一个对象，且用属性来描述对象的状态，使用方法和事件来处理对象的各种行为。

- 属性：用来描述对象的状态。通过定义属性值，可以改变对象的状态。如图 4.1 中，可以定义字符串 HungryOrNot 来表示该自然人肚子的状态，HungryOrNot 成为自然人的某个属性；
- 方法：由于对象行为的复杂性，对象的某些行为可用通用的代码来处理，这些通用的代码称为方法。如图 4.1 中，可以定义方法 Eat( )来处理自然人肚子很饿的情况，Eat( )成为自然人的某个方法；
- 事件：由于对象行为的复杂性，对象的某些行为不能使用通用的代码来处理，需要用户根据实际情况爱编写处理该行为的代码，该代码称为事件。在图 4.1 中，可以定义事件 DrinkBeforeEat( )来处理自然人肚子很饿同时嘴巴很渴需要先喝水后进食的情况。

了解了什么是对象，下面来看看什么是面向对象编程。

4.1.2 面向对象编程

面向对象编程（OPP）是一种计算机编程架构，其基本原则：计算机程序由单个能够起到子程序作用的单元或对象组合而成。具有三个最基本的特点：重用性、灵活性和扩展性。这种方法将软件程序的每个元素构成对象，同时对象的类型、属性和描述对象的方法。为了实现整体操作，每个对象都能够接收信息、处理数据和向其它对象发送信息。

面向对象编程主要包含有以下重要的概念：

1. 继承

允许在现存的组件基础上创建子组件，典型地说就是用类来对组件进行分组，而且还可以定义新类（子类）为现存的类（父类）的扩展，子类继承了父类的全部属性、方法和事件而不必重新定义；同时通过扩展，子类可以获得专属自己的属性、方法和事件（不影响父类的属性、方法和事件等），这样就可以将所有类拓扑成树形或网状结构。以动物“虎”类为例，拓扑成的树状结构如图 4.3 所示。

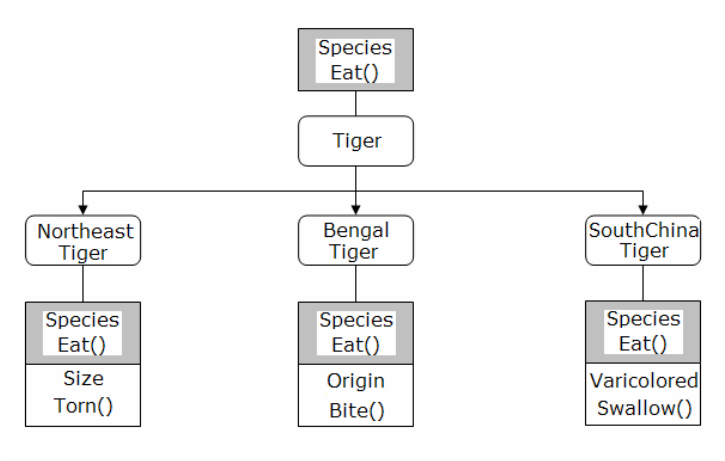

图 4.3 通过继承形成的树形结构

其中灰色框内为“虎”科共有的属性和方法，在生成子类的同时被子类继承，白色长方形框内的为经子类扩展的而特有的属性和方法，同时子类对父类的扩展并不影响父类的任何属性、方法和事件。

2. 封装

封装就是将对象的实现过程通过函数等方式封装起来，使用户只能通过对象提供的属性、方法和事件等接口去访问对象，而不需要知道对象的具体实现过程。封装允许对象运行的代码相对于调用者来说是完全独立的，调用者通过对象及相关接口参数来访问此接口。只要对象的接口不变，而只是对象的内部结构或实现方法发生了改变，程序的其他部分不用作任何处理。模拟吃饭的过程，例如有对象 Tom 及其属性 HungryOrNot、ThirstyOrNot 和方法 DrinkWater(CupNumber)、Eat(GramNumber)，下面的类 C 代码演示何为代码的封装：

```
If(Tom.HungryOrNot==YES)
{
    if(Tom.ThirstyOrNot==YES)
    {
       Tom.DrinkWater(1);
    }
    Tom.Eat(1);
}
```

在操作对象的过程中，用户并不需要知道 DrinkWater(CupNumber)、Eat(GramNumber)方法的具体实现过程，只需知道对象的接口，然后传递相应的参数即可操作对象，实现了对象的封装。

3. 多态

多态指一种对象类型定义多种实现的方案，具体实现的方案由使用的环境来决定。这样就形成了可复用的代码和标准化的程序。广义上讲，多态指一段程序能够处理多种类型对象的能力。仍然模拟吃饭的过程，例如有对象 Tom 及其属性 HungryOrNot、ThirstyOrNot 和方法 DrinkWater(CupNumber)、Drink Milk (CupNumber)、Eat(GramNumber)、FinalEat(FoodType)，下面的类 C 代码演示何为多态性：

```
::FinalEat(FoodType)
{
   if(Tom.ThirstyOrNot==YES)
   {
```

```
        if(FoodType==Rice)
            Tom.DrinkWater(1);
        else if(FoodType==Bread)
            Tom.DrinkMilk(1);
        else
            exit(1);
    }
    Tom.Eat(1);
}
```

Tom根据调用Tom.FinalEat(FoodType)使用的参数FoodType是米饭还是面包等其它东西决定下一步的动作，实现了多态性。

4.1.3 基于对象编程

定位 JavaScript 脚本为基于对象的脚本编程语言而不是面向对象的编程语言，是因为JavaScript 以 DOM 和 BOM 中定义的对象模型及操作方法为基础，但又不具备面向对象编程语言所必具备的显著特征如分类、继承、封装、多态、重载等，只能通过嵌入其他面向对象编程语言如 Java 生成的 Java applet 组件等实现 Web 应用程序的功能。

JavaScript 支持 DOM 和 BOM 提供的对象模型，用于根据其对象模型层次结构来访问目标对象的属性并施加对象以相应的操作。

在支持对象模型预定义对象的同时，JavaScript 支持 Web 应用程序开发者定义全新的对象类型，并通过操作符 new 来生成该对象类型的实例。但不支持强制的数据类型，任何类型的对象都可以赋予任意类型的数值。

> 注意：在 JavaScript 语言中，之所以任何类型的对象都可以赋予任意类型的数值，是因为 JavaScript 为弱类型的脚本语言，即变量在使用前不需作任何声明，在浏览器解释运行其代码时才检查目标变量的数据类型。

下面简要介绍 HTML 文档的结构和文档对象模型（DOM）。

4.2 JavaScript 对象的生成

JavaScript 是基于对象的编程语言，除循环和关系运算符等语言构造之外，其所有的特征几乎都是按照对象的处理方法进行的。JavaScript 支持的对象主要包括：

- JavaScript 核心对象：包括同基本数据类型相关的对象（如 String、Boolean、Number）、允许创建用户自定义和组合类型的对象（如 Object、Array）和其他能简化 JavaScript 操作的对象（如 Math、Date、RegExp、Function）。该部分内容将在第 6 章重点叙述。
- 浏览器对象：包括不属于 JavaScript 语言本身但被绝大多数浏览器所支持的对象，如控制浏览器窗口和用户交互界面的 window 对象、提供客户端浏览器配置信息的 Navigator 对象。该部分内容将在第 7 章重点叙述。
- 用户自定义对象：由 Web 应用程序开发者用于完成特定任务而创建的自定义对象，可自由设计对象的属性、方法和事件处理程序，编程灵活性较大。该部分内容将在本章后续小节叙述。
- 文本对象：由文本域构成的对象，在 DOM 中定义，同时赋予很多特定的处理方法，

如 insertData( )、 appendData( )等。该部分内容将在第 5 章详细叙述。

注意：ECMA-262 标准只规定 JavaScript 语言基本构成，而 W3C DOM 规范只规定了文档对象模型的访问层次及如何在 JavaScript 脚本中实现，浏览器厂商只定义用户界面并扩展 DOM 层次，基于以上原因，上述对象分类方法可能导致重叠现象出现。

本章主要初步叙述 DOM 框架，而把重点放在如何创建和使用用户自定义的对象上。首先来了解 HTML 文档的结构。

4.2.1 HTML 文档结构

在 HTML 文档中，其标记如<body>与</body>、 与 等都是成对出现的，称为标记对。文档内容通过这些成对出现的标记对嵌入到文档中，与 JavaScript 脚本等其他代码一起构成一个完整的 HTML 文档。观察如下的简单文档：

```
//源程序 4.1
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
   <meta http-equiv=content-type content="text/html; charset=gb2312">
   <title>Frist Page!</title>
</head>
<body>
   <p>DOM</p>
</body>
</html>
```

可绘制成图 4.4 所示的 HTML 元素层次结构图。

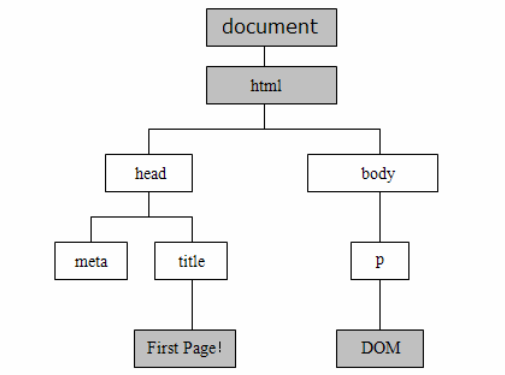

图 4.4 实例的 HTML 元素层次结构图

载入文档后，document 元素相对于该文档而言是唯一的，访问该层次结构图中任何元素都以 document 为根元素进行访问。同时<html>标记元素是<head>标记元素和<body>标记元素的父元素，同时又是 document 元素的子元素。可见如果 HTML 文档中严格使用 HTML 成对标记，则其元素是相互嵌套的，并且通过相互之间在结构图中的层次结构关系可实现相互定位，为 DOM 框架提供了理论基础。

注意：这里所说的 document 元素、<html>标记元素等，只是在单纯的 HTML 文档背景下定义的；在后面讲到的 DOM 框架中，元素又被称为对象；在浏览器载入该文档并根据 DOM 模型生成的节点

树中，元素又被称为节点。其实它们代表同一事物，只是定义的背景不同而已。

4.2.2 DOM 框架

由 HTML 文档结构可知，文档中各个元素（标记元素、文本元素等）在 HTML 元素层次结构图中都被标记为具有一定“社会”关系的成员，并可通过这种关系来访问指定的成员，那我们是不是可以设定这种结构模型的某种标准，以便实现元素访问方法的一致性呢？

DOM（文档结构模型）应运而生，其主要关注在浏览器解释 HTML 文档时如何设定各元素的这种“社会”关系及处理这种关系的方法。从实际应用的角度出发，HTML 文档根据 DOM 中定义的框架模型在浏览器解释后生成对象访问层次，而 JavaScript 脚本经常要控制其中的某个对象。DOM 基本框架如图 4.5 所示，其中灰色代表模型中的顶级对象，包括 window 对象及其下的 frames、location、document、history、navigator、screen 等对象：

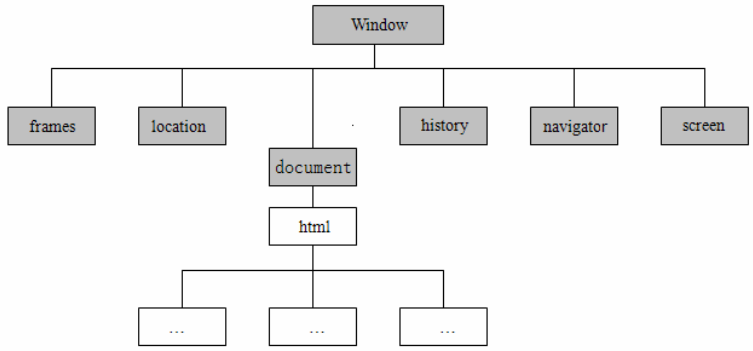

图 4.5 DOM 框架结构示意图

DOM 中几个顶级对象及其作用如表 4.1 所示。

表 4.1 DOM中的顶级对象及其作用

对象名称	作用
window	表示与当前浏览器窗口相关的最顶级对象，包含当前窗口的最小最大化、尺寸大小等信息同时具有关闭、新建窗口等方法。
frames[]	表示Window页面中的框架数组对象，每个框架都包含一个window对象
location	以URL的形式载入当前窗口，并保存正在浏览的文档的位置及其构成，如协议、主机名、端口、路径、URL的查询字符串部分等
document	包含HTML文档中的HTML标记和构成文档内容的文本的对象，客户端浏览器中每个载入的HTML文档都有一个document对象，在多框架文档中，框架集的每个成员都包含一个document对象，按照对象包含的层次进行访问
history	包含当前窗口的历史列表对象，用于跟踪窗口中曾经使用过的URL，包括其历史表的长度、历史表中上一个URL和下一个URL等信息。
navigator	包含当前浏览器的相关信息的对象，包括处理当前文档的客户端浏览器的版本号、商标等只读信息，防止脚本对客户端浏览器相关信息的恶意访问和篡改。
screen	包含当前浏览器运行的物理环境信息的对象，包含如监视器的有效像数等信息。

对 DOM 框架层次及其相关顶级对象的了解有助于更好理解浏览器载入 HTML 文档时 JavaScript 对象的生成过程。后面的“文档对象模型(DOM)”章节将详细讨论 DOM。

4.2.3 顶级对象之间的关系

我们来模拟浏览器载入某标准 HTML 的过程来阐述 window、frames[]、location 等几种常见的顶级对象之间的关系。首先参考如下包含框架集的标准 HTML 文档：

```
<html>
<head>
</head>
<body>
  <frameset>
    <frame src="a.html">
    <frame src="b.html">
  </frameset>
</body>
</html>
```

将上述代码保存为 Sample.html，鼠标双击，系统调用默认的浏览器，生成 window 对象，打开 Sample.html 文档后，生成 screen、navigator、location、history、frames[]和 document 等对象。

- window 对象包含当前浏览器窗口中的所有对象，为对象访问过程忠默认的顶级对象，如引用该对象的 alert( )方法，可将 window.alert(msg)直接改写为 alert(msg)，同样 window.document.forms[1]可改写为 document.forms[1]；
- screen 对象包含当前浏览器运行的物理环境信息，如当前屏幕分辨率；
- navigator 对象包含当前浏览器的相关信息，如浏览器版本等；
- location 对象以 URL 形式保存正在浏览的文档相关信息，如路径等；
- history 对象包含浏览器当前窗口的访问历史列表，如单击链接进入新页面，则原始页面地址列入当前窗口的访问历史列表中；
- frames[]对象包含当前 Window 页面中的框架数组成员，如实例中的两个框架，每个框架都包含一个独立的 document 对象；
- document 对象包含 HTML 文档中的 HTML 标记和构成文档内容的文本的对象，在每个单独保存的 HTML 文档中都直接包含一个 document 对象。

由上面的分析，可看出从浏览器打开文档至关闭文档期间，screen 和 navigator 对象不变（用户不改变其硬件和软件设置），且与文档无关；location 对象代表当前文档位置的相关信息，与文档地址（相对地址或绝对位置）及访问方法相关；history 对象则是当前文档页面的访问列表，与文档中链接及页面跳转情况有关；frames[]和 document 对象则是浏览器载入时根据文档结构生成的对象，决定于文档。

理解这几个顶级对象的关系有助于深入了解浏览器解释运行 HTML 文档过程中各对象的生成步骤和相互之间如何影响彼此。

4.2.4 浏览器载入文档时对象的生成

浏览器载入 HTML 文档时，根据 DOM 定义的结构模型层次，当遇到自身支持的 HTML 元素对象所对应的标记时，就按 HTML 文档载入的顺序在客户端内存中创建这些对象，并按对象创建的顺序生成对象数组，而不管 JavaScript 脚本是否真正运行这些对象。对象创建后，浏览器为这些对象提供专供 JavaScript 脚本使用的可选属性、方法和处理程序，Web 应用程序开发者通过这些属性、方法和处理程序就能动态操作 HTML 文档内容。

下面的实例说明客户端浏览器载入 HTML 文档时将按载入时对象创建的顺序生成对象

数组：

```
//源程序 4.2
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
   <title>Sample Page!</title>
   <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
<h5>Test!</h5>
<!--NOTE!-->
<p>
     Welcome to
     <em>
          DOM
     </em>
     World!
</p>
<ul>
   <li>
      Newer
   </li>
</ul>
<hr>
<br>
<script language="JavaScript" type="text/javascript">
<!-- //在支持 JavaScript 脚本的浏览器将忽略该标记
   var i,origlength,msg;
   //获取生成的对象数组长度
   origlength=document.all.length;
   msg="对象数组长度: "+origlength+"\n";
   //循环输出各节点的类型和 tagName 属性值
   for(i=0;i<origlength;i++)
   {
      if(i<10)
      {
         msg+="类型:"+typeof(document.all[i])+"编号:    "_
               +i+"名称:"+document.all[i].tagName+"\n";
      }
      else
      {
         msg+="类型:"+typeof(document.all[i])+"编号:"_
               +i+"名称:"+document.all[i].tagName+"\n";
      }
   }
   //使用警示框输出信息
   window.alert(msg);
-->       //在支持 JavaScript 脚本的浏览器将忽略该标记
</script>
</body>
</html>
```

运行上面的程序，结果如图 4.6 所示。

图 4.6 HTML 文档载入时生成的对象数组顺序图

分析上述代码，浏览器载入该文档时生成对象的顺序应为<! DOCTYPE>、<HTML>、<HEAD>、<TITLE>、<META>、<BODY>、<H5>、<!>、<P>、<EM>、<UL>、<LI>、< HR >、
及<SCRIPT>。图 4.6 表明两点：

- 浏览器载入文档时，根据当前浏览器支持的 DOM 规范级别生成对应于 HTML 标记的对象（object）；
- 浏览器根据其将各标记载入时的先后顺序生成对象数组的顺序。

对象生成后，浏览器将调用与对象相对应的属性、方法和事件供 JavaScript 脚本根据用户动作和页面动作进行相关处理。在本书的后续章节将详细讲解 HTML 通用元素对象，下面简要介绍 JavaScript 脚本中的核心对象。

4.3 JavaScript 核心对象

JavaScript 作为一门基于对象的编程语言，以其简单、快捷的对象操作获得 Web 应用程序开发者的首肯，而其内置的几个核心对象，则构成了 JavaScript 脚本语言的基础。主要核心对象如表 4.2 所示。

表 4.2 JavaScript核心对象

核心对象	附加说明
Array	提供一个数组模型，用来存储大量有序的类型相同或相似的数据，将同类的数据组织在一起进行相关操作。
Boolean	对应于原始逻辑数据类型，其所有属性和方法继承自Object对象。当值为真表示true，值为假则表示false。
Date	提供了操作日期和时间的方法，可以表示从微秒到年的所有时间和日期。使用Date读取日期和时间时，其结果依赖于客户端的时钟。
Function	提供构造新函数的模板，JavaScript中构造的函数是Function对象的一个实例，通过函数名实现对该对象的引用。
Math	内置的Math对象可以用来处理各种数学运算，且定义了一些常用的数学常数，如Math对象的实例的PI属性返回圆周率π的值。各种运算被定义为Math对象的内置方法，可直接调用。
Number	对应于原始数据类型的内置对象，对象的实例返回某数值类型。
Object	包含由所有 JavaScript 对象所共享的基本功能，并提供生成其它对象如Boolean等对象的模板和基本操作方法。
RegExp	表述了一个正则表达式对象，包含了由所有正则表达式对象共享的静态属性，用于指定字符或字符串的模式。

String	和原始的字符串类型相对应，包含多种方法实现字符串操作如字符串检查、抽取子串、连接两个字符串甚至将字符串标记为HTML等。

JavaScript 语言中，每种基本类型都构成了一个 JavaScript 核心对象，并由 JavaScript 提供其属性和方法，Web 应用程序开发者可以通过操作对象的方法来操作该基本类型的实例。

4.4 文档对象的引用

客户端浏览器载入 HTML 文档时，对于所有可以编码的 HTML 元素按照 DOM 规范和载入元素的顺序生成对象数组，然后进行初始化供 JavaScript 脚本使用。下面讨论 JavaScript 脚本访问文档对象的方法。

4.4.1 通过对象位置访问文档对象

浏览器载入 HTML 文档后，将根据该文档的结构和 DOM 规范生成对象数组，该对象数组中各对象之间的相对位置随着 HTML 文档的确定而确定下来，JavaScript 脚本可以通过这个确定的相对位置来访问该对象。考察如下的 HTML 文档：

```
//源程序 4.3
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
   <title>Sample Page!</title>
   <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
<form>
   <input type=text value="Text in Form1">
</form>
<form>
   <input type=text value="Text1 of Form2">
   <input type=text value="Text2 of Form2">
</form>
<script language="JavaScript" type="text/javascript">
<!--
   var msg="";
   msg+="通过位置访问文档对象:\n\n";
   msg+="Form[0].element[0].value: "+document.forms[0].elements[0].value+"\n\n";
   msg+="Form[1].element[0].value: "+document.forms[1].elements[0].value+"\n\n";
   msg+="Form[1].element[1].value: "+document.forms[1].elements[1].value+"\n\n";
   //使用警示框输出信息
   window.alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.7 所示。

图 4.7 通过位置访问文档对象

浏览器载入该文档时，生成 forms[]数组，第一个 form 为 form[0]，第二个 form 为 form[1]：

```
<form>
   <input type=text value="Text in Form1">
</form>
<form>
   <input type=text value="Text1 of Form2">
   <input type=text value="Text2 of Form2">
</form>
```

则通过它们的位置进行访问的方法如下：

```
document.forms[0]
document.forms[1]
```

而 form[1]下面还有两个文本框，可通过如下方式访问：

```
document.forms[1].elements[0]
document.forms[1].elements[1]
```

则访问第二个 form 里面的第二个 text 的 value 属性可通过如下的方法：

```
document.forms[1].elements[1].value
```

此种方法简单明了，但对象的位置依赖于 HTML 文档的结构，如果文档结构改变而不改变上述的访问代码，浏览器弹出“某某对象为空或无此对象”错误信息。

4.4.2 通过对象名称访问文档对象

上述通过位置访问文档对象的方法由于对象对文档结构的依赖性太大，一旦文档结构改变，就必须改变对象访问的语句，这给脚本代码维护带来了很大的难度，可以通过给对象命名的方法来解决。

1. 通过 name 属性访问文档对象

在 HTML 4 版本之前，使用元素的 name 属性来给 HTML 文档中的元素进行标注，支持 name 属性的标记有：<form>、<input>、<button>、<select>、<text>、<textarea>、<a>、<applet>、<embeds>、<frame>、<iframe>、<img>、<object>、<map>等。考察如下的 HTML 文档：

```
//源程序 4.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
   <title>Sample Page!</title>
   <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
```

```
//第一个表单
<form name="MyForm1">
   <input type=text name="MyTextOfForm1" value="Text in Form1">
</form>
//第二个表单
<form name="MyForm2">
   <input type=text name="MyText1OfForm2" value="Text1 of Form2">
   <input type=text name="MyText2OfForm2" value="Text2 of Form2">
</form>
<script language="JavaScript" type="text/javascript">
<!--
   var msg="";
   msg+="通过名称访问文档对象:\n\n";
   msg+="MyForm1.MyTextOfForm1.value:"+document.MyForm1.MyTextOfForm1.value+" \n\n";
   msg+="MyForm2.MyText1OfForm2.value:"+document.MyForm2.MyText1OfForm2.value+" \n\n";
   msg+="MyForm2.MyText2OfForm2.value:"+document.MyForm2.MyText2OfForm2.value+" \n\n";
   //使用警示框输出信息
   window.alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.8 所示。

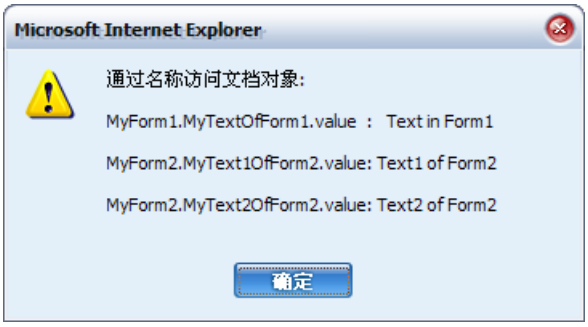

图 4.8 通过 name 属性访问文档对象

程序首先设定第一个表单的 name 属性为 MyForm1，包含在它里面的文本框的 name 属性为 MyTextOfForm1，第二个表单的 name 属性为 MyForm1，包含在它里面的两个文本框的 name 属性分别为：MyText1OfForm2 和 MyText2OfForm2。则通过如下的代码访问三个文本框的 value 属性：

```
document.MyForm1.MyTextOfForm1.value
document.MyForm2.MyText1OfForm2.value
document.MyForm2.MyText2OfForm2.value
```

2. 通过 id 属性访问文档对象

在 HTML 4 版本中添加了 HTML 元素的 id 属性来定位文档对象，考察如下的代码：

```
//源程序 4.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
```

```
<title>Sample Page!</title>
</head>
<body>
<p id="p1">Welcome to<B> DOM </B>World! </p>
<script language="JavaScript" type="text/javascript">
<!--
//返回对象的相关信息
function nodeStatus(node)
{
   var temp="";
   if(node.nodeName!=null)
   {
      temp+="nodeName: "+node.nodeName+"\n";
   }
   else temp+="nodeName: null!\n";
   if(node.nodeType!=null)
   {
      temp+="nodeType: "+node.nodeType+"\n";
   }
   else temp+="nodeType: null\n";
   if(node.nodeValue!=null)
   {
      temp+="nodeValue: "+node.nodeValue+"\n\n";
   }
   else temp+="nodeValue: null\n\n";
   return temp;
}
//处理并输出信息
//返回 id 属性值为 p1 的元素对象
var currentElement=document.getElementById('p1');
var msg=nodeStatus(currentElement);
//返回 p1 的第一个孩子，并输出相关信息
currentElement=currentElement.firstChild;
msg+=nodeStatus(currentElement);
alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.9 所示。

图 4.9 通过 id 属性访问文档对象

在段落语句中，设定 id 为 p1：

```
<p id="p1">Welcome to<B> DOM </B>World! </p>
```

然后通过这个 id 属性访问到 p 元素，即定位了该对象（元素对象）：

```
currentElement=document.getElementById('p1');
```

该 currentElement 即为通过 id 属性返回的对象 p，然后进行相关处理。关于元素节点的相关知识将在“文档结构模型(DOM)”章节中将详细讲解。

由于 id 属性为 HTML 4 新添加的属性，在老版本中得不到支持，而 name 属性则支持新版本和老版本，最为可靠的方法就是同时设置 name 属性和 id 属性，并将它们设置为相同的值，然后通过相同的访问方法进行访问。如有下列表单：

```
<form name="MyForm" id="MyForm">
   <input type=text name="MyText" id="MyText" value="MyTextOfMyFrom">
</form>
```

如果要访问 MyForm 中的文本框 MyText，可使用如下的方法：

```
document.MyForm.MyText
```

通过 id 属性和 name 属性访问文档对象的方法，有利于文档对象的精确定位，同时从根本上解决了“通过对象位置访问文档对象”方法过于依赖文档结构的问题。

4.4.3 通过联合数组访问文档对象

在 HTML 被浏览器解释执行的同时，同类型的元素将构成某个联合数组的元素，可通过一个整数或者字符串为索引参数，完全定位该对象。一般情况下使用 HTML 文档中分配给标记元素的 id 属性或 name 属性作为参数。

下面的代码演示如何通过联合数组访问文档对象：

```
//源程序 4.6
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
   <title>Sample Page!</title>
   <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
//第一个表单
<form name="MyForm1">
   <input type=text name="MyTextOfForm1" value="Text of Form1">
</form>
//第二个表单
<form name="MyForm2">
   <input type=text name="MyText1OfForm2" value="Text1 of Form2">
   <input type=text name="MyText2OfForm2" value="Text2 of Form2">
</form>
<script language="JavaScript" type="text/javascript">
<!--
   var msg="";
   msg+="通过联合数组访问文档对象:\n\n";
   msg+="Form1.MyTextOfForm1.value:"+document.forms["MyForm1"].elements[0].value+"\n\n";
   msg+="Form2.MyText1OfForm2.value:"+document.forms["MyForm2"].elements[0].value+"\n\n";
   msg+="Form2.MyText2OfForm2.value:"+document.forms["MyForm2"].elements[1].value+"\n\n";
   //使用警示框输出信息
```

```
   window.alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.10 所示。

图 4.10 使用联合数组访问文档对象

在访问文档对象的实现语句 document.forms["MyForm1"].elements[0]中，也可用如下的语句代替，可实现同样的功能：

```
document.forms["MyForm1"].elements["MyTextOfForm1"]
```

注意：在 IE 中提供专门 item( )方法作为联合数组的索引，该方法将对象集中名为参数字符串的对象从对象集中取出，如上面例子中可用 document.forms.item("MyForm1")语句实现同样功能。

4.5 创建和使用自定义对象

在 JavaScript 脚本语言中，主要有 JavaScript 核心对象、浏览器对象、用户自定义对象和文本对象等，其中用户自定义对象占据举足轻重的地位。

JavaScript 作为基于对象的编程语言，其对象实例采用构造函数来创建。每一个构造函数包括一个对象原型，定义了每个对象包含的属性和方法。对象是动态的，表明对象实例的属性和方法是可以动态添加、删除或修改的。

JavaScript 脚本中创建自定义对象的方法主要有两种：通过定义对象的构造函数的方法和通过对象直接初始化的方法。

4.5.1 通过定义对象的构造函数的方法

下面的实例是通过定义对象的构造函数的方法和使用 new 操作符所生成的对象实例，先考察其代码：

```
//源程序 4.7
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
```

```
<script>
<!--
//对象的构造函数
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=showInformation;
}
//定义对象的方法
function showInformation()
{
   var msg="";
   msg="自定义对象实例：\n"
   msg+="\n 机构名称 ："+this.name+" \n";
   msg+="所在地址 ："+this.address +"\n";
   msg+="教育层次 ："+this.grade +" \n";
   msg+="在校人数 ："+this.number
   window.alert(msg);
}
//生成对象的实例
var ZGKJDX=new School("中国科技大学","安徽・合肥","高等学府","13400");
-->
</script>
</head>
<body>
<br>
<center>
<form>
   <input type=button value=调试对象按钮  onclick="ZGKJDX.information()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 4.11 所示。

图 4.11 通过定义对象的构造函数生成自定义对象实例

在该方法中，用户必须先定义一个对象的构造函数，然后再通过 new 关键字来创建该对象的实例。

定义对象的构造函数如下：

```
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=showInformation;
}
```

当调用该构造函数时，浏览器给新的对象分配内存，并隐性地将对象传递给函数。this 操作符是指向新对象引用的关键词，用于操作这个新对象。下面的句子：

```
this.name=iName;
```

该句使用作为函数参数传递过来的 iName 值在构造函数中给该对象的 name 属性赋值，该属性属于所有 School 对象，而不仅仅属于 School 对象的某个实例如上面中的 ZGKJDX。对象实例的 name 属性被定义和赋值后，可以通过如下方法访问该实例的该属性：

```
var str=ZGKJDX.name;
```

使用同样的方法继续添加其他属性 address、grade、number 等，但 information 不是对象的属性，而是对象的方法：

```
this.information=showInformation;
```

方法 information 指向的外部函数 showInformation 结构如下：

```
function showInformation()
{
   var msg="";
   msg="自定义对象实例：\n"
   msg+="\n 机构名称 : "+this.name+" \n";
   msg+="所在地址 : "+this.address +"\n";
   msg+="教育层次 : "+this.grade +" \n";
   msg+="在校人数 : "+this.number
   window.alert(msg);
}
```

同样，由于被定义为对象的方法，在外部函数中也可使用 this 操作符指向当前的对象，并通过 this.name 等访问它的某个属性。

在构建对象的某个方法时，如果代码比较简单，也可以使用非外部函数的做法，改写 School 对象的构造函数：

```
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=function()
                                {
                                    var msg="";
                                    msg="自定义对象实例：\n"
                                    msg+="\n 机构名称 : "+this.name+" \n";
                                    msg+="所在地址 : "+this.address +"\n";
                                    msg+="教育层次 : "+this.grade +" \n";
                                    msg+="在校人数 : "+this.number;
                                    window.alert(msg);
                                };
}
```

删除源程序 4.7 中外部函数 showInformation，保存更改，程序运行结果与源程序 4.7 运

行结果相同。

4.5.2 通过对象直接初始化的方法

此方法通过直接初始化对象来创建自定义对象，与定义对象的构造函数方法不同的是，该方法不需要生成此对象的实例，改写源程序 4.7：

```
<script>
<!--
//直接初始化对象
var ZGKJDX={name:"中国科技大学",
            address:"安徽・合肥",
            grade:"高等学府",
            number:"13400",
            information:showInformation
           };
//定义对象的方法
function showInformation()
{
   var msg="";
   msg="自定义对象实例：\n"
   msg+="\n 机构名称 : "+this.name+" \n";
   msg+="所在地址 : "+this.address +"\n";
   msg+="教育层次 : "+this.grade +" \n";
   msg+="在校人数 : "+this.number
   window.alert(msg);
}
-->
</script>
```

程序运行结果与源程序 4.7 运行结果相同。

该方法在只需生成某个应用对象并进行相关操作的情况下使用时，代码紧凑，编程效率高，但致命的是，若要生成若干个对象的实例，就必须为生成每个实例重复相同的代码结构，而只是参数不同而已，代码的重用性比较差，不符合面向对象的编程思路，应尽量避免使用该方法创建自定义对象。

4.5.3 修改、删除对象实例的属性

JavaScript 脚本可动态添加对象实例的属性，同时，也可动态修改、删除某个对象实例的属性，更改源程序 4.7 中的 showInformation( )代码：

```
function showInformation()
{
   var msg="";
   msg="用户自定义的对象实例：\n\n"
   msg+="   机构名称 : "+this.name+" \n";
   msg+="   所在地址 : "+this.address +"\n";
   msg+="   教育层次 : "+this.grade +" \n";
   msg+="   在校人数 : "+this.number+" \n\n";
   //修改对象实例的 number 属性
   this.number=23500;
   msg+="修改对象实例的属性:\n\n"
```

```
    msg+="   机构名称 : "+this.name+" \n";
    msg+="   所在地址 : "+this.address +"\n";
    msg+="   教育层次 : "+this.grade +" \n";
    msg+="   在校人数 : "+this.number+" \n\n";
    //删除对象实例的 number 属性
    delete this.number;
    msg+="删除对象实例的属性:\n\n"
    msg+="   机构名称 : "+this.name+" \n";
    msg+="   所在地址 : "+this.address +"\n";
    msg+="   教育层次 : "+this.grade +" \n";
    msg+="   在校人数 : "+this.number+" \n\n";
    window.alert(msg);
}
```

程序运行结果如图 4.12 所示。

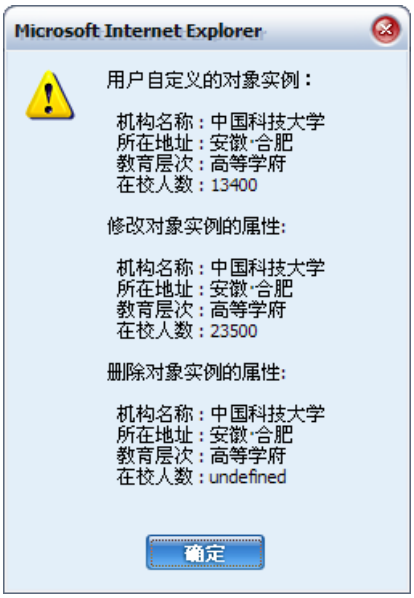

图 4.12 修改和删除对象实例的属性

从以上程序可以看出，执行 this.number=23500 语句后，对象实例的 number 属性值更改为 23500；执行 delete this.number 语句后，对象实例的 number 属性变为 undefined，同任何不存在的对象属性一样为未定义类型，但我们并不能删除对象实例本身，否则返回错误。

可见，JavaScript 动态添加、修改、删除对象实例的属性过程十分简单，之所以称之为对象实例的属性而不是对象的属性，是因为该属性只在对象的特定实例中才存在，而不能通过某种方法将某个属性赋予特定对象的所有实例。

> 注意：JavaScript 脚本中的 delete 运算符用于删除对象实例的属性，与 C++中用途不一样，C++中 delete 运算符能删除对象的实例。

4.5.4 通过原型为对象添加新属性和新方法

JavaScript 语言中所有对象都由 Object 对象派生，每个对象都有指定了其结构的原型（prototype）属性，该属性描述了该类型对象共有的代码和数据，可以通过对象的 prototype 属性为对象动态添加新属性和新方法。考察如下的代码：

```
//源程序 4.8
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script>
<!--
//对象的构造函数
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=showInformation;
}
//定义对象的方法
function showInformation()
{
   var msg="";
   msg="通过原型给对象添加新属性和新方法：\n\n"
   msg+="原始属性:\n";
   msg+="       机构名称 : "+this.name+" \n";
   msg+="       所在地址 : "+this.address +"\n";
   msg+="       教育层次 : "+this.grade +" \n";
   msg+="       在校人数 : "+this.number+" \n\n";
   msg+="新属性:\n";
   msg+="       占地面积 : "+this.addAttributeOfArea+" \n";
   msg+="新方法:\n";
   msg+="       方法返回 : "+this.addMethod+"\n";
   window.alert(msg);
}
function MyMethod()
{
   var AddMsg="New Method Of Object!";
   return AddMsg;
}
//生成对象的实例
var ZGKJDX=new School("中国科技大学","安徽・合肥","高等学府","13400");
School.prototype.addAttributeOfArea="3000";
School.prototype.addMethod=MyMethod();
-->
</script>
</head>
<body>
<br>
<center>
<form>
   <input type=button value="调试对象按钮" onclick="ZGKJDX.information()">
</form>
</center>
```

```
</body>
</html>
```

程序运行结果如图 4.13 所示。

图 4.13 通过原型给对象添加新属性和新方法

程序调用对象的 prototype 属性给对象添加新属性和新方法：

```
School.prototype.addAttributeOfArea="3000";
School.prototype.addMethod=MyMethod();
```

原型属性为对象的所有实例所共享，用户利用原型添加对象的新属性和新方法后，可通过对象引用的方法来修改。

4.5.5 自定义对象的嵌套

与面向对象编程方法相同的是，JavaScript 允许对象的嵌套使用，可以将对象的某个实例作为另外一个对象的属性来看待，考察下列代码：

```
//源程序 4.9
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script>
<!--
//对象的构造函数
//构造嵌套的对象
var SchoolData={
                    code:"0123-456-789",
                    Tel:"0551-1234567",
                    Fax:"0551-7654321"
                   };
//构造被嵌入的对象
var ZGKJDX={
                name:"中国科技大学",
```

```
                address:"安徽·合肥",
                grade:"高等学府",
                number:"13400",
                //嵌套对象 SchoolData
                data:SchoolData,
                information:showInformation
              };
//定义对象的方法
function showInformation()
{
   var msg="";
   msg="对象嵌套实例：\n\n";
   msg+="被嵌套对象直接属性值:\n"
   msg+="      机构名称 : "+this.name+"\n";
   msg+="      所在地址 : "+this.address +"\n";
   msg+="      教育层次 : "+this.grade +"\n";
   msg+="      在校人数 : "+this.number +"\n\n";
   msg+="访问嵌套对象直接属性值:\n"
   msg+="      学校代码 : "+this.data.code +"\n";
   msg+="      办公电话 : "+this.data.Tel +" \n";
   msg+="      办公传真 : "+this.data.Fax +" \n";
   window.alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form>
   <input type=button value=调试对象按钮  onclick="ZGKJDX.information()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 4.14 所示。

图 4.14 自定义对象的嵌套

首先构造对象 SchoolData 包含学校的相关联系信息，如下代码：

```
var SchoolData={
                    code:"0123-456-789",
                    Tel:"0551-1234567",
                    Fax:"0551-7654321"
                  };
```

然后构建 ZGKJDX 对象，同时嵌入 SchoolData 对象，如下代码：

```
var ZGKJDX={
            name:"中国科技大学",
            address:"安徽・合肥",
            grade:"高等学府",
            number:"13400",
            //嵌套对象 SchoolData
            data:SchoolData,
            information:showInformation
          };
```

可以看出，在构建 ZGKJDX 对象时，程序将 SchoolData 对象作为自身的某个属性 data 对应的值嵌入进去，并可通过如下的代码访问：

```
this.data.code
this.data.Tel
this.data.Fax
```

通过直接对象初始化的方法，上述代码可改写如下：

```
var ZGKJDX={
            name:"中国科技大学",
            address:"安徽・合肥",
            grade:"高等学府",
            number:"13400",
            //嵌套对象 SchoolData
            data:{
                    code:"0123-456-789",
                    Tel:"0551-1234567",
                    Fax:"0551-7654321"
                  },
            information:showInformation
          };
```

程序运行结果与源程序 4.9 相同。

下面介绍对象创建过程中内存的分配和释放问题。

4.5.6 内存的分配和释放

JavaScript 是基于对象的编程语言而不是面向对象的编程语言，缺少指针的概念，而后者在动态分配和释放内存的过程中作用巨大，那 JavaScript 中的内存如何管理呢？

创建对象的同时，浏览器自动为该对象分配内存空间，JavaScript 将新对象的引用传递给调用的构造函数，在对象清除时其占据的内存将自动回收，其实整个过程都是浏览器的功劳，JavaScript 只负责创建该对象。

浏览器中的这种内存管理机制称为“内存回收”，它动态分析程序中每个占据内存空间的数据(变量、对象等)，如果该数据对于程序标记为不可再用时，浏览器将调用内部函数将其占据的内存空间释放，实现内存的动态管理。

当然，在自定义的对象使用完后，可通过给其赋空值的方法标记对象已经使用完成：

```
ZGKJDX=null;
```

浏览器将根据此标记动态释放其占据的内存，否则将保存该对象直至当前程序再次使用它为止。

4.6 本章小结

本章主要介绍了 JavaScript 基于对象编程语言及与 C++、Java 等面向对象编程语言的异同点；初步了解了浏览器载入文档时 JavaScript 对象的产生过程及文档中对象的访问方法；重点介绍了 JavaScript 如何创建和使用自定义的对象以及程序中如何动态管理内存的问题。

JavaScript 语言使用构造函数创建新的对象实例，对象用来将该实例的属性初始化。在 JavaScript 中创建和管理对象是直线式的，并不需要对诸如内存管理（C++等面向对象编程语言概念）等方面特别注意。同时用户自定义的对象可以用于构建模块化和易于维护的脚本，但较之 JavaScript 核心对象和 DOM 规范所定义的对象而言，其功能一般较为简单，实际用途不大。下面几章将重点介绍 DOM 规范和 JavaScript 核心对象等。