第5章 客户端动态技术 JavaScript

本章学习目标

- 理解客户端动态的基本原理。
- 掌握 JavaScript 的基本语法。
- 了解 JavaScript 的内置对象。
- 理解 JavaScript 文档对象模型的原理。
- 理解 JavaScript 的事件处理机制。
- 掌握在 Dreamweaver 中编写 JavaScript 常见事件处理程序并绑定 HTML 元素的方法。

本章主要讲述了 JavaScript 的基本语法、内置对象、文档对象模型和事件处理机制，并以身份信息自动填充、表格行背景随鼠标切换、表格行全选以及自定义 URL 浮动小窗体 4 个综合实例，展示了在 Dreamweaver 中编写 JavaScript 的方法。

5.1 客户端动态技术概述

客户端动态技术是指可以在客户端执行的技术。在 Web 开发中，客户端一般指用户浏览器端。此处的动态，并不仅指网页上的各种动画，更重要的是指用户与浏览器的交互，如弹出对话框、弹出菜单、拖动层等。在诸多客户端动态技术中，JavaScript 是应用最广泛的客户端技术，也是被万维网联盟（World Wide Web Consortium，W3C）接受为前端开发的标准技术之一。

JavaScript，简称 JS，是一种大小写敏感的高级解释型脚本编程语言。解释型脚本语言意味着，用 JS 写的程序并不需要编译，可以由浏览器直接执行。JavaScript 起源于网景公司（Netscape）于 1995 年发布的 LiveScript，后借助 Java 语言的名声，将其改名为

“JavaScript”，并一直沿用至今。因此，JavaScript 虽然有一些语法与 Java 类似，但事实上，这是两种完全不同的程序设计语言。后经过多年的发展，被欧洲计算机制造商协会（European Computer Manufacturers Association，ECMA）确立为浏览器脚本语言的标准，并基于此发布了 ECMAScript 标准。该标准定义了 JavaScript 的核心语言和特性，自此 JavaScript 成为客户端浏览器的标准。进入 21 世纪之后，Google 在其搜索引擎和 Gmail 邮箱中，应用了大量 JavaScript 技术，开发出了诸如 Google Suggestion、Gmail 无刷新收邮件等效果，使得 JavaScript 技术名声大噪。这也标志着 Web 2.0 的兴起，从此 JavaScript 进入了快速发展时期。

时至今日，JavaScript 已经形成了覆盖桌面、游戏和移动应用开发等领域，在著名的开发语言排行榜 TIBOE 中，近几年 JavaScript 也稳居在前 6 名左右。同时还出现了一种新的数据结构——JSON 格式，这是一种比 XML 还要简洁的数据表示方式。由此可见，JavaScript 是一门发展势头良好的程序设计语言。虽然本书重点是关注其 Web 开发，但掌握 JavaScript 的意义远不止于此，它的程序设计风格、技巧和特性，都有助于程序员理解现代程序设计的语言思想，提升程序设计实力。

需要说明的是，JavaScript 本身是一个非常庞大的体系，如果要把它的全部内容都展示出来，可能一本平均厚度的书都是不够的，因此本章仅关注其能够撑起页面交互的核心知识。

5.2 JavaScript 基本语法

5.2.1 数据类型

JavaScript 有基本数据类型和引用数据类型两种数据类型。

1. 基本数据类型

基本数据类型也称为简单数据类型，常见的一共有 5 种，具体见表 5.1。

表 5.1 JavaScript 基本数据类型

类 型	说 明	示 例
number	数值型，包括正负小数、正负整数和 0 的所有数值	i=1，j=12.3
string	字符串型，用一对英文的双引号或者单引号括起来的任意内容	s="hello"，s='hello'
boolean	布尔型，值为固定的 true 或者 false，用于条件判断	c=true，c=false
undefined	未定义类型，该类型只有一个值，即 undefined。当声明一个变量但不给这个变量赋值时，它的值就是 Undefined	obj=undefined
null	空值，该类型也只有一个值，即 null，该值专门用来表示一个为空的对象	obj=null

从表 5.1 可知，简单数据类型的内容就是单一的一个值本身。

2. 引用数据类型

引用数据类型即 Object 类型，是一种复合类型，内部由数据和方法组成。其中，数据一

般以“键:值”对作为其基本数据结构,每个“键:值”对以逗号分隔,整个键值对集合存放在花括号中,代码如下。

```
<script language="javascript">
person={
    id:1001,
    name:"张三"
};
</script>
```

上述代码即创建了一个名为 person 的 Object 类变量。完成定义后,则可以通过 person.id 的方式来调用 id 的值。这种键值对的方式,还可以采取嵌套对象的方式进行数据的定义,代码如下。

```
<script language="javascript">
person={
    id:1001,
    name:"张三",
    courses: {
        english:100,
        JavaScript:100
    }
};
</script>
```

其中,courses 依然可以视为一个键,但是它的值却是另一个对象。对于其中的值依然可以通过“对象.键”的方式来引用,只不过嵌套几层,就多几个“.”,如获取 english 的值,即 person.courses.english。

除了数据之外,在 Object 中,还可以加入函数,代码如下。

```
<script language="javascript">
person={
    id:1001,
    name:"张三",
    courses:{
        english:100,
        JavaScript:100
    },
    study(){alert('study');},
    work(){}
};
</script>
```

其中,study()和 work()两个函数即成为这个对象的方法,其访问依然可以通过 person.study()的方式来访问。这种结构与 Java 中类的定义极为相似。由此可知,也可认为 JavaScript 是一种面向对象的程序设计语言。上述方式还可以通过创建空的 Object,再通过添加属性和方法的方式来实现,代码如下。

```
<script language="javascript">
person=new Object();
person.id=1001;
person.name="张三";
person.courses={english:100,JavaScript:100};
person.study=function(){alert('study')};
person.work=function(){};
person.study();
</script>
```

其中，study 和 work 方法的创建则采取了类似匿名函数的定义方法。这个例子虽然简单，但展示了 Object 对象创建及数据和方法调用的过程。本节之所以花了大量篇幅来讲述该过程，是因为了解这一过程，将有助于加强对后续章节中对象创建和引用的理解。

3. 弱类型

弱类型并不是指一种特定的类型，而是指定义变量时不必指定类型，其类型会根据值的类型自动判断，如图 5.1 所示。

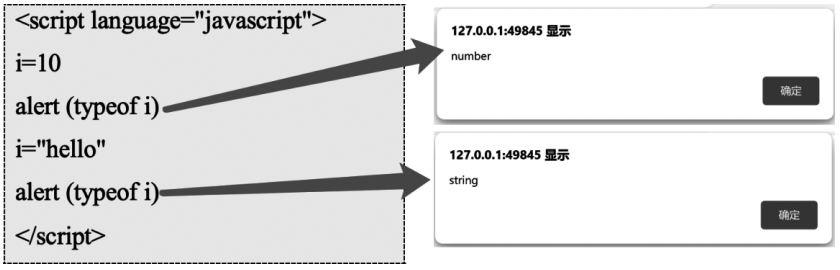


图 5.1　弱类型示例

该代码使用了 typeof 方法来判断变量的类型，从截图可以看到，第一次判断时，i 的类型是 number，第二次则变成了 string。

虽然 JavaScript 并不需要声明类型，但在 ECMA5 的标准中，使用 var 来声明变量，而在 ECMA6 的标准中，则新增了 let 来声明变量。这三种方式的区别主要在于作用范围不同，一般没有任何修饰的是全局变量，var 修饰的变量作用域为函数范围，而 let 修饰的则为程序块。作用域的概念会在函数中再做详细解释。在此只需知道这三种变量的大致区别即可。事实上，JavaScript 是面向浏览器元素的程序设计语言，一般不会涉及复杂的作用域的应用。因此，使用没有任何类型修饰的变量，也已经能满足绝大多数的应用场合。

5.2.2　运算符与表达式

运算符是指针对变量执行各种操作的符号或关键字，由变量和运算符组成的等式即表达式。JavaScript 中主要有算术运算符、赋值运算符、比较运算符、逻辑运算符、字符串连接运算符。

1. 算术运算符

算术运算符用于执行变量之间的算术运算，给定变量 x＝20，y＝10，具体计算规则如表 5.2 所示。

表 5.2 算术运算符示例

运算符	作　用	表　达　式	z 的值
+	加法	z=x+y	30
-	减法	z=x-y	10
*	乘法	z=x*y	200
/	除法	z=x/y	2
		z=y/x	0.5
%	取模(余数求)	z=x%y	0
		z=y%x	20
++	自增	z=++x,将 x 的值加 1,并赋值给 z	z=21,x=21
		z=x++,将 x 的值赋值给 z,再将 x 加 1	z=20,x=21
--	自减	z=--x,将 x 的值减 1,并赋值给 z	z=19,x=19
		z=x--,将 x 的值赋值给 z,再将 x 减 1	z=20,x=19

2. 赋值运算符

赋值运算符用于给 JavaScript 变量赋值,给定变量 x=20,y=10,赋值运算符具体规则如表 5.3 所示。

表 5.3 赋值运算符示例

运算符	表达式	作用	结果
=	x=y		x=10
+=	x+=y	x=x+y	x=30
-=	x-=y	x=x-y	x=10
*=	x*=y	x=x*y	x=200
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

3. 比较运算符

比较运算符用于变量之间的比较,其结果返回 boolean 值:true 和 false。给定变量 x=20,比较运算符的具体规则如表 5.4 所示。

表 5.4 比较运算符示例

运　算　符	作　用	表　达　式	返回值
==(双等号)	判断值是否相等	x==8	false
		x==20	true
===(三等号)	判断值和类型是否相等	x==="20"	false
		x===20	true

续表

运　算　符	作　　用	表　达　式	返回值
!=	判断是否不等	x!=8	true
!==	不绝对等于(值和类型有一个不相等,或两个都不相等)	x!=="20"	true
		x!==20	false
>	大于	x>30	false
<	小于	x<30	true
>=	大于或等于	x>=30	false
<=	小于或等于	x<=20	true

4. 逻辑运算符

逻辑运算符用于测定变量或值之间的逻辑,结果也是返回 boolean 值:true 和 false。给定变量 x=20,y=10,逻辑运算符的具体规则如表 5.5 所示。

表 5.5　逻辑运算符示例

运　算　符	作　　用	表　达　式	返回值
&&(逻辑与)	运算符左右两个值均为 true,则返回 true	x<30 && y>1	true
||(逻辑或)	运算符左右两个值只要有一个为 true,则返回 true	x<20 || y==10	true
!(逻辑非)	逻辑反转,右侧为 true,则返回 false,否则返回 true	!x<20	true

5. 字符串连接运算符

字符串连接运算符即普通的加号“+”,只是当参与计算的其中一项为字符串时,相加的功能就变成了连接功能。具体规则如表 5.6 所示。

表 5.6　字符串连接运算符示例

表　达　式	结　　果
x="5"+5	x="55"
x="hello"+5	x="hello5"

5.2.3　控制语句

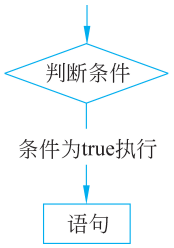
控制语句即控制代码执行流程的语句,JavaScript 中控制语句主要有条件语句和循环语句。

1. 条件语句

条件语句是根据指定的条件,选择性地执行特定代码的控制语句。在 JavaScript 中常见的条件语句有 if 和 switch 语句。

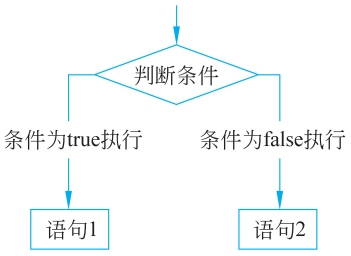
if 有 4 种形式:单分支、双分支、多分支和嵌套。单分支,即只有一个条件,满足条件则执行对应代码,具体规则如图 5.2 所示。

双分支结构,则有两个分支,根据 true 或者 false 来执行其中一个,具体规则如图 5.3 所示。




```
<script language="javascript">
x=10
if(x==10)
    alert("hello world!")
</script>
```

图 5.2　单分支 if 语句




```
<script language="javascript">
x=60
if(x<60)
    alert("不及格")
else
    alert("及格")
</script>
```

图 5.3　双分支 if 语句

双分支结构还有一种三元表达式的写法，语法结构为：判断条件？表达式 1:表达式 2。如果判断条件为真，则执行表达式 1，否则执行表达式 2。示例代码如图 5.4 所示。

```
<script language="javascript">
x=60
result=x>=60?"及格":"不及格"
alert(result)
</script>
```

图 5.4　三元表达式

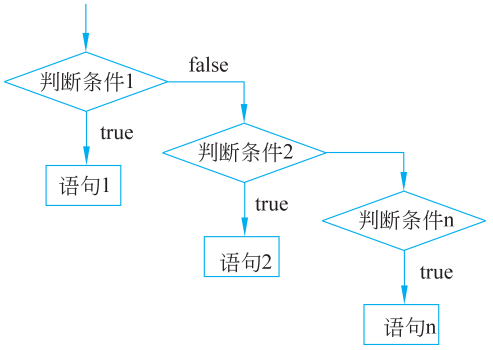
多分支结构，则是有多个分支，这些不同的分支是通过不同的具有互斥性的条件来执行的。互斥性是指在同一个条件语句中，这些条件只会执行一次，具体规则如图 5.5 所示。

嵌套分支，则是在满足某一分支条件的情形下，又出现了其他的分支语句，代码如下。

```
<script language="javascript">
x=100
if(x>=90){
   if(x==100)
       alert("满分")
}
</script>
```

从这些代码可知，嵌套分支结构并没有固定写法，根据不同的需求，可以在分支中嵌套更多的 if 语句。

除了 if 语句外，还有一种条件语句 switch。switch 语句和多分支 if 语句类似，但是 switch 是根据具体的值来判断执行语句。




```
<script language="javascript">
x=70
if(x<60)
    alert("不及格")
else if(x>=60 && x<70)
    alert("及格")
else if(x>=70 && x<80)
    alert("中等")
else if(x>=80 && x<90)
    alert("良好")
else if(x>=90)
    alert("优秀")
</script>
```

图 5.5　多分支 if 语句

```
<script language="javascript">
x=70
switch (x) {
   case 60:
        alert("及格")
        break
   case 70:
        alert("中等")
        break
   case 80:
        alert("良好")
        break
   case 90:
        alert("优秀")
        break
   default:
        alert("未匹配到")
   }
</script>
```

其中，switch 语句后的表达式类型和 case 后的类型要完全一致，并且值也要完全相同，才会执行 case 后的语句。同时，如果 case 语句的范围内出现了 break 语句，则会结束 switch 程序。如没有 break，匹配到一个 case 后，程序依然会继续往下匹配，直至程序结束或者再次碰到 break 语句。如果没匹配到任何 case，则会执行 default 后的语句。switch 可以视为一种简化的多分支结构，在每个 case 中还能嵌套其他 if 语句。switch 语句结构清晰简单，但是表达式范围较为固定，在实际的 Web 程序中并不常用。

2. 循环语句

循环语句是指按条件重复执行表达式的语句。在 JavaScript 中，循环语句主要有 for 循环、while 循环和 do-while 循环三种。

for 循环的语法结构如下。

```
for(初始化变量;条件表达式;操作表达式){
      循环体
}
```

在 for 循环中，初始化变量，即为循环声明一个变量，该变量通常作为计数器，即用于设置循环的次数。而条件表达式，即根据初始化的变量，来决定是否要执行循环体。操作表达式，一般是初始化变量的计数器，是决定循环能够正常终止的关键代码。如图 5.6 所示案例，为循环输出 1～10 的累加总和。

```
<script language="javascript">
sum=0
for(i=0;i<=10;i++)
{
    sum=sum+i
    document.writeln("第",i,"次循环 i=",i," sum 为: ",sum)
    document.writeln("<br>")
}
</script>
```

```
第0次循环 i=0 sum为: 0
第1次循环 i=1 sum为: 1
第2次循环 i=2 sum为: 3
第3次循环 i=3 sum为: 6
第4次循环 i=4 sum为: 10
第5次循环 i=5 sum为: 15
第6次循环 i=6 sum为: 21
第7次循环 i=7 sum为: 28
第8次循环 i=8 sum为: 36
第9次循环 i=9 sum为: 45
第10次循环 i=10 sum为: 55
```

图 5.6　for 循环示例

从图 5.6 中可以看出，i 变量决定了循环次数。每一次循环，循环体会执行一次。虽然每次执行的代码相同，但由于 i 不同，所以结果也不同。for 的循环体内还可以嵌套 for 循环，以实现更为复杂的操作。

while 循环的语法结构如下。

```
while(条件表达式){
    循环体
}
```

while 循环是在条件表达式为 true 的前提下，循环执行指定一段代码，直到表达式为 false 时结束循环。如图 5.6 所示案例的 while 循环代码如下。

```
i=0
sum=0
while(i<=10){
    sum=sum+i
    document.writeln("第",i,"次循环 i=",i," sum 为:",sum)
    document.writeln("<br>")
    i=i+1
}
```

它的循环逻辑基本与 for 循环类似，只是变量的声明和计数是分开的。需要注意的是，这种写法使得在实际编程中，初学者容易忘记编写循环体中的 i=i+1，导致出现无限循环，即死循环。

do-while 循环的语法结构如下。

```
do{
  循环体
}while(条件表达式)
```

do-while 和 while 最大的差别在于 do-while 至少会执行一次循环体，而 while 可能一次也不执行。

这三种循环功能类似，不存在唯一性，选择哪种主要由个人使用偏好来决定。但在实际工作中，for 循环是被广泛使用的。

在循环中，还有两个特殊的语句，用于中止或者退出循环，即 continue 语句和 break 语句。continue 语句是用于中止当前当次循环，即在某一次循环中，如果执行到 continue 语句，则 continue 之后的语句不再执行，直接进入下一次循环。图 5.7 为计算 1～20 的偶数和。

```
<script language="javascript">
sum=0
for(i=0;i<=20;i++)
{
    if(i%2!=0)
        continue
    sum=sum+i
    document.writeln("第",i,"次循环 i=",i," sum 为：",sum)
    document.writeln("<br>")
}
</script>
```

```
第0次循环 i=0 sum为：0
第2次循环 i=2 sum为：2
第4次循环 i=4 sum为：6
第6次循环 i=6 sum为：12
第8次循环 i=8 sum为：20
第10次循环 i=10 sum为：30
第12次循环 i=12 sum为：42
第14次循环 i=14 sum为：56
第16次循环 i=16 sum为：72
第18次循环 i=18 sum为：90
第20次循环 i=20 sum为：110
```

图 5.7　continue 语句示例

当 i 为奇数时，通过 if 语句会执行 continue 语句，但之后的代码就跳过了，直接进入下次循环。

break 语句则是直接结束循环，即如果在循环体中执行到 break 语句，不论还剩下几次循环，都会直接结束循环。示例如图 5.8 所示。

```
<script language="javascript">
for(i=0;i<=1000;i++)
{
    if(i==10)
    {
        document.write("循环结束")
        break
    }
    document.writeln("第",i,"次循环 i=",i)
    document.writeln("<br>")
}
</script>
```

```
第0次循环 i=0
第1次循环 i=1
第2次循环 i=2
第3次循环 i=3
第4次循环 i=4
第5次循环 i=5
第6次循环 i=6
第7次循环 i=7
第8次循环 i=8
第9次循环 i=9
循环结束
```

图 5.8　break 语句示例

原本要执行 1001 次的循环，在第 11 次时执行了 break 语句，因此剩下的 990 次就不再执行了。

本节介绍的只是基本控制语句，事实上，当循环这个机制引入程序设计后，所编写的程序就可以实现靠人工很难完成的复杂任务。例如，计算 1000 以内的素数、水仙花数等。但由于本书关注的是 Web 开发，因此对一般的算法研究并不会花太多笔墨。条件和循环语句的常用用法，会在后续章节中以实例方式讲解。

5.2.4　函数

函数是一种可重复调用的代码块，JavaScript 使用 function 关键词进行函数定义，基本语法如下。