第一部分

JavaScript 语言核心

本书该部分（第2～12章）描述JavaScript语言核心。这部分是该语言的主要参考资料。学习之初通读一遍该部分，以后在遇到JavaScript的难点时，回到这里重新查阅相关内容以巩固知识的掌握：

第2章

词法结构

编程语言的词法结构是一套基础性规则，用来描述如何使用这门语言来编写程序。作为语法的基础，它规定了诸如变量名是什么样的、怎么写注释，以及程序语句之间如何分隔等规则。本章用很短的篇幅来介绍JavaScript的词法结构。

2.1 字符集

JavaScript程序是用Unicode字符集编写的。Unicode是ASCII和Latin-1的超集，并支持地球上几乎所有在用的语言。ECMAScript 3要求JavaScript的实现必须支持Unicode 2.1及后续版本，ECMAScript 5则要求支持Unicode 3及后续版本。可以参考3.2节的“边栏”来了解更多关于Unicode和JavaScript的信息。

2.1.1 区分大小写

JavaScript是区分大小写的语言。也就是说，关键字、变量、函数名和所有的标识符（identifier）都必须采取一致的大小写形式。比如，关键字“while”必须写成“while”，而不能写成“While”或者“WHILE”。同样，“online”、“Online”、“OnLine”和“ONLINE”是4个不同的变量名。

但需要注意的是，HTML并不区分大小写（尽管XHTML区分大小写）。由于它和客户端JavaScript联系紧密，因此这点区别很容易混淆[译注1]。许多客户端JavaScript对象和属性与它们所表示的HTML标签和属性同名。在HTML中，这些标签和属性名可以使用大写也可以是小写，而在JavaScript中则必须是小写。例如，在HTML中设置事件处理程序

译注1：严格讲XHTML是区分大小写的，但由于浏览器有着非常较强大的纠错能力，即使文档中包含很多不严格的大小写，浏览器还是比较“宽容”地正确解析渲染。

时，onclick属性可以写成onClick，但在JavaScript代码（或者XHTML文档）中，必须使用小写的onclick。

2.1.2 空格、换行符和格式控制符

JavaScript会忽略程序中标识（token）[译注2]之间的空格。多数情况下，JavaScript同样会忽略换行符（2.5节提到了一种意外情形）。由于可以在代码中随意使用空格和换行，因此可以采用整齐、一致的缩进来形成统一的编码风格，从而提高代码的可读性。

除了可以识别普通的空格符（\u0020），JavaScript还可以识别如下这些表示空格的字符：水平制表符（\u0009）、垂直制表符（\u000B）、换页符（\u000C）、不中断空白(\u00A0)、字节序标记（\uFEFF），以及在Unicode中所有Zs类别的字符[译注3]。JavaScript将如下字符识别为行结束符：换行符（\u000A），回车符（\u000D），行分隔符（\u2028），段分隔符（\u2029）。回车符加换行符在一起被解析为一个单行结束符。

Unicode格式控制字符（Cf类[译注4]），比如“从右至左书写标记”（\u200F）和“从左至右书写标记”（\u200E）[译注5]，控制着文本的视觉显示，这对于一些非英语文本的正确显示来说是至关重要的，这些字符可以用在JavaScirpt的注释、字符串直接量和正则表达式直接量中，但不能用在标识符（比如，变量名）中。但有个例外，零宽连接符（\u200D）和零宽非连接符（\uFEFF）[译注6]是可以出现在标识符中的，但不能作为标识符的首字符。上文也提到了，字节序标记格式控制符（\uFEFF）被当成了空格来对待。

2.1.3 Unicode 转义序列

在有些计算机硬件和软件里，无法显示或输入Unicode字符全集。为了支持那些使用老旧技术的程序员，JavaScript定义了一种特殊序列，使用6个ASCII字符来代表任意16位

译注2：请参照*http://en.wikipedia.org/wiki/Token*。

译注3：Unicode对其所有字符做了分类，这种分类使用“通用类别值”表示，这里的“Zs”既是其中一种类别值，特指没有标志符号但不属于控制或格式字符的空格字符。更多类别值的描述请参见*http://www.unicode.org/reports/tr44/*中关于General Category Values的内容。

译注4：Cf是Unicode中的一种“通用类别值”，指代那些影响文本布局或文本处理操作但通常不会呈现的格式字符。

译注5：“从右至左书写标记”（RIGHT-TO-LEFT MARK）和“从左至右书写标记”（LEFT-TO-RIGHT MARK）均属于双向字符集语言，字符是带有方向的，比如在阿拉伯语言中，标点位于单词的左侧，而不是我们通常熟悉的右侧。

译注6：ZERO WIDTH (NON-)JOINER，零宽(非)连接符，指没有宽度的不可见连接符，在蒙文、满文、锡伯文等少数民族语言中会使用到。

Unicode内码。这些Unicode转义序列均以\u为前缀，其后跟随4个十六进制数（使用数字以及大写或小写的字母A～F表示）。这种Unicode转义写法可以用在JavaScript字符串直接量、正则表达式直接量和标识符中（关键字除外）。例如，字符é的Unicode转义写法为\u00E9，如下两个JavaScript字符串是完全一样的：

```
"café" === "caf\u00e9" // => true
```

Unicode转义写法也可以出现在注释中，但由于JavaScript会将注释忽略，它们只是被当成上下文中的ASCII字符处理，而且并不会被解析为其对应的Unicode字符。

2.1.4 标准化

Unicode允许使用多种方法对同一个字符进行编码。比如，字符“é”可以使用Unicode字符\u00E9表示，也可以使用普通的ASCII字符e跟随一个语调符\u0301。在文本编辑器中，这两种编码的显示结果一模一样，但它们的二进制编码表示是不一样的，在计算机里也不相等。Unicode标准为所有字符定义了一个首选的编码格式，并给出了一个标准化的处理方式将文本转换为一种适合比较的标准格式，JavaScript会认为它正在解析的程序代码已经是这种标准格式，不会再对其标识符、字符串或正则表达式作标准化处理。

2.2 注释

JavaScript支持两种格式的注释。在行尾“//”之后的文本都会被JavaScript当做注释忽略掉的。此外，“/*”和“*/”之间的文本也会当做注释，这种注释可以跨行书写，但不能有嵌套的注释。下面都是合法的JavaScript注释：

```
// 这里是单行注释
/* 这里是一段注释 */ // 这里是另一段注释
/*
*这又是一段注释
* 这里的注释可以连写多行
*/
```

2.3 直接量

所谓直接量（literal），就是程序中直接使用的数据值。下面列出的都是直接量：

```
12 // 数字
1.2 // 小数
"hello world"          // 字符串文本
'Hi'                   // 另一个字符串
true                   // 布尔值
false                  // 另一个布尔值
/javascript/gi         // 正则表达式直接量（用做模式匹配）
```

```
null                    // 空
```

第3章会详细讲解数字和字符串直接量。正则表达式直接量会在第10章讲解。更多复杂的表达方式（参见4.2节）可以写成数组或对象直接量，例如

```
{ x:1, y:2 } // 对象
[1,2,3,4,5] // 数组
```

2.4 标识符和保留字

标识符就是一个名字。在JavaScript中，标识符用来对变量和函数进行命名，或者用做JavaScript代码中某些循环语句中的跳转位置的标记。JavaScript标识符必须以字母、下划线（_）或美元符（$）开始。后续的字符可以是字母、数字、下划线或美元符（数字是不允许作为首字符出现的，以便JavaScript可以轻易区分开标识符和数字）。下面是合法的标识符：

```
i
my_variable_name
v13
_dummy
$str
```

出于可移植性和易于书写的考虑，通常我们只使用ASCII字母和数字来书写标识符。然而需要注意的是，JavaScript允许标识符中出现Unicode字符全集中的字母和数字。（从技术上讲，ECMAScript标准也允许在标识符的首字符后面出现Unicode字符集中的Mn类、Mc类和Pc类[译注7]）。由此，程序员也可以使用非英语语言或数学符号来书写标识符：

```
var sí = true;
var π = 3.14;
```

和其他任何编程语言一样，JavaScript保留了一些标识符为自己所用。这些“保留字”不能用做普通的标识符，下面会讲到。

保留字

JavaScript把一些标识符拿出来用做自己的关键字。因此，就不能再在程序中把这些关键字用做标识符了：

译注7：Unicode对其所有字符做了分类，这种分类使用“通用类别值”表示，这里的“Mn”、“Mc”和“Pc”就是其中三种类别值，Mn表示基字符的修改中出现的非间距字符，Mc表示基字符的修改中影响了基字符标志位的宽度的间距字符，Pc指连接两个字符的连接符或标点符号。更多类别值的描述请参见*http://www.unicode.org/reports/tr44/*中关于General Category Values的内容。

```
break       delete      function      return      typeof
case        do          if            switch      var
catch       else        in            this        void
continue    false       instanceof    throw       while
debugger    finally     new           true        with
default     for         null          try
```

JavaScript同样保留了一些关键字，这些关键字在当前的语言版本中并没有使用，但在未来版本中可能会用到。ECMAScript 5保留了这些关键字：

```
class  const  enum  export  extends  import  super
```

此外，下面这些关键字在普通的JavaScript代码中是合法的，但是在严格模式下是保留字：

```
implements let private public yield
interface package protected static
```

严格模式同样对下面的标识符的使用做了严格限制，它们并不完全是保留字，但不能用做变量名、函数名或参数名：

```
arguments  eval
```

ECMAScript 3将Java的所有关键字都列为自己的保留字，尽管这些保留字在ECMAScript 5中放宽了限制，但如果你希望代码能在基于ECMAScript 3实现的解释器上运行的话，应当避免使用这些关键字作为标识符：

```
abstract    double      goto          native      static
boolean     enum        implements    package     super
byte        export      import        private     synchronized
char        extends     int           protected   throws
class       final       interface     public      transient
const       float       long          short       volatile
```

JavaScript预定义了很多全局变量和函数，应当避免把它们的名字用做变量名和函数名：

```
arguments           encodeURI           Infinity    Number          RegExp
Array               encodeURIComponent  isFinite    Object          String
Boolean             Error               isNaN       parseFloat      SyntaxError
Date                eval                JSON        parseInt        TypeError
decodeURI           EvalError           Math        RangeError      undefined
decodeURIComponent  Function            NaN         ReferenceError  URIError
```

JavaScript的具体实现可能定义独有的全局变量和函数，每一种特定的JavaScript运行环境（客户端、服务器端等）都有自己的一个全局属性列表，这一点是需要牢记的。参照第四部分的Window对象来了解客户端JavaScript中定义的全局变量和函数列表。

2.5 可选的分号

和其他许多编程语言一样，JavaScript使用分号（;）将语句（参见第5章）分隔开。这对增强代码的可读性和整洁性是非常重要的：缺少分隔符，一条语句的结束就成了下一条语句的开始，反之亦然。在JavaScript中，如果语句各自独占一行，通常可以省略语句之间的分号（程序结尾或右花括号“}”之前的分号也可以省略）。许多JavaScript程序员（包括本书中的示例代码）使用分号来明确标记语句的结束，即使在并不完全需要分号的时候也是如此。另一种风格就是，在任何可以省略分号的地方都将其省略，只有在不得不用的时候才使用分号。不管采用哪种编程风格，关于JavaScript中可选分号的问题有几个细节需要注意。

考虑如下代码，因为两条语句用两行书写，第一个分号是可以省略掉的：

```
a = 3;
b = 4;
```

如果按照如下格式书写，第一个分号则不能省略掉：

```
a = 3; b = 4;
```

需要注意的是，JavaScript并不是在所有换行处都填补分号：只有在缺少了分号就无法正确解析代码的时候，JavaScript才会填补分号。换句话讲（类似下面代码中的两处异常），如果当前语句和随后的非空格字符不能当成一个整体来解析的话，JavaScript就在当前语句行结束处填补分号。看一下如下代码：

```
var a
a
=
3
console.log(a)
```

JavaScript将其解析为：

```
var a; a = 3; console.log(a);
```

JavaScript给第一行换行处添加了分号，因为如果没有分号，JavaScript就无法解析代码`var a a`。第二个a可以单独当做一条语句“a;”，但JavaScript并没有给第二行结尾填补分号，因为它可以和第三行内容一起解析成“a=3;”。

这些语句的分隔规则会导致一些意想不到的情形，这段代码写成了两行，看起来是两条独立的语句：

```
var y = x + f
(a+b).toString()
```

但第二行的圆括号却和第一行的f组成了一个函数调用，JavaScript会把这段代码看做：

```
var y = x + f(a+b).toString();
```

而这段代码的本意并不是这样。为了能让上述代码解析为两条不同的语句，必须手动填写行尾的显式分号。

通常来讲，如果一条语句以“(”、“[”、“/”、“+”或“-”开始，那么它极有可能和前一条语句合在一起解析。以“/”、“+”和“-”开始的语句并不常见，而以“(”和“[”开始的语句则非常常见，至少在一些JavaScript编码风格中是很普遍的。有些程序员喜欢保守地在语句前加上一个分号，这样哪怕之前的语句被修改了、分号被误删除了，当前语句还是会正确地解析：

```
var x = 0 // 这里省略了分号
;[x,x+1,x+2].forEach(console.log) // 前面的分号保证了正确地语句解析
```

如果当前语句和下一行语句无法合并解析，JavaScript则在第一行后填补分号，这是通用规则，但有两个例外。第一个例外是在涉及return、break和continue语句（参见第5章）的场景中。如果这三个关键字后紧跟着换行，JavaScript则会在换行处填补分号。例如，这段代码：

```
return
true;
```

JavaScript会解析成：

```
return; true;
```

而代码的本意是这样：

```
return true;
```

也就是说，在return、break和continue和随后的表达式之间不能有换行。如果添加了换行，程序则只有在极特殊的情况下才会报错，而且程序的调试非常不方便。

第二个例外是在涉及“++”和“--”运算符（见4.8节）的时候。这些运算符可以作为表达式的前缀，也可以当做表达式的后缀。如果将其用做后缀表达式，它和表达式应当在同一行。否则，行尾将填补分号，同时“++”或“--”将会作为下一行代码的前缀操作符并与之一起解析，例如，这段代码：

```
x
++
y
```

这段代码将解析为“x; ++y”，而不是“x++; y”。