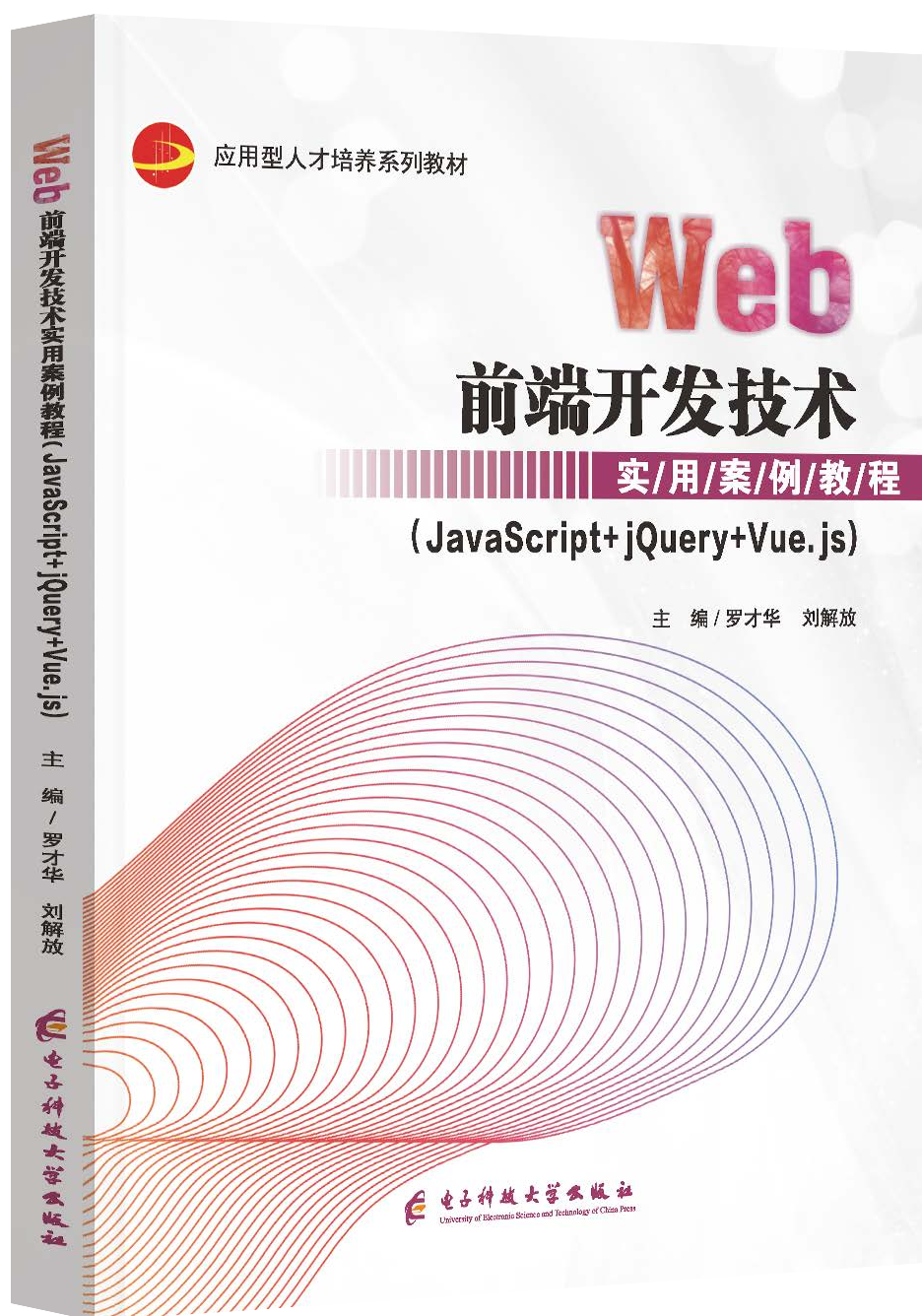




书名：Web前端开发技术案例教程（JavaScript+jQuery+Vue.js）（双色）

ISBN：978-7-5647-9395-1

作者：罗才华 刘解放

出版社：电子科技大学出版社

定价：65.80元

Web 前端开发是计算机类专业的一门核心课程,内容繁多、复杂,主要包括 HTML 5、CSS 3 和 JavaScript 三部分,以及围绕这三者开发出来的大量技术框架和解决方案。

本书立足于 JavaScript 原生语言基础,对其语法、数组、函数、对象和事件等做了详细介绍,然后对 JavaScript 的基础性框架 jQuery 和三大前端主流框架之一的 Vue.js 进行了详细讲解,并提供了大量实战案例来巩固重点知识,训练综合应用能力。全书共分为八章,各章主要内容说明如下:

第一章讲解 JavaScript 语言基础,主要介绍了 JavaScript 脚本语言的主要特征、组成和基本语法。

第二章讲解 JavaScript 语言进阶,主要介绍了 JavaScript 中的数组的定义与使用,函数的定义和调用,对象的创建、内置对象、浏览器对象、JSON 对象和 RegExp 对象的使用。

第三章讲解 BOM 与 DOM 编程,主要介绍了 BOM 常用对象的使用、文档对象模型、DOM 操作 HTML 元素和节点、常用事件和事件处理程序、如何使用 JavaScript 语言进行 DOM 编程。

第四章讲解 jQuery 基础,详细介绍了 jQuery 的下载与使用、jQuery 选择器、jQuery 中的各种 DOM 操作、jQuery 事件、jQuery 中各种动画效果的实现方法。

第五章讲解 jQuery 进阶,详细介绍了 Ajax 的概念、原理与优缺点,jQuery 中 Ajax 请求的几种方法,综合运用 JavaScript+Ajax+JSON+PHP 等技术解决实际问题。

第六章讲解 Vue.js 基础,详细介绍了 Vue.js 发展历程、Vue.js 的下载与使用、MVVM 原理、Vue 的生命周期、双向绑定原理、Vue 事件绑定与组件。

第七章讲解 Vue.js 进阶,主要讲解 Vue.js 的过渡与动画,Vue.js 中的路由和 HTTP 库 axios。

第八章讲解 Vue.js 高级应用,详细介绍了 Vue CLI 脚手架的安装与创建 Vue 项目、vue-cli 的项目结构与项目配置、Element-UI 插件及其应用。

由于编者水平有限,加之编写时间仓促,书中难免有疏漏和不足之处,恳请广大读者和同行批评指正。

编者

第一章

JavaScript 语言基础 1

◎ 1.1 JavaScript 语言概述 1

◎ 1.2 JavaScript 基本语法 6

◎ 1.3 实战与提高 22

◎ 课后练习 27

第二章

JavaScript 语言进阶 28

◎ 2.1 数组 28

◎ 2.2 函数 33

◎ 2.3 对象 41

◎ 2.4 实战与提高 53

◎ 课后练习 58

第三章

BOM 与 DOM 编程 59

◎ 3.1 BOM 59

◎ 3.2 DOM 概述 63

◎ 3.3 事件与事件处理 76

◎ 3.4 实战与提高 90

◎ 课后练习 100

第四章

jQuery 基础 101

◎ 4.1 jQuery 快速入门 101

◎ 4.2 jQuery 选择器 106

◎ 4.3 jQuery 操作 DOM 113

◎ 4.4 jQuery 事件 115

◎ 4.5 jQuery 动画与特效 119

◎ 4.6 实战与提高 122

◎ 课后练习 136

第五章

jQuery 进阶 137

◎ 5.1 Ajax 技术在 jQuery 中的应用 137

◎ 5.2 实战与提高 152

◎ 课后练习 167

第六章

Vue.js 基础 168

◎ 6.1 Vue.js 快速入门 168

	◎ 6.2 Vue.js 基础知识	174
	◎ 6.3 实战与提高	188
	◎ 课后练习	196
第七章	Vue.js 进阶	197
	◎ 7.1 Vue.js 过渡与动画	197
	◎ 7.2 Vue.js 路由	203
	◎ 7.3 axios	211
	◎ 7.4 实战与提高	215
	◎ 课后练习	223
第八章	Vue.js 高级应用	224
	◎ 8.1 VueCLI 脚手架	224
	◎ 8.2 vue-cli 项目结构与配置	229
	◎ 8.3 Element-UI	235
	◎ 8.4 实战与提高	237
	◎ 课后练习	256
参考文献		257

第一章

JavaScript 语言基础



【学习目标】

- 了解 JavaScript 的概念、特点与组成。
- 掌握 JavaScript 的数据类型及类型转换。
- 掌握 JavaScript 的运算符的使用。
- 掌握 JavaScript 的流程控制语句的使用。

1.1 JavaScript 语言概述

JavaScript 是目前 Web 应用程序开发者使用最为广泛的客户端脚本程序语言。对于网页开发而言，HTML、CSS 和 JavaScript 分别代表了结构、样式和行为，结构是网页的骨架，样式是网页的外观，而行为则是网页的交互逻辑。JavaScript 内嵌于 HTML 页面，通过浏览器内置的 JavaScript 引擎进行解释执行，从交互的角度实现业务逻辑和页面控制，提升用户体验。在计算机、手机等设备上浏览的网页，大多数的交互逻辑都是由 JavaScript 实现的。

1.1.1 JavaScript 的发展简史

JavaScript 起源于 Netscape（网景）公司 1995 年推出的 LiveScript 语言。Netscape 和 Sun 公司合作后借用 Java 的名称将其改名为 JavaScript，并在 Netscape Navigator 2 中实现了 JavaScript 脚本规范的第一个版本，即 JavaScript 1.0 版。实际上，JavaScript 与 Java 本质上是两种不同的编程语言。在设计之初，JavaScript 是一种可以嵌入网页中的编程语言，用来控制浏览器的行为，用于解决诸如表单合法性验证等简单而实用的问题，避免为了验证一个表单的有效性，在客户端与服务器端进行反复多次的数据交换，从而节省了时间和网络资源。

1997 年，为了避免无序竞争，同时解决多个 JavaScript 语言版本中语法、特性等各方面的混乱问题，JavaScript 1.1 作为草案提交给 ECMA（欧洲计算机厂商协会），作为中立的 ECMA 开始了标准化脚本语言之路，将其命名为“ECMAScript”。从此，ECMAScript 作为 JavaScript 脚本的基础，开始得到越来越多的浏览器厂商不同程度的支持。

1999 年 6 月，ECMA 发布 ECMA-290 标准，主要添加了使用 ECMAScript 来开发可复用组件内容。

2005 年 12 月，ECMA 发布 ECMA-357 标准（ISO/IEC 22537），主要增加了对扩展标记语言 XML 的有效支持。

2015 年，ECMA 发布新版本 ECMAScript 2015（人们习惯称为“ECMAScript 6”或“ES6”），相比前一版本做了大量的改进。

1.1.2 JavaScript 的特点与组成

1. JavaScript 的特点

JavaScript 是一种基于对象和事件驱动的客户端脚本语言，具有相对的安全性，主要用于创建交互性较强的动态页面。如今，JavaScript 已不再局限于简单的数据验证，而是具备与浏览器窗口及其内容等几乎所有方面进行交互的能力。JavaScript 已经成为一门功能全面的编程语言，能够处理复杂的计算和交互，



也拥有了闭包、匿名函数、元编程等特性。它是一门既简单又复杂的语言，简单是因为学会使用它只需片刻，而复杂是因为要真正掌握它则需要数年时间。JavaScript 具有以下几个特点。

(1) JavaScript 是一种基于对象采用事件驱动方式的解释性脚本语言

JavaScript 是基于对象的脚本编程语言，能通过 DOM（文档结构模型）及自身提供的对象和操作方法来实现所需的功能。JavaScript 采用事件驱动方式，能响应键盘、鼠标及浏览器窗口等事件并执行指定的操作。JavaScript 无须使用专门的编译器进行编译，嵌入 JavaScript 脚本的 HTML 文档在被浏览器载入时逐行地解释，可大量节省客户端与服务器端进行数据交互的时间。

(2) JavaScript 具有实时性和动态性

JavaScript 事件处理是实时性的，无须经服务器就可对客户端的事件做出响应，并用处理结果实时更新目标页面。同时，JavaScript 提供简单高效的语言流程，灵活处理对象的各种方法和属性，及时响应文档页面事件，实现页面的交互性和动态性。

(3) JavaScript 可以跨平台

JavaScript 脚本的正确运行依赖于浏览器，与具体的操作系统无关。只要客户端装有支持 JavaScript 脚本的浏览器，JavaScript 脚本运行结果就能正确反映在客户端浏览器平台上。目前，几乎所有的浏览器都支持 JavaScript，JavaScript 的跨平台性使其在移动端也承担着重要的职责。

(4) JavaScript 具有相对安全性

JavaScript 是客户端脚本，通过浏览器解释执行。它不允许直接访问本地计算机，并且不能将数据存到服务器上，它也不允许对网络文档进行修改和删除，只能通过浏览器实现信息浏览或动态交互，从而有效地防止数据的丢失，确保安全性。

2.JavaScript 的组成

虽然 JavaScript 和 ECMAScript 通常被人们用来表达相同的含义。但现在的 JavaScript 的含义却要比 ECMAScript 中规定的要丰富得多。一个完整的 JavaScript 是由 ECMAScript、DOM、BOM 三部分组成，如图 1-1 所示。



图 1-1 JavaScript 的组成

(1) ECMAScript: JavaScript 的核心

ECMAScript 是一套 JavaScript 语法工业标准，是对实现该标准所规定的各个方面、各类内容的语言描述。它规定了这门语言的语法、类型、语句、对象、关键字、操作符等组成部分。

(2) DOM: 文档对象模型

DOM 是 W3C 组织推荐用于 HTML 的应用程序编程接口，它把整个页面映射为一个多层节点结构。通过 DOM 提供的接口，可以对页面上的各种元素进行位置、大小、颜色等操作。

(3) BOM: 浏览器对象模型

BOM 提供独立于内容的可以与浏览器窗口进行互动的对象结构。开发人员使用 BOM 可以控制浏览器显示页面以外的部分，如弹出框、控制浏览器导航跳转等。从根本上讲，BOM 只处理浏览器窗口和框架。

1.1.3 JavaScript 入门

1.代码书写位置

在 HTML 页面中，可以通过行内式、嵌入式和外链式等三种方式来引入 JavaScript 代码。

(1) 行内式

行内式是将单行或者简单少量的 JavaScript 代码直接写到 HTML 的标签属性中，示例如下：

【示例程序 1-1】 创建一个简单的 HTML 页面 1.html，页面上设置一个按钮。程序代码清单如下：



1-1.html 代码清单

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>行内式</title>
  </head>
  <body>
    <input type="button" value="点击" onclick="alert(行内式)" />
  </body>
</html>
```

上例使用行内式编写 JavaScript 代码，实现了单击“点击”按钮时，弹出一个警示框提示“行内式”，运行结果如图 1-2 所示。



图 1-2 行内式

为了便于网页维护，现代网页开发提倡结构、样式、行为的分离，即分离 HTML、CSS、JavaScript 的代码，因此，在实际开发中并不推荐使用行内式。

(2) 嵌入式

嵌入式是指使用<script>标签包裹 JavaScript 代码，将其直接写在 HTML 文件的<head>或<body>标签中，其语法结构如下：

```
<script type="text/javascript">
  JavaScript 语句
</script>
```

<script>标签的 type 属性用于告知浏览器脚本的类型。由于 HTML5 中该属性默认值为“text/javascript”，因此，在编写时可以省略 type 属性。

【示例程序 1-2】创建 2.html，编写嵌入式 JavaScript 代码。程序代码清单如下：

1-2.html 代码清单

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>嵌入式</title>
    <script type="text/javascript">
      alert("嵌入式");
    </script>
  </head>
  <body>
  </body>
</html>
```

在上述代码中，“alert(“嵌入式”)”为一条 JavaScript 语句，末尾用分号“;”代表该语句结束。



页面打开时，将会弹出一个提示信息为“嵌入式”的警示框，如图 1-3 所示。

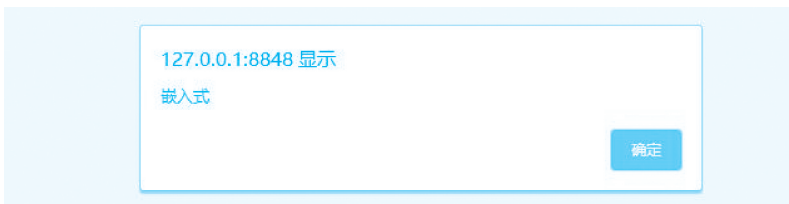


图 1-3 嵌入式

(3) 外链式

外链式是指将 JavaScript 代码写在一个单独的文件中，在 HTML 页面中使用<script>标签的 src 属性引入该文件。这个文件通常使用“js”作为文件扩展名，适合实践开发中需要编写大量、逻辑复杂、特有功能 JavaScript 代码的情况。

外链式有利于 HTML 页面代码结构化。把大段的 JavaScript 代码独立到 HTML 页面之外，既美观也方便了代码的重复使用。需要注意的是，外链式的<script>标签内不能再编写 JavaScript 代码。

【示例程序 1-3】外链式案例。分别创建一个 HTML 文件 3.html 和一个 js 文件 test.js，程序代码清单如下：

1-3.html 代码清单

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>外链式</title>
    <script src=" test.js" ></script>
  </head>
  <body>
  </body>
</html>
```

test.js 代码清单

```
alert("外链式");
```

通过浏览器访问 3.html，将会弹出一个提示信息为“外链式”的警示框，如图 1-4 所示。

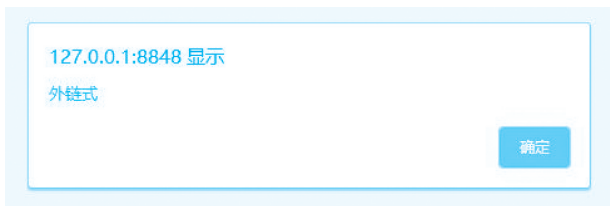


图 1-4 外链式

2. 注释

为了提高代码的可读性，JavaScript 支持单行注释和块级（多行）注释。注释在程序解释时会被 JavaScript 解释器自动忽略。

(1) 单行注释

单行注释以两个斜杠（“//”）开头，示例如下：

```
<script type="text/javascript">
  alert("Hello");    // 输出 Hello（这个单行注释）
</script>
```

(2) 多行注释

多行注释以一个斜杠和一个星号（“/*”）开头，以一个星号和一个斜杠（“*/”）结尾，示例如下：

```
<script type="text/javascript">
/*
    alert("Hello");
*/
</script>
```

其中，2~4 行就是一个多行注释。多行注释中可以嵌套单行注释，但不能再嵌套多行注释。

3.常用输入输出语句

为了实现在网页中与用户的交互效果，方便信息的输入和输出，JavaScript 提供了 4 种常用的输入输出语句，见表 1-1 所列。

表 1-1 常用的输入输出语句

名称	描述
alert（）	浏览器弹出警示框
prompt（）	浏览器弹出输入框，用于用户输入内容
console.log（）	浏览器控制台输出信息
document.write（）	在 HTML 文档页面中输出内容

【示例程序 1-4】常用输入输出语句的使用。程序关键代码如下：

1-4.html 关键 JavaScript 代码

```
<script type="text/javascript">
    alert("弹出警示框");
    prompt("弹出输入框");
    console.log("控制台输出信息");
    document.write("页面输出内容");
</script>
```

通过浏览器访问测试，alert（）语句的效果如图 1-5 所示，rompt（）语句的效果如图 1-6 所示。

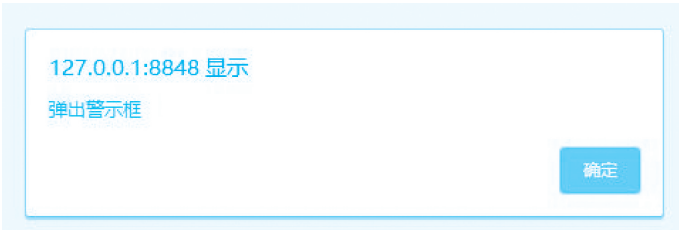


图 1-5 alert（）效果



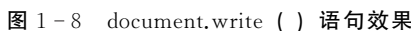
图 1-6 prompt（）语句效果

console.log（）语句的执行结果在浏览器控制台中查看。在 Chrome 浏览器中按 F12 键或在网页



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The message '控制台输出信息' (Control output information) is displayed in the console log. The message is in Chinese and appears to be a system message related to the control output.

document.write () 语句的显示效果如图 1-8 所示。



JavaScript 脚本语言同其他语言一样，有它自身的基本数据类型、表达式、运算符和基本程序框架。JavaScript 提供了六种数据类型用来处理数据和文字，利用变量提供存放信息的地方，使用表达式完成较复杂的信息处理。

变量可以看作存储数据的容器，是程序在内存中申请的一块用于存放数据的空间，每个变量都有唯一的名称。在 JavaScript 中创建变量通常称为“声明”变量。声明变量使用 `var` 关键词，如要一个语句同时声明多个变量，多个变量名之间需用英文逗号分隔，示例如下：

变量声明之后，在没有赋值的情况下，该变量是空的，默认值会被设定为 `undefined`。使用赋值符号（“=”）表示将右边的数据赋值给左边的变量，示例如下：

也可以在声明变量时对其进行赋值，这个过程称为变量的初始化，示例如下：

在对变量进行命名时，需要遵循变量的命名规范，具体如下。

- 在计算机中，不同的数据所需占用的存储空间是不同的，为了充分利用存储空间，便于区分所需内存大小不同的数据，JavaScript 定义了不同的数据类型。ECMAScript 中有 String、Number、Null、

Boolean、Undefined 等五种基本数据类型和 Object 一种复杂（引用）数据类型（ES6 中新增了一种 Symbol 数据类型，这种类型的对象永不相等，即使创建的时候传入相同的值，可以用于解决属性名冲突的问题），如图 1-9 所示。

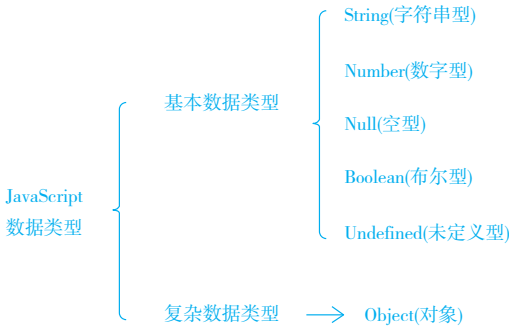


图 1-9 数据类型

(1) 字符串型（String）

String 类型是用来表示文本的数据类型。字符串变量用来存储字符串，可以是零或多个 Unicode 字符、数字组成的字符序列。在 JavaScript 中，字符串由双引号（"）或单引号（'）包裹，示例如下：

```
var str1='';           //空字符串
var str2= 'abc';       //用单引号包括字符串 abc
var str3= "小明";      //用双引号包括字符串小明
```

在 JavaScript 中，用双引号和用单引号包裹的字符串效果完全相同。不过，以双引号开头的字符串必须以双引号结尾，而以单引号开头的字符串也必须以单引号结尾。例如，下面这种字符串的表示法会导致语法错误。

```
var firstName = 'Nicholas";    // 语法错误（左右引号必须匹配）
```

①转义符

String 数据类型包含一些特殊的字符字面量，也叫“转义符号”，用于表示非打印字符或具有其他用途的字符，如换行、Tab 等。比如要在单引号中使用单引号，或在双引号中使用双引号，就需要使用转义字符“\”对其进行转义，常用转义符见表 1-2 所列。

表 1-2 常用转义符

转义符	含义	转义符	含义
\n	换行	\f	进纸
\t	制表	\\	斜杠
\b	空格	\v	跳格（Tab）
\r	回车	\0	Null 字节
\'	单引号（'）	\"	双引号（"）
\xnn	以十六进制代码 nn 表示的一个字符（其中 n 为 0~F）。例如，\x41 表示 "A"	\unnnn	以十六进制代码 nnnn 表示的一个 Unicode 字符（其中 n 为 0~F）。例如，\u03a3 表示希腊字符 Σ

这些字符字面量可以出现在字符串中的任意位置，并被作为一个字符来解析。

【示例程序 1-5】转义符的使用。程序关键代码如下：

1-5.html 关键代码

```
var str1= '\m OK ';      //输出结果:Im OK
var str2= "\"TOM\"";     //输出结果:"TOM"
var str3= 'I am a\n girl'; //输出结果:I am a(换行) girl
var str4= 'D:\\JavaScript\\'; //输出结果:D:\\JavaScript\\
var str5= '\x41BC';      //输出结果:ABC
```

②字符串长度



任何字符串的长度都可以通过访问其 length 属性取得。

【示例程序 1-6】访问字符串中的字符，程序关键代码如下：

1-6.html 关键代码

```
var text = "This is the letter sigma: \u03a3.";
alert (text.length);           // 输出 28
```

这个例子中的变量 text 有 28 个字符，其中 6 个字符长的转义序列 “\u03a3” 表示为 1 个字符，一个空格算一个字符。

③访问字符串中的字符

可以使用索引位置 “[index]” 来访问字符串中的每个字符。字符串的索引从 0 开始，即字符串的第一个字符索引值为 [0]，第二个为 [1]，以此类推。如果超出了字符串的长度最大值，则返回 undefined。

【示例程序 1-7】访问字符串中的字符，程序关键代码如下：

1-7.html 关键代码

```
var text = "HelloWord";
console.log (text [0] );           //输出结果：H
console.log (text [1] );           //输出结果：e
console.log (text [18] );          //输出结果：undefined
```

(2) 数字型 (Number)

Number 类型是最基本的数据类型，包括整型值和浮点型值，可以用来保存整数或小数。所谓浮点型值，就是该数值中必须包含一个小数点，并且小数点后面必须至少有一位数字，示例如下：

```
var num1=20;                       //整型值
var num2= 3.14159;                 //浮点型值
```

①数字型的进制

一般情况下，最基本的数值字面量格式是十进制整数，除了以十进制表示外，整数还可以通过八进制（在数字开头加 0 表示，由 0~7 组成）或十六进制（在数字开头加 0x 表示，由 0~9 和 a~f 组成）的字面值来表示。

【示例程序 1-8】数字型的进制。程序关键代码如下：

1-8.html 关键代码

```
var num1=07;                       //八进制表示的 7
var num2= 010;                     //八进制表示的 8
var num3=0x9;                      //十六进制表示的 9
var num4= 0x10;                    //十六进制表示的 16
```

②数字型范围

数字型的最大值和最小值可以用如下代码获取，结果用科学记数法表示。

```
alert (Number.MAX_VALUE);          //输出结果：1.7976931348623157e+308
alert (Number.MIN_VALUE);          //输出结果：5e-324
```

③数字型的三个特殊值

数字型的三个特殊值分别为 Infinity（无穷大）、-Infinity（无穷小）和 NaN（Not a number，非数值），示例如下：

```
console.log (Number.MAX_VALUE * 2); //输出结果：Infinity
console.log (Number.MIN_VALUE * 2); //输出结果：-Infinity
console.log ('abc'+10);              //输出结果：NaN
```

④isNaN

函数 isNaN（）接受一个参数，用于判断这个参数是不是非数字值，返回一个布尔值，如果参数是数字则返回 false，否则返回 true，示例如下：

```
console.log (isNaN (10) );           //输出结果：false
console.log (isNaN ('abc') );        //输出结果：true
```

(3) 布尔型 (Boolean)



Boolean 类型常用于逻辑判断，只有 true 和 false 两个字面值来表示事物的“真”与“假”。由于 JavaScript 中大小写敏感，true 和 false 只有为小写时才表示为布尔型。

```
var bool1=true;           //为变量 bool1 赋一个布尔值 true
var bool2=false;         //为变量 bool2 赋一个布尔值 false
```

(4) 空型 (Null)

Null 类型是只有一个值的数据类型，这个特殊的值是 null，用于表示一个不存在或无效的对象或地址。由于 JavaScript 中大小写敏感，变量的值只有为小写的 null 时才表示为空型 (Null)。

```
var a=null;               //变量 a 的值为 null
```

(5) 未定义型 (Undefined)

Undefined 类型也是只有一个值的数据类型，这个特殊的值是 undefined。如果一个变量声明后没有赋值，则变量的值默认为 undefined。任何变量都可以通过设置为 undefined 来清空。与 null 不同的是 undefined 表示没有为变量赋值。

```
var b;                   //声明变量 b，未赋值，此时 b 的值就是 undefined
var c=undefined;         //给变量 c 赋值为 undefined
```

2. 数据类型检测

在开发过程中，一般利用 typeof 操作符来检测变量或值的数据类型。typeof 操作符以字符串的形式返回当前操作数的数据类型，可以使用比较运算符“==”来判断 typeof 返回的检测结果是否符合预期。

对变量或值调用 typeof 运算符，将返回下列值之一：

- (1) undefined：变量或值是 Undefined 类型。
- (2) boolean：变量或值是 Boolean 类型。
- (3) number：变量或值是 Number 类型。
- (4) string：变量或值是 String 类型。
- (5) object：变量或值是一种引用类型或 Null 类型。

【示例程序 1-9】数据类型检测。程序关键代码如下：

1-9.html 关键代码

```
console.log(typeof "John");           //输出结果：string
console.log (typeof 20);               //输出结果：number
console.log (typeof false);           //输出结果：boolean
console.log (typeof undefined);       //输出结果：undefined
console.log (typeof null);            //输出结果：object
var a=12;                             //声明变量 a 并赋值
var b='34';                           //声明变量 b 并赋值
var sum=0;                            //声明变量 sum 并赋值
sum=a+b;                             //变量进行相加运算
console.log (sum);                    //输出结果：1234
console.log (typeof a);               //输出结果：number
console.log (typeof b);               //输出结果：string
console.log (typeof sum);             //输出结果：string
console.log (typeof sum=='string');   //输出结果：true
console.log (typeof sum=='number');   //输出结果：false
```

从程序运行结果可以看到，typeof sum 的返回结果是 string，在与字符串 string 比较时，结果为 true，说明 sum 是 string 类型（数字型 a 和字符串型 b 做加运算时，进行的是字符连接操作，运算结果为字符串“1234”）。另外，需要注意的是 typeof 检测 null 值时返回的结果是 object，而不是 null，这是最初 JavaScript 遗留的历史问题，后来一直被沿用下来。

3. 数据类型转换

数据类型转换，就是把某一种数据类型转换为另一种数据类型。对两个数据进行操作时，若其数据类型不同，则需要进行数据类型转换。JavaScript 数据类型转换有隐式转换和强制转换两种方式。隐



式转换是基于强制转换之上，自动进行的转换；强制转换主要指通过 Number ()、String () 和 Boolean () 三个函数，手动将各种类型的值分别转换成数字、字符串或布尔值。

(1) 数字型转换为字符串型

在开发中，将数字型转换为字符串型，有字符串拼接、toString () 和 String () 三种常用的方式，下面举例来说明。

【示例程序 1-10】数字型转换为字符串型，程序关键代码如下：

1-10.html 关键代码

```
//声明一个变量 num
var num=3.14;
//方式 1: 利用字符串拼接 “+”
var str1=num+'';
console.log (str1, typeof str1);           //输出结果: 3.14 string
//方式 2: 利用 toString () 转换为字符串
var str2=num.toString ();
console.log (str2, typeof str2);           //输出结果: 3.14 string
//方式 3: 利用 String () 强制转换为字符串
var str3= String (num);
console.log (str3, typeof str3);           //输出结果: 3.14 string
```

从程序运行结果可以看到，console.log () 可以同时输出多个值，值之间用 “,” 分隔。其中，方式 1 为隐式转换，是最常用的方式。隐式转换是自动发生的，当操作的两个数据类型不同时，JavaScript 会根据既定的规则针对不同的数据类型进行自动转换。方式 2 和方式 3 属于显式转换，即强制转换，为开发人员进行的主动转换。



知识拓展

- null 和 undefined 无法使用 toString () 方式进行转换。
- 对于数字型变量，可以在 toString () 的括号中传入参数来进行进制转换，如变量 a 的值为 9，则 a.toString (2) 表示将 9 转换为二进制，结果为 1001。

(2) 数据转换为数字型

将数据转换为数字型，有四种常见的方式。除了隐式转换外，有 Number ()、parseInt () 和 parseFloat () 等三个函数将非数值转换为数值。Number () 函数可以用于任何数据类型，而另两个函数则专门用于把字符串转换成数值。parseInt () 函数在转换字符串时，更多的是看其是否符合数值模式。转换纯数字时，数字前面如果有 “+” 会看成正数，“-” 会看成负数，前面有 0 会自动忽略，后面跟有单位会自行去掉，下面举例来说明。

【示例程序 1-11】数据转换为数字型案例，程序关键代码如下：

1-11.html 关键代码

```
//方式 1: 利用算术运算符 (-、*、/) 进行隐式转换
console.log ("18"-1);           //输出结果: 17
//方式 2: 利用 Number () 进行转换
console.log (Number ("3.14159")); //输出结果: 3.14159
//方式 3: 利用 parseInt () 将字符串转换为整数
console.log (parseInt ("23.15")); //输出结果: 23
//方式 4: 利用 parseFloat () 将字符串转换为小数
console.log (parseFloat ("23.15")); //输出结果: 23.15
var num1 = parseInt ("1234blue"); //输出结果: 1234
```




```
var num2 = parseInt("");           //输出结果:NaN
var num3 = parseInt("03.14");      //输出结果:3
var num4 = parseInt(22.5);          //输出结果:22
var num5 = parseInt("-70");         //输出结果:-70
var num6 = parseInt("70");          //输出结果:70
var num7 = parseInt("100px");       //输出结果:100
var num8 = parseInt("a100");        //输出结果:NaN
```

另外, 还可以通过 `parseInt()` 函数的第 2 个参数值设置进制的转换, 如将字符 “F” 转换为十六进制数, 代码如下:

```
var num = parseInt("F", 16);        //输出结果:15
```

(3) 数据转换为布尔型

使用 `Boolean()` 函数可以对任何数据类型的值进行布尔型数据转换, 结果会返回一个 `Boolean` 值, 该值取决于要转换数据的数据类型及其实际值, 见表 1-3 所列。

表 1-3 数据转换为布尔型

数据类型	转换为 true 的值	转换为 false 的值
Boolean	true	false
String	任何非空字符串	" " (空字符串)
Number	任何非零数字值 (包括无穷大)	0 和 NaN
Object	Object	null
Undefined		undefined

从表 1-3 可以看出, 数据转换为布尔型时, 代表空、否定的值 (如空字符串、0、NaN、null 和 undefined 等) 会被转换为 false, 其余则转换为 true。

1.2.3 运算符与表达式

1. 表达式

JavaScript 表达式是用运算符将数据 (如常量、变量、函数等) 按一定的规则连接起来的、有意义的式子。表达式是一个会产生结果值的式子, 它的结果值有多种类型 (如数值型、字符串型或布尔型等)。根据结果值的类型, 可以将 JavaScript 表达式分为算术表达式、字符串表达式和逻辑表达式等几种类型, 示例如下:

```
1+2*3           //算术表达式
"abc"+"123"      //字符串表达式
9<10&&8.3>=2     //逻辑表达式
```

表达式是一个相对的概念, 在表达式中可以含有若干个子表达式, 比如 $1+2*3$ 这个表达式中 $2*3$ 就是一个子表达式, 表达式中的一个常量或变量也可以看作是一个表达式。当一个表达式含有多个运算符时, 这些运算符会按照运算符的优先级进行运算。

2. 运算符

在程序中, 经常会对数据进行运算, 为此 JavaScript 提供了多种类型的运算符。运算符也称为操作符, 是用于实现赋值、比较或执行算术运算等功能的符号。根据运算符的作用, 可以将运算符大致分为算术运算符、递增递减运算符、逻辑运算符、比较运算符、赋值运算符和三元运算符六类。

(1) 算术运算符

算术运算符用于对两个变量或值进行算术运算, 见表 1-4 所列。

表 1-4 算术运算符

运算符	运算	示例	结果
+	加	$2+3$	6
-	减	$6-2$	4
*	乘	$4*8$	32
/	除	$15/3$	5
%	取模	$5\%3$	2



算术运算符在实际应用中需要注意以下几点：

- ①在算术运算时可以使用“+”和“-”表示正数或负数。如 $(+5) + (-2)$ 的运算结果为3。
- ②在进行四则混合运算时，需遵循“先乘除后加减”的数学运算规则。
- ③取模运算多用于判断一个数能否被整除的情况。在进行取模运算时，运算结果的正负取决于取模运算符“%”左边的那个数的符号。如， $(-10) \% 3 = -1$ ，而 $10 \% (-3) = 1$ 。

④在开发中，应尽量避免对浮点数进行相等运算，即不要直接判断两个浮点数是否相等。因为，这样的操作有可能会因JavaScript的精度问题而导致结果的偏差。

(2) 递增递减运算符

递增(++)、递减(--)运算符属于一元运算符，只对一个表达式进行操作，可以快速地对变量的值进行递增和递减操作。使用递增和递减运算符需区分前置型和后置型。前置型是符号写在要操作变量的前面，后置型是符号写在要操作变量的后面。递增和递减运算符在对布尔型数据进行操作时，会将布尔值true当作1，false当作0。递增递减运算符具体运算规则见表1-5所列。

表 1-5 递增递减运算符

运算符	运算	规则	示例	结果
++	递增（前置）	先运算再取值	a=2, b=++a;	a=3; b=3
++	递增（后置）	先取值再运算	a=2, b=a++;	a=3; b=2
--	递减（前置）	先运算再取值	a=2, b=--a;	a=1; b=1
--	递减（后置）	先取值再运算	a=2, b=a--;	a=1; b=2

从表1-5可以看出，前置型的运算规则就是先运算再取值，如++num前置递增是先将num的值加1，后取num的值进行其他操作；后置型的运算规则就是先取值再运算，num++后置递增是取num的值进行其他操作，后将num的值加1。

【示例程序 1-12】递增递减运算符应用案例。程序关键代码如下：

1-12.html 关键代码

```
var num = 10;
console.log (++num + 10);           //输出结果：21
var num = 10;
console.log (10 + num++);           // 输出结果：20
var a=10;
var b=++a + 2;                       //b=11+2, a=11
var c=b+++ 2;                         //c=13+2, b=14
var d=c+++++ a;                       //d=15+12, c=16, a=12
```

(3) 逻辑运算符

逻辑运算符用于判断变量或值之间的逻辑关系，常用于多个条件的判断，其返回值是布尔值，见表1-6所列。

表 1-6 逻辑运算符

运算符	运算	示例	结果
&&	与	a&& b	当a与b都为true时，结果为true，否则为false
	或	a b	当a与b中至少有一个为true时，结果为true，否则为false
!	非	! a	当a为true时，结果为false；当a为false时，结果为true

【示例程序 1-13】我们设定x=6且y=3，解释以上逻辑运算符。

1-13.html 代码清单

```
var a= (x < 10 && y > 1)
console.log (a);                     // 输出结果：true
var a=(x >10 || y> 5)
console.log (a);                     // 输出结果：false
console.log (! a);                   // 输出结果：true
```



知识拓展

逻辑与运算符短路现象

• 逻辑与运算符短路现象。语法格式为：表达式 1 && 表达式 2。当逻辑与运算的第一个操作数（表达式 1）的值是 false 时，直接返回 false，不再对第二个操作数进行计算（发生短路）；如果表达式 1 的值为 true，则返回表达式 2 的值。

• 逻辑或运算符短路现象。语法格式为：表达式 1 || 表达式 2。当逻辑或运算的第一个操作数（表达式 1）的值是 true 时，直接返回 true，不再对第二个操作数进行计算（发生短路）；如果表达式 1 的值为 false，则返回表达式 2 的值。

(4) 比较运算符

比较运算符用于对两个变量或数值进行比较，其返回值是布尔值，见表 1-7 所列。

表 1-7 比较运算符

运算符	运算	示例	结果
==	等于	5==8	false
===	绝对等于 (值和类型均相等)	5=== '5'	false
!=	不等于	5!=4	true
!==	不绝对等于	5!== '5'	true
>	大于	5>8	false
>=	大于等于	5>=8	false
<	小于	5<8	true
<=	小于等于	5<=8	true

比较运算符在实际应用中需要注意以下两点。

①不同的数据类型在进行“==”和“!=”比较的时候，会自动将其转换成相同类型的数据再进行比较。如字符串“123”与数字 123 进行比较，首先会将字符串“123”转换为数字型，再与 123 进行比较。

②运算符“==”和“!=”与运算符“===”和“!==”在进行比较时，前两个运算符只需要比较数据的值是否相等，不考虑数据的类型是否相同，而后两个运算符不仅要比较值是否相等，还要比较数据的类型是否相同。

(5) 赋值运算符

简单的赋值运算符用等于号（“=”）表示，其作用就是把右侧的值赋给左侧的变量。除此之外，还可以使用“+=”相加并赋值、“*=”相乘并赋值、“-=”相减并赋值等赋值运算符，常用的赋值运算符见表 1-8 所列。

表 1-8 赋值运算符

运算符	运算	示例	结果
=	赋值	x=5	x=5
+=	加并赋值	x=5, x+=2;	x=7
-=	减并赋值	x=5, x-=2;	x=3
=	乘并赋值	x=5, x=2;	x=10
/=	除并赋值	x=5, x/=2;	x=2.5
%=	取余并赋值	x=5, x%=2;	x=1
+=	连接并赋值	x="abc", x+="d" ;	x="abcd"



运算符“+=”在执行时，若其操作数都是非字符型数据，则表示相加后再赋值，否则表示先拼接字符串再赋值，下面举例说明。

【示例程序 1-14】赋值运算符案例。程序关键代码如下：

1-14.html 关键代码

```
var a=2,b="2";
a+=3;
b+=3;
console.log(a,b);           //输出结果为 5 "23"
```

(6) 三元运算符

三元运算符又称为“三目运算符”，是一种需要三个操作数的运算符，会根据不同的条件，执行不同的操作或返回不同的值，具体语法结构如下：

条件表达式？操作1：操作2

三元运算过程是先判断条件表达式的值，如果条件表达式为 true，则执行操作 1 并返回执行结果，否则执行操作 2 并返回执行结果。下面举例说明。

【示例程序 1-15】利用三元运算符比较两个整数的大小。程序关键代码如下：

1-15.html 关键代码

```
var a=10;
var b=20;
a>b? alert("a大!"); alert("b大!");
```

上面代码先声明 a 和 b 两个变量并分别赋值为 10 和 20。然后对“a>b”进行判断，得出判断结果为 false，接着执行后面的 alert() 函数，弹出警示框输出“b大!”，程序运行效果如图 1-10 所示。

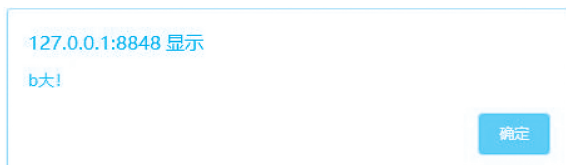


图 1-10 比较结果

3. 运算符的优先级

JavaScript 中的运算符优先级是一套规则，该规则在计算表达式时控制运算符执行的顺序，具有较高优先级的运算符先于较低优先级的运算符执行。表 1-9 按 JavaScript 运算符的优先级从最高到最低排列。

表 1-9 运算符优先级

运算符	描述
()	圆括号
., []	字段访问、数组下标、函数调用以及表达式分组
++ -- - ~ ! delete new typeof void	一元运算符、返回数据类型、对象创建、未定义值
* / %	乘法、除法、取模
+ - +	加法、减法、字符串连接
<< >> >>>	移位
< <= > >= instanceof	小于、小于等于、大于、大于等于、instanceof
== != === !==	等于、不等于、严格相等、非严格相等
&	按位与
^	按位异或
	按位或
&&	逻辑与

续表

运算符	描述
	逻辑或
?:	条件
= op=	赋值、运算赋值
,	多重求值

圆括号可以调高其内部运算符的优先级，一般用来改变运算符优先级法则所决定的求值顺序。为复杂的表达式适当地添加圆括号，可避免复杂的运算符优先级法则，让代码逻辑更清晰，避免错误的发生，示例如下：

```
z = 78 * (96 + 3 + 45)
```

在上面的表达式中有四类运算符：=、*、（）和+。根据运算符优先级的规则，首先对圆括号内的表达式求值，圆括号中有两个加法运算符，因为两个加法运算符具有相同的优先级，所以表达式“(96 + 3 + 45)”的计算顺序为从左到右，先将96和3相加，再将加运算的结果与45相加，得到的结果为144，然后是乘法运算78乘以144，得到结果为11232，最后是赋值运算，将11232赋值给z。

1.2.4 流程控制

在一个程序执行的过程中，代码的执行顺序对程序的结果是有直接影响的。很多时候需要通过控制代码的执行顺序来实现要完成的功能。简单来说，流程控制就是用来控制代码按照一定结构顺序来执行。

流程控制主要有顺序结构、分支结构和循环结构三种，分别代表三种代码执行的顺序，如图1-11所示。

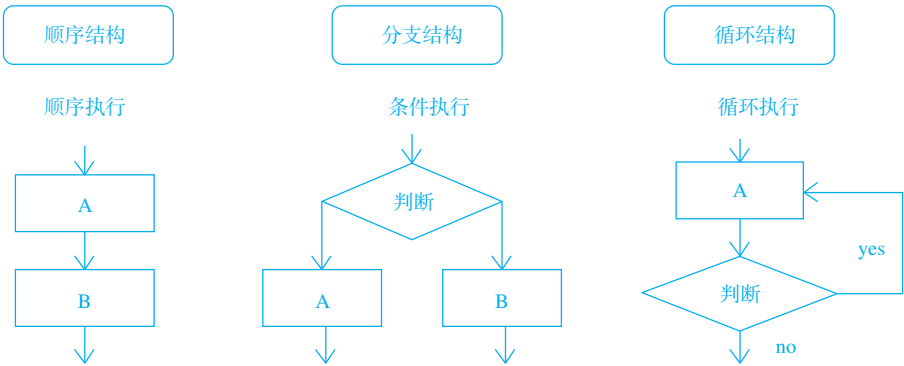


图 1-11 流程控制

(1) 顺序结构是程序中最简单、最基本的流程控制，它没有特定的语法结构，程序会按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。

(2) 分支结构是从上到下执行代码的过程中，根据不同的条件，执行不同的分支路径代码，从而得到不同的结果。常用的分支结构语句有 if、if...else、if...else if 和 switch。

(3) 循环结构是通过循环语句反复执行一段代码。常用的循环结构语句有 for、while 和 do...while。

1.分支结构

分支结构根据给出的条件来决定是否执行某个分支代码，分支结构可分为单分支语句（if）、双分支语句（if...else）和多分支语句（if...else if 和 switch）进行讲解。

(1) if 语句

if 语句也称为单分支语句、条件语句，常用于对一个变量进行判断。其语法结构如下：

```
if ( 条件表达式 ) {  
    //当条件表达式为 true 时执行的代码  
}
```



if 语句执行时，会先对括号里的条件表达式进行求值判断，如果条件表达式的值为 true，则执行 if 后的语句块，否则不会执行 if 后的语句块。if 语句的执行流程图如图 1-12 所示。

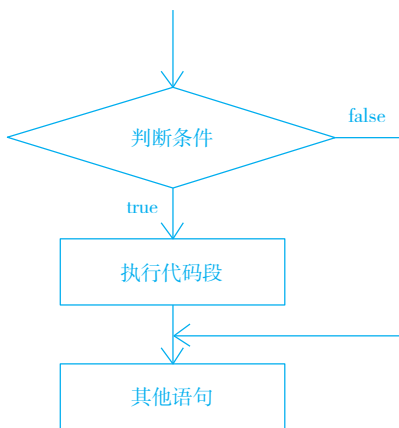


图 1-12 if 语句流程图

【示例程序 1-16】使用 if 语句实现当输入年龄大于等于 18 时，控制台输出“已成年”。程序关键代码如下：

1-16.html 关键代码

```

var age=parseInt(prompt("请输入你的年龄:"));
if (age>=18) {
    console.log("已成年");
}
  
```

(2) if...else 语句

if...else 语句也称为“双分支语句”，当满足某种条件时执行某种操作，否则执行另外一种操作。其语法结构如下：

```

if ( 条件表达式 ) {
    //当条件表达式为 true 时执行的代码
} else {
    //当条件表达式为 false 时执行的代码
}
  
```

if...else 语句执行时，会先对括号里的条件表达式进行求值判断，如果该值为 true，则执行 if 后的语句块，否则执行 else 后的语句块。if...else 语句的执行流程图如图 1-13 所示。

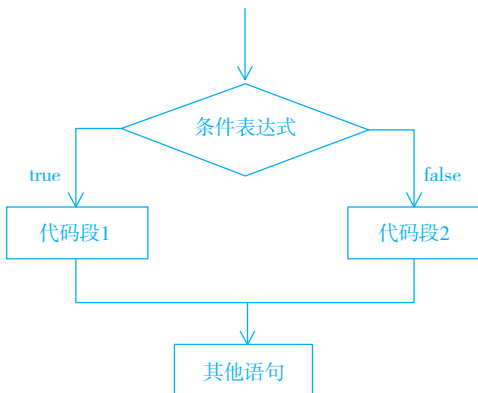


图 1-13 if...else 语句流程图

【示例程序 1-17】获取两个数中的更大值。程序关键代码如下：



1-17.html 关键代码

```
var num1=100;
var num2=20;
if (num1>num2) {
    console.log (num1);
} else {
    console.log (num2);
}
```

(3) if...else if 语句

if...else if 语句也称为“多分支语句”，使用该语句可针对不同的情况进行不同的处理，主要用来选择多个代码块之一来执行。其语法结构如下：

```
if ( 条件 1 ) {
    //当条件 1 为 true 时执行的代码
} else if ( 条件 2 ) {
    //当条件 2 为 true 时执行的代码
} else if ( 条件 3 ) {
    //当条件 3 为 true 时执行的代码
} else {
    //当条件 1、条件 2 和条件 3 都不为 true 时执行的代码，最后默认执行语句
}
```

if...else if 语句执行时，会从上到下依次对条件表达式进行求值判断，如果值为 true，则执行当前语句；如果值为 false，则继续向下判断，依次类推。如果所有的条件都为 false，则执行最后一个 else 里的语句（在这种情况下，如果 if...else if 语句的最后没有 else，则表示什么都不执行）。需要指出的是，在 if...else if 语句中，最多只有一个代码块会被执行，一旦某个代码块执行完，则代表结束该 if...else if 语句。if...else if 语句的执行流程图如图 1-14 所示。

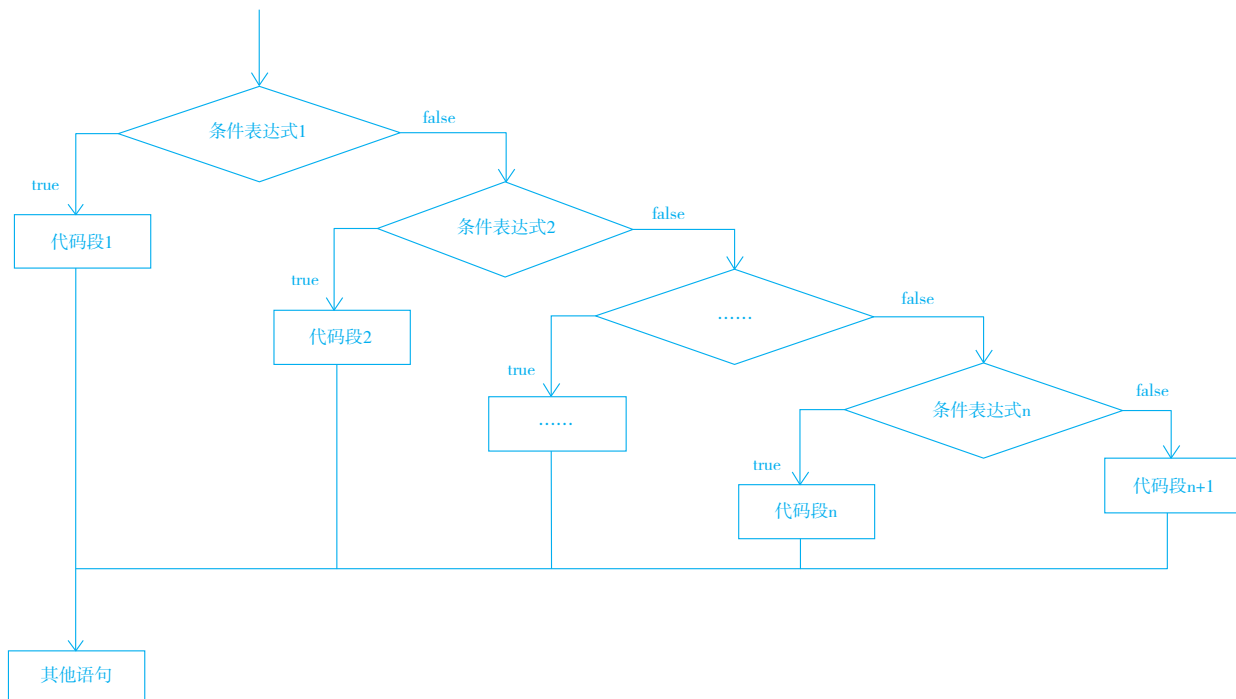


图 1-14 if...else if 语句流程图

【示例程序 1-18】 利用 if...else if 语句，对一个学生的考试成绩进行等级的划分。如果成绩是在 90 分（含 90 分）以上的，则显示为优秀，成绩大于等于 80 分的为良好，成绩大于等于 70 分的为中等，成绩大于等于 60 分的为及格，成绩小于 60 分的为不及格。程序关键代码如下：

1-18.html 关键代码

```
var score = parseInt(prompt("请输入您的成绩:(0—100 之间)"));
if (!isNaN(score) && score <= 100) {
    if (score >= 90) {
        console.log("您的成绩为优秀");
    } else if (score >= 80) {
        console.log("您的成绩为良好");
    } else if (score >= 70) {
        console.log("您的成绩为中等");
    } else if (score >= 60) {
        console.log("您的成绩为及格");
    } else {
        console.log("您的成绩为不及格");
    }
} else {
    console.log("您输入的成绩有误!");
}
```

需要注意的是，if…else if 语句在使用时，“else if”中间要有空格，否则程序执行后会显示语法错误。

(4) switch 语句

switch 语句也是多分支语句，功能与 if…else if 语句类似。它是基于不同的条件来执行不同的代码，常用于分支条件值特定（常量）的情况。其语法结构如下：

```
switch (表达式) {
    case 常量 1:
        语句 1;
        break;
    case 常量 2:
        语句 2;
        break;
    ...
    case 常量 n:
        语句 n;
        break;
    default:
        语句 n+1;
        break;
}
```

switch 语句执行时，首先计算表达式的值，然后将获得的值与关键字 case 中的常量依次进行比较，如果存在匹配全等（“===”），则执行与该 case 关联的代码块，并在遇到 break 时停止，结束整个 switch 语句；如果所有 case 的值都和表达式的值不匹配，则执行 default 里的代码。需要注意的是，代码里的 break 可以省略，如果省略，代码会继续执行下一个 case。switch 语句在比较值时使用的是全等操作符，不会发生类型转换（如字符串‘10’不等于数字 10）。switch 语句的执行流程图如图 1-15 所示。

【示例程序 1-19】利用 switch 语句对一个学生的成绩级别进行判定，如果成绩等级是 A，则提示分数在 90 分以上，如果成绩等级是 B，则提示分数在 80 分以上，如果成绩等级是 C，则提示分数在 70 分以上，如果成绩等级是 D，则提示分数在 60 分以上，否则提示不及格。程序关键代码如下：

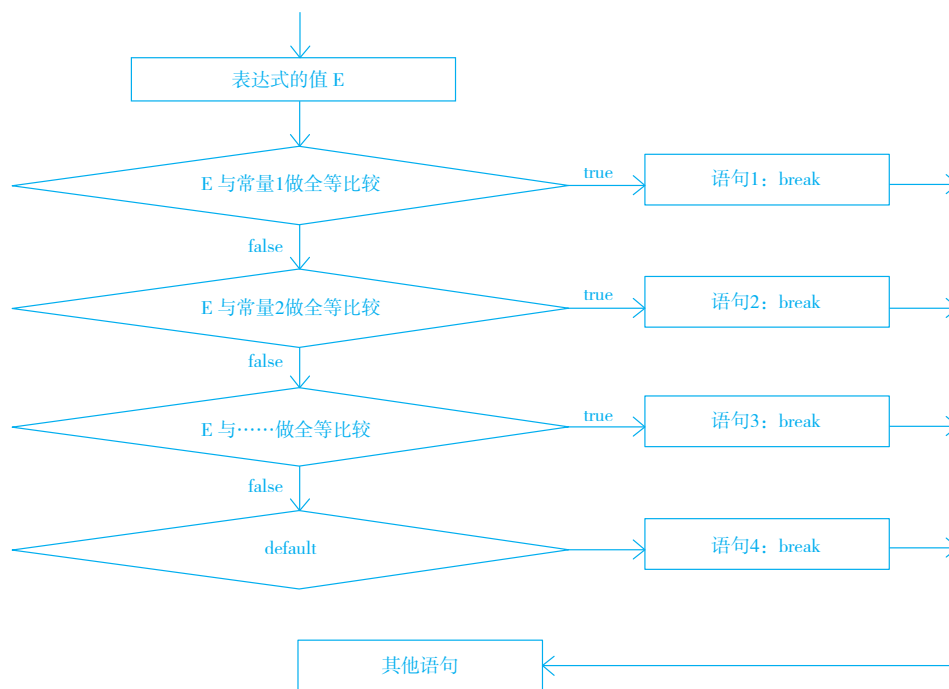


图 1-15 switch 语句流程图

1-19.html 关键代码

```
var jiBie = prompt("请输入您的成绩级别:(大写字母 A-D)");
switch (jiBie) {
    case "A" :
        console.log("分数在 90 分以上的");
        break;
    case "B" :
        console.log("分数在 80 分以上的");
        break;
    case "C" :
        console.log("分数在 70 分以上的");
        break;
    case "D" :
        console.log("分数在 60 分以上的");
        break;
    default :
        console.log("不及格");
}
```

一般情况下，switch 语句和 if...else if 语句可以相互替换。当分支条件值特定时，一般选用 switch 语句；而当分支条件值为一个范围时，一般选用 if...else if 语句。

2. 循环结构

循环结构用来实现重复执行一段代码。例如，要连续输出 100 句相同的语句，如果不使用循环结构，需要编写 100 行代码才能实现，而使用循环语句，只需要几行代码就可以实现。JavaScript 常用的循环语句有 for、while 和 do...while 三种。

(1) for 语句

for 循环语句是最常用的循环语句，一般在循环次数确定的情况下使用。其语法结构如下：

```
for (初始化表达式; 循环条件; 操作表达式) {
    // 循环体
}
```



在 for 循环语句中，表达式之间需用分号分隔，被重复执行的一段语句（代码块）称为“循环体”。这段代码是否能够重复执行，取决于循环条件的真假。“初始化表达式”一般用来初始化（如循环计数器的值），“循环条件”决定了下一次循环是否能够继续，“操作表达式”是在每次循环体执行过后，才被执行的代码，通常利用递增或递减对计数器变量进行更新。需要特别指出的是，初始化表达式只会执行一次，for 语句的流程图如图 1-16 所示。

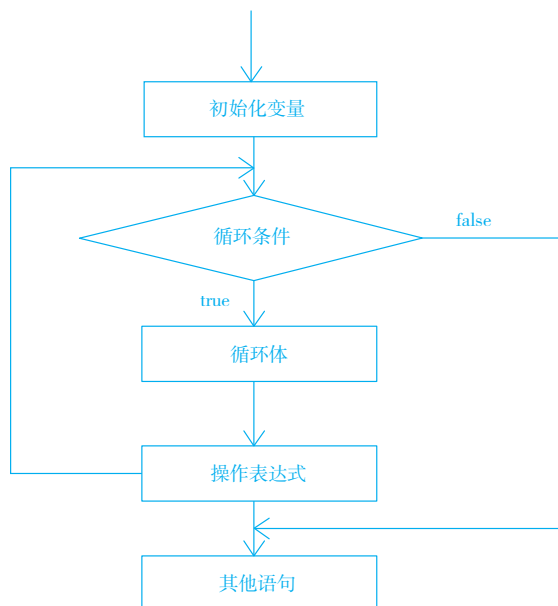


图 1-16 for 语句流程图

【示例程序 1-20】使用 for 语句输出 1~100 的整数，并求出这些整数的和。程序关键代码如下：
1-20.html 关键代码

```

var sum = 0;
for (var i=1; i<=100; i++) {
    console.log (i);
    sum += i;
}
console.log (sum); //计算结果：5050
  
```

在上面的 for 语句中，首先执行“var i=1”，初始化循环计数变量 i，然后判断“i<=100”是否成立，如果成立则执行循环体，否则结束循环语句。循环体“console.log (i); sum += i;”分别表示输出当前变量 i 的值和将当前 i 的值叠加到 sum 中。对上面的代码，可以将 sum 初始化放到 for 循环的初始化表达式中，具体如下：

```

for (var sum = 0, i=1; i<=100; i++) {
    .....
}
  
```

(2) while 语句

while 语句一般用来解决无法确认循环次数的情况，其语法结构如下：

```

while (循环条件表达式) {
    //循环体
    .....
}
  
```

while 语句在执行时，先对循环条件表达式进行求值判断，如果值为 true，则执行循环体，循环体执行完毕以后，继续对循环条件表达式进行判断，如果为 true，则继续执行循环体，以此类推；如果值为 false，则终止循环。while 循环语句流程图如图 1-17 所示。

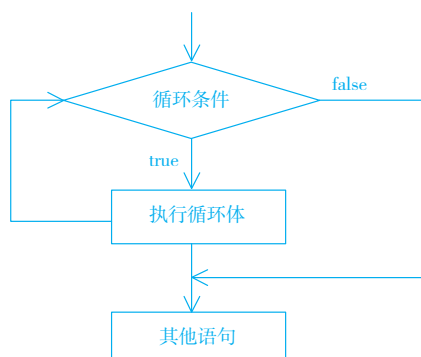


图 1-17 while 语句流程图

【示例程序 1-21】用 while 语句求 1~100 所有整数的和。程序关键代码如下：

1-21.html 关键代码

```
var i = 1;
var sum = 0;
while (i <= 100) {           // 判断条件
    sum += i;                // 循环体
    i++;                     // 自增
}
console.log (sum);           //计算结果：5050
```

(3) do...while 语句

do...while 语句和 while 语句非常像，二者经常可以相互替代。do...while 语句的语法结构如下：

```
do {
    // 循环体;
} while (循环条件);
```

从 do...while 的语法结构可以看出，do...while 语句的特点是不管循环条件是否成立，都会先执行一次循环体。因此，do...while 语句的循环体至少会被执行一次，这是它与 while 语句的最大区别（while 语句是先判断循环条件，再执行循环体）。do...while 语句的流程图如图 1-18 所示。

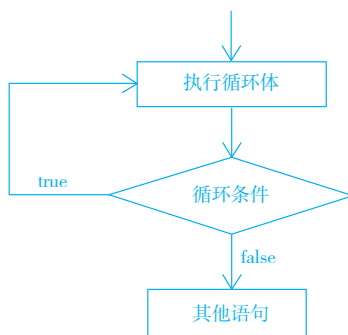


图 1-18 do...while 语句流程图

【示例程序 1-22】用 do...while 语句求 1~100 所有整数的和。程序关键代码如下：

1-22.html 关键代码

```
var i = 0;
var sum = 0;
do {
    sum += i;
    i++;
} while (i <= 100);           //循环条件，注意后面的分号
console.log (sum);           //计算结果：5050
```



3. 跳转语句

跳转语句使得 JavaScript 的执行可以从一个位置跳转到另一个位置，用于实现程序执行过程中的流程跳转，常用的跳转语句有 break 语句和 continue 语句。其中，break 语句是跳转到循环或者其他语句的结束，表示立即跳出整个循环（当前循环），即这一层的循环结束，并开始执行循环后面的内容，而 continue 语句是终止本次循环的执行，并开始下一次循环的执行（提前结束本次循环，不影响下一次循环的执行）。

【示例程序 1-23】break 语句和 continue 语句的使用案例。程序关键代码如下：

1-23.html 关键代码 1

```
//求 200-300 之间第一个能被 7 整除的数。
for (var i=200; i<=300; i++) {
    if (i % 7 == 0) {
        console.log (i);           //计算结果：203
        break;
    }
}
```

当 i=203 时，i 能被 7 整除，因此，先在控制台输出当前 i 值，然后执行 break 语句，跳出当前 for 循环，循环结束。如果将程序中的 break 语句修改为 continue 时，则表示结束当次循环（不会结束当前循环），并继续执行下一次循环，将会得到 200~300 中的所有能被 7 整除的数。

1-23.html 关键代码 2

```
//求 1-100 之间所有不能被 7 整除的整数的和。
var s = 0;
for (var i = 0; i < 100; i++) {
    if (i % 7 == 0) {
        continue;
    }
    s += i;
}
console.log (s);           //计算结果：4215
```

1.3 实战与提高

1.3.1 求三个数的最大数

在进行数学运算时，有时候需要进行数值的比较。下面通过简单的三元运算符代码来求三个数的最大数。

本案例需求分析如下：

- (1) 通过对话框分别输入三个整数。
- (2) 运用三元运算符做比较。
- (3) 输出最大值。

具体代码清单如下：

1-24.html 代码清单

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>三元运算符比较最大值</title>
</head>
```



```
<body>
  <script type="text/javascript">
    // 使用三元运算符结构:表达式? 操作一:操作二;
    // a > b ? alert(a) : alert(b);
    // 判断 a 是否大于 b,如果 a 大于 b 成立,输出 a,如果不成立。输出 b。
    var a = parseInt(prompt("请输入第一个数:"));
    var b = parseInt(prompt("请输入第二个数:"));
    var c = parseInt(prompt("请输入第三个数:"));
    //让 a 和 b 进行比较,找出大的,再用大的跟 c 比较
    (a > b ? a : b) > c ? alert(a > b ? a : b); alert(c);
  </script>
</body>
</html>
```

在浏览器中运行程序,并在对话框分别输入 88、66、28 三个整数后,消息框会输出最大值 88,如图 1-19 所示。

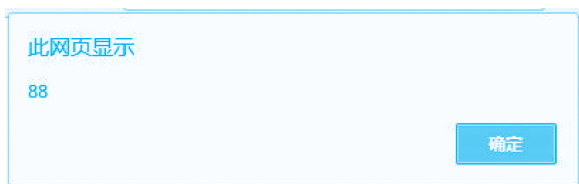


图 1-19 求三个数的最大数

1.3.2 计算圆的周长和面积

数学中根据圆的半径即可求出圆的周长和面积。下面将在程序中根据指定的半径实现圆的周长和面积的计算。

本案例需求分析如下:

- (1) 页面弹出输入半径 r 的输入框。
- (2) 判断半径 r 是否合法。如果合法,则计算圆的周长和圆的面积;否则,给出输入错误提示。
- (3) 在页面中显示半径 r 、圆的周长和圆的面积。

根据上面的分析,程序代码清单如下:

1-25.html 代码清单

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>输入圆的半径计算面积和周长</title>
</head>
<body>
  <div>圆的半径:<input id="r" type="text"></div>
  <div>圆的周长:<input id="cir" type="text"></div>
  <div>圆的面积:<input id="area" type="text"></div>
  <script type="text/javascript">
    var r = prompt('请输入圆的半径');
    if(! isNaN(r)){
      var cir = 2 * Math.PI * r;
      var area = Math.PI * r * r;
      // var area = Math.PI.toFixed(2) * Math.pow(r, 2)
      document.getElementById('r').value = r;
```

```

        document.getElementById('cir').value = cir.toFixed(2);
        document.getElementById('area').value = area.toFixed(2);
    }else{
        alert('请输入正确的数字')
    }
}
</script>
</body>
</html>

```

在上述代码中，“Math.PI”表示一个圆的周长与直径的比例，即 π 约为3.14159。toFixed(2)方法表示保留2位小数。在浏览器中运行程序，输入半径并点击“确定”按钮后页面效果如图1-20和图1-21所示。



图 1-20 输入圆的半径

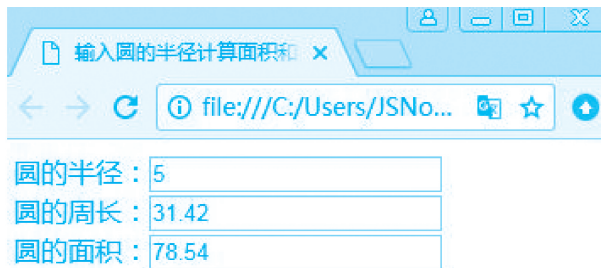


图 1-21 输入圆的半径

1.3.3 打印实心、空心 and 倒金字塔

金字塔是世界建筑的奇迹之一，其形状呈三角形，下面将利用双重循环语句以及条件判断语句实现打印实心、空心 and 倒金字塔。

以实心金字塔为例，由效果图可以看出，金字塔是由空格和星星“*”组成的三角形。假设最上面的一个星星作为金字塔的第一层，并且每层中星星的数量都为奇数。所以需实现以下条件：

- (1) 每层中星星的数量 = 当前层数 * 2 - 1；
- (2) 每层星星前的空格 = 金字塔层数 - 当前层数。

空心金字塔和倒金字塔同理，分别考虑好星星数量和空格数量与层数之间的关系即可。

空心金字塔思路：每一行的边上打印星星，其他的打印空格，最后一行打印星星。

倒金字塔思路：倒金字塔的第*i*行，空格为*i*-1，星星为 $2 * (n - i + 1) - 1$ 。

根据上面的分析，具体代码清单如下：

1-26.html 代码清单（实心金字塔）

```

<script type="text/javascript">
    //输入金字塔的层数
    var n = prompt("请输入一个正整数:");
    n = parseInt(n);
    document.write("实心金字塔:<br/>");
    for (var i = 1; i <= n; i++) {
        //输出星星前的空格
        for (var k = n-1; k >= i; k--) {
            document.write("&nbsp;");
        }
        //打印星星
        for (var j = 1; j <= 2 * i - 1; j++) {
            document.write("* ");
        }
        //换行
    }

```




```
document.write("<br>");
}
</script>
```

1-27.html 代码清单 (空心金字塔)

```
<script type="text/javascript">
//输入金字塔的层数
var n = prompt("请输入一个正整数:");
n = parseInt (n);
document.write("空心金字塔:<br/>");
for(i=1;i<=n;i++){
    for(j=1;j<=n-i;j++){
        document.write("&nbsp;");
    }
    for(k=1;k<=2*i-1;k++){
        if(i==n){                //最后一行星号全打印出来
            document.write(" * ")
        }
        else if(k==1||k==2*i-1){    //打印两边上的星号
            document.write(" * ")
        }
        else{
            document.write("&nbsp;");    //打印空心
        }
    }
    document.write("<br/>");
}
</script>
```

1-28.html 代码清单 (倒金字塔)

```
<script type="text/javascript">
var n = prompt("请输入一个正整数:");
n = parseInt (n);
document.write("倒金字塔:<br/>");
for(i=1;i<=n;i++){
    for(j=1;j<=i-1;j++){
        document.write("&nbsp;");
    }
    for(k=1;k<=2*(n-i+1)-1;k++){
        document.write(" * ");
    }
    document.write("<br/>");
}
</script>
```

程序运行结果如图 1-22、图 1-23 和图 1-24 所示。

```

      *
     **
    ***
   ****
  *****
 *****
*****

```

图 1-22 实心金字塔

```

      *
     * *
    *  *
   *   *
  *    *
 *     *
*      *
*****

```

图 1-23 空心金字塔

```

*****
*****
*****
*****
*****
*****
*****

```

图 1-24 倒金字塔

九九乘法表体现了数字之间乘法的规律，是学生学习数学时必不可少的一项内容。下面案例使用 JavaScript 打印九九乘法表。

(1) 九九乘法表的表格是由 9 行、每行最多 9 列的单元格组成。乘法表的层数 = 表格的行数 = 每行中的列数

(3) for 里层循环列数 j 。且满足 $j \leq i$ 。

根据上面的分析，可以通过简单的 JavaScript 代码来实现这一效果。具体代码清单如下：

[illegible]

The screenshot shows a web browser window with the address bar displaying "file:///C:/Users/JSNoob.admin-PC/Desktop/JavaScript前端开发案例...". The page content is a multiplication table on a yellow background. The table lists products of numbers from 1 to 9, arranged in rows and columns. The first row is "1*1=1". The second row is "2*1=2" followed by "2*2=4". The third row is "3*1=3", "3*2=6", and "3*3=9". The fourth row is "4*1=4", "4*2=8", "4*3=12", and "4*4=16". The fifth row is "5*1=5", "5*2=10", "5*3=15", "5*4=20", and "5*5=25". The sixth row is "6*1=6", "6*2=12", "6*3=18", "6*4=24", "6*5=30", and "6*6=36". The seventh row is "7*1=7", "7*2=14", "7*3=21", "7*4=28", "7*5=35", "7*6=42", and "7*7=49". The eighth row is "8*1=8", "8*2=16", "8*3=24", "8*4=32", "8*5=40", "8*6=48", "8*7=56", and "8*8=64". The ninth row is "9*1=9", "9*2=18", "9*3=27", "9*4=36", "9*5=45", "9*6=54", "9*7=63", "9*8=72", and "9*9=81".

1*1=1									
2*1=2	2*2=4								
3*1=3	3*2=6	3*3=9							
4*1=4	4*2=8	4*3=12	4*4=16						
5*1=5	5*2=10	5*3=15	5*4=20	5*5=25					
6*1=6	6*2=12	6*3=18	6*4=24	6*5=30	6*6=36				
7*1=7	7*2=14	7*3=21	7*4=28	7*5=35	7*6=42	7*7=49			
8*1=8	8*2=16	8*3=24	8*4=32	8*5=40	8*6=48	8*7=56	8*8=64		
9*1=9	9*2=18	9*3=27	9*4=36	9*5=45	9*6=54	9*7=63	9*8=72	9*9=81	

— 26 —



【本章小结】

本章首先介绍了 JavaScript 的用途、发展状况。接着针对 JavaScript 的入门知识进行介绍,包括引入方式、输入输出语句、注释等,然后介绍了 JavaScript 的基础语法知识,包括数据类型的概念及数据类型的转换、表达式以及运算符的使用,讲解了如何使用流程控制语句实现条件判断和代码的重复执行,并通过案例巩固加强学习,最后以案例讲解的形式进行实战和提高,深化对以上知识点的理解与应用。



课后练习

一、选择题

1. 下列选项中, 可以实现警告框的是 ()。
A.alert () B.prompt () C.document.write () D.console.log ()
2. 下面关于 for 语句的描述错误的是 ()。
A.for 循环语句小括号内的每个表达式都可以为空
B.for 循环语句小括号内的分号分割符可以省略
C.for 适合循环次数已知的情況
D.for 循环语句小括号中依次包括初始化表达式、循环条件和操作表达式
3. 下面链接外部 JavaScript 正确的是 ()。
A.<script src="animation.js"></script>
B.<link src="animation.js">
C.<script href="animation.js"></script>
D.<style src="animation.js"></style>
4. 下列选项中, 不属于比较运算符的是 ()。
A.== B.=== C.!= D.==
5. 下列选项中, 与三元运算符的功能相同的是 ()。
A.if 语句 B.if...else 语句
C.if...else if...else 语句 D.以上答案皆正确

二、填空题

1. 若 num 等于 2, 则执行完 num += 3 后, num 的值等于_____。
2. 流程控制主要有 3 种结构, 分别是顺序结构、分支结构和_____。
3. _____ 语句会无条件执行一次循环体后, 再判断是否符合条件。
4. 数字型可以用来保存_____或浮点数。
5. JavaScript 中的数据类型分为两大类, 分别是基本数据类型和_____。
6. JavaScript 由_____、_____和_____三部分组成。

三、上机练习题

1. 根据用户输入的数字 (数字 1~7), 返回对应的星期几。例如 1 代表星期一, 2 代表星期二, 依次类推。
2. 求 1~100 的所有整数的偶数和、奇数和。
3. 根据用户输入的年份, 判断是闰年还是平年。闰年: 能被 4 整除, 但不能被 100 整除的年份或者能被 400 整除的年份。