高等院校产教融合创新应用系列

JavaScript 迭代渐进式前端开发实践

梁晓晖　著

清華大學出版社
北　京

内 容 简 介

JavaScript 是一种轻量级、解释型编程语言，也是深受广大编程者喜爱的、能够实现跨领域开发的“多面能手”。本书以前端开发为应用领域，精选《成绩转换系统》《验证码及其应用》《网站换肤》《用户注册与数据提交》《打地鼠游戏》五个实战主题，通过多版迭代，生动有趣地介绍了 JavaScript 语言和软件开发的核心知识，包括 JavaScript 编程基础、数组、函数、对象、DOM、正则表达式与数据交互、BOM 及第三方工具、ES6 等。通过本书的学习，读者不仅能在知识层面有所收获，而且可以潜移默化地提高软件开发能力和个人综合素养。

本书可以作为高职高专、应用型本科、培训机构 JavaScript 语言课程的教材，也可作为 Web 前端开发人员的参考书，以及 JavaScript 语言爱好者的自学用书。

图书在版编目 (CIP) 数据

JavaScript 迭代渐进式前端开发实践 / 梁晓晖著 .
北京 : 清华大学出版社 , 2025. 7. -- (高等院校产教融合创新应用系列). -- ISBN 978-7-302-69796-1

Ⅰ. TP312.8

中国国家版本馆 CIP 数据核字第 2025SD8830 号

责任编辑：王　定
封面设计：周晓亮
版式设计：思创景点
责任校对：成凤进
责任印制：宋　林

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**14.25　　**字　　数：**320千字
版　　次：2025年8月第1版　　**印　　次：**2025年8月第1次印刷
定　　价：59.80元

产品编号：112401-01

前 言 PREFACE

JavaScript 是一种轻量级、免费、开源且跨平台的脚本语言，功能强大而灵活。作为现代 Web 开发的基石语言，JavaScript 已经连续多年位居“全球使用最广泛的编程语言”前列。它不仅驱动着前端领域 90% 以上的动态网页交互，支撑着 React、Vue 等主流框架的生态繁荣，而且成功跻身于服务端开发、桌面开发、嵌入式开发等多个领域，堪称“一次学习，多端应用”的典范。JavaScript 的亮眼表现，使得许多高校纷纷为计算机类专业开设 JavaScript 程序设计相关课程，并将其列为核心专业课程，尤其是软件开发类专业。

本书以前端应用领域为研究对象，主要介绍使用 JavaScript 语言进行 Web 前端开发的相关知识，同时融入一些软件开发的相关常识、基本思想、思维技巧，以及程序员职业道德、工匠精神等，本书的宗旨是：不仅要教会读者 JavaScript 语言，更要教会读者如何编程，以及如何成为一名优秀的程序员。

我本科和研究生均就读于计算机应用专业，热爱教育，喜欢编程，是一名不折不扣的双师型教师。二十多年的从教经历和多家大、中、小型企业项目研发经历，使我十分了解软件行业的人才需求，也十分了解学生的学情和一般计算机语言学习者的“痛点”。在历经十余载的 JavaScript 相关课程教学沉淀后，我决定重构 JavaScript 知识体系，编写这部教材，精选经典案例与实战高频内容，汇集成册，以期能够符合读者学习和认知规律，贴近实战。

本书特点

1. 素养引领，项目育人

软件行业作为赋能各行各业的“数字生产力”，其增长势头强劲。工业和信息化部发布的数据显示，2024 年，我国软件业务收入 137 276 亿元，同比增长 10.0%，软件业利润总

额 16 953 亿元，同比增长 8.7%，这一数据凸显了软件行业对国民经济发展的关键推动作用，不仅助力技术进步，还为就业和投资提供了新的机会。优质软件犹如清泉，润泽社会；而劣质软件乃至病毒、木马，则如同毒瘤，贻害无穷。党的二十大报告强调：“育人的根本在于立德。”一名优秀的软件从业者需要拥有炽热的爱国情怀、高尚的道德情操、正确的价值取向和精益求精的工匠精神，并坚守遵纪守法的行为准则。本书将这些素养元素巧妙地融于项目之中，引导读者树立正确的人生观，争当优秀的 IT 人。

2. 重构知识，打破困境

本书通过重构 JavaScript 知识体系，巧妙地将各个知识点融入实战场景，打破学生“似懂非懂，学而不会”的困境。学习知识旨在应用，所以，为了解决实际问题而组织知识，而不是为了编写者方便而堆砌知识，这才是编程类书籍的编写奥义所在！

那么在知识点安排上，如何既科学合理又兼顾读者的学习和认知规律呢？在翻阅了大量书籍之后，我终于找到了答案。清华大学出版社经典计算机语言书籍《汇编语言》(王爽 著) 的知识屏蔽与线索化创作理念，点亮了我的思路，使我豁然开朗，茅塞顿开。

零散的知识点难以记忆并付诸实践，因此，本书以项目为载体，通过一条条清晰的需求线索，将知识点紧密串联，使读者既能掌握知识，又能学会应用，一举两得；每个知识点学习和理解难度有所不同，有些知识点放在前面很难理解透彻，放在后面就水到渠成，易于理解。因此，我根据知识点自身的特性对其进行组织和安排。在具体的知识介绍和例题分析中，按照“知识屏蔽”的原则，所有用于解决问题的知识，必须是已经讲解过的知识，尽量避免使用后面还没介绍的知识，以减轻读者的学习困扰。一步一步，循序渐进。五大实战主题的先后顺序安排，也是按照难度层层递进，而非平级罗列，充分考虑读者的学习特点和认知规律。

3. 实战导向，迭代渐进

所谓实战导向，是指教材中的主题实战气息浓厚。既然学习就是为了应用，那就干脆把企业项目中的真实内容直接拿来或适当裁剪后当作知识载体，尽量缩小和填补课堂与职场之间的差距。

事实上，本书就是这样做的，小到知识学习环节中的例题，大到五大实战主题，再到课后编程实践题。模拟“百度搜索”效果的例题，源自我研发语料库系统时检索数据的真实案例；“图片能放大能缩小”是我为企业研发项目时客户的真实需求；而删除数据前确认弹窗、注册账户合法性验证、动态地址配置、验证码等案例，更是实战味道满格，稍加修改甚至不用修改就可以拿到企业级项目中使用。课后习题中的电子相册、秒表 DIY、倒计时、音乐播放器等，不仅能够激发读者的创造潜能，提高读者的实战能力，而且在生活中也能直接使用，“学以致用”的理念得到了充分体现。

对于初学者而言，在学习语言的同时掌握企业级项目模块的开发，这似乎是个遥不可及的梦想。为了破解这个难题，本书创造性地将难度分解，使用“迭代渐进”的编写模式：每个大的实战主题都分为三版迭代，每迭代一版，会增加若干个维度的知识点和难度级别，

对应的代码功能也越来越强大。从简约原型版到基本完善版，再到终极应用版，就像是沿“之”字形路爬山一样，每一步不是特别累，却能轻松登顶！看似遥不可及的目标，被努力与智慧化解！读者能否达到预期的效果呢？我们拭目以待！

4. 掌握语言，学会编程

本书的编写目标不仅是让读者掌握使用 JavaScript 语言编写前端项目的基本知识和常用技巧，更是让读者学会如何编写程序。因此，读者将在本书中发现例题分析、编程引导，以及软件开发相关的常识、程序员职业素养、经典编程思想和实战经验小贴士等丰富内容，这些内容均为读者掌握通用编程技能提供有力支持。

了解“软件工程”的读者会发现，本书每一个实战场景的内容安排都有一根隐藏的主线，那就是软件开发生命周期：“任务描述”对应“需求分析”；“执行效果图”对应“概要设计”；“技术分析”对应“详细设计”；“编程实现”对应“编码实现”；“多版迭代”对应“测试阶段”和“维护阶段”。这种“润物细无声”的方式，可以让读者在潜移默化中得到专业的技术熏陶。

5. 一书多用，适配灵活

为了兼具编程作品集的实用性和传统教材的系统性，本书配备一主一辅双重目录，前者以场景的实现环节为索引对象，后者则以具体的知识点为呈现目标。高校学生可以将本书用作教材，前端开发人员可以将本书用作参考书，JavaScript 语言爱好者可以将本书用作自学书籍。

本书作为教材使用时，可根据实际情况灵活安排教学内容：课时够用或学情良好时，可进行完整学习；课时紧张或学情欠佳时，可采用选择性组合学习，如每个实战主题 V1.0 实现 +V2.0、V3.0 知识点学习。这种方式灵活适配多种学情，进可攻，退可守，对柔性分层教学和因材施教非常友好。

配套资源与答疑服务

为了方便读者学习和教师教学，本书提供教学大纲、教学课件、电子教案、教学进度表、课后习题参考答案、模拟试卷、程序源代码等配套教学资源，可扫描下列二维码或书中二维码获取。如果读者在本书阅读过程中发现问题或者存有疑问，可发邮件到我的邮箱：71304690@qq.com。

教学大纲　教学课件　电子教案　教学进度表　课后习题参考答案　模拟试卷

鸣谢

在本书编写过程中，北京楷斯科技有限公司技术总监马翔和清华大学出版社编辑王定对本书样张给予了中肯的建议，河北软件职业技术学院张莹、胡金扣、闫绍惠、杨宁侠、崔秀艳等老师参与了本书电子资源制作、辅助资源建设及后续课程服务。我的家人与朋友们同样对本书的编写给予了极大的支持与理解，他们慷慨地腾出自己的宝贵时间去分担我的其他工作，让我得以心无旁骛地雕琢和完善本书内容，在此，对大家一并表示最诚挚的感谢！

本书小彩蛋

为什么让一只地鼠出现在压轴项目而非企业级项目需求呢？

因为几乎全书都是企业项目啊。“打地鼠游戏”是“麻雀虽小，五脏俱全”，完全能够承载足够多的知识点来进行综合实践，更重要的是，打地鼠游戏可以增加编程乐趣，读者还可以 DIY 声音、背景等效果呢。唯有热爱，可抵山海。愿本书的每一位读者，都因为阅读本书而喜欢上编程，喜欢上用代码去美化世界的感觉。

让我们以代码为利剑，一路披荆斩棘，勇往直前，最终抵达理想的彼岸！

本书如有疏漏之处，欢迎大家批评指正！若对本书有更好的建议，也欢迎大家不吝赐教！

梁晓晖
2025 年 4 月 28 日凌晨于保定

目 录 CONTENTS

JavaScript 知识点索引

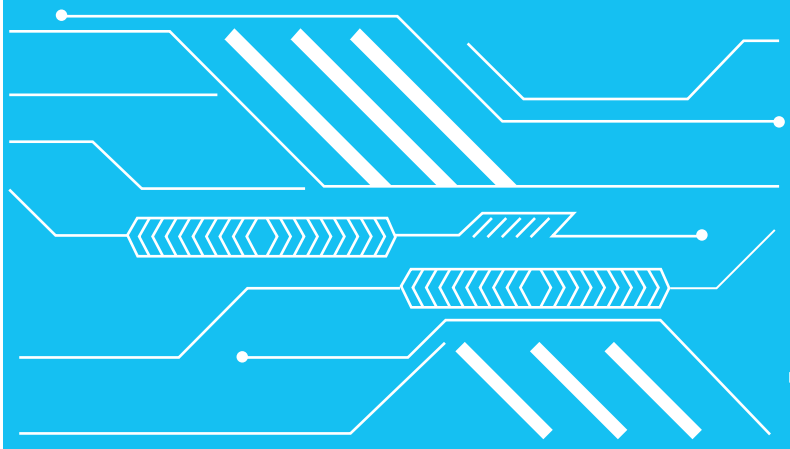
JavaScript 编程基础

JavaScript 编程进阶

JavaScript 对象编程

JavaScript 编程高级

JavaScript 综合与拓展



实战主题 1

成绩转换系统

计算机最基本的功能之一是对原始数据进行加工，进而得到对用户有用的信息。作为诸多加工形式之一的数据转换，几乎是所有应用程序不可或缺的组成部分。

本主题以将成绩从百分制转换为等级制为应用场景，通过三个版本的 JavaScript 代码迭代实现，带领读者使用 JavaScript 语言编写代码，实现数据输入、加工、输出的全部过程。

千里之行，始于足下。基础牢固，未来的学习之路才能更加顺畅。

通过本主题的学习，读者将从开发环境、调试技能、软件开发常识、基本 JavaScript 语法、软件素养等几个方面有所收获，并为后续知识的学习打下坚实的基础。

知识目标

- 了解 JavaScript 的基本常识。
- 掌握 JavaScript 标准与组成。
- 掌握 JavaScript 在浏览器上的执行过程。
- 掌握将 JavaScript 引入网页中的常用方法。
- 掌握 JavaScript 的基本语法和输入输出方法。
- 掌握 VSCode 编程 IDE 的使用方法及技巧。

能力目标

- 能够熟练使用 VSCode IDE 编写 JavaScript 代码。
- 能够熟练使用 JavaScript 输入方法实现数据输入。
- 能够使用基本的程序控制语句实现数据加工。
- 能够熟练利用浏览器控制台调试 JavaScript 代码。

- 能够使用 JavaScript 输出方法实现数据输出。
- 能够遵循行业主流命名规范科学合理地为变量命名。
- 能够遵循特定编程风格编写和组织代码。

素养目标

- 培养基本的软件设计思想。
- 提升软件设计的用户体验维度。
- 了解软件健壮性并提升思维缜密性。
- 培养和践行精益求精的工匠精神。
- 培养良好的编码习惯和编码风格。
- 培养思考与分析能力。

思维导图

1.1 《成绩转换系统 V1.0》需求与技术分析

成绩转换系统体现了一个典型的数据处理流程，包括数据输入、数据加工和数据输出三大环节。在 V1.0 版本的实现过程中，读者将从零开始接触 JavaScript 语言，慢慢熟悉 JavaScript 的特点，逐步开启 JavaScript 编程的学习之旅。

1.1.1 《成绩转换系统 V1.0》任务描述

本任务将实现弹窗版成绩转换系统，属于基础版本，目的是让读者体验使用 JavaScript 语言编写、运行和调试程序的基本流程，掌握 JavaScript 弹窗的几种应用方式及基本的流程控制语句。

具体需求如下：

(1) 用户输入 0~100 的百分制成绩。

(2) 点击“确定”按钮，显示转换后的等级制成绩。

(3) 要求弹窗实现。

1.1.2 《成绩转换系统 V1.0》任务效果

《成绩转换系统 V1.0》任务效果如图 1-1 所示。

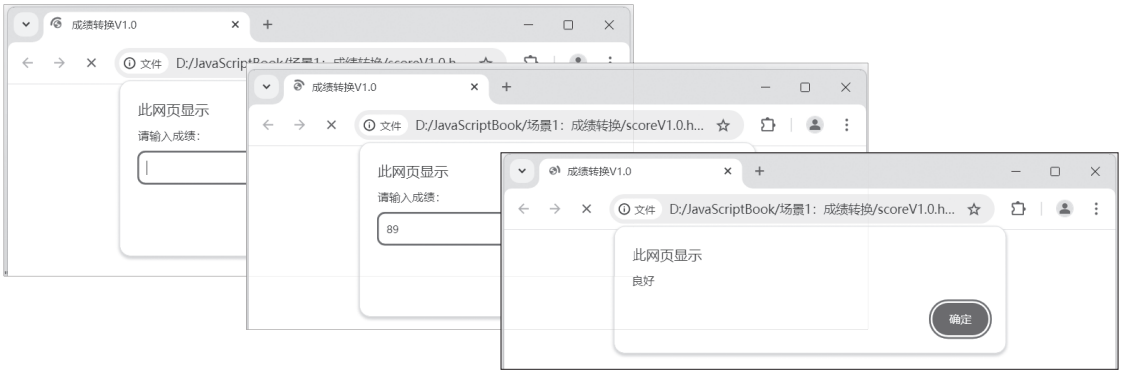


图 1-1 《成绩转换系统 V1.0》任务效果

1.1.3 《成绩转换系统 V1.0》技术分析

《成绩转换系统 V1.0》需求简单，就是经典的输入数据、加工数据、输出数据三个基本步骤，使用风格为醒目弹窗式，因而实现逻辑相对容易。成绩转换系统作为 JavaScript 编程初体验的第一个实战主题，涵盖知识的难度不大，但是涉及内容较广，属于偏基础部分。因此，这是培养良好的编程习惯、编码风格，以及进行编程思想启蒙的最佳时机。

下面以一问一答的形式来解决技术层面的问题。

(1) 如何使用 JavaScript 语言编写网页代码?

对应知识：JavaScript 语言编程常识。

(2) 如何使用 VSCode IDE ?

对应知识：VSCode IDE 使用方法及常用技巧。

(3) 如何使用输入弹窗让用户将数据录入系统并通过警告弹窗将加工结果呈现出来?

对应知识：JavaScript 输入输出。

(4) 如何将获取到的用户数据妥善保管?

对应知识：JavaScript 变量及作用域。

(5) 如何将用户数据转换成逻辑上可用的数据?

对应知识：JavaScript 数据类型及数据类型转换。

(6) 如何根据用户数据进行百分制到等级制的加工转换?

对应知识：JavaScript 运算符及流程控制语句。

1.2 《成绩转换系统 V1.0》知识学习

通过上述介绍，相信读者对于要完成的任务已经有了大致的了解。下面对实现该任务所需的知识进行介绍，以便尽快展开程序编写工作。

1.2.1 JavaScript 简介

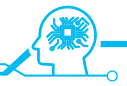
JavaScript(简称JS)是一种具有函数优先特性的轻量级、解释型编程语言。它最初作为开发Web页面的脚本语言而闻名，是网页开发三剑客(HTML、CSS、JavaScript)中唯一真正具备编程能力的语言。

随着不断地发展演进，JavaScript 已经被用到很多非浏览器环境中。现在的 JavaScript 不仅能够进行 Web 全栈开发，还能进行桌面开发和嵌入式开发，可谓“软硬通吃”，是编程领域一个几乎无所不能的重量级存在。精通一门语言，便可以在整个软件领域大展身手，这并非遥不可及的神话。

本书将以 JavaScript 在 Web 前端开发领域的应用为载体，向大家介绍 JavaScript 基本语法和核心编程技术。

1.2.2 JavaScript 历史

JavaScript 是由网景通信公司 (Netscape Communications Corporation，简称网景) 的布兰登 • 艾奇于 1995 年在 Netscape 浏览器上首次设计实现的，它最初的设计目标是改善网页的用户体验。



提起 JavaScript，很多人都会联想到两种与其名称相似的语言，并怀有如下疑问：

(1) JavaScript 与 Java 是什么关系？

(2) JavaScript 与 JScript 是什么关系？

从语言本身来讲，它们各自有各自的语法和研发团队，背后的设计理念和设计思想也不尽相同，是三种完全不同的独立语言。但是，从语言背后的公司及其营销策略上来讲，它们之间的关系就非常微妙了。任何技术都是人创造的，既然与人相关，自然离不开人文因素。

JavaScript 最初名为 LiveScript，后来网景与 Sun 合作，将其更名为 JavaScript。众所周知，Sun 公司最著名的产品是 Java。JavaScript 与 Java 名称上的相似，是当时网景出于营销考虑与 Sun 达成协议的结果。微软同时期推出 JScript 来迎战 JavaScript。

1.2.3　JavaScript 标准与组成

JavaScript 标准是 ECMAScript，ECMAScript 的演进过程如下。

1996 年 11 月，网景正式向 ECMA(欧洲计算机制造商协会) 提交语言标准。

1997 年 6 月，ECMA 以 JavaScript 语言为基础制定了 ECMAScript 标准规范 ECMA-262。JavaScript 成为 ECMAScript 最著名的实现之一。

1999 年 12 月，ECMAScript 3 发布。

2009 年 12 月，ECMAScript 5 发布。

2015 年 6 月，ECMAScript 2015 发布，该版本也被称为 ECMAScript 6 或 ES6。

2016 年，ECMAScript 7 发布。

随着时间的推移，ECMAScript 继续演进。

本书以 ES5 为主，稍微涉及一些 ES6。只要打好了 ES5 基础，掌握 ES6 等后续版本就十分容易，很多 ES6 的新增内容不过是 ES5 中相关内容的语法糖而已。如果说 ES5 是“走”，那么 ES6 就是“跑”，不会走就想跑，注定要摔跤。学习是一个循序渐进的过程，如果知识顺序安排得当，基本功扎实，学习效果自然理想。反之，欲速则不达。

完整的 JavaScript 实现由三个部分组成，如图 1-2 所示。

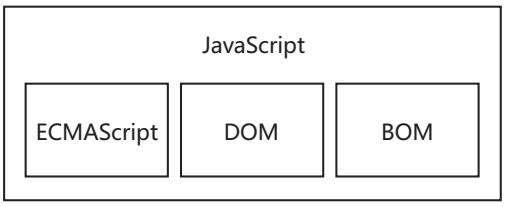


图 1-2　JavaScript 组成

其中，

(1) ECMAScript 是 JavaScript 的标准，也是核心。

(2) DOM(文档对象模型) 是一套操作页面元素的 API，可以把 HTML 看作文档树，通过 DOM 提供的 API 对树上的节点进行操作。

(3) BOM(浏览器对象模型) 是一套操作浏览器功能的 API，通过 BOM 可以操作浏览器窗口，例如：弹窗、控制浏览器跳转、获取分辨率等。

1.2.4 JavaScript 在浏览器上的执行过程

浏览器内核分为两部分：渲染引擎和 JS 引擎。

(1) 渲染引擎用来解析 HTML 和 CSS，如 Chrome 的 Blink 内核。

(2) JS 引擎也叫 JS 解释器，用来读取和解析网页中的 JavaScript 代码，如 Chrome 的 V8 引擎。

JS 引擎是逐行解释 JavaScript 代码的。也就是说，读取一行源码，将其转换为计算机可识别的二进制编码，交由计算机执行。随后读取下一行源码，重复以上过程。

JS 引擎的执行过程与渲染引擎紧密相关，因为 JavaScript 可以修改页面元素和结构，从而影响页面的渲染结果。因此，在编写代码时，要特别留意 JavaScript 代码的书写位置。例如：document 对象的 getElementById() 方法，可以实现根据 id 值获取页面元素，但是如果这行代码放在页面元素尚未渲染之前执行，就无法得到想要的结果，而是得到一个空值 null。

1.2.5 编程 IDE：VSCode

Visual Studio Code(简称 VSCode) 是一款由微软开发的、轻量级的、免费的、跨平台的代码编辑器。它支持多种编程语言和文件格式，包括但不限于 JavaScript、TypeScript、Python、Java、PHP、Go 等，并且拥有丰富的扩展生态系统，允许用户根据需要添加额外支持，例如安装各种插件。

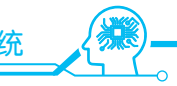
VSCode 内置了对 JavaScript 和 TypeScript 的支持，并提供了针对其他语言和运行时的丰富扩展生态系统，使得开发者能够轻松地编写、调试和部署代码。此外，VSCode 还具有强大的搜索、Git 管理、调试模块、错误警告等功能，能够显著提升开发效率和代码质量。凭借卓越的性能和广泛的功能集，VSCode 迅速获得了广大开发者的青睐，成为全球最受欢迎的代码编辑器之一。

支持编写 JavaScript 程序的 IDE 还有很多，如 HBuilder、WebStorm 等，它们各有千秋，读者可以根据自己喜好进行选择。本书代码全部采用 VSCode 编写。

VSCode 的常规使用界面如图 1-3 所示。单击侧菜单中从上往下数第 1 个图标，可以展开和折叠资源区；单击侧菜单中从上往下数第 4 个图标，可以安装各种插件。

图 1-3 VSCode 使用界面

使用 VSCode 有很多技巧，例如：输入 li:small*9 可以自动生成 9 对 class 值为 small 的 li 标签；输入 btn 并按 Enter 键可以自动生成一对 button 标签等。在编写 JavaScript 代码的过程中，有意识地使用这些技巧，可以大幅提高代码编写速度。



1.2.6　在网页中使用 JavaScript 的方法

在网页中使用 JavaScript 有三种方式。

1. 采用内部 JavaScript 方式

直接通过一对 script 标签，将 JavaScript 代码嵌入页面中，我们称这种方式为内部 JavaScript 方式，语法示例如下：

```
<script>
   JavaScript 代码 ...
</script>
```

2. 采用外部 JavaScript 方式

首先创建一个单独的 JavaScript 文件，然后在网页中通过 script 标签的 src 属性指定该文件，即可将文件中的 JavaScript 代码引入网页，我们称这种方式为外部 JavaScript 方式，语法示例如下：

```
<script src="JavaScript 文件的路径 "></script>
```

实战小贴士

在一个真正意义上的网站中，往往存在多种类型的文件，为了合理且清晰地安放这些资源，通常根据功能将网站资源存放到不同的文件夹中。例如，js 文件夹专门存放 JavaScript 脚本文件，Images 文件夹专门存放图片文件，CSS 文件夹专门存放样式文件。

3. 采用行内 JavaScript 方式

HTML 文档可以在标签的属性中，使用 JavaScript 脚本作为其属性值，此类属性通常为事件属性，这种方式被称为行内 JavaScript 方式，语法示例如下：

```
<button onclick="alert('Hello, world!')"> 点我 </button>
```

注意，因为属性值使用的是双引号，所以 alert() 函数中的字符串参数采用单引号表示。当单击“点我”按钮时，将看到一个显示“Hello, world!”的 alert 弹窗。在这个示例中，onclick 属性的值直接为 JavaScript 代码。但是，通常情况下，事件处理代码不止一行，所以可以将其封装到函数中，然后再将 onclick 属性的值设置为该函数。

假设按钮的单击事件处理代码被封装到 clickMe() 函数中，可以通过下面的方式对其进行调用：

```
<button onclick="clickMe()"> 点我 </button>
```

1.2.7　JavaScript 编程常识与命名规范

1. 代码缩进

代码缩进是指在编写代码时，在每一行代码前空出一定的空白区域，用来表示代码之

间的包含和层次关系。这种空白区域可以通过 Tab 键或空格键来实现。良好的代码缩进可以增强代码的可读性，进而帮助我们更好地理解代码。

2. 代码注释

注释是对代码添加解释和说明。为变量、语句、程序片段、函数等添加注释，能够提高代码的可读性和可维护性。一个拥有良好注释的程序，即使更换了开发者，甚至若干年后，依然可以被很快读懂，反之，如果一个程序没有注释，则可能像天书一样晦涩难懂，最后只能落得被抛弃的下场。注意：注释只是为了提高程序的可读性而存在，并不会被计算机编译，因此不需要担心添加注释会影响程序的执行效率。

在 JavaScript 中，注释有两种：单行注释和多行注释。其中，单行注释用 // 表示，多行注释用 /* */ 表示。

在 VSCode 中添加注释有以下两个技巧：

(1) 按住鼠标左键，选择打算注释或取消注释的代码，然后按下 ctrl+“/”键，即可为其添加或取消注释。

(2) 在为函数添加注释时，建议添加标准化注释，由于不同函数的功能和参数各不相同，建议所有的函数注释中都要包括函数的功能和参数含义描述。在 VSCode 中，为函数添加标准化注释的方法是输入 /**，再按 Enter 键，这种方法会自动生成标准化注释框架，非常方便。

3. 代码命名规范

良好的命名规范可以提高代码的可读性和可维护性，以下是一些常见的命名规范。

(1) 小驼峰命名法 (lower camel case)：第一个单词以小写字母开始，后续单词的首字母大写，例如 myVariable。

(2) 大驼峰命名法 (pascal case)：每个单词的首字母都大写，例如 MyVariable。

(3) 下画线命名法：下画线命名法是指将多个单词组合在一起时，单词之间用下画线连接，例如 my_variable_name。

在 JavaScript 中，推荐使用小驼峰命名法。

4. JavaScript 语法基本特点

JavaScript 语句的基本特点主要包含如下几个方面。

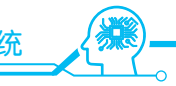
(1) JavaScript 程序的执行单位为行 (line)，也就是一行一行地执行。一般情况下，每一行就是一个语句。语句 (statement) 是为了完成某种任务而进行的操作，多条语句最终构成了程序源代码。

(2) 语句一般以分号结尾，一个分号就表示一个语句的结束。多个语句可以写在一行内。但是为了清晰起见，并不提倡这么做。

(3) 分号前可以没有任何内容，JavaScript 引擎将其视为空语句。

(4) 表达式不需要分号结尾。一旦在表达式后面添加分号，则 JavaScript 引擎会将表达式视为语句，这样会产生一些无意义的语句。

(5) JavaScript 区分大小写，即 JavaScript 语句和变量都对大小写敏感。



实战小贴士

JavaScript 中一行代码结尾的分号是可选的，以分号结尾和不以分号结尾只是不同的编程风格而已。但是，在编写代码时最好不要省略分号。虽然省略分号可以使代码更紧凑，但这可能会导致出现错误和维护问题。

1.2.8　JavaScript 变量、常量及作用域

变量、常量及作用域是 JavaScript 语言的基础内容。

1. 变量的概念

JavaScript 变量是用于存储数据的容器。可以给变量起一个简短名称，如 x，或者更有描述性的名称，如 userName。为了增加程序的可读性，建议选择后者，因为可以见名知意。通过使用变量名，我们可以获取或者修改变量中的数据。

JavaScript 变量可以保存值(如 x=5)，也可以保存表达式(如 z=x+y)。

注意，变量名建议采用英文单词或汉语拼音全拼进行命名，优先推荐前者。最好不要使用汉语拼音缩写命名，例如：xsxm，一般人很难猜出来这个变量用于存放“学生姓名”。

2. 变量声明及赋值

在 JavaScript 中，创建变量通常称为“声明”变量。在 ES5 中，使用 var 关键字来声明变量。

(1) 单纯声明一个变量，语法示例如下：

```
var userName;// 声明一个空的变量（它没有值，实际上是 undefined）
```

(2) 向变量赋值，语法示例如下：

```
userName=" 张三 "; // 等号是赋值符号
```

(3) 声明一个变量并为其赋值，语法示例如下：

```
var userName=" 张三 ";
```

(4) 在一条语句中声明、赋值多个变量。以 var 开头，使用逗号分隔变量，语法示例如下：

```
var lastname="Jack", age=23, job="carpenter";
```

3. 变量命名规范

在 JavaScript 中，对变量进行命名，需要遵循以下规则。

(1) 变量由字母 (a~z，A~Z)、数字、美元符号 ($) 和下画线 (_) 组成。

(2) 第一个字母必须是字母、下画线 (_) 或美元符号 ($)。

(3) 变量名称对大小写敏感 (y 和 Y 是不同的变量)。

(4) 变量名不能包含空格。

(5) 变量名不能与系统关键字、保留字重名。

(6) 变量名最好见名知意。

(7) 变量最好遵循某种特定的编程规范。

上述规则中，前面五条是语法层面的硬性要求，后面两条是编程风格层面的软性要求。

实战小贴士

一名优秀的程序员不仅要注重代码质量，还要注重代码风格。良好的编程风格能让代码具有更高的可读性，编程风格良好的程序员也更容易被团队成员接受和喜爱。

下面我们通过简单的例子，使用 JavaScript 变量来解决一个实际问题。

【例 1-1】使用变量模拟交换杯中饮料。

本例的本质为交换两个变量的值。交换两个杯子中的饮料，需要借助第三个杯子作为中转。同样地，交换两个变量的值，需要引入第三个变量作为中转变量。具体实现代码如图 1-4 所示。补充说明：console.log() 可以在控制台上输出其参数内容。

例 1-1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>交换果汁</title>
</head>
<body>
    <script>
        var yellowCup="Orange Juice";
        var pinkCup="dragonfruit Juice";
        //交换两个杯子中的饮料
        var whiteCup=yellowCup;
        yellowCup=pinkCup;
        pinkCup=whiteCup;
        console.log(yellowCup);
        console.log(pinkCup);
    </script>
</body>
</html>
```

图 1-4　交换杯中饮料的代码

其中：

第 11 行代码表示用黄杯子盛放橘子汁；

第 12 行代码表示用粉杯子盛放火龙果汁；

第 14 行代码用一个中间存储杯子 (白杯子) 来临时存放黄杯子中的橘子汁；

第 15 行代码让黄杯子存放粉杯子中的火龙果汁；

第 16 行代码，让粉杯子存放临时存放在白杯子中的橘子汁；

第 17、18 行代码，将黄杯子、粉杯子的值输出到控制台，以验证结果是否正确。

将代码保存后，在谷歌浏览器中查看，笔记本电脑按 Fn+F12 组合键，台式机直接按 F12 键，切换到控制台，可以看到输出结果如图 1-5 所示，在每条输出数据的右侧，显示了导致该结果出现的代码所处行数。

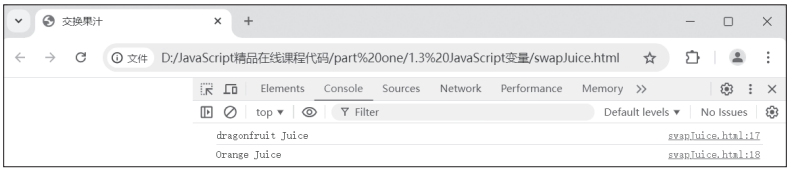


图 1-5　交换杯中饮料的执行效果

4. 作用域及数据类型

通常来说，一段代码中用到的名字 (如变量名) 并不总是有效和可用的，而限定这个名字的可用性的代码范围就是这个名字的作用域。作用域机制可以有效减少命名冲突情况的发生。

通过前面的学习，我们可知变量需要先声明后使用，但这并不意味着声明变量后就可以在任意位置对其进行使用。如果在函数内部声明一个变量，在函数外部对其进行访问，就会出现变量未定义的错误。

【例 1-2】变量作用域使用“反面教材”。

本例展示了一个错误使用作用域的案例，读者应在实践中引以为戒。具体代码如图 1-6 所示，执行效果如图 1-7 所示。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        function stuInfo(){
            var score=85;
            console.log("该生成绩为"+score+"分");
        }
        stuInfo();
        console.log(score);
    </script>
</body>
</html>
```

例 1-2

图 1-6　错误使用作用域的代码

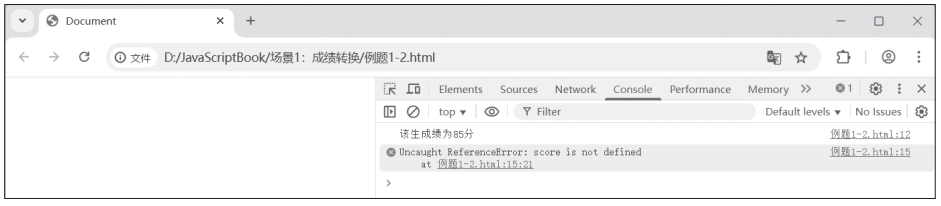


图 1-7　错误使用作用域的执行效果

从例 1-2 可以看出，变量需要在它的作用范围内才可以被正确使用。

在 ES5 及其之前版本的 JavaScript 实现中，根据作用域范围的不同，作用域被划分为全局作用域和函数作用域。根据声明所处的作用域，JavaScript 将变量分为以下两种。

(1) 全局变量：不在任何函数内声明的变量 (显式定义) 或在函数内省略 var 声明的变

量 (隐式定义) 都称为全局变量，全局变量在同一个页面文件中的所有脚本内都可以使用。

(2) 局部变量：在函数内利用 var 关键字定义的变量称为局部变量，它仅在该函数体内有效。

在 ES6 中，引入了块级作用域的概念，于是有了第 3 种变量：块级变量。

(3) 块级变量：使用 let 关键字声明的变量称为块级变量，仅在定义它的语句块 (以“{}”为标识) 内有效。

对于初学者而言，重点是理解全局变量和局部变量的区别，块级变量和 let 关键字属于 ES6 新增内容。本主题以 ES5 为主要学习内容，目的是让读者掌握原生 JavaScript 语言的基本语法，以及编程领域的一些基本概念和编程技巧，而非语法糖或框架等，所以简单了解即可，读者如感兴趣，可参考本书附录或自行查阅相关资料，这里不再赘述。

5. 作用域提升

在 JavaScript 中，存在作用域提升机制。JavaScript 引擎在对 JavaScript 代码进行解释执行之前，会对 JavaScript 代码进行预解析，在预解析阶段，会将以关键字 var 和 function 开头的语句块提前进行处理，它们的声明会被提升到其对应作用域的最顶端。这就是作用域提升机制。

例如：图 1-8 所示代码与图 1-9 所示代码的实际运行过程一模一样。

```
<script>
    console.log(a);//undefined
    var a=10;
</script>
```

图 1-8　作用域提升的示例

```
<script>
   var a;
   console.log(a);//undefined
   a=10;
</script>
```

图 1-9　作用域提升后的代码

注意： var 声明的变量具有作用域提升效果，let 声明的变量没有作用域提升效果。

6. 常量的概念

常量用于表示一些固定不变的数据，也有人将其称为字面量。在 JavaScript 中，主要有以下五种类型的常量。

- 整型常量，例如 1。
- 实型常量，例如 3.14。
- 字符串常量，例如“Hello”。
- 布尔常量，例如 true。
- 自定义常量 (它是 ES6 中引入的内容)，例如 const PI=3.14。

【例 1-3】编写 JavaScript 代码实现求圆的面积。

本例为常量的典型应用案例，圆周率可以被定义为一个常量。具体代码如图 1-10 所示。

```
<script>
    // 因为圆的半径可以变化，所以将其定义为变量。
    var r=10;
    //因为圆周率是永远不变的，我们将其定义为常量。
    const PI = 3.14; //不可以修改值
     //修改常量会报错
     // PI = 4.66; //报错invalid assignment to const 'PI'
     console.log("半径为",r,"的圆的面积为：",PI*r*r); //半径为 10 的圆的面积为： 314
</script>
```

图 1-10　求圆的面积

例 1-3

在使用常量时，需要注意以下几点。

(1) 如果尝试修改自定义常量的值，程序会报错，因为常量一旦定义就不能改变值了。

(2) 通常情况下，为了便于区分常量和变量，建议常量用大写字母命名。 如果包含多个单词，则使用下画线 _ 进行分割。

(3) 自定义常量不存在作用域提升。

1.2.9　JavaScript 数据类型

如果 JavaScript 是读者学习的第一门编程语言，建议充分调动理解记忆功能；如果读者之前学习过其他的编程语言，建议采用对比学习法和类比学习法以提高学习效率。

1. 数据类型的概念

在计算机中，不同数据占用的存储空间是不同的，因此，引入数据类型的概念是十分必要的。

(1) 从计算机的角度来看：

① 内存大小的确定方便系统分配空间。

② 内存位置的确定方便系统进行定向。

③ 内存需求的确定可以充分利用内存空间。

(2) 从人类的角度来看：

① 为程序员提供了一个方便访问内存的接口。

② 为程序员理解变量的含义提供方便。

③ 为程序员表达现实生活中的场景提供量身定制的帮助。

④ 对程序员来讲，操作一个任意形式的变量，很不好掌握，非常容易出错。引入数据类型的概念可以限制人类操作，从而降低操作难度和出错率。

数据类型是固定内存块大小的别名，也可以通俗地理解为数据的类别型号。例如，商品名称“苹果”，价格 10 元，这些属于不同的数据类型。前者擅长展示信息内容，后者擅长表达数值大小。前文介绍了变量拥有变量名和变量值两个元素。数据类型的引入，让变量拥有了第三个元素，该元素决定了如何将变量的值存储到计算机的内存中。

JavaScript 是一种弱类型语言，这意味着不用提前声明变量的类型，在程序运行过程中，类型会被自动确定，例如：

```
var studentAge=20;                          // 数字类型
var studentName=" 张三 ";                   // 字符串类型
```

JavaScript 拥有动态类型，同时也意味着相同的变量，可以用作不同的数据类型，例如：

```
var msg=8;                                  // 数字类型
var msg=" 中国有五千年的文明和历史 ";       // 字符串类型
```

实战小贴士

JavaScript 虽然允许变量类型灵活多变，但还是建议编程者在使用变量时，先声明再使用，并且一个变量对应一种数据类型。尽管 JavaScript 提供了灵活性，但使用 JavaScript 的人有权选择遵守一般的规则，严谨行事。

2. 数据类型种类

根据在内存中存储机制的不同，JavaScript 的数据类型分为基本数据类型 (值类型) 和引用类型，每种类型又细分为若干种不同的具体数据类型。

(1) 基本数据类型 (值类型)。包括字符串类型 (string)、数字类型 (number)、布尔类型 (boolean)、空类型 (null)、未定义 (undefined)、Symbol(ES6 新加)。

(2) 引用数据类型。包括对象和函数，其中，对象类型又分为对象 (Object)、字符串对象 (String)、数组对象 (Array)、正则对象 (RegExp)、日期对象 (Date)、数学对象 (Math) 等。

3. 基本数据类型和引用数据类型的存储机制

JavaScript 基本数据类型和引用数据类型存在着很大的不同，主要体现在如下几个方面。

(1) 存储位置不同。基本数据类型的数据存储在栈内存 (stack，又称堆栈) 中，引用数据类型的数据存储在堆内存 (heap) 中。

(2) 存储内容不同。数据存储时，基本数据类型在变量中存储的是值，引用数据类型在变量中存储的是空间地址。

(3) 进行数据操作时有所不同。基本数据类型操作的是值，引用数据类型操作的是空间地址。

【例 1-4】值类型的存储机制应用举例。示例代码如图 1-11 所示。

例 1-4

```
1 <script>
2     var a=1;
3     var b=a;
4     b=2;
5     console.log(a,b);//1 2
6 </script>
```

图 1-11　值类型存储机制示例代码

当第 2 行代码被执行后，在堆栈中申请一块空间，将数值 1 存入其中。

当第 3 行代码被执行后，在堆栈中申请一块空间，将 a 中的数据复制一份，存入其中，执行过程如图 1-12 中左侧堆栈所示。

当第 4 行代码被执行后，将数值 2 存入 b 所代表的堆栈空间，该空间中的原值被新值替换。执行过程如图 1-12 中右侧堆栈所示。

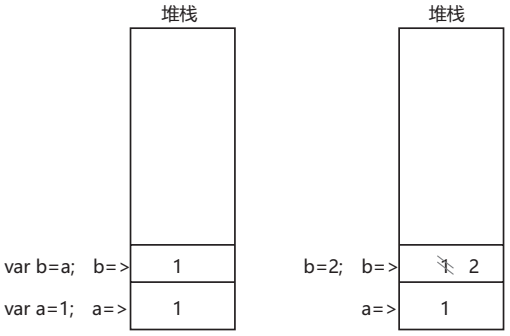


图 1-12　值类型存储机制执行过程

由此可见，值类型的数据在赋值时，只是把源变量 (a) 中的值复制了一份给目标变量 (b)，两个变量各自独立，互不干扰。当执行类似 var b=a; 的值类型赋值操作时，本质上是重新创建了一个内存空间来存储数据。修改两个独立空间中的一个数据，另外一个自然不受影响。

【例 1-5】引用类型的存储机制应用举例。示例代码如图 1-13 所示。

```
1 <script>
2   var a=new Object();
3   a.name="张三";
4   var b=a;
5   b.name="李四";
6   console.log(a.name);//"李四"
7   console.log(b.name);//"李四"
8 </script>
```

图 1-13　引用类型存储机制示例代码

例 1-5

第 2 行代码定义了一个对象类型的变量 a，该变量的数据类型属于引用类型。

第 3 行代码为变量 a 添加了 name 属性，并将该属性赋值为“张三”。

第 4 行代码定义了一个变量 b，并将其赋值为 a，注意，这里赋值的是 a 的引用，也就是说，b 中存储的和 a 中存储的都不是数据本身，而是数据的引用，即内存地址。第 2~4 行代码的执行内存图如图 1-14 所示。

可以看到，a 和 b 指向的是同一块内存空间。

第 5 行代码对 b 的 name 属性进行了重新赋值，执行内存图如图 1-15 所示。

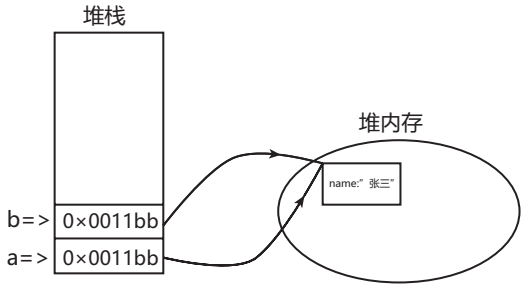


图 1-14　第 2~4 行代码执行内存图

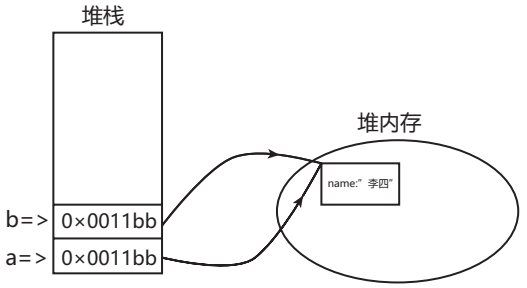


图 1-15　第 5 行代码执行内存图

第 6 行代码在控制台上显示 a 的 name 属性，发现其值已经随着 b 的 name 属性的修改，变成了李四。

由此可见，a、b 两个引用类型的变量是相互影响、相互关联的。因为本质上变量 a 和变量 b 指向的是同一块内存空间。

4. 查看 JavaScript 数据类型

可以使用 typeof 操作符来查看 JavaScript 变量或常量的数据类型，例如：

```
typeof "John";            //"string"
typeof 123;               //"number"
typeof true;              //"boolean"
typeof undefined;         //"undefined"
typeof null;              //"object"
```

5. 数字类型 (number)

当变量需要存储与数字相关的数据时，通常将其定义为数字类型。例如，游戏结束后向用户反馈本次游戏得分，这个分值可以定义为数字类型。

JavaScript 只有一种数字类型：number 类型。该类型既可以用来保存整数值，也可以用来保存小数 (浮点数) 值。例如：

```
var num1=34;              // 整数
var num2=34.00;           // 浮点数
```

极大或极小的数字可以通过科学 (指数) 记数法来书写，例如：

```
var num3=1.23e5;          //123000
var num4=1.23e-5;         //0.0000123
```

通过 typeof 运算符对上述四个变量进行类型查看，我们可以发现，返回结果均为 number 类型。

下面介绍一下在 JavaScript 中如何表示不同进制的数字。

计算机中常见的进制有二进制、八进制、十进制、十六进制，其中，我们人类使用最多的是十进制。JavaScript 通过在数字前添加前导字符来区分不同进制的数据，具体规则如下。

(1) 二进制数据：数字前加 0b 或 0B 来标识。

(2) 八进制数据：数字前加 0 来标识。

(3) 十六进制数据：数字前加 0x 或 0X 来标识。

示例如下：

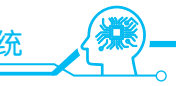
```
var num1=0b1010;          // 二进制表示的十进制数 10
var num2=016;             // 八进制表示的十进制数 14
var num3=0xA;             // 十六进制表示的十进制数 10
```

注意：

- 二进制数字序列范围：0~1。
- 八进制数字序列范围：0~7。
- 十六进制数字序列范围：0~9、A~F。

JavaScript 数值类型中，还存在着几个特殊的量值，具体如下。

- Number.MAX_VALUE 表示最大值，大小为 1.7976931348623157e+308。



- Number. MIN_VALUE 表示最小值，大小为 5e-324。
- Infinity 表示无穷大。
- -Infinity 表示无穷小。
- NaN 表示非数字 (Not a Number)。

示例如下：

```
console.log(Number.MAX_VALUE*2);          //Infinity
console.log(-Number.MIN_VALUE*2);         //-1e-323
console.log("abc"-123);                   //NaN
```

可以通过 isNaN() 函数来判断某个变量是不是数字。当参数不是数字时，该函数返回值为 true；当参数是数字时，该函数返回值为 false。这在要求变量必须为数字类型的需求中非常好用。例如：

```
console.log(isNaN("abc"));                //true
console.log(isNaN(12));                   //false
```

6. 字符串类型 (string)

计算机最主要的功能之一，就是为人类提供有价值的信息，而这些信息通常是以人类可识别的字符串的形式呈现的。所以，字符串类型是一类使用率非常高的数据类型。JavaScript 中不仅提供了字符串类型，而且为这种数据类型提供了非常丰富的内置函数。

string 类型用于表示由零或多个 16 位 Unicode 字符组成的字符序列，即字符串。在 JavaScript 中，字符串可以用双引号或单引号表示。例如：

```
var stuName=" 张三 ";
var stuName=' 张三 ';
```

上述代码的功能是一样的，即声明一个字符串类型的变量并为其赋值。ECMAScript 中的这两种语法形式没有什么区别。用双引号表示的字符串和用单引号表示的字符串完全相同。但是，以双引号开头的字符串必须以双引号结尾，以单引号开头的字符串也必须以单引号结尾，即单引号和双引号不能交叉使用。

string 类型有一个名为 length 的属性，用于指定字符串中的字符数，也就是获取一个字符串的长度。例如：

```
var stuName=" 东方红 ";
console.log(stuName.length);              //3
var nickName="Jack";
console.log(nickName.length);             //4
```

在 JavaScript 中，还存在着一类特殊的字符：转义字符。

转义字符是字符的一种间接表示方式。在特殊语境中，无法直接使用字符自身。例如，在字符串内容中包含双引号或单引号时，由于双引号或单引号是字符串的特殊标识，不能直接写出，必须借助转义字符。

假如要显示的内容是：他大声说：“去过这么多国家，我最爱的还是我的祖国！”，可以使用转义字符 \” 来表示双引号，代码如下：

```
var info=" 他大声说：\" 去过这么多国家，我最爱的还是我的祖国！ \"";
```

或者利用单引号来标识字符串，那么双引号就可以像普通字符一样直接写出了，实现同样效果的代码如下：

```
var info=' 他大声说：" 去过这么多国家，我最爱的还是我的祖国！ "';
```

在 JavaScript 中，反斜杠加上字符可以表示字符自身。注意，一些字符加上反斜杠后会表示特殊字符，而不是原字符本身，这些特殊转义字符被称为转义序列，具体内容如表 1-1 所示。

表 1-1　转义字符序列

序列	代表字符
\0	Null 字符 (\u0000)
\b	退格符 (\u0008)
\t	水平制表符 (\u0009)
\n	换行符 (\u000A)
\v	垂直制表符 (\u000B)
\f	换页符 (\u000C)
\r	回车符 (\u000D)
\"	双引号 (\u0022)
\'	撇号或单引号 (u0027)
\\	反斜杠 (\u005C)

注意：如果在一个正常字符前添加反斜杠，JavaScript 会忽略该反斜杠。例如：

```
document.write(" 他兴奋地说 :\" 我终于 \ 看 \ 到天安门啦！ \"");
```

等同于下面的代码：

```
document.write(" 他兴奋地说 :\" 我终于看到天安门啦！ \"");
```

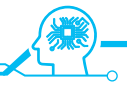
JavaScript 还为字符串类型提供了丰富的内置函数，借助这些函数，我们可以方便地完成字符串操作。例如，可以使用 concat() 函数将两个字符串拼接在一起。需要注意的是，调用 concat() 函数不会改变调用者字符串，返回的字符串是新创建的字符串。示例代码如下：

```
var str1="hello";
var str2="world";
var str3=str1.concat(str2);      //helloworld
```

使用 + 也可以完成字符串拼接的效果，上述代码也可以采用下面的方式实现：

```
var str1="hello";
var str2="world";
var str3=str1+str2;              //helloworld
```

关于更多的字符串内置函数，可以查阅本书《实战主题 2　验证码及其应用》或菜鸟教程等互联网资源。由于内置函数开箱即用，无须关注底层实现逻辑，只要按照函数要求填充参数即可得到所需结果，使用起来非常简单，这里不再赘述。



实战小贴士

随着所学知识的增多，完成一件事情的方法也越来越多，择优而用，久而久之代码质量会越来越高。在实践中要不断积累，胸有丘壑，方能游刃有余。人生中的每一步，都意义非凡，过程甚至比结果还要重要。

7. 布尔类型

布尔类型是基本类型的一种，该类型只有两个值：true 和 false，其中，true 表示真 (即对)，false 表示假 (即错)。布尔类型的值也可以进行运算，但是最终有效的运算结果依然是 true 和 false 二者之一。

例如，下述代码定义了一个值为 false 的布尔值：

```
var isValid=false;
```

布尔类型的值是逻辑值，通常用于程序执行条件判断。在编写程序时，经常需要根据具体情况采取不同处理方案，此类需求就是布尔类型的用武之地。例如，在非此即彼的 if 语句中，经常采用如下逻辑去编写代码：

```
if （判定条件）{
  方案 1 代码
} else{
  方案 2 代码
}
```

其中的判定条件就是布尔值，它可以是一个布尔变量，也可以是若干个布尔变量组成的布尔表达式。

布尔类型的值通常进行的有效运算是布尔运算，也就是逻辑运算，具体如何计算在“1.2.10 JavaScript 运算符”一节中会有专门介绍。布尔类型的值也可以进行其他运算，如加法运算。当布尔值参与加法运算时，true 被当成 1 来使用，false 被当成 0 来使用，示例如下：

```
console.log(true+5);      //6
console.log(false+5);     //5
```

8. 空值 null

null 是一种基本类型，表示有意不包含任何对象值。如果看到 null(分配给变量或由函数返回)，那么在那个位置原本应该是一个对象，但由于某种原因，该对象没有创建。null 通常用作一个对象类型变量的初始值。

如果将变量想象成一个盒子，那么：如果这个盒子里存放了东西，比如苹果，则该变量拥有一个具体值；如果这个盒子里什么也没有，则该变量的值为 null。

示例代码如下：

```
var user=null;
console.log(user);        //null
```

9. 未定义 undefined

如果一个变量声明了，但是未赋值，那么该变量就是 undefined(未定义) 类型。

例如，对于下述代码，第一行代码只是声明了变量 user，并没有为其赋值，所以，第二行代码的输出结果为 undefined。

```
var user;
console.log(user);                          //undefined
```

那么，undefined 类型值可以进行运算吗？答案是肯定的，只是与将布尔值进行逻辑运算之外的其他运算如出一辙，非常少见。

示例代码如下：

```
var he;
console.log(he);
console.log(he+"is a student。");          //undefined is a student。
console.log(he+10);                         //NaN
```

10. 对象类型

在 ES5 中，流传着“一切皆对象”的说法，可见对象在 JavaScript 语言中的地位非常重要。这里主要对 JavaScript 对象的基础知识进行介绍，更深层次的知识会在《实战主题 5 打地鼠游戏》，即本书最后一章中进行介绍。

JavaScript 对象是拥有属性和方法的数据。从本质上来讲，对象就是一组“键值对”(key-value) 的集合，属于一种无序的复合数据结构。

在 JavaScript 对象中，键名与键值之间采用半角冒号分隔，多个键值对之间用半角逗号分隔，最后一组键值对后面无须加逗号。

根据键值的不同，将对象的成员分为两种：属性和方法。

- 属性用于封装对象的数据，表示与对象有关的值。
- 方法用于封装对象的行为，表示对象可以执行的动作或可以完成的功能。

【例 1-6】定义一个学生对象，使其拥有 2 个属性和 1 个方法。示例代码如图 1-16 所示。

例 1-6

```
var student={
    studentName:"张三",
    studentAge:20,
    showInfo:function(){
        console.log("我是学生");
    }
}
```

图 1-16　对象示例代码

图 1-16 所示的代码中，花括号定义了一个对象，该对象被赋值给变量 student，即 student 指向一个对象。

对象内部包含 3 个成员：2 个属性和 1 个方法。

第一个键值对为属性，属性名为 studentName，属性值为“张三”。

第二个键值对也为属性，属性名为 studentAge，属性值为 20。

第三个键值对为方法，方法名为 showInfo，方法的值 (也就是定义) 为冒号后面的内容。

可以通过以下两种方法使用对象成员。

方法 1：点表示法。

可以通过“对象名 . 成员名”来访问对象的成员。

【例 1-7】通过点运算符使用对象成员。代码如图 1-17 所示。

```
var student = {
     studentName: "张三",
     studentAge: 20,
     showInfo: function () {
         console.log("我是学生");
     }
}
console.log(student.studentName);//输出：张三
console.log(student.studentAge);//输出：20
student.showInfo();//输出：我是学生
```

图 1-17　通过点运算符使用对象成员

例 1-7

方法 2：方括号表示法。

对象的成员也可通过对象 [" 键名 "] 来进行调用，注意方括号里面的键名必须加引号。

【例 1-8】通过键名使用对象成员。代码如图 1-18 所示。

```
var student = {
     studentName: "张三",
     studentAge: 20,
     showInfo: function () {
         console.log("我是学生");
     }
}
console.log(student["studentName"]);//输出：张三
console.log(student["studentAge"]);//输出：20
student["showInfo"]();//输出：我是学生
```

图 1-18　通过键名使用对象成员

例 1-8

对象主要分为三类：

(1) 内置对象 / 原生对象。指 JavaScript 语言本身预定义的对象，在 ECMAScript 标准中定义，由所有的浏览器厂家来提供具体实现，由于标准统一，这些对象的浏览器兼容性问题相对较小。

(2) 宿主对象。指 JavaScript 运行环境 (即浏览器) 提供的对象，由浏览器厂家自行定义并实现，早期存在较大的兼容性问题，目前其中一些主要的对象已经被大部分浏览器兼容，具体分为 BOM 对象和 DOM 对象两大类。

(3) 自定义对象。指由用户创建的对象，兼容性问题需要编程者注意。

在使用对象类型数据时，需要注意如下内容：

- 对象可以嵌套对象。即对象的成员的值又是一个对象。例如：班级对象里面嵌套着班主任对象。

【例 1-9】嵌套对象。代码如图 1-19 所示。


例 1–9

```
var myClass={
    className:"网站规划与开发设计2022-02班",
    classTutor:{
        tuTorName:"张三",
        tuTorTel:"15923456789"
    }
}
console.log(myClass.className);//"网站规划与开发设计2022-02班"
console.log(myClass.classTutor.tuTorName);//"张三"
myClass.className="人工智能2022-02班";
console.log(myClass.className);//"人工智能2022-02班"
```

图 1-19　嵌套对象

- 对象的属性具有唯一性。对象属性如果被多次赋值，那么后面的属性值将覆盖该属性的原值。例如，例 1-9 中第 10 行代码对 className 属性进行了重新赋值，因此，第 11 行的执行结果为最新的赋值内容：人工智能 2022-02 班。

实战小贴士

每一种数据类型都有其最擅长的本领和最适合的工作场景，所谓术业有专攻，在软件开发过程中，为变量定义数据类型时，建议选择合适的数据类型，而非随性而为。

1.2.10　JavaScript 运算符

通常所说的代码，是指一条表达式语句。而运算符则是构成表达式的基本元素。

1. 算术运算符

算术运算符是指主要用于数学计算的运算符。JavaScript 中包含的算术运算符如表 1-2 所示。

表 1-2　算术运算符

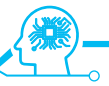
运算符	描述	x	y	举例
+	加法运算符	5	2	x+y // 执行加法操作，结果为 7
-	减法运算符	5	2	x-y // 执行减法操作，结果为 3
*	乘法运算符	5	2	x*y // 执行乘法操作，结果为 10
/	除法运算符	5	2	x/y // 执行除法操作，结果为 2.5
%	取模 (余数)	5	2	x%y // 执行取模操作，结果为 1

++	自加	5		y=x++; //x 以原值参与运算，执行加 1 操作。结果为 x=6，y=5
				y=++x; //x 先加 1 之后，再参与运算，结果为 x=6，y=6

--	自减	5		y=x--; //x 以原值参与运算，执行减 1 操作。结果为 x=4，y=5
				y=--x; //x 先减 1 之后，再参与运算，结果为 x=4，y=4

表 1-2 中，加、减、乘、除、取模与数学课上所学的概念相同，故不再赘述。这里重点介绍自加和自减操作。

当自加或自减符号放到操作数的前面时，操作数本身先进行自加或自减操作，然后再以操作后的值来参与运算。



当自加或自减符号放到操作数的后面时，操作数先以本身的原值参与运算，然后再进行自加或自减操作。

另外，还需注意：当进行加法运算的操作数为字符串类型数据时，+ 当作字符串串联符号来使用。

2. 赋值运算符

在 JavaScript 中，运算符“=”用于赋值操作。它是使用率最高的运算符之一。例如，在开发打地鼠游戏的过程中，每次游戏期间，都需要记录用户打中地鼠的次数。在游戏刚开始时，需要将游戏数据复位，这些都需要用到赋值操作。

赋值运算符举例如下：

```
count=0;            // 变量初始化
count=count+1;      // 变量被重新赋值
```

注意：赋值运算符不是比较大小中的等于运算符，后者是比较运算符，有专门的符号，后续会有相关介绍。

在 JavaScript 中，还可以利用赋值运算符来简写算术表达式。例如：

```
x+=5 等同于 x=x+5;
x-=5 等同于 x=x-5;
x*=5 等同于 x=x*5;
x/=5 等同于 x=x/5;
x%=5 等同于 x=x%5;
```

3. 比较运算符

比较运算符用于操作数的比较运算。在 JavaScript 中，比较运算符及其使用规则如表 1-3 所示。

表 1-3　比较运算符

运算符	描述	x 值	举例
>	大于	3	x>5; //false
<	小于	3	x<5; //true
>=	大于或等于	3	x>=5; //false
<=	小于或等于	3	x<=5; //true
==	等于 (值相等时返回 true)	3	x==3; //true　x==4; //false
!=	不等于 (值不相等时返回 true)	3	x!=5; //true　x!="3"; //false
===	恒等 (值相等并且类型相等时，返回 true)	3	x==="3"; //false　x===3; //true
!==	不恒等 (值或类型不相等时，返回 true)	3	x!==5; //true　x!=="3"; //true

4. 逻辑运算符

JavaScript 中的逻辑运算符用于操作数的逻辑运算，其计算的结果通常用于控制分支结构的走向。

在 JavaScript 中，常用的逻辑运算符有三种，分别为逻辑非 (!)、逻辑与 (&&) 和逻辑或 (||)，具体计算规则及举例如表 1-4 所示。

表 1-4　逻辑运算符

逻辑运算符	运算规则	x	y	举例
!(逻辑非)	对一个值进行非运算，即：对一个布尔值进行取反操作	true		!x //false
&&(逻辑与)	对符号两侧的值进行与运算并返回结果，计算规则：只要有一个值为 false 就返回 false；只有两个值全为 true 时才返回 true	false	true	x&&y //false
||(逻辑或)	对符号两侧的值进行或运算并返回结果，计算规则：只要有一个值为 true 就返回 true；只有两个值全为 false 时才返回 false	true	false	x || y //true

注意：JavaScript 中的“与”属于短路的与，如果第一个值为 false，则不会检查第二个值；JavaScript 中的“或”属于短路的或，如果第一个值为 true，则不会检查第二个值。此外，JavaScript 中还包含按位逻辑运算符，例如：按位与 (&)、按位或 (|)、按位异或 (^)、按位非 (~)，此类运算符主要用于处理二进制数据，在前端开发中应用相对较少，感兴趣读者可自行查阅相关资料，这里不再赘述。

5. 条件运算符

条件运算符也叫三元运算符，它是 JavaScript 中唯一使用三个操作数的运算符。语法如下：

```
条件表达式？表达式 1：表达式 2
```

当条件表达式的返回值为真时，返回表达式 1 的值；为假时，返回表达式 2 的值。

由三元运算符构成的表达式，可以看作双分支 if 语句的简写方式。例如，图 1-20 中的代码与图 1-21 中的代码，均可用于模拟打地鼠游戏中游戏成绩的计算功能。

```
1  //模拟打地鼠成绩计算
2  var count=70;//打中地鼠的次数
3  var total=100;// 地鼠一共出来的次数
4  var grade=(count/total*100>=60)?"过关":"惨败";
5  console.log("本次游戏的战绩为",grade);
```

图 1-20　使用三元运算符模拟游戏成绩计算

```
1  //模拟打地鼠成绩计算
2  var count=70;//打中地鼠的次数
3  var total=100;// 地鼠一共出来的次数
4  var grade=0;// 成绩
5  if(count/total*100>=60){
6      grade="过关";
7  }else{
8      grade="惨败";
9  }
10 console.log("本次游戏的战绩为",grade);
```

图 1-21　使用 if 语句模拟游戏成绩计算

图 1-20 和图 1-21 中的两段代码各有千秋：三元运算符编写的代码更加简洁，而 if 语句编写的代码可读性更高。编程过程中，读者可以根据自己的喜好选择其一。

6. 运算符的优先级

JavaScript 中的运算符具有不同的优先级，这些优先级决定了表达式中运算符的执行顺序。优先级高的运算符会作为优先级低的运算符的操作数。表 1-5 按照从高到低的顺序，列举了一些常见运算符的优先级。

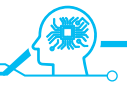


表 1-5　运算符优先级

运算符	描述
圆括号 ()	圆括号在 JavaScript 中拥有最高的优先级，可以覆盖其他所有运算符的默认优先级。编程时可以使用圆括号改变表达式的计算顺序
成员访问 . 和 函数调用 ( )	成员访问运算符 . 用来访问对象的属性。函数调用运算符 () 则用来调用函数并执行
一元运算符	++、--、+、-、!
乘性运算符	乘法 (*)、除法 (/)、取模 (%)
加性运算符	加法 (+)、减法 (-)
关系运算符	<、>、<=、>=、instanceof
相等性运算符	==、!=、===、!==
逻辑与运算符	&&
逻辑或运算符	||
条件 (三元) 运算符	? :
赋值运算符 (=)	=、+=、-=、*=、/=

注意：如果不想记忆这些琐碎的优先级顺序，可以使用优先级最高的圆括号 () 来人为确定优先级，最内层括号中的运算符优先级最高。

1.2.11　JavaScript 数据类型转换

数据类型转换是指将一种类型的数据转换为另外一种类型数据的操作。例如，将字符串类型的 "1" 转换为数字类型的 1，将字符串类型的 "2024/7/27" 转换为日期类型，表示 2024 年 7 月 27 日。

在大大小小、或简单或复杂的程序中，经常会遇到各种不同类型的数据之间相互转换的需求。因为用户提供的数据，往往与程序加工的合法数据存在一定的差异。例如，取款时，用户通过文本框输入取款金额，由于文本框的 value 值是字符串类型，必须被转换成数字类型之后，才能与账户余额进行运算。计算机最主要的工作之一，就是对输入的数据进行加工，然后返回加工后的结果。大多数情况下，粗加工就是数据类型转换，精加工则是对转换后的数据根据既定的算法进行进一步处理。

JavaScript 中的数据类型转换主要分为两种：隐式类型转换和显式类型转换。

1. 隐式类型转换

所谓隐式转换，是指在写代码时不做特定处理，但是 JavaScript 解释器会自动进行数据类型转换的操作。它是按照 JavaScript 内置的既定规则进行的。

下面介绍几种常见的隐式转换。

(1) 数字与字符串进行运算，产生结果为字符串。示例如下：

```
console.log(1+2+"3");            //"33"
console.log(true+"12");          //"true12"
console.log(false+"12");         //"false12"
console.log("12"+null);          //"12null"
console.log("12"+undefined);     //"12undefined"
```

(2) 数字与数字进行运算，产生结果为数字。示例如下：

```
console.log(1+2+3);                //6
console.log(true+12);              //13
console.log(false+12);             //12
console.log(12+null);              //12
console.log(12+undefined);         //NaN
```

(3) 数字与数字字符串进行运算，产生结果为数字。示例如下：

```
console.log(10-"20");              //-10
console.log(10*"20");              //200
console.log("10"/"20");            //0.5
```

(4) 数字与非数字字符串进行运算，产生结果为 NaN。示例如下：

```
console.log(10-"one");             //NaN
console.log(10*"102a");            //NaN
console.log("10"/"one");           //NaN
console.log("10"/"true");          //NaN
```

2. 显式类型转换

显式类型转换是指将数据类型强制转换为某种类型。有两种方式可以完成显式类型转换：使用类型转换函数和使用专用函数。

(1) 使用类型转换函数来实现数据类型转换。使用 Boolean() 函数、Number() 函数、String() 函数、Object() 函数，可完成类型转换。可以看出，这些函数以数据类型命名。示例如下：

```
console.log(Number("3"));          //3
console.log(String(false));        //"false"
console.log(Boolean("3"));         //true
console.log(Object(3));            //Number{3}
```

注意： 在使用 Boolean() 函数进行类型转换时，代表空、否定的值会被转换为 false，如""、0、NaN、null 和 undefined，其余的值会转换为 true。

(2) 使用专用函数完成数据类型转换。JavaScript 中提供了专门的函数和方法来实现数字与字符的转换。

① 不带参数的 toString() 可将数据强制转换为字符串类型。例如：

```
var a=10;
var b=a.toString();
console.log(typeof b);             // "string"
console.log(b);                    //"10" 字符串
```

② 带有进制参数的 toString() 可将数字转换为指定进制。例如：

```
var a=10;
var b=a.toString(2);
console.log(b);                    //1010   将十进制数字 10 转换为二进制
```

③ toFixed() 可以接收一个小数位数参数，将数值转换为字符串，并按照四舍五入规则保留指定位数的小数。在财务相关的系统中经常用到。例如：富士苹果 10 块钱 3 斤，买 5 斤需要多少钱？可用如下代码实现。

```
var price=10/3;
var total=(5*price).toFixed(2);
console.log(total);                    //16.67
```

上面介绍了将数字类型数据转换为字符串类型数据的方法，下面介绍一些将字符串类型数据转换为数字类型数据的方法。

④ parseInt(string) 函数采用下取整算法，将字符串强制转换为对应的整数。例如：

```
var price="10.99";
var total=5*parseInt(price);
console.log(total);                    //50
```

注意： parseInt() 从被转换字符串的起始位开始转换，直到遇到非数字字符为止。例如：parseInt("54a") 的结果为 54。如果被转换字符串的第一位就是非数字字符，则返回 NaN。布尔值、undefined、null 使用该函数转换为整型，会变成 NaN。

⑤ parseInt(string, radix) 函数可以将指定进制的字符串转换为整数。其中，参数 string 为要转换的字符串，radix 为进制，默认为十进制。如果不指定进制参数 (radix) 或将 radix 指定为 0，则按十进制进行解析，否则按 radix 指定的进制进行解析。如果指定了进制参数 (radix)，则可以省略前缀 "0" "0o" "0x"。该函数在涉及硬件的控制类应用中使用较多。将 110 从二进制转化成十进制的代码如下：

```
console.log(parseInt("110", 2)); // 6
```

⑥ parseFloat() 函数可将字符串转换为浮点数。示例代码如下：

```
var a="2.23456";
var b=parseFloat(a);
console.log(typeof b);                 //number
console.log(b);                        //2.23456
```

1.2.12　数据输入输出

数据输入和数据输出是编写程序的必备知识。在 JavaScript 中进行输入输出操作主要有六种方式，分别为普通输入 (prompt)、选择式输入 (confirm)、输出到警告框 (alert)、输出到 HTML 文档、输出到 HTML 元素、输出到浏览器控制台。

1. 普通输入

JavaScript 中可以通过 prompt() 函数以弹窗方式输入信息，该函数可以接收两个参数，其中：

(1) 第一个参数为提示字符串，用于提示用户需要输入信息的内容。

(2) 第二个参数为可选的默认值，用于在输入框中显示一个默认值。

例如：var yourName=prompt(" 请输入你的姓名：","爱国者 "); 执行结果如图 1-22 所示。

图 1-22　prompt 普通输入

2. 选择式输入

在 JavaScript 中可以通过 confirm() 函数以确认框的方式输入信息，常用于确认用户是否接受某种操作。当确认框弹出时，用户可以单击“确认”或“取消”按钮来确定操作意向。

(1) 当单击“确认”按钮时，确认框返回 true。

(2) 当单击“取消”按钮时，确认框返回 false。

例如：var yourChoice=confirm(" 选择男款，单击确定，选择女款，单击取消。"); 将得到如图 1-23 所示的执行效果。

3. 输出到警告框

JavaScript 中可以使用 alert() 函数将数据输出到警告框。例如：alert("Welcome to JavaScript World!"); 将得到图 1-24 所示的执行效果。

图 1-23　confirm 选择式输入

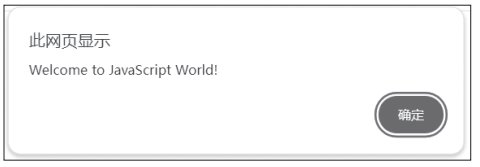


图 1-24　alert 警告框输出

实战小贴士

prompt()、confirm() 和 alert() 函数都是 window 对象的自带方法。因为 window 对象是一个全局对象，所以可以省略掉 window。这三个函数都是阻塞函数，如果不对其进行响应，后面的内容就不会加载出来。

4. 输出到 HTML 文档

在 JavaScript 中可以使用 document.write() 方法将内容写到 HTML 文档中。

语法：document.write(content);

参数：content 是必选项，类型为字符串，可以是变量或值为字符串的表达式，写入的内容常常包括 HTML 标签。

【例 1-10】将信息输出到页面文档。示例代码如图 1-25 所示，执行效果如图 1-26、图 1-27 所示。

例 1-10

```
1 var msg=prompt("请输入姓名");
2 document.write("欢迎你，"+msg);
3 var str="<h1 style='color:green;'>"+msg+"</h1>"
4 document.write(str);
```

图 1-25　将信息输出到页面文档示例代码

图 1-26　输入姓名

图 1-27　将输入的姓名输出到页面

5. 输出到 HTML 元素

JavaScript 允许通过为 HTML 元素属性赋值的方式，将数据输出到 HTML 元素中。例如，可以将 div 的 innerHTML 赋值为一个 HTML 字符串，或者将 div 的 innerText 属性赋值为具体的文本。

【例 1-11】将信息输出到页面元素。示例代码如图 1-28 所示。执行效果如图 1-29 所示。

本例列举了通过为 innerHTML 和 innerText 两个属性赋值来实现将信息输出到页面的方法。前者可以包含 HTML 标签，后者只能包含纯文本。

```
<body>
    <div id="content1"></div>
    <div id="content2"></div>
    <div id="content3"></div>
    <script>
        var div1 = document.getElementById("content1");
        div1.innerHTML = "我是中国人，我热爱我的祖国。";
        var div2 = document.getElementById("content2");
        div2.innerText = "我是中国人，我热爱我的祖国。";
        var div3 = document.getElementById("content3");
        div3.innerHTML = "<span style='color:green'>我是中国人，我热爱我的祖国。</span>";
    </script>
</body>
```

图 1-28　将信息输出到页面元素代码

例 1-11

注意：

- document.getElementById() 函数可根据 id 值获取对应的页面元素。
- 页面元素通过点标识符 + 属性名的方式获取或设置相关属性。
- 只能为能够显示文本的页面元素设置 innerText 和 innerHTML 属性。

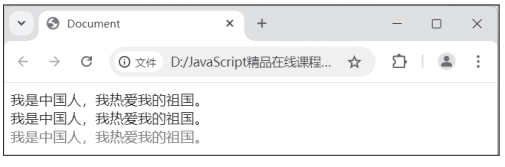


图 1-29　将信息输出到页面元素执行效果

6. 输出到浏览器控制台

JavaScript 允许通过 console.log() 语句将数据输出到浏览器控制台。

语法：console.log(参数1，[参数2,参数3...]);

console.log() 是一个非阻塞函数，因此常被用于程序后台调试。

【例 1-12】将数据输出到控制台。示例核心代码如图 1-30 所示，执行效果如图 1-31 所示。

```
<script>
    var count=10;
    console.log(count);
    console.log("本次得分：",count);
</script>
```

图 1-30　数据输出到控制台示例代码

例 1-12

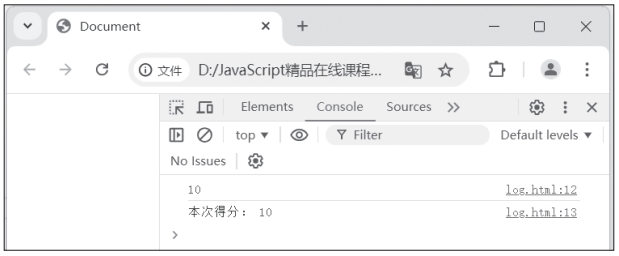


图 1-31　将数据输出到控制台执行效果

实战小贴士

当程序需要进行排错时，通常使用 console.log() 方法进行数据追踪。可以直接将数据输出到控制台，也可以加上特定前缀后再输出到控制台，以便对不同的变量进行区分。

1.2.13　if 语句

组成程序的代码可能有很多行，那么它们到底谁先执行谁后执行呢？或者说，执行完一行代码后，接着执行哪一行代码呢？一个程序有条不紊地按照既定的顺序执行，从而达到程序预期的效果，“流程控制”这个超级指挥官功不可没。

所谓流程控制，就是控制代码的执行顺序。所有语言的流程控制都分为如下三大类。

(1) 顺序结构。按照定义的顺序，从上往下依次执行代码。

(2) 分支结构。根据不同的情况，执行不同的分支代码。

(3) 循环结构。重复执行某些代码块，直到满足特定条件为止。

这些流程控制语句的执行逻辑在所有的编程语言中都是相同的，只是不同语言中的语法可能有所不同。如果读者曾经学习过其他编程语言，那么只需关注具体语法即可。如果这是读者学习的第一门编程语言，那么请务必搞清其中的执行逻辑。

顺序结构的逻辑较为简单，只是单一的线性顺序，因此这里不再赘述。下面主要介绍 JavaScript 中用于实现分支结构的 if 语句。

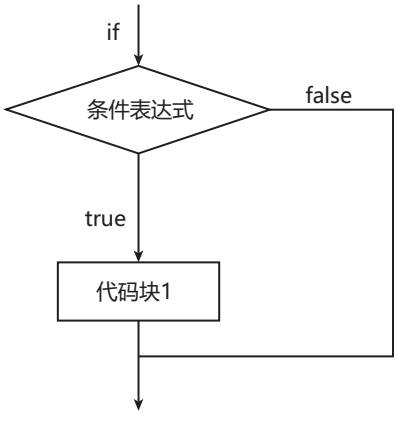


图 1-32　单分支 if 语句执行流程

1. 单分支 if 语句

单分支 if 语句是最简单的 if 语句，语法如下：

```
if （条件表达式）{
    代码块 1
}
```

当条件表达式的值为真时，执行代码块 1 中的语句，否则直接跳过代码块 1，执行花括号后面的语句，执行流程如图 1-32 所示。

【例 1-13】单分支 if 语句应用举例。代码如图 1-33 所示。

```
<script>
  var score = prompt("请输入成绩！");
  if (score >= 60) {
     alert("你已经通过科目2考试！恭喜！");
  }
 alert("请有序离开考场！");
</script>
```

例 1-13

图 1-33　单分支 if 语句示例代码

上述代码运行后，只有当通过 prompt 输入框输入的成绩大于或等于 60 时，才会收到恭喜信息和有序离场信息，否则只能收到离场信息。

2. 双分支 if 语句 (if...else... 语句)

双分支 if 语句是非此即彼的条件语句，语法如下：

```
if （条件表达式）{
    语句块 1
}else{
    语句块 2
}
```

当条件表达式的值为真时，执行语句块 1 中的代码，否则执行语句块 2 中的代码，执行流程如图 1-34 所示。

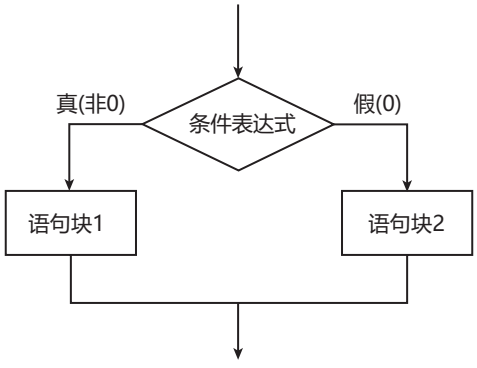


图 1-34　双分支 if 语句执行流程

【例 1-14】参加了 MCSE 认证考试的人员，会根据成绩的不同，得到不同的信息。示例代码如图 1-35 所示。

```
<script>
  var score = prompt("请输入成绩！");
  if (score >= 60) {
    alert("你已经通过MCSE考试！pass！");
  } else {
    alert("你没有通过MCSE考试！failed！");
  }
  alert("请有序离开考场！");
</script>
```

例 1-14

图 1-35　双分支 if 语句示例代码

上述代码运行后，会弹出一个 prompt 弹框，提示用户输入百分制成绩，如果成绩大于或等于 60，则弹出考试通过的相关信息；否则，弹出考试失败的相关信息。无论是否通过，都会显示“请有序离开考场！”的信息。

3. 多分支 if 语句（嵌套 if 语句）

在 JavaScript 中，if…else if…else 语句可用于对多个条件进行判断，并根据判断结果进行多种不同的处理，语法如下：

```
if （表达式 1){
  语句 1
}else if( 表达式 2){
  语句 2
}...
else if （表达式 n-1){
     语句 n-1
}else{
   语句 n
}
```

当条件表达式 1 的值为真时，执行语句块 1 中的代码，否则查看条件表达式 2 的值。当其值为真时，执行语句块 2 中的代码，否则继续查看其他条件表达式的值，以此类推，直至查看表达式 n-1 的值 。若其值为真，则执行语句块 n-1 中的代码；否则，执行语句块 n 中的代码。多分支 if 语句的执行流程如图 1-36 所示。

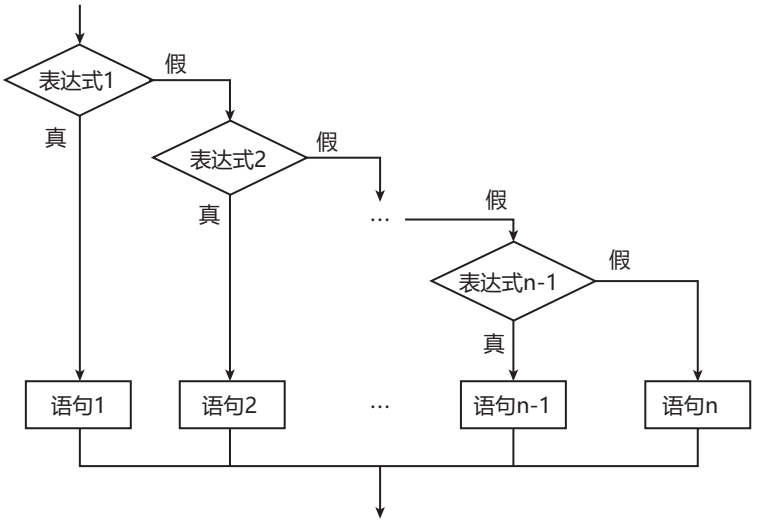


图 1-36　多分支 if 语句执行流程

【例 1–15】多分支 if 语句应用，具体需求如下：

- 当积分 >=5000 时，显示“您的身份为超级会员！”
- 当积分 >=3000 时，显示“身份为白金会员！”
- 当积分 >=2000 时，显示“身份为青铜会员！”
- 当积分 <2000 时，显示“身份为普通会员！”

示例代码如图 1-37 所示。

```
<script>
  var points = 5900;
    if (points >= 5000) {
       alert("您的身份为超级会员！");
    } else if (points >= 3000) {
       alert("身份为白金会员！");
    } else if (points >= 2000) {
       alert("身份为青铜会员！");
    } else {
       alert("身份为普通会员！");
    }
</script>
```

例 1-15

图 1-37　多分支 if 语句示例代码

在示例代码中，points >= 3000 出现在 else if 分支的控制条件中，因此，它隐含了 points<5000 的信息 (即不满足 points>=5000 的情况)。必须明确 else 的含义，否则可能会出现将该条件写成 points >= 3000&&points<5000 的情况。

1.3 《成绩转换系统 V1.0》编程实现

在了解 JavaScript 的基础编码常识之后，就可以分步骤实现《成绩转换系统 V1.0》了。

(1) 创建 scoreV1.0.html 并迅速生成如下代码框架：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
</body>
</html>
```

(2) 在 <head></head> 区域修改 title，添加样式：

```
<title> 成绩转换系统 V1.0</title>
```

(3) 在 <body></body> 区域添加如下代码：

```
<script>
 var score = prompt(" 请输入成绩：");
 if(score >= 90){
     alert(" 优秀 ");
 }else if(score >= 80){
     alert(" 良好 ");
 }else if(score >= 70){
     alert(" 中等 ");
 }else if(score >= 60){
```

scoreV1.0