目 录

Contents

第 1 章

Java Web 应用开发简介 …… 1

1.1 Web 发展历程简介 …… 2
1.1.1 Web 的起源 …… 2
1.1.2 Web 的发展 …… 2
1.1.3 Web 的影响 …… 3
1.2 Web 应用程序的工作原理 …… 3
1.2.1 程序开发体系结构 …… 3
1.2.2 静态网站与动态网站 …… 4
1.3 Web 开发技术 …… 5
1.3.1 客户端技术 …… 5
1.3.2 服务器端技术 …… 5
1.4 常用上网资源 …… 6

第 2 章

JavaScript 脚本语言 …… 7

2.1 JavaScript 概述 …… 8
2.2 JavaScript 基础语法 …… 8
2.2.1 JavaScript 的基本用法 …… 8
2.2.2 JavaScript 的变量 …… 11
2.2.3 JavaScript 的关键字 …… 12
2.2.4 JavaScript 的数据类型 …… 12
2.2.5 JavaScript 的运算符 …… 13
2.3 函数 …… 15
2.3.1 函数的定义与调用 …… 16
2.3.2 函数参数 …… 18
2.3.3 函数范围 …… 19
2.4 事件 …… 20
2.4.1 常见事件 …… 20
2.4.2 事件操作 …… 20

第 3 章

Java Web 开发环境配置 …… 21

3.1 Web 容器简介 …… 22
3.2 Tomcat 安装与配置 …… 22
3.2.1 Tomcat 下载 …… 22
3.2.2 Tomcat 安装 …… 23
3.2.3 Tomcat 配置 …… 25
3.3 Eclipse 简介 …… 29
3.3.1 Eclipse 下载 …… 29
3.3.2 Eclipse 安装 …… 30
3.3.3 Eclipse 配置 …… 31
3.3.4 Eclipse 配置 Tomcat …… 32
3.4 IDEA 配置 Tomcat 服务器 …… 34
3.4.1 下载 IDEA …… 34
3.4.2 安装 IDEA …… 35
3.4.3 IDEA 配置 Tomcat …… 37
3.5 开发环境测试 …… 42

第 4 章

JSP 基础语法 …… 43

4.1 JSP 概述 …… 44

4.2 JSP 注释 ····· 45
4.3 Scriptlet 标签 ····· 45
4.4 指令标识 ····· 46
4.4.1 page 指令 ····· 47
4.4.2 include 指令 ····· 48
4.4.3 taglib 指令 ····· 50
4.5 脚本标识 ····· 50
4.5.1 JSP 表达式 ····· 50
4.5.2 声明标识 ····· 51
4.5.3 代码片段 ····· 51
4.6 动作标识 ····· 52
4.6.1 文件包含标识 ····· 52
4.6.2 请求转发标识 ····· 53
4.6.3 传递参数标识 ····· 54

第 5 章
JSP 内置对象 ····· 55

5.1 JSP 内置对象概述 ····· 56
5.2 page 对象 ····· 56
5.3 request 对象 ····· 57
5.3.1 HTTP 协议 ····· 57
5.3.2 GET 请求传递参数 ····· 59
5.3.3 POST 请求 ····· 60
5.3.4 resquest 对象常用方法 ····· 61
5.4 response 对象 ····· 63
5.4.1 response 四种功能 ····· 63
5.4.2 操作 Cookie ····· 64
5.5 session 对象 ····· 65
5.5.1 获取 session ····· 65
5.5.2 session 添加、获取、移除数据 ····· 67
5.5.3 session 超时、设置缺省时间 ····· 67
5.6 application 对象 ····· 68
5.7 out 对象 ····· 69
5.8 pageContext 对象 ····· 69
5.9 config 对象 ····· 70
5.10 exception 对象 ····· 70

第 6 章
JavaBean 技术 ····· 71

6.1 JavaBean 概述 ····· 72
6.2 JavaBean 的应用 ····· 72
6.3 JSP 中使用 JavaBean ····· 73
6.3.1 添加属性值 ····· 74
6.3.2 获取属性值 ····· 74
6.4 JavaBean 的保存范围 ····· 74
6.5 删除 JavaBean ····· 75

第 7 章
Servlet 技术 ····· 77

7.1 Servlet 概述 ····· 78
7.1.1 Servlet 技术体系 ····· 78
7.1.2 Servlet 特点 ····· 79
7.1.3 Servlet 与 JSP 的区别 ····· 79
7.2 第一个 Servlet 程序 ····· 80
7.3 Servlet 生命周期 ····· 81
7.4 Servlet 常用 API 接口和类 ····· 82
7.4.1 Servlet 实现相关的 Servlet 接口 ····· 82
7.4.2 与请求和响应相关的接口 ····· 85
7.4.3 与 Servlet 配置相关的接口 ····· 86
7.4.4 Servlet 上下文 ····· 86
7.4.5 请求转发 ····· 86
7.5 Servlet 开发 ····· 88
7.5.1 init( ) 方法 ····· 88
7.5.2 service( ) 方法 ····· 89
7.5.3 destroy( ) 方法 ····· 89
7.6 过滤器 ····· 89
7.7 监听器 ····· 91
7.7.1 监听 ServletContext 域的创建与销毁 ····· 91
7.7.2 监听 session 域的创建与销毁 ····· 91
7.7.3 监听 request 域的创建与销毁 ····· 92

第 8 章

EL 表达式语言 …… 93

8.1 EL 概述 …… 94
8.2 EL 运算符 …… 94
8.2.1 访问数据运算 …… 95
8.2.2 empty 运算 …… 96
8.2.3 算术运算 …… 97
8.2.4 条件运算 …… 98
8.2.5 逻辑运算 …… 99
8.3 EL 内置对象 …… 100
8.3.1 页面上下文对象 …… 100
8.3.2 访问作用域范围的内置对象 …… 103
8.3.3 访问环境信息的内置对象 …… 104
8.4 禁用 EL …… 107

第 9 章

JSP 标签库——JSTL …… 109

9.1 JSTL 概述 …… 110
9.2 JSTL 安装与配置 …… 110
9.3 JSTL 核心标签 …… 113
9.3.1 表达式标签 …… 113
9.3.2 流程控制标签 …… 117
9.3.3 循环控制标签 …… 120
9.3.4 url 相关标签 …… 124
9.4 格式化标签库 …… 127
9.4.1 数字格式化标签 …… 127
9.4.2 日期格式化标签 …… 129
9.4.3 国际化标签 …… 132
9.5 函数标签库 …… 133

第 10 章

Ajax 技术 …… 135

10.1 Ajax 概述 …… 136
10.2 XMLHttpRequest 对象 …… 136
10.2.1 创建 XMLHttpRequest 对象 …… 136
10.2.2 XMLHttpRequest 对象常用方法 …… 137
10.2.3 XMLHttpRequest 对象的常用属性 …… 138
10.3 Ajax 技术应用 …… 139
10.3.1 打招呼 …… 139
10.3.2 用户验证 …… 141
10.3.3 解析 XML 数据 …… 142
10.3.4 解决中文乱码问题 …… 144

第 11 章

Java Web 数据库编程 …… 145

11.1 JDBC 概述 …… 146
11.2 JDBC 常用类和接口 …… 146
11.2.1 Driver 接口 …… 146
11.2.2 DriverManager 类 …… 146
11.2.3 Connection 接口 …… 147
11.2.4 Statement 接口 …… 148
11.2.5 PreparedStatement 接口 …… 148
11.2.6 ResultSet 接口 …… 149
11.3 准备数据库 …… 150
11.4 JDBC 连接数据库 …… 150
11.5 JDBC 操作数据库 …… 152
11.5.1 插入数据 …… 152
11.5.2 查询数据 …… 154
11.5.3 更新数据 …… 159
11.5.4 删除数据 …… 161
11.5.5 批处理 …… 164
11.5.6 调用存储过程 …… 166
11.6 事务处理 …… 168
11.6.1 事务的概念 …… 168
11.6.2 JDBC 的事务支持 …… 169
11.7 Java Web 中应用 JDBC …… 171

第 12 章

Sping MVC 框架 …… 173

12.1 MVC 概述 …… 174
12.2 Spring MVC 概述 …… 174
12.2.1 Spring MVC 简介 …… 174
12.2.2 Spring MVC 运行流程 …… 174
12.3 Spring MVC 开发环境配置 …… 175
12.3.1 Maven 简介 …… 175
12.3.2 Maven 搭建 Spring MVC 项目 … 177
12.4 处理器和适配器 …… 185
12.4.1 非注解的处理器 …… 185
12.4.2 非注解的适配器 …… 187
12.4.3 注解的处理器和适配器 …… 188
12.5 前端控制器和视图解析器 …… 190
12.5.1 前端控制器与视图解析器的简介 … 190
12.5.2 前端控制器的分析 …… 190
12.5.3 视图解析器的分析 …… 195
12.6 请求映射和参数绑定 …… 200
12.6.1 @Controller …… 200
12.6.2 @RequestMapping …… 201
12.6.3 绑定过程 …… 202
12.7 拦截器 …… 204
12.7.1 拦截器概述 …… 204
12.7.2 拦截器使用 …… 204
12.7.3 拦截器链 …… 206
12.8 Spring MVC 与 RESTful …… 207
12.8.1 RESTful 概述 …… 207
12.8.2 Spring MVC 实现 RESTful 风格 … 207

第 13 章

MyBatis 框架 …… 209

13.1 MyBatis 概述 …… 210
13.1.1 MyBatis 介绍 …… 210
13.1.2 MyBatis 运行流程 …… 210
13.2 操作数据库 …… 211
13.2.1 准备数据库 …… 211
13.2.2 配置工程环境 …… 211
13.2.3 数据库连接配置 …… 213
13.2.4 编写数据库映射配置文件 …… 214
13.2.5 操作数据库 …… 215
13.3 SqlConfig 配置文件 …… 217
13.4 Mapper 映射文件 …… 218
13.5 MyBatis 高级映射 …… 220
13.5.1 一对一查询 …… 220
13.5.2 一对多查询 …… 222
13.5.3 多对多查询 …… 222
13.6 MyBatis 缓存 …… 223
13.6.1 一级缓存 …… 223
13.6.2 二级缓存 …… 224
13.7 Spring MVC 与 MyBatis 整合 …… 226
13.7.1 配置文件 …… 226
13.7.2 编写业务 …… 228
13.7.3 JSON 数据格式 …… 230

第 14 章

简单用户信息管理系统案例 …… 233

14.1 SSM 框架概述 …… 234
14.2 需求与开发步骤 …… 234
14.3 环境搭建 …… 235
14.4 Model 层实现与测试 …… 238
14.4.1 数据表建立 …… 238
14.4.2 实体类编写 …… 238
14.4.3 Mapper 层 DAO 类编写 …… 240
14.4.4 Mapper 层 DAO 类测试 …… 241
14.4.5 业务层实现 …… 243
14.4.6 Spring 与 MyBatis 整合配置 …… 244
15.4.7 业务层测试 …… 245
14.5 Controller 层实现 …… 246
14.6 View 层实现 …… 249
14.7 Web 配置与运行测试 …… 253

参考文献 …… 255

第 1 章 Java Web 应用开发简介

学习目标

1. 了解 Web 的发展历程。
2. 了解 Web 开发相关技术。
3. 了解学习 Web 开发常用网上资源。
4. 掌握 Web 应用程序的工作原理。

知识导图

本章导读

随着互联网的迅猛发展，Web 技术的应用已经越来越广泛，如今越来越多的计算机语言也开始支持 Web 的开发。本章将介绍 Web 的发展历程和 Web 开发的相关技术。

1.1 Web 发展历程简介

WWW（World Wide Web）即全球广域网，也称为万维网、Web。它是一种基于超文本和 HTTP 的、全球性的、动态交互的、跨平台的分布式图形信息系统。是建立在 Internet 上的一种网络服务，为用户在 Internet 上查找和浏览信息提供了图形化的、易于访问的直观界面，其中的文档及超链接将 Internet 上的信息节点组织成一个互为关联的网状结构。

1.1.1 Web 的起源

1989 年，在 CERN（欧洲粒子物理研究所）中由 Tim Berners-Lee 领导的小组提交了一个针对 Internet 的新协议和一个使用该协议的文档系统，该小组将这个新系统命名为 World Wide Web，它的目的在于使全球的科学家能够利用 Internet 交流自己的工作文档，

该系统被设计为允许 Internet 上任意一个用户从大量文档服务计算机的数据库中搜索和获取文档。1990 年末，CERN 已经开发出该系统的基本框架，并于 1991 年将其移植到其他计算机平台，进行正式发布。

1.1.2 Web 的发展

1. Web 1.0 时代

1980 年，诞生了最早的网络构想。该构想来源于 Tim Berners-Lee 构建的 ENQUIRE 项目，这是一个超文本在线编辑数据库，尽管与现在使用的互联网不太一样，但是在许多核心思想上却是一致的。

1994 年，Web 1.0 时代开始。其主要特征是大量使用静态的 HTML 网页来发布信息，并开始使用浏览器来获取信息，这个时候主要是单向的信息传递。通过 Web，互联网上的资源可以在一个网页里比较直观地表示出来，而且资源之间可以在网页上任意链接。

Web 1.0 的本质是聚合、联合、搜索，其聚合的对象是巨量、无序的网络信息。Web 1.0 只解决了人对信息搜索、聚合的需求，而没有解决人与人之间沟通、互动和参与的需求。

2. Web 2.0 时代

Web 2.0 的概念始于 2004 年出版社经营者 Tim O'Reilly 和 Media Live International 之间的一场头脑风暴论坛。之后 Tim O'Reilly 在发表的“What Is Web2.0”一文中概括了 Web 2.0 的概念，并给出了描述 Web 2.0 的框图——Web 2.0 Meme Map，Web 2.0 至此诞生，该文之后成为 Web 2.0 研究的经典文章。此后关于 Web 2.0 的相关研究与应用迅速发展，Web 2.0 的理念与相关技术日益成熟和发展，推动了 Internet 的变革与应用的创新。

在 Web 2.0 中，软件被当成一种服务，Internet 从一系列网站演化成一个成熟的为最终用户提供网络应用的服务平台，强调用户的参与、在线的网络协作、数据储存的网络化、社会关系网络、RSS 应用以及文件的共享等成为 Web 2.0 发展的主要支撑和表现。

Web 2.0 模式大大激发了创造和创新的积极性，使 Internet 重新变得生机勃勃。Web 2.0 的典型应用包括 Blog、Wiki、RSS、Tag、SNS、P2P、IM 等。

3. Web 3.0 时代

Web 3.0 是 Internet 发展的必然趋势，是 Web 2.0 的进一步发展和延伸。

Web 3.0 在 Web 2.0 的基础上，将杂乱的微内容进行最小单位的继续拆分，同时进行词义标准化、结构化，实现微信息之间的互动和微内容间基于语义的链接。Web 3.0 能够进一步深度挖掘信息并使其直接从底层数据库进行互通。并把散布在 Internet 上的各种信息点以及用户的需求点聚合和对接起来，通过在网页上添加元数据，使机器能够理解网页内容，从而提供基于语义的检索与匹配，使用户的检索更加个性化、精准化和智能化。

Web 3.0 的定义是：网站内的信息可以直接和其他网站相关信息进行交互，能通过第三方信息平台同时对多家网站的信息进行整合使用；用户在 Internet 上拥有直接的数据，并能在不同网站上使用；完全基于 Web，用浏览器即可以实现复杂的系统程序才具有的功能。

Web 3.0 浏览器会把网络当成一个可以满足任何查询需求的大型信息库。Web 3.0 的本质是深度参与、生动体验以及体现网民参与的价值。

1.1.3 Web 的影响

万维网使得全世界的人们以史无前例的巨大规模进行相互交流。相距遥远的人们、不同年龄的人们可以通过网络来发展亲密的关系或者使彼此思想境界得到升华，甚至改变他们对待小事的态度以及精神。情感经历、政治观点、文化习惯、表达方式、商业建议、艺术、摄影、文学等信息都可以以人类历史上从来没有过的低投入实现数据共享。

尽管使用万维网仍然要依靠存在自身缺陷的物化的工具，但至少它的信息保存方式不是使用人们熟悉的方式如图书馆、出版物等。因此信息传播是经由万维网和互联网来实现，而无须被搬运具体的书卷，或者手工的或实物的复制而限制。而且数字储存方式的优点是，你可以比查阅图书馆或者纸质版书籍更有效率地查询到网络上的信息资源。

1.2 Web 应用程序的工作原理

1.2.1 程序开发体系结构

随着信息运用，更多的计算机技术的日新月异，单机的软件程序已经越来越难以满足用户的需求。因此，诞生了各种各样的程序开发体系结构。其中，应用最为广泛的程序开发体系结构大致分为两类：C/S 体系结构和 B/S 体系结构。

1. C/S 体系结构

C/S 体系结构（Client/Server 结构）：即客户端 / 服务器结构。在这种体系结构中，每个客户端都需要安装相应的工具软件，服务器通常采用高性能的 PC 机或工作站，同时采用大型数据库系统（如 Oracle、SQL Server 等），如图 1-1 所示。

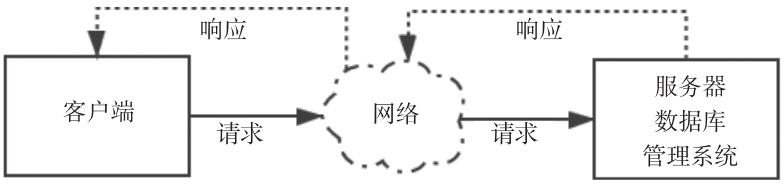


图 1-1　C/S 结构

C/S 体系结构的优点是能充分发挥客户端计算机的处理能力，很多工作可以在客户端处理后再提交给服务器。C/S 体系结构的缺点是开发比较麻烦，在对软件进行管理和维护时，客户端和服务器端需要同时更改。

2. B/S 体系结构

B/S 体系结构（Browser/Server 结构）：即浏览器 / 服务器结构。在这种体系结构中，客户端不需要安装任何软件，而是在服务器端安装对应的软件，客户端通过浏览器访问服务器，从而实现信息、资源的交互和共享，如图 1-2 所示。

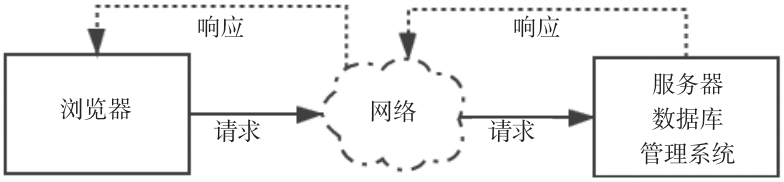


图 1-2　B/S 结构

B/S 体系结构的优点是成本低、维护方便、分布性强、开发简单，可以不用安装任何专门的软件就能实现在任何地方进行操作，客户端零维护，系统的扩展非常容易。B/S 体系结构的缺点是通信开销大、系统和数据的安全性较难保障。

C/S 结构与 B/S 结构两种模式各自拥有其特色优势，在不同的系统环境与操作平台下，选择较为接近或交叉进行混合模式的使用，可以保证数据的敏感性、安全性和稳定发展，还可以加强对数据库的修改与新增记录的操作。对客户端程序进行保护，提高资源数据的交互性能，实现系统维护成本较低、维护方式较简便、布局更合理、网络数据使用效率较高的目的，所以采用 C/S 结构与 B/S 结构混合模式才是最佳方案。

1.2.2 静态网站与动态网站

Web 应用程序大致分为两类，即静态网站和动态网站。

早期的 Web 应用主要是以静态页面的方式进行浏览，即静态网站。这些网站通过 HTML 语言进行编写，并部署至 Web 服务器，用户通过使用浏览器发送 HTTP 协议请求服务器上的 Web 页面，Web 服务器接收到用户请求处理后，将页面发送并显示给用户。如图 1-3 所示。

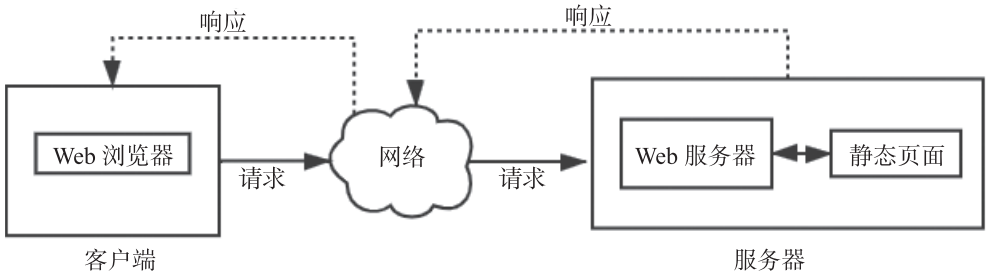


图 1-3　静态网站处理流程

然而，随着网络的不断发展，很多线下的业务开始往线上发展，基于互联网的 Web 应用也日渐复杂，用户所访问的资源已不再局限于服务器上保存的静态网页，更多的内容需要根据用户的请求动态生成页面信息，即动态网站。动态网站通常使用 HTML 语言和动态脚本语言（如 JSP、ASP、PHP 等）编写，并将编写后的程序部署到 Web 服务器上，由 Web 服务器对动态脚本代码进行处理，并转化为浏览器可以解析的 HTML 代码，返回客户端浏览器，显示给用户。如图 1-4 所示。

带有动画效果的网页不一定是动态网页，动态网页是指具有交互性、内容可以自动更新，同时内容会根据访问的时间和访问者而改变。这里的交互性是指服务器会自动根据用户请求的不同而显示不同的结果。

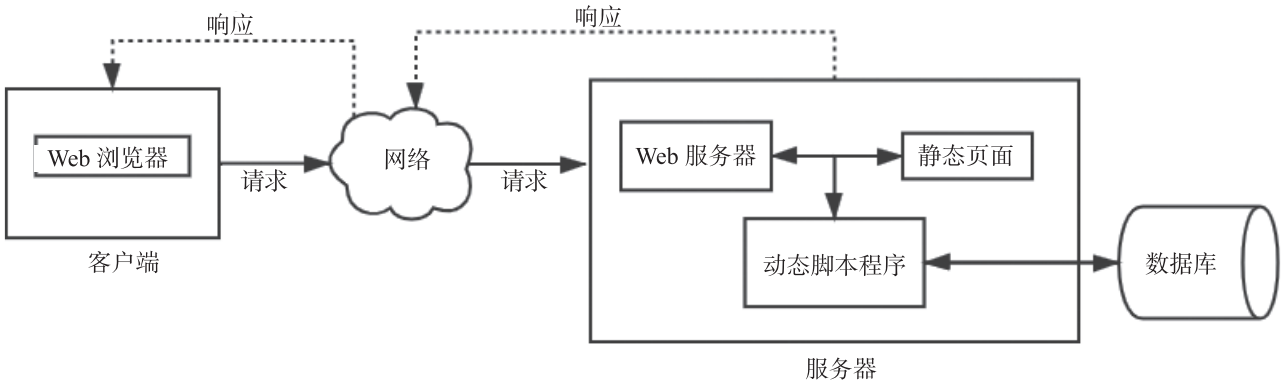


图 1-4　动态网站处理流程

1.3 Web 开发技术

在进行 Web 应用程序开发时，通常需要应用到客户端和服务器端两方面的技术。其中，客户端应用的技术主要用于展示信息内容，服务器端应用的技术主要用于进行业务逻辑的处理与数据库的交互等。

1.3.1 客户端技术

在日常 Web 项目的开发中，离不开客户端技术的支持。目前，常用的客户端技术包括 HTML 语言、CSS 和客户端脚本技术。

1. HTML 语言

HTML 语言即超文本标记语言，是一种标记语言。它是客户端技术的基础，主要用于显示网页信息。它包括一系列标签．通过这些标签可以将网络上的文档格式统一，使分散的 Internet 资源连接为一个逻辑整体。HTML 文本是由 HTML 命令组成的描述性文本，HTML 命令可以说明文字、图形、动画、声音、表格、链接等。

2. CSS 样式

CSS（Cascading Style Sheets）即层叠样式表，是一种用来表现 HTML 或 XML（标准通用标记语言的一个子集）等文件样式的计算机语言。CSS 不仅可以静态地修饰网页，还可以配合各种脚本语言动态地对网页各元素进行格式化。在目前比较流行的 CSS+DIV 布局的网站中，CSS 更是极大地提高了开发者对信息展现格式的控制能力。

3. 客户端脚本技术

客户端脚本技术指的是嵌入 Web 页面中的程序代码，这些程序代码是一种解释性的语言，浏览器对客户端脚本进行解释。通过脚本语言可以实现以编程的方式对页面元素进行控制，从而增加页面的灵活性。常用的客户端脚本语言有 JavaScript 和 VBScript。

1.3.2 服务器端技术

在进行动态 Web 项目开发时，也离不开服务器端技术的支持。目前，比较常用的服务器端技术主要有 5 种：CGI、ASP、PHP、ASP.NET 和 JSP。

1. CGI（Common Gateway Interface，公共网关接口）

CGI 是最早出现的实现动态 Web 的操作标准，可以采用任何语言实现（如 C 或 VB），但是这种传统的 CGI 程序本身是采用多进程的机制进行处理的。每当一个新用户连接到服务器上时，服务器都会为其分配一个新的进程，很明显，这种程序的执行效率是很低的。

2. ASP（Active Server Pages，动态服务页）

ASP 是一个动态 Web 服务器端的开发环境，利用它可以产生和运行动态的、交互的、高性能的 Web 服务应用程序。由于 ASP 技术出现较早，所以一直到今天还在被使用着，但是 ASP 技术本身有一个最大的问题就是平台的支持，ASP 只能运行在 IIS（Internet Information Services，互联网信息服务）服务器上，且只能在 SQL Server 数据库上才可以得到最大限度发挥。但是这套开发技术相对于使用 Java 开发而言，性能是很差的，所以一般用于个人或中小型项目开发。

3. PHP（Hypertext Preprocessor，超文本预处理）

PHP 是一种跨平台的服务器端的嵌入式脚本语言。它大量地借用 C、Java 和 Perl 语言的语法，并结合 PHP 自身的特性，使 Web 开发者能够迅速地写出动态页面。而且 PHP 是完全免费的，用户可以从 PHP 官方站点自由下载。但是 PHP 本身也有缺点，就是需要运行在 Apache 服务器下，只有在使用 MySQL 数据库时才可以达到性能的最大限度发挥，所以一般都只适合于个人或小型项目开发。

4. ASP.NET

ASP.NET 是微软公司继 ASP 之后推出的新一代动态网站开发技术。ASP.NET 基于 .NET 框架平台，用户可以选择 .NET 框架下自己喜欢的语言进行开发。ASP.NET 技术是 ASP 技术的更新，也是微软公司目前主推的技术，但是由于微软的产品会受到平台的限制，所以此技术往往用于中型项目的开发。

5. JSP（Java Server Page，Java 服务页）

使用的 Java 完成的动态 Web 开发，代码风格与 ASP 类似，都属于在 HTML 代码中嵌入其他代码以实现功能。JSP 嵌入代码为 Java，由于 Java 语言的跨平台特性，因此 JSP 不会受到操作系统或开发平台的制约，而且有多种服务器可以支持，如 Tomcat、WebLogic、JBoss、Websphere 等，所以经常在中大型项目开发中使用。JSP 的前身是 Servlet（服务器端小程序），但是由于 Servlet 开发过于复杂，所以 Sun 公司的开发人员根据 ASP 技术的特点，将 Servlet 程序重新包装，而形成新的一门开发技术——JSP。

1.4 常用上网资源

在 Java Web 应用程序的日常开发中，通常需要用到很多的资源，也经常碰到许多一时无法处理的问题，以下给出一些比较常用的资源下载网址和技术社区网址：

（1）JDK 官方网站 https://www.oracle.com/java/technologies。

（2）Tomcat 官方网站 https://tomcat.apache.org。

（3）MySQL 数据库官方网站 https://www.mysql.com。

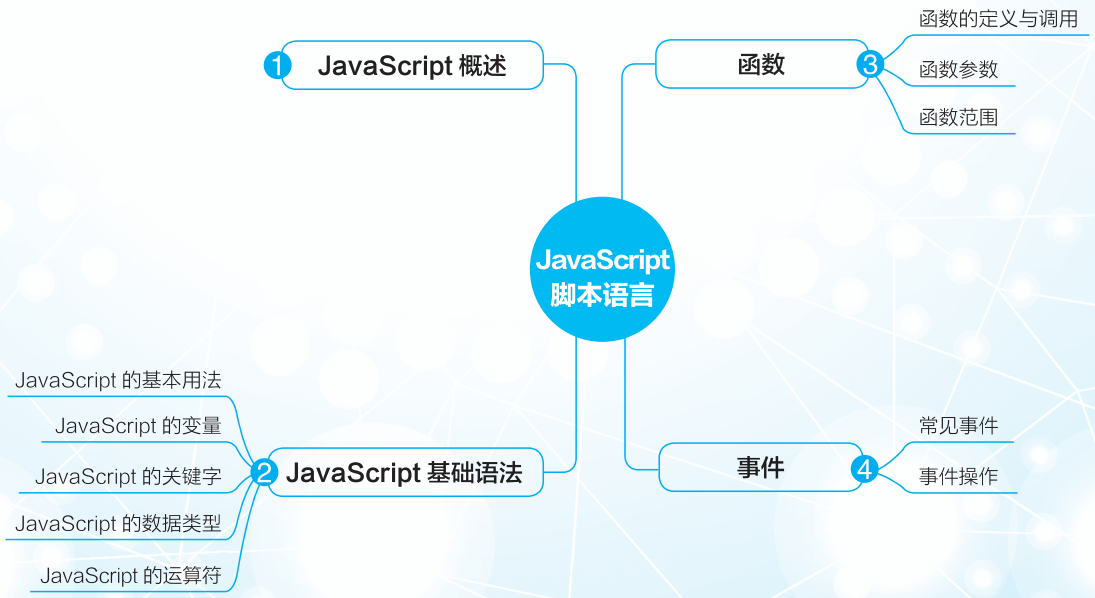
（4）CSDN 社区中心 https://www.csdn.net。

第 2 章 JavaScript 脚本语言

学习目标

1. 了解 JavaScript 的特点。
2. 掌握 JavaScript 的基本语法。
3. 掌握 JavaScript 的常用函数。

知识导图



本章导读

JavaScript 是目前互联网上最流行的脚本语言，这门语言可用于 HTML 和 Web，更可广泛用于服务器、计算机、笔记本电脑、平板电脑和智能手机等设备。

JavaScript 是 Web 页面中一种比较流行的脚本语言，它由客户端浏览器解释执行，可以应用在 JSP、PHP、ASP 等网站中。同时，随着 Ajax 进入 Web 开发的主流市场，JavaScript 已经被推到了“舞台”的中心，因此熟练掌握并应用 JavaScript 对于网站开发人员来说尤为重要。

本章将简单介绍 JavaScript 的基本用法，更深入的知识由于篇幅原因本章不过多介绍，读者可参考相关资料自行学习。

2.1 JavaScript 概述

JavaScript 是一种基于对象和事件驱动并具有安全性能的脚本语言。JavaScript 在 1995 年由 Netscape 公司的 Brendan Eich 在网景导航者浏览器上首次设计实现而成。因为 Netscape 与 Sun 公司合作，Netscape 管理层希望它外观看起来像 Java，因此取名为 JavaScript。

JavaScript 适用于静态或动态网页，是一种被广泛使用的客户端脚本语言，它具有以下的特点：

（1）解释性。JavaScript 是一种解释型的脚本语言，C、C++ 等语言先编译后执行，而 JavaScript 是在程序的运行过程中逐行进行解释。

（2）轻量性。JavaScript 语言中采用的是弱类型的变量类型，对使用的数据类型未做出严格的要求，是基于 Java 基本语句和控制的脚本语言，其设计简单紧凑。

（3）基于对象。JavaScript 是一种基于对象的脚本语言，它不仅可以创建对象，也能使用现有的对象。

（4）动态性。JavaScript 是一种采用事件驱动的脚本语言，它不需要经过 Web 服务器就可以对用户的输入做出响应。在访问一个网页时，鼠标在网页中进行单击或上下移动、窗口移动等操作，JavaScript 都可直接对这些事件给出相应的响应。

（5）跨平台性。JavaScript 脚本语言与操作系统无关，仅需要浏览器的支持。因此，一个 JavaScript 脚本在编写后可以带到任意机器上使用，前提是机器上的浏览器支持 JavaScript 脚本语言，目前 JavaScript 已被大多数的浏览器所支持。

2.2 JavaScript 基础语法

JavaScript 的语法与 Java 类似，但相对于 Java 来说语法要简单得多，就是包含了一些变量及函数的声明操作。

与 Java 语言相同的是，JavaScript 是区分大小写的。与 Java 语言不同的是，首先，JavaScript 不要求必须以“;”（英文分号）作为语句结束标记。如果语句末尾没有分号，则 JavaScript 会自动将该行代码的结尾作为语句的结尾。其次，所有 JavaScript 代码是在 HTML 页面中编写的，使用 <script> 标记完成。

2.2.1 JavaScript 的基本用法

如果需要在 HTML 页面中插入 JavaScript，则需要使用 <script> 标签。<script> 和 </script> 会告诉 JavaScript 在何处开始和结束。一般而言，<script> 标记都是出现在 <head> 标记中的，当然，也可以在任意位置上编写，但是最好在调用其操作之前进行编写。

【例 2.1】第一个 JavaScript 程序。代码如下：

```
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <script>
      // 弹出会话框
      alert("Hello World!");
    </script>
  </head>
  <body>
  </body>
</html>
```

运行该程序后浏览器会弹出一个会话框，程序运行结果如图 2-1 所示。

图 2-1　第一个 JavaScript 程序

在一个 HTML 文件中，也可以同时定义多个 <script> 标签，执行时会采用顺序执行的方式进行。

【例 2.2】定义多个 <script> 标签。代码如下：

```
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <script>
      alert("Hello World!");   // 第一个会话框
    </script>
  </head>
  <body>
    <script>
      alert("Hello JavaScript!");    // 第二个会话框
    </script>
  </body>
</html>
```

运行该程序后浏览器会弹出两个会话框，程序运行结果如图 2-2 所示。

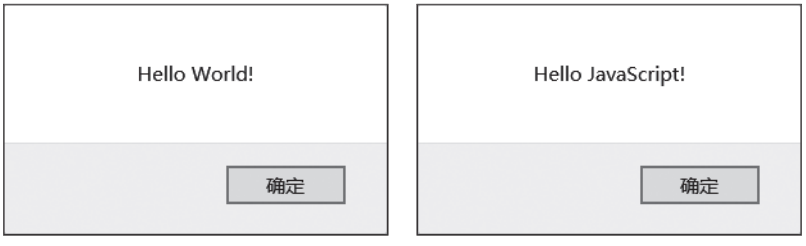


图 2-2　定义多个 <script> 标签

在 HTML 中编写 JavaScript 时，还可以使用 document.write( ) 向 HTML 页面输出内容。

【例 2.3】document.write( ) 语句使用示例。代码如下：

```
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <script>
      document.write("Hello World!");  // 页面输出
    </script>
  </body>
</html>
```

运行结果如图 2-3 所示。

图 2-3　使用 document.write( ) 语句

在编写 JavaScript 时，也可以把脚本保存到外部文件中。外部 JavaScript 文件的文件扩展名是 .js，通常包含被多个网页使用的代码。如需使用外部文件，则在 <script> 标签的 "src" 属性中设置该 .js 文件。

【例 2.4】定义外部 JavaScript 文件，代码如下：

```
alert(" 运行外部 .js 文件 ");
```

本外部 .js 文件保存在 js 目录中。

在 HTML 文件中引用外部 .js 文件，代码如下：

```
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <script src="./js/ 例 2.4.js"></script>
  </head>
  <body>
  </body>
</html>
```

本程序在 js 目录中创建了一个例 2.4.js 文件，然后在 HTML 页面中引用。运行结果如图 2-4 所示。

图 2-4　定义外部 JavaScript 文件

在 JavaScript 中，有两种注释，分别是单行注释和多行注释。

单行注释以“//”双斜线开头，注释内容写在其后，注释内容在程序运行时不起任何作用。

多行注释则以“/*”和“*/”作为开头和结尾，注释内容则位于“/*”和“*/”之间，同样在程序执行时不进行编译。注释使用如例 2.5 所示。

【例 2.5】JavaScript 中的单行和多行注释使用示例。代码如下：

```
/*
定义函数：
 功能：弹出会话框
 */
function getMsg( ){
   // 弹出框提示“error”
      alert("error");
}
```

2.2.2 JavaScript 的变量

在 JavaScript 中也可以定义变量，且定义变量的语法相比其他语言更加简单，只需使用关键字 var 声明变量即可。定义变量时需要注意以下几点：

（1）变量命名必须以字母、下划线“_”或者“$”为开头。其他字符可以是字母、_、美元符号或数字。

（2）变量名中不允许使用空格和其他标点符号，首个字不能为数字。

（3）变量名长度不能超过 255 个字符。

（4）变量名区分大小写。

（5）变量名必须放在同一行中，且最好能见名知意。

（6）不能使用脚本语言中保留的关键字、保留字、true、false 和 null 作为标识符。

定义变量的格式如下：

```
var 变量名 ;
var 变量名 = 初始值 ;
var 变量名 1, 变量名 2,
```

【例 2.6】在 JavaScript 中定义变量。代码如下：

```
var num=123;        // 将变量 num 初始化为 123
var str="Hello";    // 将变量 str 初始化为 Hello
var num1=22,str1="haha"; // 定义多个变量并进行赋值
```

在 JavaScript 中，根据变量的作用域不同，JavaScript 中的变量可以分为全局变量和局部变量两种。全局变量是作用于整个脚本代码的变量，定义在所有函数之外；局部变量是只作用于函数体内的变量，定义在函数体内。

【例 2.7】定义局部变量和全局变量。代码如下：

```
var num=1;          // 定义全局变量 num 并初始化为 1
function msg( ){
    var num1=2;     // 定义局部变量 num1 并初始化为 2
    alert(num1);    // 将局部变量 num1 以会话框方式输出
}
```

2.2.3 JavaScript 的关键字

JavaScript 中的关键字可用于表示控制语句的开始或结束，或者用于执行特定操作等。

关键字是 JavaScript 语言保留的，不能用作标识符。JavaScript 中的部分关键字，如表 2-1 所示。

表 2-1　JavaScript 中的部分关键字

abstract	const	export	instranceof	new	short	typeof
boolean	class	finally	int	null	static	throws
byte	default	for	interface	package	super	var
break	do	function	implements	private	synchronized	void
catch	debugger	final	import	protected	true	while
case	double	float	if	public	this	with
continue	else	flase	long	return	throw	
char	enum	in	native	switch	try	

需要注意的是，关键字不能定义为 JavaScript 中的变量或函数名。

2.2.4 JavaScript 的数据类型

JavaScript 的数据类型主要有引用类型和基本类型。引用类型主要包括对象（Object）、数组（Array）和函数（Function），基本类型主要包括字符型、数值型、布尔型、空值、转义字符和未定义值 6 种。

1. 字符型

字符型数据是用于存储单引号或多引号括起来的一个或多个字符的变量。

【例 2.8】定义字符型变量，代码如下：

```
var str1='a';       // 使用单引号括起来的单个字符
var str2='aa';      // 使用单引号括起来的多个字符

var str3="b";       // 使用双引号括起来的单个字符
var str4="bb";      // 使用双引号括起来的多个字符
```

2. 数值型

JavaScript 中的数值型数据分为两种，分别为整型和浮点型。整型数据可以是正整数、负整数和 0，且可以用十进制、八进制和十六进制进行表示。浮点型数据由整数部分和小数部分组成，只能采用十进制，可以使用科学计数法或标准法表示。

【例 2.9】定义数值型数据，代码如下：

```
var num=123      // 定义整型数据
var fnum=12.34   // 定义浮点型数据
var bnum=1.2E3   // 采用科学计数法表示浮点数，代表 1200
```

3. 布尔型

布尔型数据只有 true 和 false 两个值，在编写时主要是用于说明或代表一种状态或标志。在 JavaScript 中，也可以使用整数 0 表示 false，非 0 整数来表示 true。

4. 转义字符

转义字符是以反斜杠“\”开头的不可显示的特殊字符，也称为转义字符。转义字符常用于在字符串中添加不可显示的特殊字符，用以防止引号匹配混乱问题。在 JavaScript 中常用的转义字符如表 2-2 所示。

表 2-2　JavaScript 中常用的转义字符

转义字符	含义	转义字符	含义
\b	退格	\"	双引号
\f	换页	\\	反斜杠
\n	换行	\t	Tab 符
\r	回车	\'	单引号

5. 空值

在 JavaScript 中，空值数据（null）主要用于定义空的或不存在的引用。如果引用没有定义的变量，则会返回一个 null 值。另外，空值不等于空的字符串（""）或 0。

6. 未定义值

未定义值（undefined）是当使用一个未声明的变量或者已经声明但没有赋值的变量时返回的数据。

2.2.5 JavaScript 的运算符

在 JavaScript 中常用的运算符按类型分主要有赋值运算符、算术运算符、逻辑运算符、比较运算符、条件运算符和字符串运算符。

1. 算术运算符

算术运算主要用于在程序中的加、减、乘、除等运算。常用的算术运算符如表 2-3 所示。

表 2-3　JavaScript 中常用的算术运算符

运算符	描述	例子	结果
+	加法	6+6	12
-	减法	3-2	1
*	乘法	5*6	30
/	除法	6/2	3
%	取模（除余）	6%4	2
++	自增	x=2, y=++x	x=3,y=3
		x=2, y=x++	x=3,y=2
--	自减	x=2, y=--x	x=1,y=1
		x=2,y=x--	x=1,y=2

2. 赋值运算符

在 JavaScript 中，赋值运算又可以分为简单赋值运算和复合赋值运算。简单赋值运算是将赋值运算符（=）右边表达式的值保存到左边的变量值中；复合赋值运算则是混合了其他运算（位运算、算术运算）以及赋值操作。常见的赋值运算符如表 2-4 所示。

表 2-4　JavaScript 中常见的赋值运算符

运算符	例子	运算符	例子
=	x=y	*=	x*=y // 即 x=x*y
+=	x+=y // 即 x=x+y	/=	x/=y // 即 x=x/y
-=	x-=y // 即 x=x-y	%=	x%=y // 即 x=x%y

3. 比较运算符

在 JavaScript 中，比较运算符常用于在逻辑语句中判断变量或值是否相等。JavaScript 中的比较运算符如表 2-5 所示。

表 2-5　JavaScript 中的比较运算符

运算符	描述	例子	结果
==	等于	x=3,x==3	true
		x=3,x==4	false
===	绝对等于（值和类型相等）	x=3,x===“3”	false
		x=3,x===3	true
!=	不等于	x=3,x!=2	true
!==	不绝对等于（值和类型有一个不相等，或两个都不相等）	x=3,x!==“3”	true
		x=3,x!==3	false
>	大于	x=3,x>5	true
<	小于	x=3,x<2	false
>=	大于等于	x=3,x>=2	true
<=	小于等于	x=3,x<=4	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，常和比较运算符一起使用，用来表示复杂的比较运算。JavaScript 中的逻辑运算符如表 2-6 所示。

表 2-6　JavaScript 中的逻辑运算符

运算符	描述	例子	结果
&&	逻辑与。当条件都为真时为真。	x=2,y=3 (x<3 && y>2)	true
||	逻辑或。当条件都为假时为假。	x=2,y=3 (x<3 && y>4)	true
!	逻辑非。否定条件	!0	true

5. 条件运算符

条件运算符是 JavaScript 中支持的一种特殊的三目运算符，其使用格式为：

```
操作数 ? 值 1: 值 2
```

其判定方式为，如果“操作数”为真，则表达式结果为“值 1”，反之，则表达式结果为“值 2”。

【例 2.10】JavaScript 中的条件运算符使用，代码如下：

```
var a=2,b=3;
var c=a<b ? a:b
```

本程序的运行结果为：c=2。

6. 字符串运算符

字符串运算符是用于两个字符型数据之间的运算符，除了比较运算符外，也可以是“+”和“+=”运算符。“+”运算符用于连接两个字符串，“+=”运算符则用于连接两个字符串，并将结果赋给第一个字符串。

【例 2.11】JavaScript 的字符串运算符使用示例。代码如下：

```
<html>
    <head>
            <meta charset="utf-8">
            <script >
                    var str1="An apple",str2="a day ";
                    var str3=str1+" "+str2;    // 在 str1 和 str2 之间添加空格
                    str3+=" doctor away";
                    alert(str3);
            </script>
            <title></title>
    </head>
    <body>
    </body>
</html>
```

本程序的功能是将两个字符串使用“+”运算符拼接在一起形成一个新的字符串。运行结果如图 2-5 所示。

图 2-5　JavaScript 的字符串运算符

说明：

在 JavaScript 中流程控制结构可分为三类，分别为顺序结构、分支结构、循环结构。用法和 C 语言、Java 语言等类似，由于篇幅原因，此处不再详述。

2.3 函数

函数是 JavaScript 中另一个基本概念，它允许你在一个代码块中存储一段用于处理单任务的代码，然后在任何你需要的时候调用，这样实现了代码的复用。本节主要探索函数的基本概念，如函数的定义和调用、函数

的参数和范围。

2.3.1 函数的定义与调用

在 JavaScript 中函数定义有多种方式，如函数声明、函数表达式、函数构造器。

1. 函数声明

JavaScript 函数通过 function 关键词进行定义，其后是函数名，定义函数参数的括号与定义函数内容的大括号。函数名可包含字母、数字、下画线和美元符号。括号可包括由逗号分隔的自定义数量的参数，这些参数不用指定对应的类型。需要注意的是，函数声明后不会立即执行，而是在我们需要的时候进行调用。函数的语法格式如下：

```
function 函数名 ( 参数 1, 参数 2, ...)
{
   函数体 ;
}
```

【例 2.12】第一个函数。代码如下：

```
function myFuntion1( )
{
   alert(' 我的第一个函数 !');
}
```

这里定义了一个函数，用来弹出一个警告框。运行结果如图 2-6 所示。

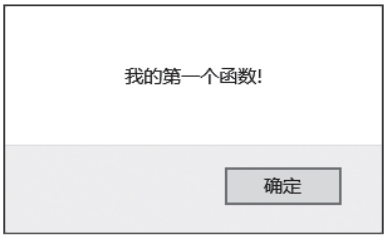


图 2-6　调用函数—警告框

另外，还可以在函数中通过 return 关键字返回值。这里不需要指定返回值类型，直接将值返回即可。如果函数中没有 return 返回值，则默认返回 undefined。其使用格式如下：

```
function 函数名 ( 参数 1, 参数 2, ...)
{
    函数体 ;
    return 返回值 ;
}
```

【例 2.13】带有返回值的函数使用示例。代码如下：

```
function myFunction2(a,b)
{
    return a+b;
}
```

这里定义了一个作加法函数，它将参数 a 与 b 的和返回。

2. 函数表达式

JavaScript 函数可以通过一个表达式定义。函数表达式可以存储在变量中。语法格式如下：

```
var myFunction3 = function (a, b) {
    return a * b
}
```

这个函数没有函数名，叫作匿名函数，它储存在变量 x 中，定义了 a,b 两个参数，并将 a*b 的结果返回。

3. 函数构造器

函数同样可以通过函数构造器定义，需要在 Function 构造器的参数中先后指定函数参数与函数体，并且函数体是最后一个参数。

【例 2.14】使用函数构造器定义函数，代码如下：

```
var myFunction4 = new Function("a", "b", "return a / b");
```

4. 调用函数

在调用函数时，使用函数名并紧跟一个括号，并在括号内顺序传入参数。如果是匿名函数，则使用函数变量调用函数。

【例 2.15】调用函数示例，代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <title> 调用函数 </title>
    <meta charset="utf-8">
    <script>
      function myFuntion1( )
      {
        alert(' 我的第一个函数 !');
      };

      function myFunction2(a, b)
      {
        return a + b;
      };
      var myFunction3 = function (a, b) {
        return a * b
      };
      var myFunction4 = new Function("a", "b", "return a / b");
    </script>
  </head>
  <body>
    <script>
      console.log(myFuntion1( ),myFunction2(1,2),myFunction3(1,2),myFunction4(4,2))
    </script>
  </body>
</html>
```

本程序的运行结果如图 2-7 所示。

图 2-7　调用函数—控制台日志

2.3.2 函数参数

一些函数需要在调用它们时指定参数，在定义函数时指定的参数叫作显式参数，在调用函数时传递的参数叫作隐式参数。它们需要放在函数括号内，才能正确地完成其工作，当然也可以不指定参数。

1. 参数规则

JavaScript 是一种弱类型的语言，不仅可以在定义参数时不用指定参数的类型，而且在调用函数时，也可传入任意个参数，如表 2-7 所示。

表 2-7　JavaScript 参数个数

当显式参数少于隐式参数时	多余的隐式参数为 undefined
当显式参数等于隐式参数时	显式参数与隐式参数一一对应
当显式参数多于隐式参数时	缺失的显式参数为 undefined

2. 自带参数的函数

从 ES6 开始，JavaScript 开始支持函数带有默认值。可以同时指定参数带有默认值或没有带默认值。

【例 2.16】自带参数的函数使用，代码如下：

```
function myFunction(a, b = 1) {
    return a + b;
}
// 当参数 b 指定了值时，参数 b 的隐式参数被指定为该值
console.log(myFunction(1, 2));
// 当参数 b 未指定值时，参数 b 的隐式参数被指定为默认值
console.log(myFunction(1));
```

运行本程序，结果如图 2-8 所示。

图 2-8　自带参数的函数

3. arguments 对象

为了方便地得到参数的值，JavaScript 函数提供了一个内置的 arguments 对象。arguments 对象包含了函数调用的参数数组，这样可以从数组中遍历参数值。

【例 2.17】arguments 对象的调用使用，代码如下：

```
function myFunction ( ) {
    for (i = 0; i < arguments.length; i++) {
      console.log(arguments[i]);
    }
}
myFunction(1, 2, 3, 4, 5); // 这个函数将所有的参数在控制台打印出来
```

运行本程序，在控制台日志的结果如图 2-9 所示。

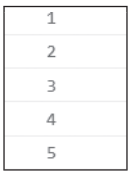


图 2-9　arguments 对象的调用

2.3.3 函数范围

在 JavaScript 中，函数中的变量是有作用域的。

1. 参数的作用域

通过值传递的参数只是函数内的局部变量，函数只能获取参数值。在内部修改参数的值时，外部传入的参数变量的值不会修改。

JavaScript 也支持通过对象传递参数，此时在函数内部修改对象的属性就会修改其初始的值，这些对象的属性则是全局变量。

2. 变量的作用域

在 JavaScript 中，用 var 申明的变量实际上是有作用域的。如果一个变量在函数体内部申明，则该变量的作用域为整个函数体，在函数体外不可引用该变量。如果两个不同的函数各自申明了同一个变量，那么该变量只在各自的函数体内起作用。换句话说，不同函数内部的同名变量互相独立，互不影响。

由于 JavaScript 的函数可以嵌套，此时，内部函数可以访问外部函数定义的变量，反过来则不行。JavaScript 的函数在查找变量时从自身函数定义开始，从“内”向“外”查找。如果内部函数定义了与外部函数重名的变量，则内部函数的变量将“屏蔽”外部函数的变量。

3. 变量提升

JavaScript 的函数定义有个特点，它会先扫描整个函数体的语句，把所有申明的变量“提升”到函数顶部。通常是先定义函数再调用函数，但因为有变量提升，可以先调用函数再定义函数。

【例 2.18】变量提升示例，代码如下：

```
myFunction( );          // 先调用函数
function myFunction ( ) {  // 定义函数，先使用变量
    alert(' 变量提升示例 ');
}
```

本程序在浏览器中运行结果如图 2-10 所示。

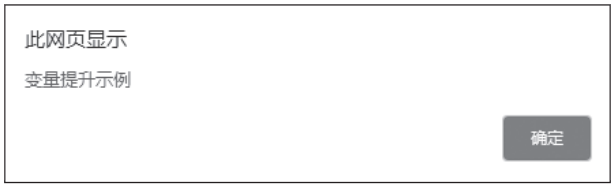


图 2-10　变量提升示例

2.4 事件

事件是发生在 HTML 元素上的所有事情。JavaScript 则可以在触发这些事件时，实现相关的功能。在 JavaScript 的事件处理中主要是围绕函数展开的，一旦发生事件后，则会根据事件的类型来调用相应的函数，以完成事件的处理。

2.4.1 常见事件

下面列举一些常见的事件。

1. HTML 事件

HTML 事件主要用于处理和响应 HTML 中的事件。HTML 事件如表 2-8 所示。

表 2-8　HTML 事件及说明

事件	说明
onchange	HTML 元素改变
onclick	用户单击 HTML 元素
onmouseover	用户在一个 HTML 元素上移动鼠标
onmouseout	用户从一个 HTML 元素上移开鼠标
onkeydown	用户按下键盘按键
onload	浏览器已完成页面的加载

2. 触发事件

在下面的示例中，演示了按钮被单击的事件触发时的动作。

【例 2.19】JavaScript 触发事件使用，代码如下：

```
<button onclick='document.getElementById("currentTime").innerHTML=Date( )'>
    现在的时间
</button>
```

当单击按钮时，JavaScript 会找到 id 为 currentTime 的 HTML 元素，并把它的值更改为当前的时间。

2.4.2 事件操作

1. 事件冒泡

当一个 HTML 元素产生事件时，该事件会从当前元素（事件源）开始，往上冒泡直到页面的根元素，所以经过的节点都会收到该事件并执行。特点是先触发子级元素的事件，再触发父级元素的事件。可以通过 event.stopPropagation( ) 或 event.cancelBubble=true; 方法来阻止事件冒泡。

2. 事件默认行为

当一个事件发生时浏览器自己默认做的事情，如单击链接时会默认跳转，右击时默认会弹出菜单。可以通过 event.preventDefault( ); 方法来阻止事件的默认行为。