第 5 章　JavaScript 技术实践

5.1　知 识 简 介

5.1.1　JavaScript 概述


视频讲解

JavaScript(简称 JS)是一种高级编程语言,用于 Web 开发和其他应用程序中的客户端和服务器端编程。JavaScript 最初由 Netscape 公司的 Brendan Eich 在 1995 年创建,它是一种面向对象的脚本语言,不需要编译就可以在 Web 浏览器中运行,并且可以与 HTML 和 CSS 一起使用来实现具有交互性的 Web 页面。JavaScript 由三部分组成,下面对这三部分进行简单介绍。

(1) ECMAScript:一种标准化的脚本语言规范,最初由欧洲计算机制造商协会(ECMA)制定。其定义了 JavaScript 的核心语法、数据类型、控制结构、函数、对象等基本特性,以及一些新的语言特性,例如,模块化、箭头函数、类、异步函数等。如果读者想要查看 ECMAScript 的最新规范,可参考 ECMAScript 官方网站。

(2) DOM(Document Object Model,文档对象模型):定义如何访问和操作 HTML 文档中的元素,使得 JavaScript 可以与 Web 页面进行交互。

(3) BOM(Browser Object Model,浏览器对象模型):定义与浏览器交互的对象和方法,用户可以通过 BOM 对浏览器窗口进行操作(例如,刷新窗口、弹出对话框等)。

要使用 JavaScript,需要将其引入 HTML 文件中。JavaScript 的引入方式主要有三种,分别是内部引入、外部引入和行内引入。

5.1.2　HTML DOM 操作

在 Web 开发中,文档对象模型(Document Object Model,DOM)扮演着举足轻重的角色。它是网页的结构化表示形式,允许程序和脚本动态地访问、修改网页内容、结构和样式。通过 JavaScript 操作 DOM,可以赋予网页交互性与生命力。

DOM 将 HTML 文档视作一个节点树结构,其中每个元素、属性、文本都是树中的一个节点,如图 5-1 所示。根节点通常是<html>元素,从它开始,整个文档被组织成一系列嵌套的节点。理解这一结构是进行 DOM 操作的前提。

要对页面上的元素进行操作,先得获取它们。JavaScript 提供了多种方法来选取 DOM 元素,包括但不限于以下几种。

(1) 通过 ID 获取:使用 document.getElementById('elementId'),精确获取带有指定 ID 的元素。

(2) 通过类名获取:使用 document.getElementsByClassName('className'),可以获取具有特定类名的所有元素集合。

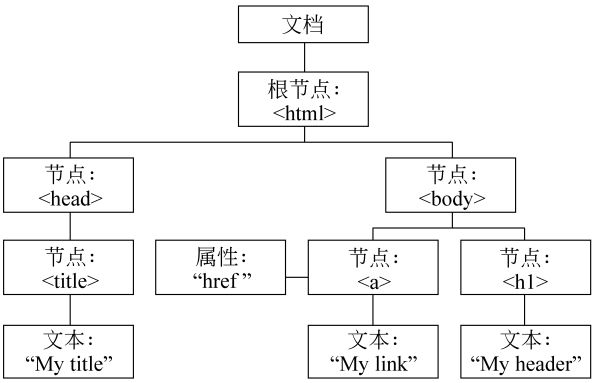

图 5-1　HTML DOM 树的结构

(3) 通过标签名获取：使用 document.getElementsByTagName('tagName')，返回所有指定标签名的元素集合。

(4) 使用 querySelector 系列：更加灵活的选择器，例如，使用 document.querySelector('.className') 可以返回匹配指定 CSS 选择器的第一个元素，而使用 document.querySelectorAll()可以返回所有匹配的元素集合。

获取元素后，即可对其进行修改，从而实现动态效果。常见的修改方式包括以下几种。

(1) 修改内容：通过.textContent 或.innerHTML 属性，可以改变元素的文本内容或 HTML 结构。例如，使用"element.textContent = '新内容'"仅可以替换文本，而使用"element.innerHTML = '新内容'"则可以插入 HTML 标签。

(2) 修改属性：使用.setAttribute('attributeName','value')可以设置元素的属性值，例如，改变图片的 src 属性。使用.removeAttribute('attributeName')则可以移除属性。

(3) 添加/删除元素：可以使用 document.createElement('tagName')创建新元素，然后使用 appendChild()或 prependChild()将其添加到父元素内。移除元素则可以通过 removeChild()实现。

DOM 操作往往与事件处理紧密相关。通过为元素绑定事件监听器，可以响应用户的动作，例如，点击、鼠标指针悬停等。

5.2　实践目的

通过该实践，加深读者对 DOM 操作及事件处理机制的掌握，培养 JavaScript 编程技能和逻辑思维能力，提高网页的动态交互体验。

5.3　实践范例

在现代高等教育体系中，毕业生的就业状况是衡量教育质量与市场需求匹配度的重要指标之一。为了高效地收集、管理和分析毕业生的就业数据，提高就业指导工作的针对

性和有效性，本次实践旨在开发一个简单的毕业生就业信息管理模块，具体功能如下所示。

（1）录入功能：允许管理员或毕业生便捷地录入个人基本信息及就业信息。

（2）数据显示：实时展示录入的就业信息，包括姓名、年龄、性别、薪资、就业城市等，以表格形式呈现。

（3）数据验证：在信息录入过程中，实施基本的数据验证，确保提交的信息完整且格式正确。

（4）删除功能：提供选项以便于管理已录入的就业信息，允许删除不再需要的记录。

5.3.1 界面设计与布局

本次实践通过 HTML 与 CSS 设计并实现一个既实用又美观的界面，用于毕业生就业信息的录入与展示。界面将包含两个核心部分：一个是表单，用于收集基本信息，另一个是表格，用于展示已录入的就业数据。

表单部分涵盖了姓名、年龄、性别、薪资以及就业城市的输入项，性别与就业城市通过下拉选择框提供选项，最后配备一个提交按钮以完成数据录入。紧接着是一个表格，用于展示录入的就业信息，包含学号、姓名、年龄、性别、薪资、就业城市及操作列，其中操作列预留空间以供未来实现数据管理功能，HTML 代码如下所示。

```
<!DOCTYPE html>
<htmllang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="./style.css">
</head>
<body>
<h1>毕业生就业情况统计</h1>
<form class="info">
    姓名：<input type="text" class="uname" name="uname" />
    年龄：<input type="text" class="age" name="age" />
    性别：<select name="gender" class="gender">
          <option value="男">男</option>
          <option value="女">女</option>
         </select>
    薪资：<input type="text" class="salary" name="salary" />
    就业城市：<select name="city" class="city">
    <option value="北京">北京</option>
    <option value="上海">上海</option>
    <option value="深圳">深圳</option>
    <option value="广州">广州</option>
</select>
    <button class="add">录入</button>
```

```
</form>
<h1>就业信息</h1>
<table>
    <thead>
    <th>学号</th>
    <th>姓名</th>
    <th>年龄</th>
    <th>性别</th>
    <th>薪资</th>
    <th>就业城市</th>
    <th>操作</th>
    </thead>
    <tbody>
    </tbody>
</table>
</body>
</html>
```

根据以上 HTML 页面代码，为提升用户体验，通过外部 CSS 文件 styles.css 对界面进行美化，确保布局合理、视觉舒适。CSS 代码如下所示。

```
/* 设置整体布局 */
body {
    font-family: Arial, sans-serif;
}
/* 标题样式 */
h1 {
    text-align: center;
    margin-bottom: 20px;
}
/* 表单样式 */
.info form {
    display: flex;
    flex-direction: column;
    align-items: center;
}
.info input[type="text"],
.info select {
    width: 150px;
    padding: 5px;
    margin-bottom: 20px;
}
.info button.add {
    background-color: #4CAF50;
    color: white;
```

```
    padding: 10px 20px;
    border: none;
    cursor: pointer;
}
/* 按钮样式 */
.info button:hover {
    background-color: #45a049;
}
/* 表格样式 */
table {
    width: 100%;
    border-collapse: collapse;
}
th, td {
    border: 1px solid #ddd;
    padding: 4px;
    text-align: left;
}
th {
    background-color: #f2f2f2;
    font-weight: bold;
}
tr:nth-child(even) {
    background-color: #f2f2f2;
}
tr:hover {
    background-color: #ddd;
}
/* 删除按钮样式 */
td a {
    display: inline-block;
    padding: 2px 5px;
    background-color: red;
    color: white;
    border-radius: 5px;
    text-decoration: none;
}
td a:hover {
    background-color: darkred;
}
```

在当前目录下新建一个 styles.css 文件，将以上 CSS 代码复制到该文件中，并在 HTML 文件中引用该 CSS 文件。在 HTML 文件的<head>部分添加以下代码。

```
<link rel="stylesheet" href="styles.css">
```

通过上述 HTML 结构与 CSS 样式的结合，构建了一个清晰、易用的界面，既便于用户录入就业信息，又能直观展示数据。最终，呈现在浏览器中的整体界面效果图将如图 5-2 所示。

图 5-2　整体界面效果图

5.3.2　表单提交事件

本节将介绍通过 JavaScript 增强网页的交互性，专注于处理毕业生就业信息表单的提交事件，为后续的表单验证、数据处理与展示打下基础，具体步骤如下所示。

创建一个名为 script.js 的文件，用于存放 JavaScript 逻辑代码。这个文件将与 HTML 文件协同工作，实现对表单提交行为的控制。以下是在 script.js 文件中应添加的代码。

```
//1.获取 HTML 的元素
const uname=document.querySelector('.uname')
const age=document.querySelector('.age')
const gender=document.querySelector('.gender')
const salary=document.querySelector('.salary')
const city=document.querySelector('.city')
const tbody=document.querySelector('tbody')
//获取所有带 name 属性的元素
const items=document.querySelectorAll('[name]')
//声明一个空数组，保存毕业生就业信息数据
const arr=[];
//获取 form 表单
const info=document.querySelector('.info')
//编写监听表单提交事件，阻止自动提交
info.addEventListener('submit', function (e){
    //阻止默认行为，不跳转
    e.preventDefault()
})
```

上述代码通过 querySelector 选取了表单元素，并声明了一个数组 arr 用于存储就业信息。接着，通过 addEventListener 为表单绑定了 submit 事件监听器，当表单提交时，会触发一个事件处理器。在该处理器中，event.preventDefault()阻止了表单的默认提交动作，避免页面刷新。

在当前目录下新建一个 script.js 文件，将以上的 JavaScript 代码复制到该文件中，并在 HTML 文件中正确引用该 JS 文件。在 HTML 文件的<head>或<body>标签结束前

添加如下代码。

```
<!-- 引入 script.js 文件 -->
<script src="script.js"></script>
```

本次实践范例中提供的 HTML 模板中包含了引入外部 JavaScript 文件的基本结构。代码<script src="script.js"></script>就是用来引用 script.js 文件的，而这句代码的前提是 script.js 文件与 HTML 文件位于同一目录下。如果 script.js 位于其他目录，请确保使用正确的相对路径或绝对路径来指定文件的位置。例如，如果 script.js 位于一个名为 js 的子目录中，路径应改为<script src="js/script.js"></script>。

5.3.3 实现表单录入验证

通过 JavaScript 增强表单的交互性，实现用户输入数据的基本验证，以确保所有必填项均已被恰当填写。在已有的 script.js 文件中，为表单提交事件绑定了一个监听器，旨在阻止默认的页面刷新行为，并在此基础上添加了数据验证逻辑。在表单监听事件中的新增代码如下所示。

```
//编写监听表单提交事件,阻止自动提交
info.addEventListener('submit', function (e){
    //阻止默认行为,不跳转
    e.preventDefault()
    //进行表单验证,如果不通过直接中断,不需要添加数据
    for(let i=0;i<items.length;i++){
        if (items[i].value===''){
            return alert('输入内容不能为空')
        }
    }
    //如果通过验证, 此处可添加处理逻辑, 如将数据添加到数组或发送至服务器
})
```

录入验证包括以下两部分。

(1) 遍历验证：通过 for 循环，对表单中的每个元素进行逐一检查，确保没有遗漏，items 数组中包含了表单中所有需要验证的输入项。

(2) 空值检测：使用 if (items[i].value === '')判断用户是否留下了空值，以此作为必填项的标准。一旦发现空项，立即通过 alert 弹窗提醒用户“请输入完整信息后再提交”，并使用 return 语句中断函数执行，有效阻止了未完整填写表单的提交。

通过上述功能的实现，不仅提升了用户体验，确保了数据的完整性，也为后续的数据处理流程打下了坚实的基础。在浏览器中预览此页面，用户将直观地看到一个响应式的表单验证机制，如图 5-3 所示，任何未完整填写的表单都将被即时拦截并给用户清晰的反馈。

5.3.4 实现信息录入功能

在用户成功提交表单并通过验证之后，根据用户录入的相关数据，新增一个用户对

图 5-3　表单输入验证效果图

象，并将该用户追加到 arr 数组中。在已有的表单提交事件处理逻辑之后，追加如下代码。

```
//创建新对象
    const obj={
        stuId:arr.length+1,
        uname:uname.value,
        age:age.value,
        gender:gender.value,
        salary:salary.value,
        city:city.value
    }
    //在数组中追加数据
    arr.push(obj)
    //清空表单
    this.reset()
```

上述代码相关功能如下所示。

(1) 数据存储：obj 用于存储用户提交的就业信息。每个属性对应表单中的一个用户输入的姓名、年龄、性别、薪资和就业城市。

(2) 数据持久化：通过 arr.push(obj)将封装好的用户对象添加到全局数组 arr 中，实现了信息的持久存储，使得每次收到表单后，能记住并累积所有用户输入的历史记录。

(3) 表单重置：this.reset()调用表单的重置方法，清除了所有输入字段，为下一轮数据录入做好准备，提高了用户体验，使得用户无须手动清理输入框，即可开始新的数据录入过程。

通过以上步骤，不仅实现了用户信息的即时捕获与存储，还确保了界面的即时反馈，使得用户在连续录入信息时更加流畅与便捷。

5.3.5　实现数据显示功能

本节的主要目的是实现数据的直观展示，确保用户录入的信息能够即时反映在网页界面上。核心在于设计一个高效的渲染机制，以数据数组 arr 为蓝本，更新表格视图，代码如下所示。

```
//渲染函数
function render(){
    //先清空 tbody, 把最新数组里面的数据渲染完毕
    tbody.innerHTML=''
    for(let i=0;i<arr.length;i++){
        //创建一个新的表格行<tr>元素
        const tr=document.createElement('tr')
        tr.innerHTML=`
        <td>${arr[i].stuId}</td>
        <td>${arr[i].uname}</td>
        <td>${arr[i].age}</td>
        <td>${arr[i].gender}</td>
        <td>${arr[i].salary}</td>
        <td>${arr[i].city}</td>
        <td>
          <a href="javascript:" data-id=${i}>删除</a>
        </td>
        `
        tbody.appendChild(tr)
    }
```

通过上述 render 函数，系统能够根据 arr 数组的实时状态重构表格内容，确保界面展示与底层数据保持同步。此过程先清空旧的表格行，避免数据重复；随后逐条迭代数组，为每条记录生成一行表格，其中包含详细的用户信息及一个便捷的“删除”链接；最后，利用 data-id 属性标记，以便进行精确的交互控制。

为了保证新录入的信息即刻反映于界面，需要在表单提交逻辑的末尾调用 render 函数。这一步骤确保了每当有新数据加入数组 arr，页面上的表格便会刷新，展现出最新数据集，因此新增如下代码。

```
//在数组中追加数据
arr.push(obj)
//清空表单
this.reset()
//重新渲染
render()
```

完成上述配置后，实现了一个响应式的表格界面，不仅实时反映了所有用户录入的信息，还为每条记录配备了删除选项，如图 5-4 所示。

5.3.6 实现删除功能

删除功能实现了用户界面与数据模型的交互，允许用户通过单击表格中的删除链接来移除特定的数据行，并自动更新界面视图，保持界面展示与后台数据的一致性，代码如下所示。

图 5-4　表格数据展示效果图

```
    //删除操作
    tbody.addEventListener('click', function (e){
        if(e.target.tagName==='A'){
            //得到当前元素的自定义属性 data-id
            arr.splice(e.target.dataset.id, 1)
            render()
        }
    })
}
</script>
```

上述代码采用事件委托技术，在 tbody 上设置一个事件监听器，而非为每个删除按钮单独设置。这样可以减少事件监听器的数量，提高效率，尤其适用于动态生成的元素。

此外，还要判断触发事件的目标元素（e.target）是否为<a>标签，因为只有当用户单击删除链接时，才需要执行删除操作。

如果用户单击了删除链接，这行代码会从被单击的链接（a 标签）获取其自定义属性 data-id 的值，该属性存储了数组中对应元素的索引。然后，使用 Array.prototype.splice() 方法根据这个索引从 arr 数组中移除对应的数据项。上述代码中，splice(index,1)的意思是从 index 位置开始删除 1 个元素。

删除数组中的元素后，立即调用 render 函数重新渲染表格，确保界面上的数据与当前 arr 数组中的内容保持一致，即删除操作的结果能够即时反映在表格上。在浏览器中打开该页面，删除操作效果图如图 5-5 所示。

图 5-5　删除操作效果图