教程使用说明

本教程采用渐进式学习方式，每个章节都包含完整的HTML页面示例。您可以将代码复制到本地文件中运行，或直接在浏览器开发者工具中练习。每个练习都包含"练习前"和"练习后"的对比代码，帮助您理解最佳实践的实际效果。

开始前的准备：

1. 创建一个名为 `js-best-practices` 的文件夹
2. 在其中创建各章节对应的HTML文件
3. 使用现代浏览器（Chrome、Firefox、Safari、Edge）运行示例
4. 打开浏览器开发者工具观察效果

第1章：DOM操作基础与最佳实践

练习1.1：高效的DOM批量操作

创建文件： `dom-batch-operations.html`

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM批量操作练习</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
        }
        .container {
            margin: 20px 0;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 5px;
        }
        .user-list {
            list-style: none;
            padding: 0;
        }
        .user-item {
            padding: 10px;
            margin: 5px 0;
            background: #f5f5f5;
            border-radius: 3px;
        }
        .performance-info {
            background: #e7f3ff;
            padding: 10px;
            margin: 10px 0;
            border-radius: 3px;
        }
        button {
            padding: 10px 20px;
            margin: 5px;
            border: none;
            border-radius: 3px;
            background: #007bff;
            color: white;
            cursor: pointer;
        }
        button:hover {
            background: #0056b3;
        }
        .bad-example {
            border-left: 4px solid #dc3545;
        }
        .good-example {
            border-left: 4px solid #28a745;
        }
    </style>
</head>
<body>
    <h1>DOM批量操作最佳实践</h1>

    <div class="container bad-example">
        <h2>❌ 不好的做法：频繁DOM操作</h2>
        <button onclick="updateUserListPoorly()">低效方式添加用户</button>
        <ul id="user-list-bad" class="user-list"></ul>
        <div id="performance-bad" class="performance-info"></div>
    </div>

    <div class="container good-example">
        <h2>✅ 好的做法：批量DOM操作</h2>
        <button onclick="updateUserListEfficiently()">高效方式添加用户</button>
        <ul id="user-list-good" class="user-list"></ul>
        <div id="performance-good" class="performance-info"></div>
    </div>

    <div class="container">
        <h3>练习任务</h3>
        <p>1. 点击两个按钮，观察性能差异</p>
        <p>2. 打开开发者工具的Performance选项卡，录制并对比两种方法</p>
        <p>3. 修改代码，尝试添加更多用户（如1000个）观察差异</p>
    </div>

    <script>
        // 模拟用户数据
        const users = [
            { id: 1, name: '张三', email: 'zhangsan@example.com' },
            { id: 2, name: '李四', email: 'lisi@example.com' },
            { id: 3, name: '王五', email: 'wangwu@example.com' },
            { id: 4, name: '赵六', email: 'zhaoliu@example.com' },
            { id: 5, name: '钱七', email: 'qianqi@example.com' }
        ];

        // ❌ 不好的做法：每次操作都直接修改DOM
        function updateUserListPoorly() {
            const startTime = performance.now();
            const userList = document.getElementById('user-list-bad');

            // 清空列表
            userList.innerHTML = '';

            // 每次迭代都进行DOM操作
            users.forEach(user => {
                userList.innerHTML += `
                    <li class="user-item" data-user-id="${user.id}">
                        <strong>${user.name}</strong><br>
                        <small>${user.email}</small>
                    </li>
                `;
            });

            const endTime = performance.now();
            const performanceDiv = document.getElementById('performance-bad');
            performanceDiv.innerHTML = `
                <strong>执行时间：${(endTime - startTime).toFixed(2)}ms</strong><br>
                <small>DOM操作次数：${users.length + 1}次（每个用户1次 + 清空1次）</small>
            `;
        }

        // ✅ 好的做法：使用DocumentFragment批量操作
        function updateUserListEfficiently() {
            const startTime = performance.now();
            const userList = document.getElementById('user-list-good');
            // 创建文档片段
            const fragment = document.createDocumentFragment();

            // 在内存中构建所有元素
            users.forEach(user => {
                const li = document.createElement('li');
                li.className = 'user-item';
                li.dataset.userId = user.id;
                li.innerHTML = `
                    <strong>${user.name}</strong><br>
                    <small>${user.email}</small>
                `;
                fragment.appendChild(li);
            });

            // 一次性更新DOM
            userList.innerHTML = '';
            userList.appendChild(fragment);

            const endTime = performance.now();
            const performanceDiv = document.getElementById('performance-good');
            performanceDiv.innerHTML = `
                <strong>执行时间：${(endTime - startTime).toFixed(2)}ms</strong><br>
                <small>DOM操作次数：2次（清空1次 + 批量添加1次）</small>
            `;
        }

        // 页面加载时的提示
        window.addEventListener('load', () => {
            console.log('💡 练习提示：');
            console.log('1. 观察两种方法的执行时间差异');
            console.log('2. 在开发者工具的Elements面板中观察DOM变化');
            console.log('3. 尝试修改users数组，添加更多数据测试性能');
        });
    </script>
</body>
</html>
```

练习1.2：现代DOM查询和事件委托

创建文件：`dom-events-delegation.html`

```
<!DOCTYPE html>
<html lang="zh-CN">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>事件委托练习</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
        }
        .container {
            margin: 20px 0;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 5px;
        }
        .todo-item {
            display: flex;
            justify-content: space-between;
            align-items: center;
            padding: 10px;
            margin: 5px 0;
            background: #f8f9fa;
            border-radius: 3px;
            border-left: 4px solid #007bff;
        }
        .todo-item.completed {
            background: #d4edda;
            border-left-color: #28a745;
            text-decoration: line-through;
            opacity: 0.7;
        }
        .todo-actions {
            display: flex;
            gap: 10px;
        }
        button {
            padding: 5px 10px;
            border: none;
            border-radius: 3px;
            cursor: pointer;
            font-size: 12px;
        }
        .btn-complete {
            background: #28a745;
            color: white;
        }
        .btn-edit {
            background: #ffc107;
            color: black;
        }
        .btn-delete {
            background: #dc3545;
            color: white;
        }
        .btn-add {
            background: #007bff;
            color: white;
            padding: 10px 20px;
            font-size: 14px;
        }
        input[type="text"] {
            padding: 8px;
            border: 1px solid #ddd;
            border-radius: 3px;
            width: 200px;
        }
        .event-log {
            background: #f8f9fa;
            padding: 10px;
            border-radius: 3px;
            max-height: 150px;
            overflow-y: auto;
            font-family: monospace;
            font-size: 12px;
        }
    </style>
</head>
<body>
    <h1>事件委托最佳实践</h1>

    <div class="container">
        <h2>任务管理应用</h2>
        <div>
            <input type="text" id="new-todo" placeholder="输入新任务...">
            <button class="btn-add" onclick="addTodo()">添加任务</button>
        </div>

        <div id="todo-list" class="todo-list">
            <!-- 任务项将动态添加到这里 -->
        </div>

        <h3>事件日志</h3>
        <div id="event-log" class="event-log"></div>
        <div style="margin-top: 20px;">
            <h3>练习要点</h3>
            <ul>
                <li>使用事件委托处理动态添加的元素</li>
                <li>利用event.target.closest()进行元素查找</li>
                <li>观察事件冒泡的工作原理</li>
                <li>体验性能优势：无需为每个按钮单独绑定事件</li>
            </ul>
        </div>
    </div>

    <script>
        class TodoApp {
            constructor() {
                this.todos = [
                    { id: 1, text: '学习JavaScript最佳实践', completed: false
},
                    { id: 2, text: '练习DOM操作', completed: true },
                    { id: 3, text: '掌握事件委托', completed: false }
                ];
                this.nextId = 4;

                this.init();
            }

            init() {
                this.render();
                this.bindEvents();
                this.logEvent('应用初始化完成');
            }

            // ✅ 使用事件委托 - 只需要一个事件监听器
            bindEvents() {
                const todoList = document.getElementById('todo-list');

                todoList.addEventListener('click', (event) => {
                    // 使用closest查找目标元素
                    const todoItem = event.target.closest('.todo-item');
                    if (!todoItem) return;

                    const todoId = parseInt(todoItem.dataset.todoId);
                    this.logEvent(`点击事件触发，目标：
${event.target.className}`);

                    // 根据点击的按钮执行不同操作
                    if (event.target.matches('.btn-complete')) {
                        this.toggleComplete(todoId);
                    } else if (event.target.matches('.btn-edit')) {
                        this.editTodo(todoId);
                    } else if (event.target.matches('.btn-delete')) {
                        this.deleteTodo(todoId);
                    }
                });

                // 为输入框绑定回车事件
                document.getElementById('new-todo').addEventListener('keypress', (event) => {
                    if (event.key === 'Enter') {
                        this.addTodo();
                    }
                });
            }

            render() {
                const todoList = document.getElementById('todo-list');

                if (this.todos.length === 0) {
                    todoList.innerHTML = '<p style="text-align: center; color: #666;">暂无任务</p>';
                    return;
                }

                // 使用DocumentFragment优化性能
                const fragment = document.createDocumentFragment();

                this.todos.forEach(todo => {
                    const div = document.createElement('div');
                    div.className = `todo-item ${todo.completed ? 'completed' : ''}`;
                    div.dataset.todoId = todo.id;

                    div.innerHTML = `
                        <span class="todo-text">${todo.text}</span>
                        <div class="todo-actions">
                            <button class="btn-complete">
                                ${todo.completed ? '取消完成' : '完成'}
                            </button>
                            <button class="btn-edit">编辑</button>
                            <button class="btn-delete">删除</button>
                        </div>
                    `;

                    fragment.appendChild(div);
                });

                todoList.innerHTML = '';
                todoList.appendChild(fragment);

                this.logEvent(`渲染了 ${this.todos.length} 个任务项`);
            }

            addTodo() {
                const input = document.getElementById('new-todo');
                const text = input.value.trim();

                if (!text) return;

                this.todos.push({
                    id: this.nextId++,
                    text: text,
                    completed: false
                });

                input.value = '';
                this.render();
                this.logEvent(`添加新任务：${text}`);
            }

            toggleComplete(id) {
                const todo = this.todos.find(t => t.id === id);
                if (todo) {
                    todo.completed = !todo.completed;
                    this.render();
                    this.logEvent(`切换任务状态：ID ${id}`);
                }
            }

            editTodo(id) {
                const todo = this.todos.find(t => t.id === id);
                if (todo) {
                    const newText = prompt('编辑任务:', todo.text);
                    if (newText !== null && newText.trim()) {
                        todo.text = newText.trim();
                        this.render();
                        this.logEvent(`编辑任务：ID ${id}`);
                    }
                }
            }

            deleteTodo(id) {
                if (confirm('确定要删除这个任务吗？')) {
                    this.todos = this.todos.filter(t => t.id !== id);
                    this.render();
                    this.logEvent(`删除任务：ID ${id}`);
                }
            }

            logEvent(message) {
                const log = document.getElementById('event-log');
                const time = new Date().toLocaleTimeString();
                log.innerHTML += `[${time}] ${message}\n`;
                log.scrollTop = log.scrollHeight;
            }
        }

        // 初始化应用
        let todoApp;
        window.addEventListener('load', () => {
            todoApp = new TodoApp();

            console.log('💡 练习提示：');
            console.log('1. 添加几个任务，观察事件如何通过委托处理');
            console.log('2. 在开发者工具中检查事件监听器数量');
            console.log('3. 尝试修改代码，为每个按钮单独绑定事件，对比差异');
        });
    </script>
</body>
</html>
```

第2章：浏览器存储最佳实践

练习2.1：本地存储管理器

创建文件： `storage-manager.html`

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>浏览器存储练习</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 1000px;
            margin: 0 auto;
            padding: 20px;
        }
        .storage-demo {
            display: grid;
            grid-template-columns: 1fr 1fr 1fr;
            gap: 20px;
            margin: 20px 0;
        }
        .storage-section {
            border: 1px solid #ddd;
            border-radius: 5px;
            padding: 15px;
        }
        .storage-section h3 {
            margin-top: 0;
            color: #333;
        }
        .localStorage { border-left: 4px solid #007bff; }
        .sessionStorage { border-left: 4px solid #28a745; }
        .indexedDB { border-left: 4px solid #ffc107; }

        .form-group {
            margin: 10px 0;
        }
        .form-group label {
            display: block;
            margin-bottom: 5px;
            font-weight: bold;
        }
        .form-group input, .form-group textarea, .form-group select {
            width: 100%;
            padding: 8px;
            border: 1px solid #ddd;
            border-radius: 3px;
            box-sizing: border-box;
        }
        button {
            padding: 8px 16px;
            margin: 5px 5px 5px 0;
            border: none;
            border-radius: 3px;
            cursor: pointer;
            font-size: 12px;
        }
        .btn-primary { background: #007bff; color: white; }
        .btn-success { background: #28a745; color: white; }
        .btn-warning { background: #ffc107; color: black; }
        .btn-danger { background: #dc3545; color: white; }

        .storage-display {
            margin: 10px 0;
            padding: 10px;
            background: #f8f9fa;
            border-radius: 3px;
            max-height: 200px;
            overflow-y: auto;
            font-family: monospace;
            font-size: 12px;
        }
        .quota-info {
            background: #e7f3ff;
            padding: 10px;
            margin: 10px 0;
            border-radius: 3px;
            font-size: 12px;
        }
    </style>
</head>
<body>
    <h1>浏览器存储最佳实践演练</h1>

    <div class="storage-demo">
        <!-- LocalStorage 演示 -->
        <div class="storage-section localStorage">
            <h3>📁 LocalStorage</h3>
            <p><small>持久化存储，适合用户偏好设置</small></p>

            <div class="form-group">
                <label>主题设置</label>
                <select id="theme-select">
                    <option value="light">浅色主题</option>
                    <option value="dark">深色主题</option>
                    <option value="auto">自动</option>
                </select>
            </div>

            <div class="form-group">
                <label>用户名</label>
                <input type="text" id="username" placeholder="输入用户名">
            </div>

            <div class="form-group">
                <label>语言偏好</label>
                <select id="language">
                    <option value="zh-CN">中文</option>
                    <option value="en-US">English</option>
                    <option value="ja-JP">日本語</option>
                </select>
            </div>
            <button class="btn-primary" onclick="savePreferences()">保存偏好</button>
            <button class="btn-danger" onclick="clearPreferences()">清除偏好</button>

            <div id="localStorage-display" class="storage-display"></div>
        </div>

        <!-- SessionStorage 演示 -->
        <div class="storage-section sessionStorage">
            <h3>🕒 SessionStorage</h3>
            <p><small>会话存储，适合临时数据</small></p>

            <div class="form-group">
                <label>临时笔记</label>
                <textarea id="temp-note" rows="3" placeholder="输入临时笔记..."></textarea>
            </div>

            <div class="form-group">
                <label>购物车商品</label>
                <input type="text" id="cart-item" placeholder="商品名称">
                <input type="number" id="cart-quantity" placeholder="数量" min="1" value="1">
            </div>

            <button class="btn-success" onclick="saveToSession()">保存到会话</button>
            <button class="btn-success" onclick="addToCart()">添加到购物车</button>
            <button class="btn-danger" onclick="clearSession()">清除会话</button>

            <div id="sessionStorage-display" class="storage-display"></div>
        </div>

        <!-- IndexedDB 演示 -->
        <div class="storage-section indexedDB">
            <h3>🗃️ IndexedDB</h3>
            <p><small>大容量结构化存储</small></p>

            <div class="form-group">
                <label>文档标题</label>
                <input type="text" id="doc-title" placeholder="输入文档标题">
            </div>

            <div class="form-group">
                <label>文档内容</label>
                <textarea id="doc-content" rows="4" placeholder="输入文档内容..."></textarea>
            </div>

            <div class="form-group">
                <label>分类</label>
                <select id="doc-category">
                    <option value="work">工作</option>
                    <option value="personal">个人</option>
                    <option value="study">学习</option>
                </select>
            </div>

            <button class="btn-warning" onclick="saveDocument()">保存文档</button>
            <button class="btn-warning" onclick="loadDocuments()">加载文档</button>
            <button class="btn-danger" onclick="clearDocuments()">清除所有</button>

            <div id="indexedDB-display" class="storage-display"></div>
        </div>
    </div>

    <!-- 存储配额信息 -->
    <div class="quota-info">
        <h3>存储配额信息</h3>
        <div id="quota-display">加载中...</div>
        <button class="btn-primary" onclick="checkStorageQuota()">检查存储配额</button>
    </div>

    <script>
        // ✅ LocalStorage 最佳实践
        class UserPreferences {
            static save(preferences) {
                try {
                    localStorage.setItem('userPreferences', JSON.stringify(preferences));
                    this.displayLocalStorage();
                    return true;
                } catch (e) {
                    if (e.name === 'QuotaExceededError') {
                        alert('存储空间不足！');
                    } else {
                        console.error('保存偏好设置失败:', e);
                    }
                    return false;
                }
            }

            static load() {
                try {
                    const stored = localStorage.getItem('userPreferences');
                    return stored ? JSON.parse(stored) : null;
                } catch (e) {
                    console.error('加载偏好设置失败:', e);
                    return null;
                }
            }

            static displayLocalStorage() {
                const display = document.getElementById('localStorage-display');
                const prefs = this.load();
                if (prefs) {
                    display.innerHTML = `
                        <strong>已保存的偏好:</strong><br>
                        ${JSON.stringify(prefs, null, 2)}
                    `;
                } else {
                    display.innerHTML = '<em>暂无保存的偏好设置</em>';
                }
            }
        }

        // ✅ SessionStorage 最佳实践
        class SessionManager {
            static setTempData(key, value) {
                try {
                    sessionStorage.setItem(key, JSON.stringify({
                        value,
                        timestamp: Date.now()
                    }));
                    this.displaySessionStorage();
                } catch (e) {
                    console.error('保存会话数据失败:', e);
                }
            }

            static getTempData(key, maxAge = 3600000) { // 默认1小时
                try {
                    const stored = sessionStorage.getItem(key);
                    if (!stored) return null;

                    const { value, timestamp } = JSON.parse(stored);
                    if (Date.now() - timestamp > maxAge) {
                        sessionStorage.removeItem(key);
                        return null;
                    }

                    return value;
                } catch (e) {
                    console.error('读取会话数据失败:', e);
                    return null;
                }
            }

            static displaySessionStorage() {
                const display = document.getElementById('sessionStorage-
display');
                const items = [];

                for (let i = 0; i < sessionStorage.length; i++) {
                    const key = sessionStorage.key(i);
                    const value = sessionStorage.getItem(key);
                    items.push(`${key}: ${value}`);
                }

                display.innerHTML = items.length > 0 ?
                    `<strong>会话存储:</strong><br>${items.join('<br>')}` :
                    '<em>暂无会话数据</em>';
            }
        }

        // ✅ IndexedDB 最佳实践
        class DocumentDatabase {
            constructor() {
                this.dbName = 'DocumentDB';
                this.version = 1;
                this.db = null;
            }

            async init() {
                return new Promise((resolve, reject) => {
                    const request = indexedDB.open(this.dbName,
this.version);

                    request.onerror = () => reject(request.error);
                    request.onsuccess = () => {
                        this.db = request.result;
                        resolve(this.db);
                    };
                    request.onupgradeneeded = (event) => {
                        const db = event.target.result;
                        if (!db.objectStoreNames.contains('documents')) {
                            const store = db.createObjectStore('documents',
{
                                keyPath: 'id',
                                autoIncrement: true
                            });
                            store.createIndex('category', 'category', {
unique: false });
                            store.createIndex('title', 'title', { unique:
false });
                        }
                    };
                });
            }

            async saveDocument(document) {
                if (!this.db) await this.init();

                return new Promise((resolve, reject) => {
                    const transaction = this.db.transaction(['documents'],
'readwrite');
                    const store = transaction.objectStore('documents');

                    const request = store.add({
                        ...document,
                        createdAt: new Date().toISOString()
                    });

                    request.onsuccess = () => resolve(request.result);
                    request.onerror = () => reject(request.error);
                });
            }

            async getAllDocuments() {
                if (!this.db) await this.init();

                return new Promise((resolve, reject) => {
                    const transaction = this.db.transaction(['documents'],
'readonly');
                    const store = transaction.objectStore('documents');
                    const request = store.getAll();

                    request.onsuccess = () => resolve(request.result);
                    request.onerror = () => reject(request.error);
                });
            }
            async clearAllDocuments() {
                if (!this.db) await this.init();

                return new Promise((resolve, reject) => {
                    const transaction = this.db.transaction(['documents'],
'readwrite');
                    const store = transaction.objectStore('documents');
                    const request = store.clear();

                    request.onsuccess = () => resolve();
                    request.onerror = () => reject(request.error);
                });
            }

            async displayDocuments() {
                try {
                    const documents = await this.getAllDocuments();
                    const display = document.getElementById('indexedDB-
display');

                    if (documents.length === 0) {
                        display.innerHTML = '<em>暂无保存的文档</em>';
                        return;
                    }

                    const docList = documents.map(doc =>
                        `<div style="margin: 5px 0; padding: 5px;
background: white; border-radius: 2px;">
                            <strong>${doc.title}</strong> (${doc.category})
<br>
                            <small>${doc.content.substring(0, 50)}...
</small><br>
                            <small>创建时间: ${new
Date(doc.createdAt).toLocaleString()}</small>
                        </div>`
                    ).join('');

                    display.innerHTML = `<strong>已保存的文档
(${documents.length}):</strong><br>${docList}`;
                } catch (error) {
                    document.getElementById('indexedDB-display').innerHTML =
                        `<em style="color: red;">加载失败: ${error.message}
</em>`;
                }
            }
        }
        // 全局实例
        const docDB = new DocumentDatabase();

        // 界面交互函数
        function savePreferences() {
            const prefs = {
                theme: document.getElementById('theme-select').value,
                username: document.getElementById('username').value,
                language: document.getElementById('language').value,
                savedAt: new Date().toISOString()
            };

            if (UserPreferences.save(prefs)) {
                alert('偏好设置保存成功！');
            }
        }

        function clearPreferences() {
            localStorage.removeItem('userPreferences');
            UserPreferences.displayLocalStorage();
            alert('偏好设置已清除！');
        }

        function saveToSession() {
            const note = document.getElementById('temp-note').value;
            if (note.trim()) {
                SessionManager.setTempData('tempNote', note);
                alert('临时笔记保存成功！');
            }
        }

        function addToCart() {
            const item = document.getElementById('cart-item').value;
            const quantity = document.getElementById('cart-quantity').value;

            if (item.trim()) {
                const cart = SessionManager.getTempData('shoppingCart') ||
[];
                cart.push({ item, quantity: parseInt(quantity), addedAt: new
Date().toISOString() });
                SessionManager.setTempData('shoppingCart', cart);

                document.getElementById('cart-item').value = '';
                document.getElementById('cart-quantity').value = '1';
                alert('商品已添加到购物车！');
            }
        }
        function clearSession() {
            sessionStorage.clear();
            SessionManager.displaySessionStorage();
            alert('会话数据已清除！');
        }

        async function saveDocument() {
            const title = document.getElementById('doc-title').value;
            const content = document.getElementById('doc-content').value;
            const category = document.getElementById('doc-category').value;

            if (title.trim() && content.trim()) {
                try {
                    await docDB.saveDocument({ title, content, category });
                    document.getElementById('doc-title').value = '';
                    document.getElementById('doc-content').value = '';
                    await docDB.displayDocuments();
                    alert('文档保存成功！');
                } catch (error) {
                    alert('文档保存失败：' + error.message);
                }
            } else {
                alert('请填写标题和内容！');
            }
        }

        async function loadDocuments() {
            await docDB.displayDocuments();
        }

        async function clearDocuments() {
            if (confirm('确定要清除所有文档吗？')) {
                try {
                    await docDB.clearAllDocuments();
                    await docDB.displayDocuments();
                    alert('所有文档已清除！');
                } catch (error) {
                    alert('清除失败：' + error.message);
                }
            }
        }

        async function checkStorageQuota() {
            if ('storage' in navigator && 'estimate' in navigator.storage) {
                try {
                    const estimate = await navigator.storage.estimate();
                    const used = (estimate.usage / 1024 / 1024).toFixed(2);
                    const total = (estimate.quota / 1024 / 1024).toFixed(2);
                    const percent = ((estimate.usage / estimate.quota) * 100).toFixed(2);

                    document.getElementById('quota-display').innerHTML = `
                        <strong>存储配额信息:</strong><br>
                        已使用：${used} MB<br>
                        总配额：${total} MB<br>
                        使用率：${percent}%<br>
                        <small>注意：配额可能根据可用空间动态调整</small>
                    `;
                } catch (error) {
                    document.getElementById('quota-display').innerHTML =
                        '无法获取配额信息：' + error.message;
                }
            } else {
                document.getElementById('quota-display').innerHTML =
                    '此浏览器不支持存储配额API';
            }
        }

        // 页面初始化
        window.addEventListener('load', async () => {
            // 加载已保存的偏好设置
            const prefs = UserPreferences.load();
            if (prefs) {
                document.getElementById('theme-select').value = prefs.theme || 'light';
                document.getElementById('username').value = prefs.username || '';
                document.getElementById('language').value = prefs.language || 'zh-CN';
            }
            UserPreferences.displayLocalStorage();

            // 显示会话存储
            SessionManager.displaySessionStorage();

            // 初始化IndexedDB并显示文档
            try {
                await docDB.init();
                await docDB.displayDocuments();
            } catch (error) {
                console.error('IndexedDB初始化失败：', error);
            }

            // 检查存储配额
            checkStorageQuota();
            console.log('💡 练习提示：');
            console.log('1. 尝试保存不同类型的数据到各种存储中');
            console.log('2. 刷新页面观察数据持久性差异');
            console.log('3. 打开新标签页观察sessionStorage的行为');
            console.log('4. 在开发者工具的Application面板查看存储数据');
        });
    </script>
</body>
</html>
```

第3章：网络请求和错误处理

练习3.1：现代API客户端

创建文件： `api-client.html`

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>API客户端最佳实践</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 1200px;
            margin: 0 auto;
            padding: 20px;
        }
        .api-demo {
            display: grid;
            grid-template-columns: 1fr 1fr;
            gap: 20px;
            margin: 20px 0;
        }
        .demo-section {
            border: 1px solid #ddd;
            border-radius: 5px;
            padding: 20px;
        }
        .demo-section h3 {
            margin-top: 0;
        }
        .request-form {
            background: #f8f9fa;
            padding: 15px;
            border-radius: 5px;
            margin: 10px 0;
        }
        .form-row {
            display: flex;
            gap: 10px;
            margin: 10px 0;
            align-items: center;
        }
        .form-row label {
            min-width: 80px;
            font-weight: bold;
        }
        .form-row input, .form-row select {
            flex: 1;
            padding: 8px;
            border: 1px solid #ddd;
            border-radius: 3px;
        }
        button {
            padding: 10px 20px;
            margin: 5px;
            border: none;
            border-radius: 3px;
            cursor: pointer;
        }
        .btn-primary { background: #007bff; color: white; }
        .btn-success { background: #28a745; color: white; }
        .btn-warning { background: #ffc107; color: black; }
        .btn-danger { background: #dc3545; color: white; }
        .btn-secondary { background: #6c757d; color: white; }

        .response-display {
            background: #f8f9fa;
            border: 1px solid #ddd;
            border-radius: 5px;
            padding: 15px;
            margin: 10px 0;
            max-height: 300px;
            overflow-y: auto;
            font-family: monospace;
            font-size: 12px;
        }
        .loading {
            color: #007bff;
            font-style: italic;
        }
        .error {
            color: #dc3545;
            font-weight: bold;
        }
        .success {
            color: #28a745;
        }
        .request-info {
            background: #e7f3ff;
            padding: 10px;
            margin: 10px 0;
            border-radius: 3px;
            font-size: 12px;
        }
        .active-requests {
            background: #fff3cd;
            padding: 10px;
            margin: 10px 0;
            border-radius: 3px;
        }
    </style>
</head>
<body>
    <h1>网络请求最佳实践演练</h1>

    <div class="api-demo">
        <!-- 基本API请求演示 -->
        <div class="demo-section">
            <h3>📡 基本API请求</h3>

            <div class="request-form">
                <div class="form-row">
                    <label>方法:</label>
                    <select id="request-method">
                        <option value="GET">GET</option>
                        <option value="POST">POST</option>
                        <option value="PUT">PUT</option>
                        <option value="DELETE">DELETE</option>
                    </select>
                </div>

                <div class="form-row">
                    <label>URL:</label>
                    <input type="text" id="request-url"

value="https://jsonplaceholder.typicode.com/posts/1"
                           placeholder="输入API地址">
                </div>

                <div class="form-row">
```

```
                    <label>请求体:</label>
                    <textarea id="request-body" rows="3"
                              placeholder='{"title": "示例标题", "body": "示例内容"}'></textarea>
                </div>

                <button class="btn-primary" onclick="makeRequest()">发送请求</button>
                <button class="btn-secondary" onclick="clearResponse('basic')">清除响应</button>
            </div>

            <div class="request-info">
                <strong>预设API端点:</strong><br>
                • GET /posts/1 - 获取单个文章<br>
                • GET /posts - 获取所有文章<br>
                • POST /posts - 创建新文章<br>
                • PUT /posts/1 - 更新文章<br>
                • DELETE /posts/1 - 删除文章
            </div>

            <div id="basic-response" class="response-display">
                <em>响应将显示在这里...</em>
            </div>
        </div>

        <!-- 请求取消演示 -->
        <div class="demo-section">
            <h3>⏹️ 请求取消与超时</h3>

            <div class="request-form">
                <div class="form-row">
                    <label>延迟 (秒):</label>
                    <input type="number" id="delay-seconds" value="3" min="1" max="10">
                </div>

                <div class="form-row">
                    <label>超时 (秒):</label>
                    <input type="number" id="timeout-seconds" value="5" min="1" max="10">
                </div>

                <button class="btn-success" onclick="makeDelayedRequest()">发送延迟请求</button>
                <button class="btn-warning" onclick="makeCancellableRequest()">可取消请求</button>
                <button class="btn-danger" onclick="cancelAllRequests()">取消所有请求</button>
            </div>

            <div class="active-requests">
                <strong>活跃请求:</strong>
                <div id="active-requests-list">无活跃请求</div>
            </div>

            <div id="cancel-response" class="response-display">
                <em>响应将显示在这里...</em>
            </div>
        </div>
    </div>

    <!-- 错误处理演示 -->
    <div class="demo-section" style="grid-column: 1 / -1;">
        <h3>🚨 错误处理最佳实践</h3>

        <div style="display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 10px;">
            <button class="btn-primary" onclick="testSuccessRequest()">正常请求</button>
            <button class="btn-warning" onclick="testNetworkError()">网络错误</button>
            <button class="btn-warning" onclick="testHttpError()">HTTP错误</button>
            <button class="btn-warning" onclick="testTimeoutError()">超时错误</button>
            <button class="btn-warning" onclick="testJsonError()">JSON解析错误</button>
            <button class="btn-secondary" onclick="clearResponse('error')">清除日志</button>
        </div>

        <div id="error-response" class="response-display" style="height: 200px;">
            <em>错误处理日志将显示在这里...</em>
        </div>
    </div>

    <script>
        // ✅ 现代API客户端最佳实践
        class APIClient {
            constructor(baseURL = '') {
                this.baseURL = baseURL;
                this.defaultHeaders = {
                    'Content-Type': 'application/json',
                };
                this.controllers = new Map(); // 用于请求取消
                this.requestId = 0;
            }

            async request(endpoint, options = {}) {
                const requestId = ++this.requestId;
                const url = `${this.baseURL}${endpoint}`;

                // 创建AbortController用于取消请求
                const controller = new AbortController();
                this.controllers.set(requestId, controller);

                const config = {
                    ...options,
                    headers: {
                        ...this.defaultHeaders,
                        ...options.headers,
                    },
                    signal: controller.signal,
                };

                // 设置超时
                const timeoutId = options.timeout ?
                    setTimeout(() => controller.abort(), options.timeout) : null;

                try {
                    this.logRequest(requestId, options.method || 'GET', url);

                    const response = await fetch(url, config);

                    // 清除超时
                    if (timeoutId) clearTimeout(timeoutId);

                    // 检查HTTP状态
                    if (!response.ok) {
                        throw new HTTPError(response.status, response.statusText, response);
                    }

                    // 处理响应数据
                    const contentType = response.headers.get('content-type');
                    let data;
                    if (contentType &&
contentType.includes('application/json')) {
                        data = await response.json();
                    } else {
                        data = await response.text();
                    }

                    this.logResponse(requestId, response.status, data);
                    return data;

                } catch (error) {
                    if (timeoutId) clearTimeout(timeoutId);

                    if (error.name === 'AbortError') {
                        this.logCancel(requestId);
                        throw new CancelError('请求已被取消');
                    }

                    if (error instanceof HTTPError) {
                        this.logError(requestId, error);
                        throw error;
                    }

                    // 网络错误或其他fetch失败
                    const networkError = new NetworkError('网络请求失败',
error);
                    this.logError(requestId, networkError);
                    throw networkError;

                } finally {
                    this.controllers.delete(requestId);
                    this.updateActiveRequestsList();
                }
            }

            async get(endpoint, params = {}, options = {}) {
                const queryString = new URLSearchParams(params).toString();
                const url = queryString ? `${endpoint}?${queryString}` :
endpoint;
                return this.request(url, { ...options, method: 'GET' });
            }

            async post(endpoint, data, options = {}) {
                return this.request(endpoint, {
                    ...options,
                    method: 'POST',
                    body: JSON.stringify(data),
                });
            }

            async put(endpoint, data, options = {}) {
                return this.request(endpoint, {
                    ...options,
                    method: 'PUT',
                    body: JSON.stringify(data),
                });
            }

            async delete(endpoint, options = {}) {
                return this.request(endpoint, { ...options, method: 'DELETE' });
            }

            cancel(requestId) {
                const controller = this.controllers.get(requestId);
                if (controller) {
                    controller.abort();
                    this.controllers.delete(requestId);
                }
            }

            cancelAll() {
                this.controllers.forEach(controller => controller.abort());
                this.controllers.clear();
                this.updateActiveRequestsList();
            }

            logRequest(id, method, url) {
                console.log(`🚀 请求 ${id}: ${method} ${url}`);
                this.updateActiveRequestsList();
            }

            logResponse(id, status, data) {
                console.log(`✅ 响应 ${id}: ${status}`, data);
            }

            logError(id, error) {
                console.error(`❌ 错误 ${id}:`, error);
            }

            logCancel(id) {
                console.warn(`⏹️ 取消 ${id}: 请求已取消`);
            }

            updateActiveRequestsList() {
                const list = document.getElementById('active-requests-list');
                const count = this.controllers.size;
                list.textContent = count > 0 ?
                    `${count} 个活跃请求 (ID: ${Array.from(this.controllers.keys()).join(', ')})` :
                    '无活跃请求';
            }
        }

        // 自定义错误类
        class HTTPError extends Error {
            constructor(status, statusText, response) {
                super(`HTTP ${status}: ${statusText}`);
                this.name = 'HTTPError';
                this.status = status;
                this.statusText = statusText;
                this.response = response;
            }
        }

        class NetworkError extends Error {
            constructor(message, originalError) {
                super(message);
                this.name = 'NetworkError';
                this.originalError = originalError;
            }
        }

        class CancelError extends Error {
            constructor(message) {
                super(message);
                this.name = 'CancelError';
            }
        }

        // 全局API客户端实例
        const apiClient = new APIClient();

        // 界面交互函数
        async function makeRequest() {
            const method = document.getElementById('request-method').value;
            const url = document.getElementById('request-url').value;
            const bodyText = document.getElementById('request-body').value;
            const responseDiv = document.getElementById('basic-response');

            responseDiv.innerHTML = '<span class="loading">请求中...</span>';
            try {
                let result;

                if (method === 'GET') {
                    result = await apiClient.get(url);
                } else if (method === 'POST') {
                    const body = bodyText ? JSON.parse(bodyText) : {};
                    result = await apiClient.post(url, body);
                } else if (method === 'PUT') {
                    const body = bodyText ? JSON.parse(bodyText) : {};
                    result = await apiClient.put(url, body);
                } else if (method === 'DELETE') {
                    result = await apiClient.delete(url);
                }

                responseDiv.innerHTML = `
                    <span class="success">✅ 请求成功</span><br><br>
                    <strong>响应数据:</strong><br>
                    <pre>${JSON.stringify(result, null, 2)}</pre>
                `;

            } catch (error) {
                responseDiv.innerHTML = `
                    <span class="error">❌ 请求失败</span><br><br>
                    <strong>错误类型:</strong> ${error.name}<br>
                    <strong>错误信息:</strong> ${error.message}<br>
                    ${error.status ? `<strong>HTTP状态:</strong> ${error.status}<br>` : ''}
                `;
            }
        }

        async function makeDelayedRequest() {
            const delay = document.getElementById('delay-seconds').value;
            const timeout = document.getElementById('timeout-seconds').value * 1000;
            const responseDiv = document.getElementById('cancel-response');

            responseDiv.innerHTML = '<span class="loading">发送延迟请求...</span>';

            try {
                const result = await apiClient.get(
                    `https://httpbin.org/delay/${delay}`,
                    {},
                    { timeout }
                );
                responseDiv.innerHTML = `
                    <span class="success">✅ 延迟请求完成</span><br><br>
                    延迟: ${delay}秒<br>
                    超时设置: ${timeout/1000}秒<br><br>
                    <pre>${JSON.stringify(result, null, 2)}</pre>
                `;

            } catch (error) {
                responseDiv.innerHTML = `
                    <span class="error">❌ 延迟请求失败</span><br><br>
                    <strong>错误:</strong> ${error.message}<br>
                    ${error.name === 'CancelError' ?
                        '<strong>原因:</strong> 请求被取消' :
                        '<strong>原因:</strong> 可能是超时或网络错误'}
                `;
            }
        }

        async function makeCancellableRequest() {
            const responseDiv = document.getElementById('cancel-response');
            responseDiv.innerHTML = '<span class="loading">发送可取消请求...
</span>';

            try {
                const result = await
apiClient.get('https://httpbin.org/delay/5');
                responseDiv.innerHTML = `
                    <span class="success">✅ 可取消请求完成</span><br><br>
                    <pre>${JSON.stringify(result, null, 2)}</pre>
                `;
            } catch (error) {
                responseDiv.innerHTML = `
                    <span class="error">❌ 请求失败或被取消</span><br><br>
                    <strong>错误:</strong> ${error.message}
                `;
            }
        }

        function cancelAllRequests() {
            apiClient.cancelAll();
            document.getElementById('cancel-response').innerHTML =
                '<span class="error">⏹️ 所有请求已取消</span>';
        }

        // 错误处理测试函数
        async function testSuccessRequest() {
            logToErrorDisplay('测试正常请求...');
            try {
                const result = await
apiClient.get('https://jsonplaceholder.typicode.com/posts/1');
                logToErrorDisplay('✅ 正常请求成功', 'success');
                logToErrorDisplay(JSON.stringify(result, null, 2));
            } catch (error) {
                logToErrorDisplay(`❌ 意外错误: ${error.message}`, 'error');
            }
        }

        async function testNetworkError() {
            logToErrorDisplay('测试网络错误...');
            try {
                await apiClient.get('https://nonexistent-domain-
12345.com/api');
            } catch (error) {
                logToErrorDisplay(`❌ 网络错误捕获成功: ${error.name}`,
'success');
                logToErrorDisplay(`错误信息: ${error.message}`);
            }
        }

        async function testHttpError() {
            logToErrorDisplay('测试HTTP错误...');
            try {
                await
apiClient.get('https://jsonplaceholder.typicode.com/posts/999999');
            } catch (error) {
                logToErrorDisplay(`❌ HTTP错误捕获成功: ${error.name}`,
'success');
                logToErrorDisplay(`状态码: ${error.status}, 信息:
${error.message}`);
            }
        }

        async function testTimeoutError() {
            logToErrorDisplay('测试超时错误...');
            try {
                await apiClient.get('https://httpbin.org/delay/10', {}, {
timeout: 2000 });
            } catch (error) {
                logToErrorDisplay(`❌ 超时错误捕获成功: ${error.name}`,
'success');
                logToErrorDisplay(`错误信息: ${error.message}`);
            }
        }

        async function testJsonError() {
            logToErrorDisplay('测试JSON解析错误...');
            try {
                await apiClient.get('https://httpbin.org/html');
            } catch (error) {
                logToErrorDisplay(`❌ JSON错误捕获成功`, 'success');
                logToErrorDisplay(`错误类型: ${error.name}, 信息:
${error.message}`);
            }
        }

        function logToErrorDisplay(message, type = '') {
            const display = document.getElementById('error-response');
            const timestamp = new Date().toLocaleTimeString();
            const className = type ? ` class="${type}"` : '';
            display.innerHTML += `<div${className}>[${timestamp}] ${message}
</div>`;
            display.scrollTop = display.scrollHeight;
        }

        function clearResponse(type) {
            const displays = {
                'basic': 'basic-response',
                'error': 'error-response'
            };

            if (displays[type]) {
                document.getElementById(displays[type]).innerHTML =
                    '<em>响应将显示在这里...</em>';
            }
        }

        // 页面初始化
        window.addEventListener('load', () => {
            console.log('💡 练习提示: ');
            console.log('1. 尝试不同的HTTP方法和端点');
            console.log('2. 观察请求取消和超时的行为');
            console.log('3. 测试各种错误情况的处理');
            console.log('4. 在Network面板观察实际的网络请求');

            // 设置请求方法改变时的处理
            document.getElementById('request-
method').addEventListener('change', (e) => {
                const bodyTextarea = document.getElementById('request-
body');
                if (e.target.value === 'GET' || e.target.value === 'DELETE')
{
                    bodyTextarea.style.display = 'none';
                } else {
                    bodyTextarea.style.display = 'block';
                }
            });

            // 预填充一些示例数据
            document.getElementById('request-body').value = JSON.stringify({
                title: '我的新文章',
                body: '这是文章内容',
                userId: 1
            }, null, 2);
        });
    </script>
</body>
</html>
```

练习总结与进阶指南

完成所有练习后的检查清单

DOM操作掌握度检查：

- □ 能够使用DocumentFragment优化批量DOM操作
- □ 理解事件委托的原理和优势
- □ 熟练使用现代DOM查询方法
- □ 掌握closest()和matches()方法的使用

存储技术应用检查：

- □ 能根据数据特性选择合适的存储方式
- □ 理解各种存储的生命周期和限制
- □ 掌握错误处理和配额管理
- □ 能够实现数据的序列化和反序列化

网络请求最佳实践检查：

- □ 能够实现完整的错误处理机制
- □ 掌握请求取消和超时控制
- □ 理解不同错误类型的处理方式
- □ 能够设计可复用的API客户端

下一步学习建议

1. **继续完成剩余章节**：按照相同的模式完成性能优化、安全实践、浏览器API等章节的练习
2. **实际项目应用**：将学到的最佳实践应用到真实项目中

3. **性能测试**：使用浏览器开发者工具分析和优化应用性能
4. **安全审计**：检查应用的安全漏洞并实施防护措施

这个改进版教程通过实际可运行的代码示例，让新手能够直观地理解和练习JavaScript最佳实践，每个练习都提供了明确的学习目标和检验方法。