PDF 预览插件 - 实施指南

📋 目录

项目介绍

这是一个功能完善的 PDF 预览插件项目，包含两个版本：

- **Vue3 版本**：使用现代化的 Vue3 框架和 Composition API
- **原生 JS 版本**：纯 JavaScript 实现，展示原生开发能力

核心功能

✅ PDF 文件加载与预览
✅ 翻页控制（上一页/下一页/跳转）
✅ 缩放功能（放大/缩小/重置）
✅ 页面旋转（90 度旋转）
✅ 缩略图导航
✅ 响应式设计
✅ 键盘快捷键

技术准备

需要掌握的技术

基础知识

- HTML5 & CSS3
- JavaScript ES6+（箭头函数、Promise、async/await、模块化）
- DOM 操作和事件处理

Vue3 相关（Vue 版本）

- Composition API（ref、reactive、computed、watch）
- 组件开发
- 生命周期钩子

工具链

- Node.js 和 npm
- Vite（构建工具）
- Git（版本控制）

核心库

- **PDF.js**：Mozilla 开发的 PDF 渲染引擎
- Canvas API：用于 PDF 页面渲染

实施步骤

第一步：环境准备

1. **安装 Node.js**
   - 访问 https://nodejs.org/
   - 下载并安装 LTS 版本（推荐 v18 或 v20）
   - 验证安装：

     ```
     node -v
     npm -v
     ```

2. **安装代码编辑器**
   - 推荐使用 VS Code
   - 安装 Vue 相关插件：Volar、ESLint

第二步：启动 Vue3 版本

```
# 1. 进入Vue3项目目录
cd vue3-version

# 2. 安装依赖
npm install

# 3. 启动开发服务器
npm run dev

# 4. 浏览器会自动打开 http://localhost:3000
```

第三步：启动原生 JS 版本

```
# 1. 进入原生JS项目目录
cd vanilla-js-version

# 2. 安装依赖（主要是PDF.js）
npm install

# 3. 启动开发服务器
npm run dev

# 4. 浏览器打开 http://localhost:3000
```

第四步：测试功能

1. **上传 PDF 文件**
   - 点击"选择 PDF 文件"按钮
   - 选择本地 PDF 文件
2. **使用示例 PDF**
   - 点击"加载示例 PDF"
   - 自动加载在线示例文件
3. **测试所有功能**
   - ✅ 翻页：点击工具栏按钮或使用键盘方向键
   - ✅ 缩放：点击放大/缩小按钮或使用+/-键
   - ✅ 旋转：点击旋转按钮
   - ✅ 跳转：输入页码后按回车
   - ✅ 缩略图：点击缩略图按钮查看

代码详解

1. PDF.js 核心原理与高级配置

1.1 完整的PDF.js配置与初始化

```
import * as pdfjsLib from 'pdfjs-dist';
import pdfjsWorker from 'pdfjs-dist/build/pdf.worker.entry?url'; // Vite环境下的正确导入方式

// 1. 配置Worker（关键步骤）
pdfjsLib.GlobalWorkerOptions.workerSrc = pdfjsWorker;

// 2. 高级配置选项
const pdfOptions = {
  // 源数据（URL、ArrayBuffer或DocumentInitParameters）
  url: pdfUrl,

  // 密码保护PDF的处理
  password: pdfPassword,
```

```
  // 性能优化配置
  cMapUrl: 'https://cdn.jsdelivr.net/npm/pdfjs-dist@3.4.120/cmaps/', // 支持中文和特殊字符
  cMapPacked: true,

  // 加载配置
  enableXfa: true, // 支持XFA表单
  nativeImageDecoderSupport: 'display', // 原生图像解码器支持
  disableFontFace: false, // 是否禁用字体加载

  // 渲染配置
  renderInteractiveForms: true, // 渲染交互式表单
};

// 3. 创建加载任务
const loadingTask = pdfjsLib.getDocument(pdfOptions);

// 4. 进度监控
loadingTask.onProgress = (progressData) => {
  const { loaded, total } = progressData;
  const percent = Math.round((loaded / total) * 100);
  console.log(`加载进度: ${percent}%`);
  // 更新UI进度条
};

// 5. 加载完成获取文档对象
const pdfDoc = await loadingTask.promise;
console.log(`PDF文档已加载，共${pdfDoc.numPages}页`);

// 6. 页面渲染函数
async function renderPage(pageNum, scale = 1.5) {
  // 获取页面
  const page = await pdfDoc.getPage(pageNum);

  // 设置视口
  const viewport = page.getViewport({
    scale: scale,
    rotation: 0 // 可设置旋转角度：0, 90, 180, 270
  });

  // 处理高DPI屏幕
  const outputScale = window.devicePixelRatio || 1;

  // 获取Canvas并设置尺寸
  const canvas = document.querySelector(`#page-${pageNum}`);
  if (!canvas) return;

  const context = canvas.getContext('2d');
  if (!context) return;

  // 设置Canvas实际像素尺寸（考虑设备像素比）
  canvas.width = Math.floor(viewport.width * outputScale);
  canvas.height = Math.floor(viewport.height * outputScale);

  // 设置Canvas显示尺寸
```

```
  canvas.style.width = `${viewport.width}px`;
  canvas.style.height = `${viewport.height}px`;

  // 应用缩放变换
  const transform = [outputScale, 0, 0, outputScale, 0, 0];
  context.setTransform(...transform);

  // 渲染参数
  const renderContext = {
    canvasContext: context,
    viewport: viewport,
    // 优化选项
    enableWebGL: true, // 启用WebGL加速（如果可用）
    renderInteractiveForms: true,
    annotationMode: 2, // 0 = 禁用, 1 = 仅显示, 2 = 可交互
  };

  // 执行渲染
  const renderTask = page.render(renderContext);

  try {
    await renderTask.promise;
    console.log(`页面 ${pageNum} 渲染完成`);
  } catch (error) {
    console.error(`页面 ${pageNum} 渲染失败:`, error);
    // 实现重试逻辑
  }
}

### 2. Vue3 Composition API 核心实现

#### 2.1 完整的Vue3 PDF预览组件实现

```vue
<template>
  <div class="pdf-viewer-container">
    <!-- 工具栏 -->
    <div class="toolbar">
      <button @click="goToPreviousPage" :disabled="currentPage === 1">上一页
</button>
      <span>{{ currentPage }} / {{ totalPages }}</span>
      <button @click="goToNextPage" :disabled="currentPage === totalPages">下一页
</button>
      <input v-model.number="pageInput" @keyup.enter="goToPage" type="number"
min="1" :max="totalPages" />

      <div class="zoom-controls">
        <button @click="zoomOut">-</button>
        <span>{{ zoomPercent }}%</span>
        <button @click="zoomIn">+</button>
        <button @click="resetZoom">重置</button>
      </div>

      <button @click="rotateClockwise">旋转</button>
      <button @click="toggleThumbnails">缩略图</button>
    </div>
```
```

```
    <!-- 加载状态 -->
    <div v-if="loading" class="loading-overlay">
      <div class="loading-spinner"></div>
      <p>{{ loadingProgress }}%</p>
    </div>

    <!-- 错误提示 -->
    <div v-else-if="error" class="error-message">
      <p>{{ error.message }}</p>
      <button @click="loadPdf(props.url)">重试</button>
    </div>

    <!-- PDF 渲染区域 -->
    <div v-else class="pdf-container">
      <canvas ref="canvasRef" :id="`page-${currentPage}`"></canvas>
    </div>

    <!-- 缩略图侧边栏 -->
    <div v-if="showThumbnails" class="thumbnails-sidebar">
      <div
        v-for="pageNum in totalPages"
        :key="pageNum"
        class="thumbnail-item"
        :class="{ active: pageNum === currentPage }"
        @click="goToPage(pageNum)"
      >
        <canvas :ref="el => thumbnailRefs[pageNum] = el"></canvas>
      </div>
    </div>
  </div>
</template>

<script setup>
import { ref, computed, watch, onMounted, onUnmounted, nextTick } from 'vue';
import * as pdfjsLib from 'pdfjs-dist';
import pdfjsWorker from 'pdfjs-dist/build/pdf.worker.entry?url';

// Props
const props = defineProps({
  url: {
    type: String,
    default: ''
  },
  initialScale: {
    type: Number,
    default: 1.5
  }
});

// Emits
const emit = defineEmits(['page-change', 'scale-change', 'error']);

// 配置 Worker
pdfjsLib.GlobalWorkerOptions.workerSrc = pdfjsWorker;
```

```
// 响应式状态
const pdfDoc = ref(null);
const currentPage = ref(1);
const totalPages = ref(0);
const scale = ref(props.initialScale);
const rotation = ref(0);
const loading = ref(false);
const loadingProgress = ref(0);
const error = ref(null);
const showThumbnails = ref(false);
const pageInput = ref(1);

// 模板引用
const canvasRef = ref(null);
const thumbnailRefs = ref({});

// 计算属性
const zoomPercent = computed(() => Math.round(scale.value * 100));

// 页面导航方法
function goToPreviousPage() {
  if (currentPage.value > 1) {
    currentPage.value--;
    pageInput.value = currentPage.value;
    renderCurrentPage();
    emit('page-change', currentPage.value);
  }
}

function goToNextPage() {
  if (currentPage.value < totalPages.value) {
    currentPage.value++;
    pageInput.value = currentPage.value;
    renderCurrentPage();
    emit('page-change', currentPage.value);
  }
}

function goToPage(pageNum) {
  const page = parseInt(pageNum) || 1;
  if (page >= 1 && page <= totalPages.value) {
    currentPage.value = page;
    pageInput.value = page;
    renderCurrentPage();
    emit('page-change', page);
  }
}

// 缩放控制
function zoomIn() {
  scale.value = Math.min(scale.value + 0.1, 5);
  renderCurrentPage();
  emit('scale-change', scale.value);
}

function zoomOut() {
```

```
  scale.value = Math.max(scale.value - 0.1, 0.1);
  renderCurrentPage();
  emit('scale-change', scale.value);
}

function resetZoom() {
  scale.value = props.initialScale;
  renderCurrentPage();
  emit('scale-change', scale.value);
}

// 旋转控制
function rotateClockwise() {
  rotation.value = (rotation.value + 90) % 360;
  renderCurrentPage();
}

// 切换缩略图
function toggleThumbnails() {
  showThumbnails.value = !showThumbnails.value;
  if (showThumbnails.value && pdfDoc.value) {
    renderThumbnails();
  }
}

// 加载PDF文档
async function loadPdf(url) {
  if (!url) return;

  loading.value = true;
  error.value = null;
  loadingProgress.value = 0;

  try {
    const loadingTask = pdfjsLib.getDocument({
      url: url,
      cMapUrl: 'https://cdn.jsdelivr.net/npm/pdfjs-dist@3.4.120/cmaps/',
      cMapPacked: true
    });

    // 进度监控
    loadingTask.onProgress = (progressData) => {
      const { loaded, total } = progressData;
      loadingProgress.value = Math.round((loaded / total) * 100);
    };

    pdfDoc.value = await loadingTask.promise;
    totalPages.value = pdfDoc.value.numPages;
    currentPage.value = 1;
    pageInput.value = 1;

    // 渲染第一页
    await renderCurrentPage();

    // 如果显示缩略图，则渲染缩略图
    if (showThumbnails.value) {
```

```
      renderThumbnails();
    }
  } catch (err) {
    error.value = {
      message: `PDF加载失败: ${err.message || '未知错误'}`
    };
    emit('error', err);
    console.error('PDF加载错误:', err);
  } finally {
    loading.value = false;
  }
}

// 渲染当前页面
async function renderCurrentPage() {
  if (!pdfDoc.value || !canvasRef.value) return;

  try {
    const page = await pdfDoc.value.getPage(currentPage.value);
    const viewport = page.getViewport({
      scale: scale.value,
      rotation: rotation.value
    });

    const canvas = canvasRef.value;
    const context = canvas.getContext('2d');

    // 处理高DPI屏幕
    const outputScale = window.devicePixelRatio || 1;
    canvas.width = Math.floor(viewport.width * outputScale);
    canvas.height = Math.floor(viewport.height * outputScale);
    canvas.style.width = `${viewport.width}px`;
    canvas.style.height = `${viewport.height}px`;

    const transform = [outputScale, 0, 0, outputScale, 0, 0];
    context.setTransform(...transform);

    await page.render({
      canvasContext: context,
      viewport: viewport
    }).promise;

  } catch (err) {
    console.error('页面渲染错误:', err);
    error.value = {
      message: `页面渲染失败: ${err.message}`
    };
  }
}

// 渲染缩略图
async function renderThumbnails() {
  if (!pdfDoc.value) return;

  // 只渲染可见区域的缩略图或预加载附近几页
  const renderCount = Math.min(20, totalPages.value); // 限制一次性渲染数量
```

```
  for (let i = 1; i <= renderCount; i++) {
    await renderSingleThumbnail(i);
  }
}

async function renderSingleThumbnail(pageNum) {
  if (!pdfDoc.value || !thumbnailRefs.value[pageNum]) return;

  try {
    const page = await pdfDoc.value.getPage(pageNum);
    const viewport = page.getViewport({ scale: 0.2 }); // 缩略图使用小比例

    const canvas = thumbnailRefs.value[pageNum];
    const context = canvas.getContext('2d');

    canvas.width = viewport.width;
    canvas.height = viewport.height;

    await page.render({
      canvasContext: context,
      viewport: viewport
    }).promise;

  } catch (err) {
    console.error(`缩略图${pageNum}渲染错误:`, err);
  }
}

// 键盘事件处理
function handleKeydown(event) {
  switch (event.key) {
    case 'ArrowLeft':
      goToPreviousPage();
      break;
    case 'ArrowRight':
      goToNextPage();
      break;
    case '+':
      zoomIn();
      break;
    case '-':
      zoomOut();
      break;
    case '0':
      resetZoom();
      break;
    case 'Escape':
      if (showThumbnails.value) {
        showThumbnails.value = false;
      }
      break;
  }
}

// 窗口大小变化处理
```

```
function handleResize() {
  if (pdfDoc.value) {
    renderCurrentPage();
  }
}

// 监听URL变化
watch(
  () => props.url,
  (newUrl, oldUrl) => {
    if (newUrl && newUrl !== oldUrl) {
      loadPdf(newUrl);
    }
  },
  { immediate: true }
);

// 生命周期钩子
onMounted(() => {
  window.addEventListener('keydown', handleKeydown);
  window.addEventListener('resize', handleResize);
});

onUnmounted(() => {
  // 清理资源
  window.removeEventListener('keydown', handleKeydown);
  window.removeEventListener('resize', handleResize);

  // 释放PDF文档资源
  if (pdfDoc.value) {
    pdfDoc.value.destroy();
  }

  // 清空引用
  pdfDoc.value = null;
  canvasRef.value = null;
  thumbnailRefs.value = {};
});
</script>

<style scoped>
.pdf-viewer-container {
  position: relative;
  width: 100%;
  height: 100vh;
  display: flex;
  flex-direction: column;
  overflow: hidden;
}

.toolbar {
  display: flex;
  align-items: center;
  gap: 10px;
  padding: 10px;
  background: #f5f5f5;
```

```
  border-bottom: 1px solid #ddd;
}

.zoom-controls {
  display: flex;
  align-items: center;
  gap: 5px;
  margin-left: auto;
}

.pdf-container {
  flex: 1;
  overflow: auto;
  display: flex;
  justify-content: center;
  align-items: flex-start;
  padding: 20px;
  background: #f9f9f9;
}

canvas {
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
  background: white;
}

.loading-overlay, .error-message {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: rgba(255, 255, 255, 0.9);
  z-index: 100;
}

.thumbnails-sidebar {
  position: absolute;
  left: 0;
  top: 60px;
  bottom: 0;
  width: 200px;
  background: #f0f0f0;
  overflow-y: auto;
  padding: 10px;
  box-shadow: 2px 0 5px rgba(0,0,0,0.1);
}

.thumbnail-item {
  margin-bottom: 10px;
  cursor: pointer;
  border: 2px solid transparent;
}
```

```
.thumbnail-item.active {
  border-color: #007bff;
}

.thumbnail-item canvas {
  width: 100%;
  height: auto;
}

@media (max-width: 768px) {
  .toolbar {
    flex-wrap: wrap;
  }

  .zoom-controls {
    margin-left: 0;
    margin-top: 5px;
  }

  .thumbnails-sidebar {
    width: 150px;
  }
}
</style>

### 3. 原生 JS 类封装实现

#### 3.1 完整的PDF预览器类封装

```javascript
/**
 * PDF预览器核心类 - 原生JavaScript实现
 * 提供完整的PDF文档加载、渲染、交互和性能优化功能
 */
class PDFViewer {
  /**
   * 构造函数
   * @param {Object} options - 配置选项
   * @param {string} options.containerId - 容器元素ID
   * @param {string} options.url - PDF文件URL
   * @param {number} options.initialScale - 初始缩放比例，默认为1.5
   * @param {Object} options.workerOptions - Worker配置选项
   * @param {Object} options.callbacks - 回调函数集合
   */
  constructor(options = {}) {
    // 合并默认选项
    this.options = {
      containerId: 'pdf-container',
      url: '',
      initialScale: 1.5,
      workerOptions: {
        workerSrc: 'https://cdn.jsdelivr.net/npm/pdfjs-
dist@3.4.120/build/pdf.worker.min.js',
        cMapUrl: 'https://cdn.jsdelivr.net/npm/pdfjs-dist@3.4.120/cmaps/',
        cMapPacked: true
```
```

```
      },
      callbacks: {
        onPageChange: () => {},
        onScaleChange: () => {},
        onLoadStart: () => {},
        onLoadProgress: () => {},
        onLoadComplete: () => {},
        onError: () => {}
      },
      ...options
    };

    // 初始化PDF.js
    this._initPDFJS();

    // 初始化状态
    this._initState();

    // 创建DOM结构
    this._createDOM();

    // 绑定事件
    this._bindEvents();

    // 如果提供了URL，则加载PDF
    if (this.options.url) {
      this.loadPDF(this.options.url);
    }
  }

  /**
   * 初始化PDF.js库
   * @private
   */
  _initPDFJS() {
    // 导入PDF.js库
    if (!window.pdfjsLib) {
      throw new Error('PDF.js library is not loaded. Please include pdf.js before
initializing the PDFViewer.');
    }

    this.pdfjsLib = window.pdfjsLib;

    // 配置Worker
    this.pdfjsLib.GlobalWorkerOptions.workerSrc =
this.options.workerOptions.workerSrc;
  }

  /**
   * 初始化状态
   * @private
   */
  _initState() {
    this.pdfDoc = null;
    this.currentPage = 1;
    this.totalPages = 0;
```

```
    this.scale = this.options.initialScale;
    this.rotation = 0;
    this.loading = false;
    this.loadingProgress = 0;
    this.error = null;
    this.showThumbnails = false;

    // 缓存管理
    this.pageCache = new Map(); // 页面缓存
    this.thumbnailCache = new Map(); // 缩略图缓存
    this.maxCacheSize = 5; // 最大缓存页数

    // 性能优化
    this._isRendering = false;
    this._pendingRender = null;
    this._devicePixelRatio = window.devicePixelRatio || 1;
  }

  /**
   * 创建DOM结构
   * @private
   */
  _createDOM() {
    const container = document.getElementById(this.options.containerId);
    if (!container) {
      throw new Error(`Container element with id "${this.options.containerId}"
not found.`);
    }

    // 清除容器内容
    container.innerHTML = '';

    // 创建主容器
    this.viewerContainer = document.createElement('div');
    this.viewerContainer.className = 'pdf-viewer-container';

    // 创建工具栏
    this._createToolbar();

    // 创建加载覆盖层
    this._createLoadingOverlay();

    // 创建错误提示
    this._createErrorMessage();

    // 创建PDF渲染区域
    this._createPDFContainer();

    // 创建缩略图侧边栏
    this._createThumbnailSidebar();

    // 将所有元素添加到容器
    container.appendChild(this.viewerContainer);
  }

  /**
```

```
   * 创建工具栏
   * @private
   */
  _createToolbar() {
    const toolbar = document.createElement('div');
    toolbar.className = 'toolbar';

    // 导航按钮
    toolbar.innerHTML = `
      <button id="prev-page" disabled>上一页</button>
      <span id="page-info">1 / 0</span>
      <button id="next-page" disabled>下一页</button>
      <input id="page-input" type="number" min="1" value="1" style="width:
60px;">

      <div class="zoom-controls">
        <button id="zoom-out">-</button>
        <span id="zoom-percent">${Math.round(this.scale * 100)}%</span>
        <button id="zoom-in">+</button>
        <button id="reset-zoom">重置</button>
      </div>

      <button id="rotate">旋转</button>
      <button id="toggle-thumbnails">缩略图</button>
    `;

    // 保存工具栏元素引用
    this.prevPageBtn = toolbar.querySelector('#prev-page');
    this.nextPageBtn = toolbar.querySelector('#next-page');
    this.pageInfoEl = toolbar.querySelector('#page-info');
    this.pageInputEl = toolbar.querySelector('#page-input');
    this.zoomOutBtn = toolbar.querySelector('#zoom-out');
    this.zoomInBtn = toolbar.querySelector('#zoom-in');
    this.resetZoomBtn = toolbar.querySelector('#reset-zoom');
    this.zoomPercentEl = toolbar.querySelector('#zoom-percent');
    this.rotateBtn = toolbar.querySelector('#rotate');
    this.toggleThumbnailsBtn = toolbar.querySelector('#toggle-thumbnails');

    this.viewerContainer.appendChild(toolbar);
  }

  /**
   * 创建加载覆盖层
   * @private
   */
  _createLoadingOverlay() {
    const overlay = document.createElement('div');
    overlay.id = 'loading-overlay';
    overlay.className = 'loading-overlay';
    overlay.style.display = 'none';

    overlay.innerHTML = `
      <div class="loading-spinner"></div>
      <p id="loading-progress">0%</p>
    `;
```

```
    this.loadingProgressEl = overlay.querySelector('#loading-progress');
    this.loadingOverlay = overlay;

    this.viewerContainer.appendChild(overlay);
  }

  /**
   * 创建错误提示
   * @private
   */
  _createErrorMessage() {
    const errorEl = document.createElement('div');
    errorEl.id = 'error-message';
    errorEl.className = 'error-message';
    errorEl.style.display = 'none';

    errorEl.innerHTML = `
      <p id="error-text">加载失败</p>
      <button id="retry-btn">重试</button>
    `;

    this.errorTextEl = errorEl.querySelector('#error-text');
    this.retryBtn = errorEl.querySelector('#retry-btn');
    this.errorMessageEl = errorEl;

    this.viewerContainer.appendChild(errorEl);
  }

  /**
   * 创建PDF渲染区域
   * @private
   */
  _createPDFContainer() {
    const container = document.createElement('div');
    container.className = 'pdf-container';

    const canvas = document.createElement('canvas');
    canvas.id = 'pdf-canvas';

    container.appendChild(canvas);

    this.pdfContainer = container;
    this.canvas = canvas;

    this.viewerContainer.appendChild(container);
  }

  /**
   * 创建缩略图侧边栏
   * @private
   */
  _createThumbnailSidebar() {
    const sidebar = document.createElement('div');
    sidebar.id = 'thumbnails-sidebar';
    sidebar.className = 'thumbnails-sidebar';
    sidebar.style.display = 'none';
```

```
    this.thumbnailsContainer = sidebar;

    this.viewerContainer.appendChild(sidebar);
  }

  /**
   * 绑定事件
   * @private
   */
  _bindEvents() {
    // 工具栏事件
    this.prevPageBtn.addEventListener('click', () => this.goToPreviousPage());
    this.nextPageBtn.addEventListener('click', () => this.goToNextPage());
    this.pageInputEl.addEventListener('change', () =>
this.goToPage(this.pageInputEl.value));
    this.pageInputEl.addEventListener('keyup', (e) => {
      if (e.key === 'Enter') {
        this.goToPage(this.pageInputEl.value);
      }
    });

    this.zoomInBtn.addEventListener('click', () => this.zoomIn());
    this.zoomOutBtn.addEventListener('click', () => this.zoomOut());
    this.resetZoomBtn.addEventListener('click', () => this.resetZoom());

    this.rotateBtn.addEventListener('click', () => this.rotateClockwise());
    this.toggleThumbnailsBtn.addEventListener('click', () =>
this.toggleThumbnails());

    // 重试按钮
    this.retryBtn.addEventListener('click', () =>
this.loadPDF(this.options.url));

    // 全局事件
    window.addEventListener('keydown', (e) => this._handleKeydown(e));
    window.addEventListener('resize', () => this._handleResize());
  }

  /**
   * 加载PDF文档
   * @param {string} url - PDF文件URL
   * @returns {Promise<void>}
   */
  async loadPDF(url) {
    if (!url) {
      throw new Error('PDF URL must be provided');
    }

    // 更新URL
    this.options.url = url;

    // 重置状态
    this._showLoading();
    this._hideError();
    this.pdfDoc = null;
```

```
    this.totalPages = 0;
    this.currentPage = 1;
    this._updatePageInfo();

    // 清空缓存
    this._clearCache();

    // 触发回调
    this.options.callbacks.onLoadStart();

    try {
      // 创建加载任务
      const loadingTask = this.pdfjsLib.getDocument({
        url: url,
        cMapUrl: this.options.workerOptions.cMapUrl,
        cMapPacked: this.options.workerOptions.cMapPacked,
        enableXfa: true, // 支持XFA表单
        enableScripting: true // 启用脚本支持
      });

      // 进度监控
      loadingTask.onProgress = (progressData) => {
        const { loaded, total } = progressData;
        this.loadingProgress = Math.round((loaded / total) * 100);
        this.loadingProgressEl.textContent = `${this.loadingProgress}%`;
        this.options.callbacks.onLoadProgress(this.loadingProgress);
      };

      // 加载PDF文档
      this.pdfDoc = await loadingTask.promise;
      this.totalPages = this.pdfDoc.numPages;

      // 更新页面信息
      this._updatePageInfo();
      this.pageInputEl.max = this.totalPages;

      // 渲染第一页
      await this.renderCurrentPage();

      // 触发回调
      this.options.callbacks.onLoadComplete(this.totalPages);

      // 如果显示缩略图，则渲染缩略图
      if (this.showThumbnails) {
        this.renderThumbnails();
      }
    } catch (err) {
      console.error('PDF加载错误:', err);
      this.error = {
        message: `PDF加载失败: ${err.message || '未知错误'}`
      };

      // 显示错误
      this._showError();

      // 触发错误回调
```

```
      this.options.callbacks.onError(err);
    } finally {
      this.loading = false;
      this._hideLoading();
    }
  }

  /**
   * 渲染当前页面
   * @returns {Promise<void>}
   */
  async renderCurrentPage() {
    if (!this.pdfDoc || !this.canvas) return;

    // 防抖渲染请求
    if (this._isRendering) {
      if (this._pendingRender) {
        clearTimeout(this._pendingRender);
      }

      this._pendingRender = setTimeout(() => {
        this.renderCurrentPage();
      }, 50);

      return;
    }

    this._isRendering = true;

    try {
      // 检查缓存
      const cachedPage =
this.pageCache.get(`${this.currentPage}-${this.scale}-${this.rotation}`);
      if (cachedPage) {
        this._displayFromCache(cachedPage);
        return;
      }

      // 获取页面
      const page = await this.pdfDoc.getPage(this.currentPage);

      // 获取视口
      const viewport = page.getViewport({
        scale: this.scale,
        rotation: this.rotation
      });

      // 设置Canvas尺寸（处理高DPI）
      const outputScale = this._devicePixelRatio;
      this.canvas.width = Math.floor(viewport.width * outputScale);
      this.canvas.height = Math.floor(viewport.height * outputScale);
      this.canvas.style.width = `${viewport.width}px`;
      this.canvas.style.height = `${viewport.height}px`;

      // 获取上下文
      const context = this.canvas.getContext('2d');
```

```
      context.setTransform(outputScale, 0, 0, outputScale, 0, 0);

      // 渲染页面
      await page.render({
        canvasContext: context,
        viewport: viewport,
        // 性能优化配置
        intent: 'display',
        renderInteractiveForms: true
      }).promise;

      // 缓存渲染结果
      this._cachePage(page, viewport);
    } catch (err) {
      console.error('页面渲染错误:', err);
      this.error = {
        message: `页面渲染失败: ${err.message}`
      };
      this._showError();
    } finally {
      this._isRendering = false;
    }
  }

  /**
   * 缓存页面渲染结果
   * @param {Object} page - PDF页面对象
   * @param {Object} viewport - 视口对象
   * @private
   */
  _cachePage(page, viewport) {
    // 实现简单的LRU缓存策略
    if (this.pageCache.size >= this.maxCacheSize) {
      // 删除最早缓存的页面
      const firstKey = this.pageCache.keys().next().value;
      this.pageCache.delete(firstKey);
    }

    // 获取Canvas数据作为缓存
    const cacheKey = `${this.currentPage}-${this.scale}-${this.rotation}`;
    const imageData = this.canvas.toDataURL('image/png');

    this.pageCache.set(cacheKey, {
      pageNumber: this.currentPage,
      scale: this.scale,
      rotation: this.rotation,
      viewport: viewport,
      imageData: imageData,
      timestamp: Date.now()
    });
  }

  /**
   * 从缓存显示页面
   * @param {Object} cachedPage - 缓存的页面数据
   * @private
```

```
   */
  _displayFromCache(cachedPage) {
    const context = this.canvas.getContext('2d');
    const image = new Image();

    image.onload = () => {
      // 设置Canvas尺寸
      this.canvas.width = Math.floor(cachedPage.viewport.width *
this._devicePixelRatio);
      this.canvas.height = Math.floor(cachedPage.viewport.height *
this._devicePixelRatio);
      this.canvas.style.width = `${cachedPage.viewport.width}px`;
      this.canvas.style.height = `${cachedPage.viewport.height}px`;

      // 绘制缓存图像
      const outputScale = this._devicePixelRatio;
      context.setTransform(outputScale, 0, 0, outputScale, 0, 0);
      context.drawImage(image, 0, 0, cachedPage.viewport.width,
cachedPage.viewport.height);
    };

    image.src = cachedPage.imageData;
  }

  /**
   * 渲染缩略图
   */
  renderThumbnails() {
    if (!this.pdfDoc || !this.thumbnailsContainer) return;

    // 清空缩略图容器
    this.thumbnailsContainer.innerHTML = '';

    // 只渲染部分缩略图以提高性能
    const renderCount = Math.min(50, this.totalPages);

    for (let i = 1; i <= renderCount; i++) {
      this._createThumbnailItem(i);
    }

    // 对于页数较多的文档，添加懒加载
    if (this.totalPages > renderCount) {
      this.thumbnailsContainer.addEventListener('scroll', () =>
this._lazyLoadThumbnails());
    }
  }

  /**
   * 创建单个缩略图项
   * @param {number} pageNum - 页码
   * @private
   */
  async _createThumbnailItem(pageNum) {
    const thumbnailItem = document.createElement('div');
    thumbnailItem.className = `thumbnail-item ${pageNum === this.currentPage ?
'active' : ''}`;
```

```
    thumbnailItem.dataset.page = pageNum;

    const canvas = document.createElement('canvas');
    canvas.className = 'thumbnail-canvas';
    canvas.dataset.page = pageNum;

    thumbnailItem.appendChild(canvas);
    this.thumbnailsContainer.appendChild(thumbnailItem);

    // 点击缩略图跳转到对应页面
    thumbnailItem.addEventListener('click', () => this.goToPage(pageNum));

    // 渲染缩略图
    await this._renderThumbnail(pageNum, canvas);
  }

  /**
   * 渲染单个缩略图
   * @param {number} pageNum - 页码
   * @param {HTMLCanvasElement} canvas - Canvas元素
   * @private
   */
  async _renderThumbnail(pageNum, canvas) {
    // 检查缓存
    const cachedThumbnail = this.thumbnailCache.get(pageNum);
    if (cachedThumbnail && canvas.isConnected) {
      const context = canvas.getContext('2d');
      const image = new Image();

      image.onload = () => {
        if (canvas.isConnected) {
          canvas.width = cachedThumbnail.width;
          canvas.height = cachedThumbnail.height;
          context.drawImage(image, 0, 0);
        }
      };

      image.src = cachedThumbnail.imageData;
      return;
    }

    try {
      const page = await this.pdfDoc.getPage(pageNum);
      const viewport = page.getViewport({ scale: 0.15 });

      if (!canvas.isConnected) return; // 检查Canvas是否仍然在DOM中

      canvas.width = viewport.width;
      canvas.height = viewport.height;

      const context = canvas.getContext('2d');

      await page.render({
        canvasContext: context,
        viewport: viewport
      }).promise;
```

```
      // 缓存缩略图
      if (canvas.isConnected) {
        this.thumbnailCache.set(pageNum, {
          width: viewport.width,
          height: viewport.height,
          imageData: canvas.toDataURL('image/png')
        });
      }
    } catch (err) {
      console.error(`缩略图${pageNum}渲染错误:`, err);
    }
  }

  /**
   * 懒加载缩略图
   * @private
   */
  _lazyLoadThumbnails() {
    // 实现缩略图懒加载逻辑
    const container = this.thumbnailsContainer;
    const scrollTop = container.scrollTop;
    const containerHeight = container.clientHeight;

    // 获取当前可见区域内尚未渲染的缩略图
    const unrenderedItems = container.querySelectorAll('.thumbnail-
item:not([data-rendered])');

    unrenderedItems.forEach(item => {
      const rect = item.getBoundingClientRect();
      const containerRect = container.getBoundingClientRect();

      if (rect.top <= containerRect.bottom + 100 && rect.bottom >=
containerRect.top - 100) {
        const pageNum = parseInt(item.dataset.page);
        const canvas = item.querySelector('canvas');

        if (canvas && !item.dataset.rendered) {
          item.dataset.rendered = 'true';
          this._renderThumbnail(pageNum, canvas);
        }
      }
    });
  }

  /**
   * 上一页
   */
  goToPreviousPage() {
    if (this.currentPage > 1) {
      this.currentPage--;
      this._updatePageInfo();
      this.renderCurrentPage();
      this._updateThumbnailActiveState();
      this.options.callbacks.onPageChange(this.currentPage);
    }
```

```
}

/**
 * 下一页
 */
goToNextPage() {
  if (this.currentPage < this.totalPages) {
    this.currentPage++;
    this._updatePageInfo();
    this.renderCurrentPage();
    this._updateThumbnailActiveState();
    this.options.callbacks.onPageChange(this.currentPage);
  }
}

/**
 * 跳转到指定页面
 * @param {number} pageNum - 页码
 */
goToPage(pageNum) {
  const page = parseInt(pageNum) || 1;

  if (page >= 1 && page <= this.totalPages) {
    this.currentPage = page;
    this._updatePageInfo();
    this.renderCurrentPage();
    this._updateThumbnailActiveState();
    this.options.callbacks.onPageChange(this.currentPage);
  }
}

/**
 * 放大
 */
zoomIn() {
  this.scale = Math.min(this.scale + 0.1, 5);
  this._updateZoomInfo();
  this.renderCurrentPage();
  this.options.callbacks.onScaleChange(this.scale);
}

/**
 * 缩小
 */
zoomOut() {
  this.scale = Math.max(this.scale - 0.1, 0.1);
  this._updateZoomInfo();
  this.renderCurrentPage();
  this.options.callbacks.onScaleChange(this.scale);
}

/**
 * 重置缩放
 */
resetZoom() {
  this.scale = this.options.initialScale;
```

```
    this._updateZoomInfo();
    this.renderCurrentPage();
    this.options.callbacks.onScaleChange(this.scale);
  }

  /**
   * 顺时针旋转
   */
  rotateClockwise() {
    this.rotation = (this.rotation + 90) % 360;
    this.renderCurrentPage();
  }

  /**
   * 切换缩略图显示
   */
  toggleThumbnails() {
    this.showThumbnails = !this.showThumbnails;

    if (this.showThumbnails) {
      this.thumbnailsContainer.style.display = 'block';
      if (this.pdfDoc) {
        this.renderThumbnails();
      }
    } else {
      this.thumbnailsContainer.style.display = 'none';
    }
  }

  /**
   * 更新页面信息
   * @private
   */
  _updatePageInfo() {
    this.pageInfoEl.textContent = `${this.currentPage} / ${this.totalPages}`;
    this.pageInputEl.value = this.currentPage;

    // 更新按钮状态
    this.prevPageBtn.disabled = this.currentPage === 1;
    this.nextPageBtn.disabled = this.currentPage === this.totalPages;
  }

  /**
   * 更新缩放信息
   * @private
   */
  _updateZoomInfo() {
    this.zoomPercentEl.textContent = `${Math.round(this.scale * 100)}%`;
  }

  /**
   * 更新缩略图激活状态
   * @private
   */
  _updateThumbnailActiveState() {
```

```
    const activeItem = this.thumbnailsContainer.querySelector('.thumbnail-
item.active');
    if (activeItem) {
      activeItem.classList.remove('active');
    }

    const newActiveItem = this.thumbnailsContainer.querySelector(`.thumbnail-
item[data-page="${this.currentPage}"]`);
    if (newActiveItem) {
      newActiveItem.classList.add('active');
      // 滚动到可见区域
      newActiveItem.scrollIntoView({ behavior: 'smooth', block: 'center' });
    }
  }

  /**
   * 显示加载状态
   * @private
   */
  _showLoading() {
    this.loadingOverlay.style.display = 'flex';
  }

  /**
   * 隐藏加载状态
   * @private
   */
  _hideLoading() {
    this.loadingOverlay.style.display = 'none';
  }

  /**
   * 显示错误信息
   * @private
   */
  _showError() {
    this.errorTextEl.textContent = this.error?.message || '加载失败';
    this.errorMessageEl.style.display = 'flex';
  }

  /**
   * 隐藏错误信息
   * @private
   */
  _hideError() {
    this.errorMessageEl.style.display = 'none';
  }

  /**
   * 键盘事件处理
   * @param {KeyboardEvent} event - 键盘事件
   * @private
   */
  _handleKeydown(event) {
    // 只有当PDF容器有焦点或者没有输入元素被激活时处理键盘事件
    if (document.activeElement.tagName === 'INPUT' ||
```

```
      document.activeElement.tagName === 'TEXTAREA' ||
      document.activeElement.contentEditable === 'true') {
    return;
  }

  switch (event.key) {
    case 'ArrowLeft':
      this.goToPreviousPage();
      break;
    case 'ArrowRight':
      this.goToNextPage();
      break;
    case '+':
    case '=':
      this.zoomIn();
      break;
    case '-':
    case '_':
      this.zoomOut();
      break;
    case '0':
      this.resetZoom();
      break;
    case 'Escape':
      if (this.showThumbnails) {
        this.toggleThumbnails();
      }
      break;
  }
}

/**
 * 窗口大小变化处理
 * @private
 */
_handleResize() {
  if (this.pdfDoc && !this.loading) {
    this.renderCurrentPage();
  }

  // 更新设备像素比
  this._devicePixelRatio = window.devicePixelRatio || 1;
}

/**
 * 清空缓存
 * @private
 */
_clearCache() {
  this.pageCache.clear();
  this.thumbnailCache.clear();
}

/**
 * 销毁PDF查看器实例
 */
```

```
  destroy() {
    // 清理事件监听器
    window.removeEventListener('keydown', (e) => this._handleKeydown(e));
    window.removeEventListener('resize', () => this._handleResize());

    // 释放PDF文档资源
    if (this.pdfDoc) {
      this.pdfDoc.destroy();
    }

    // 清空缓存
    this._clearCache();

    // 清空DOM引用
    if (this.viewerContainer && this.viewerContainer.parentNode) {
      this.viewerContainer.parentNode.removeChild(this.viewerContainer);
    }

    // 重置状态
    this._initState();
  }
}

/**
 * 使用示例
 *
 * // 创建PDF查看器实例
 * const pdfViewer = new PDFViewer({
 *   containerId: 'pdf-container',
 *   url: 'example.pdf',
 *   initialScale: 1.2,
 *   workerOptions: {
 *     workerSrc: '/path/to/pdf.worker.js'
 *   },
 *   callbacks: {
 *     onPageChange: (pageNum) => console.log('当前页码:', pageNum),
 *     onLoadProgress: (progress) => console.log('加载进度:', progress),
 *     onError: (error) => console.error('发生错误:', error)
 *   }
 * });
 *
 * // 手动加载PDF
 * // pdfViewer.loadPDF('another-document.pdf');
 *
 * // 页面导航
 * // pdfViewer.goToPage(5);
 *
 * // 销毁实例
 * // window.addEventListener('beforeunload', () => {
 * //   pdfViewer.destroy();
 * // });
 */
```

3.2 原生JS版本的样式实现

```
/* PDF查看器样式 */
.pdf-viewer-container {
  position: relative;
  width: 100%;
  height: 100vh;
  display: flex;
  flex-direction: column;
  overflow: hidden;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
Ubuntu, Cantarell, sans-serif;
}

/* 工具栏样式 */
.toolbar {
  display: flex;
  align-items: center;
  gap: 10px;
  padding: 10px 20px;
  background: #f5f5f5;
  border-bottom: 1px solid #ddd;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.toolbar button {
  padding: 6px 12px;
  border: 1px solid #ccc;
  background: white;
  border-radius: 4px;
  cursor: pointer;
  font-size: 14px;
  transition: all 0.2s ease;
}

.toolbar button:hover:not(:disabled) {
  background: #e9e9e9;
  border-color: #adadad;
}

.toolbar button:disabled {
  opacity: 0.5;
  cursor: not-allowed;
}

.page-controls {
  display: flex;
  align-items: center;
  gap: 8px;
}

#page-info {
  font-size: 14px;
  min-width: 80px;
  text-align: center;
```

```
}

#page-input {
  width: 60px;
  padding: 6px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
  text-align: center;
}

/* 缩放控制样式 */
.zoom-controls {
  display: flex;
  align-items: center;
  gap: 5px;
  margin-left: auto;
}

#zoom-percent {
  font-size: 14px;
  min-width: 45px;
  text-align: center;
}

/* PDF容器样式 */
.pdf-container {
  flex: 1;
  overflow: auto;
  display: flex;
  justify-content: center;
  align-items: flex-start;
  padding: 20px;
  background: #f9f9f9;
  position: relative;
}

#pdf-canvas {
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  background: white;
  transition: transform 0.3s ease;
}

/* 加载覆盖层样式 */
.loading-overlay {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: rgba(255, 255, 255, 0.9);
  z-index: 100;
```

```
}

.loading-spinner {
  width: 50px;
  height: 50px;
  border: 5px solid #f3f3f3;
  border-top: 5px solid #3498db;
  border-radius: 50%;
  animation: spin 1s linear infinite;
  margin-bottom: 15px;
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

/* 错误消息样式 */
.error-message {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: rgba(255, 255, 255, 0.95);
  z-index: 100;
  color: #dc3545;
}

.error-message p {
  margin-bottom: 15px;
  font-size: 16px;
}

.error-message button {
  background: #dc3545;
  color: white;
  border: none;
  padding: 8px 16px;
  border-radius: 4px;
  cursor: pointer;
}

.error-message button:hover {
  background: #c82333;
}

/* 缩略图侧边栏样式 */
.thumbnails-sidebar {
  position: absolute;
  left: 0;
  top: 60px;
```

```
  bottom: 0;
  width: 200px;
  background: #f0f0f0;
  overflow-y: auto;
  padding: 10px;
  box-shadow: 2px 0 5px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s ease;
  z-index: 50;
}

.thumbnail-item {
  margin-bottom: 10px;
  cursor: pointer;
  border: 2px solid transparent;
  border-radius: 4px;
  transition: all 0.2s ease;
  overflow: hidden;
}

.thumbnail-item:hover {
  transform: scale(1.02);
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.15);
}

.thumbnail-item.active {
  border-color: #007bff;
  background: rgba(0, 123, 255, 0.1);
}

.thumbnail-canvas {
  width: 100%;
  height: auto;
  display: block;
}

/* 响应式设计 */
@media (max-width: 768px) {
  .toolbar {
    flex-wrap: wrap;
    padding: 8px 15px;
  }

  .zoom-controls {
    margin-left: 0;
    margin-top: 8px;
  }

  .thumbnails-sidebar {
    width: 150px;
  }

  .pdf-container {
    padding: 10px;
  }
}
```

```
@media (max-width: 480px) {
  .toolbar {
    gap: 5px;
  }

  .toolbar button {
    padding: 5px 8px;
    font-size: 12px;
  }

  #page-info {
    min-width: 60px;
    font-size: 12px;
  }

  .thumbnails-sidebar {
    width: 100px;
  }
}
```

3.3 使用原生JS版本的完整示例

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>PDF预览器 - 原生JS版本</title>
  <!-- 引入PDF.js库 -->
  <script src="https://cdn.jsdelivr.net/npm/pdfjs-dist@3.4.120/build/pdf.min.js">
</script>
  <!-- 引入PDF查看器样式 -->
  <style>
    /* 这里放置上面的PDF查看器样式代码 */
    /* 为了示例简洁，实际使用时应该单独引入CSS文件 */
    body, html {
      margin: 0;
      padding: 0;
      width: 100%;
      height: 100%;
      overflow: hidden;
    }

    #app {
      width: 100%;
      height: 100%;
    }

    /* 其他样式... */
  </style>
</head>
<body>
  <div id="app">
    <div id="pdf-container"></div>
  </div>
```

```
  <!-- 引入PDF查看器类 -->
  <script>
    // 这里放置PDFViewer类的完整代码
    // 为了示例简洁，实际使用时应该单独引入JS文件

    // 初始化PDF查看器
    document.addEventListener('DOMContentLoaded', function() {
      // 检查浏览器兼容性
      if (!window.PDFJS && !window.pdfjsLib) {
        alert('您的浏览器不支持PDF.js，请使用现代浏览器。');
        return;
      }

      // 实例化PDF查看器
      const pdfViewer = new PDFViewer({
        containerId: 'pdf-container',
        url: 'sample.pdf', // 替换为您的PDF文件URL
        initialScale: 1.5,
        workerOptions: {
          workerSrc: 'https://cdn.jsdelivr.net/npm/pdfjs-
dist@3.4.120/build/pdf.worker.min.js',
          cMapUrl: 'https://cdn.jsdelivr.net/npm/pdfjs-dist@3.4.120/cmaps/',
          cMapPacked: true
        },
        callbacks: {
          onPageChange: function(pageNum) {
            console.log('页码切换到:', pageNum);
            document.title = `PDF预览器 - 第${pageNum}页`;
          },
          onScaleChange: function(scale) {
            console.log('缩放比例:', scale);
          },
          onLoadStart: function() {
            console.log('开始加载PDF');
          },
          onLoadProgress: function(progress) {
            console.log('加载进度:', progress + '%');
          },
          onLoadComplete: function(totalPages) {
            console.log('PDF加载完成，总页数:', totalPages);
          },
          onError: function(error) {
            console.error('PDF加载或渲染错误:', error);
            // 可以在这里添加自定义错误处理逻辑
          }
        }
      });

      // 支持文件上传功能
      function setupFileUpload() {
        const fileInput = document.createElement('input');
        fileInput.type = 'file';
        fileInput.accept = '.pdf';
        fileInput.style.display = 'none';
```

```
        // 添加上传按钮到工具栏
        const toolbar = document.querySelector('.toolbar');
        if (toolbar) {
          const uploadBtn = document.createElement('button');
          uploadBtn.textContent = '上传PDF';
          uploadBtn.addEventListener('click', function() {
            fileInput.click();
          });
          toolbar.appendChild(uploadBtn);
        }

        document.body.appendChild(fileInput);

        // 处理文件上传
        fileInput.addEventListener('change', function(e) {
          const file = e.target.files[0];
          if (file && file.type === 'application/pdf') {
            const fileUrl = URL.createObjectURL(file);
            pdfViewer.loadPDF(fileUrl);

            // 清理对象URL
            setTimeout(function() {
              URL.revokeObjectURL(fileUrl);
            }, 60000); // 1分钟后清理
          } else {
            alert('请选择有效的PDF文件');
          }
        });
      }

      // 设置文件上传功能
      setupFileUpload();

      // 页面卸载时清理资源
      window.addEventListener('beforeunload', function() {
        if (pdfViewer) {
          pdfViewer.destroy();
        }
      });
    });
  </script>
</body>
</html>
```

```
class PdfViewer {
  constructor(options) {
    this.options = options;
    this.state = {
      currentPage: 1,
      scale: 1.5,
    };
    this.init();
  }

  init() {
```

```
    this.createUI();
    this.bindEvents();
    this.loadPdf();
  }

  async loadPdf(url) {
    // PDF加载逻辑
  }

  renderPage(pageNum) {
    // 页面渲染逻辑
  }
}
```

面试准备

可能被问到的问题

1. 项目介绍

问题：请介绍一下这个 PDF 预览插件项目？

回答要点：

- 项目背景和目标
- 使用的技术栈（Vue3/原生 JS + PDF.js）
- 实现的核心功能
- 技术难点和解决方案

2. 技术选型

问题：为什么选择 PDF.js？有考虑过其他方案吗？

回答要点：

- PDF.js 是 Mozilla 官方开源库，稳定可靠
- 纯 JavaScript 实现，跨平台兼容性好
- 不依赖后端，前端即可完成 PDF 渲染
- 也考虑过其他方案：
  - iframe 直接嵌入（功能有限）
  - 后端转图片（增加服务器负担）
  - 商业库（成本考虑）

3. 性能优化

问题：在处理大 PDF 文件时，你是如何优化性能的？

回答要点：

- **懒加载**：只渲染当前页面，不预加载所有页面
- **缩略图优化**：使用较小的 scale 渲染缩略图

- **Canvas 复用**：避免频繁创建销毁 Canvas
- **防抖节流**：缩放、滚动事件使用防抖
- **虚拟滚动**（扩展）：大文件可实现虚拟滚动

```
// 懒加载示例
async renderPage(pageNum) {
  // 只渲染当前页面
  const page = await this.pdfDoc.getPage(pageNum)
  // ... 渲染逻辑
}

// 缩略图使用小scale
const viewport = page.getViewport({ scale: 0.3 })
```

4. Vue3 vs Vue2

问题：你在项目中使用了 Vue3，和 Vue2 有什么区别?

回答要点:

- **Composition API**：更好的逻辑复用和代码组织
- **性能提升**：更快的渲染速度，更小的包体积
- **TypeScript 支持**：更好的类型推导
- **新特性**：Teleport、Fragments、Suspense 等

5. 错误处理

问题：如何处理 PDF 加载失败的情况?

回答要点:

```
try {
  const pdfDoc = await pdfjsLib.getDocument(url).promise;
} catch (error) {
  // 1. 记录错误日志
  console.error("PDF加载失败:", error);

  // 2. 用户友好提示
  alert("PDF加载失败，请检查文件格式");

  // 3. 触发错误回调
  if (this.options.onError) {
    this.options.onError(error);
  }

  // 4. 状态恢复
  this.loading = false;
}
```

6. 响应式设计

问题：如何保证在移动端也能正常使用？

回答要点：

- 使用媒体查询适配不同屏幕
- 触摸事件支持（可扩展）
- 工具栏自适应布局
- 缩放手势支持（可扩展）

```
@media (max-width: 768px) {
  .toolbar {
    flex-direction: column;
  }
  .thumbnails-sidebar {
    position: absolute;
    z-index: 10;
  }
}
```

技术亮点总结

准备面试时，强调这些亮点：

1. **技术栈多样性**
   - 既会 Vue3 框架，也能写原生 JS
   - 展示全栈思维和工程化能力
2. **代码质量**
   - 模块化设计
   - 清晰的注释
   - 错误处理完善
3. **用户体验**
   - 流畅的交互
   - 友好的提示
   - 键盘快捷键
4. **可扩展性**
   - 插件化设计
   - 配置项丰富
   - 事件回调机制
5. **问题解决能力**
   - 性能优化思路
   - Canvas 渲染理解
   - 异步编程掌握

扩展功能

完成基础功能后，可以考虑添加这些高级功能来提升项目竞争力：

1. 水印功能

```
// 在PDF渲染后添加水印
function addWatermark(canvas, text) {
  const ctx = canvas.getContext("2d");
  ctx.globalAlpha = 0.2;
  ctx.font = "30px Arial";
  ctx.fillStyle = "#999";
  ctx.rotate((-45 * Math.PI) / 180);
  ctx.fillText(text, 100, 500);
}
```

2. 文本搜索

```
async function searchText(searchTerm) {
  const page = await pdfDoc.getPage(pageNum);
  const textContent = await page.getTextContent();
  // 搜索匹配逻辑
}
```

3. 注释功能

- 允许用户在 PDF 上做标记
- 使用 Canvas 绘图 API
- 保存注释数据

4. 下载功能

```
function downloadPdf() {
  const link = document.createElement("a");
  link.href = pdfUrl;
  link.download = "document.pdf";
  link.click();
}
```

5. 打印功能

```
function printPdf() {
  window.print();
}
```

6. 分片加载（大文件优化）

```
// 使用Range请求分片加载
const loadingTask = pdfjsLib.getDocument({
  url: pdfUrl,
  rangeChunkSize: 65536, // 64KB chunks
});
```

7. 手势支持（移动端）

```
// 双指缩放
canvas.addEventListener("gesturestart", handleGestureStart);
canvas.addEventListener("gesturechange", handleGestureChange);

// 滑动翻页
canvas.addEventListener("touchstart", handleTouchStart);
canvas.addEventListener("touchmove", handleTouchMove);
```

学习资源

官方文档

- PDF.js 官方文档
- Vue3 官方文档
- MDN Canvas API

推荐阅读

- 《JavaScript 高级程序设计》
- 《Vue.js 设计与实现》
- 《深入理解 ES6》

在线资源

- Vue Mastery（Vue3 教程）
- freeCodeCamp（前端课程）
- GitHub 优秀开源项目

项目展示建议

1. GitHub 仓库

- 完善的 README
- 清晰的代码注释
- 提交记录规范
- 在线 Demo 链接

2. 简历描述

```
PDF预览插件（个人项目）
- 使用Vue3 + PDF.js开发功能完整的PDF在线预览工具
- 实现翻页、缩放、旋转、缩略图等核心功能
- 封装原生JavaScript版本，展示跨框架开发能力
- 采用响应式设计，支持移动端访问
- 技术栈：Vue3、Vite、Canvas API、ES6+
```

3. 面试演示

- 准备一个精美的 PDF 文件
- 流畅地演示所有功能
- 讲解技术难点和解决方案
- 展示代码组织结构

下一步计划

1. **第 1 周**：完成基础功能开发
2. **第 2 周**：添加缩略图和快捷键
3. **第 3 周**：优化性能和响应式
4. **第 4 周**：添加扩展功能
5. **第 5 周**：完善文档和测试
6. **第 6 周**：部署上线和准备面试

常见问题

Q1: PDF.js Worker 路径配置问题?

A: 确保 Worker 路径正确：

```
// 开发环境使用CDN
pdfjsLib.GlobalWorkerOptions.workerSrc =
  "https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.11.174/pdf.worker.min.js";

// 生产环境可以本地引入
```

Q2: PDF 加载跨域问题?

A:

- 确保 PDF 资源支持 CORS
- 开发时可以使用本地代理
- 或者使用 Vite 的代理配置

Q3: Canvas 渲染模糊?

A:

```
// 使用devicePixelRatio提高清晰度
const scale = window.devicePixelRatio;
canvas.width = viewport.width * scale;
canvas.height = viewport.height * scale;
canvas.style.width = viewport.width + "px";
canvas.style.height = viewport.height + "px";
ctx.scale(scale, scale);
```

总结

这个 PDF 预览插件项目是一个非常适合展示前端能力的实战项目。通过完成这个项目，你将：

- ✅ 掌握 Vue3 的核心特性
- ✅ 理解 Canvas API 的使用
- ✅ 学会处理异步编程
- ✅ 提升代码组织能力
- ✅ 积累实战经验

祝你求职顺利！加油！ 💪

如有问题，欢迎随时提问和交流！