当然，把 mammoth.js 和 react-doc-viewer 结合起来是个非常棒的主意！

这样做的思路是：用 mammoth 来负责把 docx 转换成 HTML，然后用 react-doc-viewer 这个现成的好看组件来展示转换后的内容。这样既解决了渲染问题，又有了良好的用户体验。

下面是一个完整的实现方案：

import React, { useState, useEffect, useRef } from 'react';

import { DocViewer } from 'react-doc-viewer';

import mammoth from 'mammoth';

// 1. 首先，创建一个自定义的 Mammoth 渲染器

const MammothRenderer = ({ document }) => {

  const [htmlContent, setHtmlContent] = useState('');

  const viewerRef = useRef(null);

  useEffect(() => {

    const loadAndRenderDoc = async () => {

      try {

        // 2. 从 URL 获取文件

        const response = await fetch(document.uri);

        const arrayBuffer = await response.arrayBuffer();

        // 3. 使用 mammoth 将 docx 转换为 HTML

        const result = await mammoth.convertToHtml({ arrayBuffer });

        // 4. 将转换后的 HTML 设置到 state 中

        setHtmlContent(result.value);

      } catch (error) {

        console.error('Error loading or rendering document:', error);

        setHtmlContent('<div style="color: red;">Failed to load document.</div>');

      }

    };

    if (document.uri) {

      loadAndRenderDoc();

    }

  }, [document.uri]);

  return (

    <div ref={viewerRef} style={{ width: '100%', height: '100%', overflow: 'auto' }}>

      <div dangerouslySetInnerHTML={{ \_\_html: htmlContent }} />

    </div>

  );

};

// 5. 最后，在你的主组件中使用

function MyDocumentViewer() {

  const documents = [

    { uri: '/path/to/your/document.docx' }, // 你的文档路径

  ];

  return (

    <div style={{ height: '800px', border: '1px solid #ccc' }}>

      <DocViewer

        documents={documents}

        // 关键一步：告诉 react-doc-viewer 使用我们自定义的渲染器

        pluginRenderers={{

          docx: MammothRenderer,

          doc: MammothRenderer, // 也可以尝试用于 .doc 格式

        }}

      />

    </div>

  );

}

export default MyDocumentViewer;

工作原理说明：

1. 自定义渲染器：我们创建了 MammothRenderer 组件，它接收一个 document 对象，里面包含文档的 URL。

2. 文件处理：在 useEffect 钩子中，我们使用 fetch 获取文件，然后用 mammoth.convertToHtml 将其转换为 HTML 字符串。

3. 内容展示：转换后的 HTML 通过 dangerouslySetInnerHTML 渲染到页面上。

4. 集成到 react-doc-viewer：在 DocViewer 组件中，通过 pluginRenderers 属性，指定对于 docx 和 doc 类型的文件，使用我们刚刚创建的 MammothRenderer 来处理。

这样一来，react-doc-viewer 负责提供文件列表、切换等界面功能，而 mammoth.js 则高效地完成了核心的文档转换工作，两者配合得天衣无缝。

这个方案应该能解决你之前遇到的安装和报错问题。

需要我为你整理一份 Mammoth 样式自定义指南 吗？这样你就可以根据自己的项目需求，调整文档预览的字体、颜色和布局了。

---------------------------------------------------2

作为前端开发者，实现DOC/DOCX文档预览有多种优秀方案。以下是推荐的高质量插件和方案：

🏆 最佳推荐方案

1. Mammoth.js (.docx)

```bash

npm install mammoth

```

```jsx

import mammoth from "mammoth";

function DocxViewer({ file }) {

  const [html, setHtml] = React.useState("");

  const previewDocx = async (file) => {

    const arrayBuffer = await file.arrayBuffer();

    const result = await mammoth.convertToHtml({ arrayBuffer });

    setHtml(result.value);

  };

  return <div dangerouslySetInnerHTML={{ \_\_html: html }} />;

}

```

优点：轻量、纯前端处理、保留格式

2. Microsoft Office Online (商业项目首选)

```jsx

function OfficeViewer({ fileUrl }) {

  return (

    <iframe

      src={`https://view.officeapps.live.com/op/embed.aspx?src=${encodeURIComponent(fileUrl)}`}

      width="100%"

      height="600px"

      frameBorder="0"

    />

  );

}

```

优点：格式完美兼容、微软官方支持

📚 其他优秀方案

3. Docx.js + React

```bash

npm install docx

```

```jsx

import { Document, Packer, Paragraph, TextRun } from "docx";

import { saveAs } from "file-saver";

// 适合文档生成，预览需要配合其他工具

```

4. React File Viewer

```bash

npm install react-file-viewer

```

```jsx

import FileViewer from 'react-file-viewer';

function MyViewer({ file, type }) {

  return (

    <FileViewer

      fileType={type}

      filePath={file}

      errorComponent={<div>预览失败</div>}

      onError={console.error}

    />

  );

}

```

🛠️ 完整实现示例

多功能文档预览组件

```jsx

import React, { useState, useCallback } from 'react';

import mammoth from 'mammoth';

const DocumentPreview = ({ file, onError }) => {

  const [previewContent, setPreviewContent] = useState('');

  const [loading, setLoading] = useState(false);

  const [previewType, setPreviewType] = useState('');

  const previewDocx = useCallback(async (file) => {

    try {

      setLoading(true);

      const arrayBuffer = await file.arrayBuffer();

      const result = await mammoth.convertToHtml({ arrayBuffer });

      // 自定义样式

      const styledHtml = `

        <div class="docx-container" style="font-family: 'Microsoft YaHei', sans-serif; line-height: 1.6;">

          ${result.value}

        </div>

      `;

      setPreviewContent(styledHtml);

      setPreviewType('html');

    } catch (error) {

      onError?.(error);

    } finally {

      setLoading(false);

    }

  }, [onError]);

  const getOfficeViewerUrl = useCallback((file) => {

    const url = URL.createObjectURL(file);

    return `https://view.officeapps.live.com/op/embed.aspx?src=${encodeURIComponent(url)}`;

  }, []);

  const renderPreview = () => {

    if (loading) return <div className="loading">加载中...</div>;

    switch (previewType) {

      case 'html':

        return <div dangerouslySetInnerHTML={{ \_\_html: previewContent }} />;

      case 'office':

        return (

          <iframe

            src={getOfficeViewerUrl(file)}

            width="100%"

            height="600"

            frameBorder="0"

            title="文档预览"

          />

        );

      default:

        return <div>选择预览方式</div>;

    }

  };

  return (

    <div className="document-preview">

      <div className="preview-options">

        <button onClick={() => previewDocx(file)}>Mammoth预览</button>

        <button onClick={() => setPreviewType('office')}>Office在线预览</button>

      </div>

      {renderPreview()}

    </div>

  );

};

export default DocumentPreview;

```

📊 方案对比

方案 优点 缺点 适用场景

Mammoth.js 纯前端、轻量、免费 复杂格式可能丢失 内部系统、简单文档

Office Online 格式完美、官方支持 需要网络、微软服务 商业项目、重要文档

React File Viewer 集成简单、多格式 依赖较多、定制性差 快速开发

💡 专业建议

1. 内部系统：优先选择 Mammoth.js，可控性强

2. 商业项目：使用 Office Online 确保兼容性

3. 大文件处理：考虑后端转换 + 前端显示方案

4. 移动端：注意 iframe 在移动端的体验

🎨 样式优化

```css

.docx-container {

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', sans-serif;

  line-height: 1.8;

  padding: 20px;

}

.docx-container h1,

.docx-container h2,

.docx-container h3 {

  color: #333;

  margin: 1.2em 0 0.6em;

}

.docx-container table {

  border-collapse: collapse;

  width: 100%;

}

.docx-container td,

.docx-container th {

  border: 1px solid #ddd;

  padding: 8px;

}

```

选择方案时考虑您的具体需求：文档复杂度、网络环境、安全性要求等。对于大多数场景，Mammoth.js + Office Online 备用方案是最佳选择。

------------------------------------------------------------------------------3

使用 Mammoth.js 读取本地 DOC/DOCX 文件的完整实现方案：

🎯 基础实现

1. 文件输入 + Mammoth 转换

```jsx

import React, { useState, useCallback } from 'react';

import mammoth from 'mammoth';

const LocalDocViewer = () => {

  const [htmlContent, setHtmlContent] = useState('');

  const [loading, setLoading] = useState(false);

  const [fileName, setFileName] = useState('');

  // 处理文件选择

  const handleFileSelect = useCallback(async (event) => {

    const file = event.target.files?.[0];

    if (!file) return;

    // 检查文件类型

    if (!file.name.match(/\.(doc|docx)$/i)) {

      alert('请选择 DOC 或 DOCX 文件');

      return;

    }

    setFileName(file.name);

    await convertDocToHtml(file);

  }, []);

  // 转换 DOC/DOCX 为 HTML

  const convertDocToHtml = async (file) => {

    try {

      setLoading(true);

      setHtmlContent('');

      // 方法1: 使用 FileReader 读取为 ArrayBuffer

      const arrayBuffer = await readFileAsArrayBuffer(file);

      // 使用 Mammoth 转换

      const result = await mammoth.convertToHtml({ arrayBuffer });

      // 应用自定义样式

      const styledHtml = applyCustomStyles(result.value);

      setHtmlContent(styledHtml);

      console.log('文档转换成功', result.messages);

    } catch (error) {

      console.error('文档转换失败:', error);

      setHtmlContent('<div class="error">文档预览失败，请重试</div>');

    } finally {

      setLoading(false);

    }

  };

  // 读取文件为 ArrayBuffer

  const readFileAsArrayBuffer = (file) => {

    return new Promise((resolve, reject) => {

      const reader = new FileReader();

      reader.onload = (e) => resolve(e.target.result);

      reader.onerror = reject;

      reader.readAsArrayBuffer(file);

    });

  };

  return (

    <div className="local-doc-viewer">

      <div className="file-input-section">

        <input

          type="file"

          accept=".doc,.docx"

          onChange={handleFileSelect}

          className="file-input"

          id="doc-file"

        />

        <label htmlFor="doc-file" className="file-input-label">

          选择 DOC/DOCX 文件

        </label>

        {fileName && <span className="file-name">已选择: {fileName}</span>}

      </div>

      {loading && <div className="loading">正在转换文档...</div>}

      {htmlContent && (

        <div

          className="doc-preview-container"

          dangerouslySetInnerHTML={{ \_\_html: htmlContent }}

        />

      )}

    </div>

  );

};

// 应用自定义样式

const applyCustomStyles = (html) => {

  return `

    <div class="mammoth-container">

      ${html}

    </div>

  `;

};

export default LocalDocViewer;

```

🎨 完整样式方案

```css

/\* 文件输入样式 \*/

.file-input {

  display: none;

}

.file-input-label {

  display: inline-block;

  padding: 12px 24px;

  background: #007bff;

  color: white;

  border-radius: 6px;

  cursor: pointer;

  transition: background 0.3s;

  font-size: 14px;

}

.file-input-label:hover {

  background: #0056b3;

}

.file-name {

  margin-left: 12px;

  color: #666;

  font-size: 14px;

}

.loading {

  text-align: center;

  padding: 20px;

  color: #007bff;

  font-size: 16px;

}

/\* 文档预览容器样式 \*/

.mammoth-container {

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Microsoft YaHei', sans-serif;

  line-height: 1.6;

  color: #333;

  max-width: 210mm; /\* A4 纸宽度 \*/

  min-height: 297mm; /\* A4 纸高度 \*/

  margin: 20px auto;

  padding: 40px;

  background: white;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

  border: 1px solid #e0e0e0;

}

/\* 标题样式 \*/

.mammoth-container h1 {

  font-size: 24px;

  font-weight: bold;

  color: #2c3e50;

  border-bottom: 2px solid #3498db;

  padding-bottom: 8px;

  margin: 24px 0 16px 0;

}

.mammoth-container h2 {

  font-size: 20px;

  font-weight: bold;

  color: #34495e;

  margin: 20px 0 12px 0;

}

.mammoth-container h3 {

  font-size: 16px;

  font-weight: bold;

  color: #7f8c8d;

  margin: 16px 0 8px 0;

}

/\* 段落样式 \*/

.mammoth-container p {

  margin: 8px 0;

  text-align: justify;

}

/\* 列表样式 \*/

.mammoth-container ul,

.mammoth-container ol {

  margin: 12px 0;

  padding-left: 24px;

}

.mamm-container li {

  margin: 4px 0;

}

/\* 表格样式 \*/

.mammoth-container table {

  border-collapse: collapse;

  width: 100%;

  margin: 16px 0;

  font-size: 14px;

}

.mammoth-container th {

  background: #f8f9fa;

  font-weight: bold;

  text-align: left;

}

.mammoth-container td,

.mammoth-container th {

  border: 1px solid #dee2e6;

  padding: 8px 12px;

}

/\* 代码块样式 \*/

.mammoth-container pre {

  background: #f8f9fa;

  border: 1px solid #e9ecef;

  border-radius: 4px;

  padding: 12px;

  overflow-x: auto;

  margin: 12px 0;

  font-family: 'Consolas', 'Monaco', monospace;

}

/\* 图片样式 \*/

.mammoth-container img {

  max-width: 100%;

  height: auto;

  display: block;

  margin: 12px auto;

}

.error {

  color: #dc3545;

  text-align: center;

  padding: 20px;

  background: #f8d7da;

  border: 1px solid #f5c6cb;

  border-radius: 4px;

}

```

🔧 高级功能扩展

2. 拖拽上传版本

```jsx

import React, { useState, useCallback } from 'react';

import mammoth from 'mammoth';

const DragDropDocViewer = () => {

  const [htmlContent, setHtmlContent] = useState('');

  const [isDragging, setIsDragging] = useState(false);

  const handleDragOver = useCallback((e) => {

    e.preventDefault();

    setIsDragging(true);

  }, []);

  const handleDragLeave = useCallback((e) => {

    e.preventDefault();

    setIsDragging(false);

  }, []);

  const handleDrop = useCallback(async (e) => {

    e.preventDefault();

    setIsDragging(false);

    const files = e.dataTransfer.files;

    if (files.length === 0) return;

    const file = files[0];

    if (!file.name.match(/\.(doc|docx)$/i)) {

      alert('请拖拽 DOC 或 DOCX 文件');

      return;

    }

    await convertAndDisplayDoc(file);

  }, []);

  const convertAndDisplayDoc = async (file) => {

    try {

      const arrayBuffer = await file.arrayBuffer(); // 现代浏览器支持

      const result = await mammoth.convertToHtml({ arrayBuffer });

      // 高级样式处理

      const processedHtml = processMammothOutput(result.value);

      setHtmlContent(processedHtml);

    } catch (error) {

      console.error('转换错误:', error);

      setHtmlContent('<div class="error">文件转换失败</div>');

    }

  };

  // 处理 Mammoth 输出，增强样式

  const processMammothOutput = (html) => {

    return html

      .replace(/<p>/g, '<p class="doc-paragraph">')

      .replace(/<table>/g, '<table class="doc-table">')

      .replace(/<img/g, '<img class="doc-image"');

  };

  return (

    <div className="drag-drop-viewer">

      <div

        className={`drop-zone ${isDragging ? 'dragging' : ''}`}

        onDragOver={handleDragOver}

        onDragLeave={handleDragLeave}

        onDrop={handleDrop}

      >

        <div className="drop-content">

          <div className="drop-icon">📄</div>

          <p>拖拽 DOC/DOCX 文件到此处</p>

          <p className="drop-hint">或点击选择文件</p>

          <input

            type="file"

            accept=".doc,.docx"

            onChange={(e) => e.target.files?.[0] && convertAndDisplayDoc(e.target.files[0])}

            className="file-input"

          />

        </div>

      </div>

      {htmlContent && (

        <div

          className="preview-content"

          dangerouslySetInnerHTML={{ \_\_html: htmlContent }}

        />

      )}

    </div>

  );

};

```

3. 带错误处理和进度提示的完整 Hook

```jsx

import { useState, useCallback } from 'react';

import mammoth from 'mammoth';

// 自定义 Hook

export const useDocViewer = () => {

  const [htmlContent, setHtmlContent] = useState('');

  const [loading, setLoading] = useState(false);

  const [error, setError] = useState(null);

  const convertDocToHtml = useCallback(async (file) => {

    if (!file) return;

    setLoading(true);

    setError(null);

    setHtmlContent('');

    try {

      // 支持 File.arrayBuffer() 和 FileReader 两种方式

      const arrayBuffer = typeof file.arrayBuffer === 'function'

        ? await file.arrayBuffer()

        : await readFileAsArrayBuffer(file);

      const result = await mammoth.convertToHtml({

        arrayBuffer,

        // 可选的转换选项

        // includeEmbeddedStyleMap: true,

        // includeDefaultStyleMap: true,

      });

      // 检查转换消息

      if (result.messages.length > 0) {

        console.warn('转换警告:', result.messages);

      }

      const styledHtml = `

        <div class="document-content">

          ${result.value}

        </div>

      `;

      setHtmlContent(styledHtml);

    } catch (err) {

      const errorMsg = `文档转换失败: ${err.message}`;

      setError(errorMsg);

      setHtmlContent(`<div class="error">${errorMsg}</div>`);

    } finally {

      setLoading(false);

    }

  }, []);

  // 传统的 FileReader 方式（兼容性更好）

  const readFileAsArrayBuffer = (file) => {

    return new Promise((resolve, reject) => {

      const reader = new FileReader();

      reader.onload = (e) => resolve(e.target.result);

      reader.onerror = () => reject(new Error('文件读取失败'));

      reader.readAsArrayBuffer(file);

    });

  };

  const reset = useCallback(() => {

    setHtmlContent('');

    setError(null);

    setLoading(false);

  }, []);

  return {

    htmlContent,

    loading,

    error,

    convertDocToHtml,

    reset

  };

};

// 使用示例

const DocViewerComponent = () => {

  const { htmlContent, loading, error, convertDocToHtml } = useDocViewer();

  return (

    <div>

      <input

        type="file"

        accept=".doc,.docx"

        onChange={(e) => convertDocToHtml(e.target.files?.[0])}

      />

      {loading && <div>转换中...</div>}

      {error && <div className="error">{error}</div>}

      {htmlContent && (

        <div dangerouslySetInnerHTML={{ \_\_html: htmlContent }} />

      )}

    </div>

  );

};

```

💡 使用提示

1. 文件大小限制：大文件可能导致性能问题，建议添加文件大小检查

2. 错误处理：始终处理转换过程中的错误

3. 样式定制：根据需求调整 CSS 样式

4. 移动端适配：确保预览区域在移动设备上显示正常

这样实现的本地 DOC/DOCX 预览功能既美观又实用，适合大多数业务场景。

---------------------------------5

**Doc preview page**

你说得对！docx-preview 默认确实是连续滚动的，不是真正的分页显示。让我解释一下并提供正确的分页方案：

🔍 docx-preview 的"分页"真相

默认行为：连续滚动

```typescript

// 这样只是模拟分页，实际还是一屏连续显示

await renderAsync(arrayBuffer, previewRef.current, null, {

  breakPages: true, // 这个只是添加分页符样式，不是真正分页

});

```

🎯 真正的分页解决方案

方案1：使用专业的PDF转换

```typescript

// 1. 在后端将DOCX转换为PDF

// 2. 使用PDF.js在前端显示真正分页

```

方案2：手动分页控制（推荐）

```typescript

// components/PagedDocxPreview.tsx

import React, { useRef, useEffect, useState, useCallback } from 'react';

import { renderAsync } from 'docx-preview';

interface PagedDocxPreviewProps {

  document: string | File;

  pageSize?: number; // 每页高度

}

const PagedDocxPreview: React.FC<PagedDocxPreviewProps> = ({

  document,

  pageSize = 1123 // A4纸高度(297mm)的像素近似值

}) => {

  const containerRef = useRef<HTMLDivElement>(null);

  const [pages, setPages] = useState<HTMLElement[]>([]);

  const [currentPage, setCurrentPage] = useState(0);

  const [loading, setLoading] = useState(false);

  // 手动分页函数

  const paginateContent = useCallback((content: HTMLElement) => {

    const pages: HTMLElement[] = [];

    let currentPage: HTMLElement[] = [];

    let currentHeight = 0;

    // 遍历所有子元素

    Array.from(content.children).forEach((child) => {

      const childElement = child as HTMLElement;

      const childHeight = childElement.offsetHeight;

      // 如果当前页高度 + 新元素高度 > 页面高度，创建新页

      if (currentHeight + childHeight > pageSize && currentPage.length > 0) {

        const pageDiv = document.createElement('div');

        pageDiv.className = 'docx-page';

        pageDiv.style.height = `${pageSize}px`;

        pageDiv.style.overflow = 'hidden';

        pageDiv.style.marginBottom = '20px';

        pageDiv.style.border = '1px solid #e0e0e0';

        pageDiv.style.padding = '20px';

        pageDiv.style.boxSizing = 'border-box';

        currentPage.forEach(element => pageDiv.appendChild(element.cloneNode(true)));

        pages.push(pageDiv);

        currentPage = [childElement];

        currentHeight = childHeight;

      } else {

        currentPage.push(childElement);

        currentHeight += childHeight;

      }

    });

    // 添加最后一页

    if (currentPage.length > 0) {

      const pageDiv = document.createElement('div');

      pageDiv.className = 'docx-page';

      pageDiv.style.height = `${pageSize}px`;

      pageDiv.style.overflow = 'hidden';

      pageDiv.style.marginBottom = '20px';

      pageDiv.style.border = '1px solid #e0e0e0';

      pageDiv.style.padding = '20px';

      pageDiv.style.boxSizing = 'border-box';

      currentPage.forEach(element => pageDiv.appendChild(element.cloneNode(true)));

      pages.push(pageDiv);

    }

    return pages;

  }, [pageSize]);

  useEffect(() => {

    const renderDocument = async () => {

      if (!containerRef.current) return;

      setLoading(true);

      setPages([]);

      setCurrentPage(0);

      try {

        let arrayBuffer: ArrayBuffer;

        if (typeof document === 'string') {

          const response = await fetch(document);

          arrayBuffer = await response.arrayBuffer();

        } else {

          arrayBuffer = await document.arrayBuffer();

        }

        // 创建临时容器进行渲染

        const tempContainer = document.createElement('div');

        await renderAsync(arrayBuffer, tempContainer);

        // 手动分页

        const paginatedPages = paginateContent(tempContainer);

        setPages(paginatedPages);

      } catch (error) {

        console.error('渲染失败:', error);

      } finally {

        setLoading(false);

      }

    };

    if (document) {

      renderDocument();

    }

  }, [document, paginateContent]);

  // 更新容器显示当前页

  useEffect(() => {

    if (!containerRef.current || pages.length === 0) return;

    containerRef.current.innerHTML = '';

    if (pages[currentPage]) {

      containerRef.current.appendChild(pages[currentPage].cloneNode(true));

    }

  }, [pages, currentPage]);

  const nextPage = () => {

    setCurrentPage(prev => Math.min(prev + 1, pages.length - 1));

  };

  const prevPage = () => {

    setCurrentPage(prev => Math.max(prev - 1, 0));

  };

  return (

    <div className="paged-docx-preview">

      {loading && <div>加载中...</div>}

      {pages.length > 0 && (

        <>

          <div className="page-controls">

            <button onClick={prevPage} disabled={currentPage === 0}>

              上一页

            </button>

            <span>

              第 {currentPage + 1} 页 / 共 {pages.length} 页

            </span>

            <button onClick={nextPage} disabled={currentPage === pages.length - 1}>

              下一页

            </button>

          </div>

          <div ref={containerRef} className="page-container" />

          <div className="page-controls">

            <button onClick={prevPage} disabled={currentPage === 0}>

              上一页

            </button>

            <span>

              第 {currentPage + 1} 页 / 共 {pages.length} 页

            </span>

            <button onClick={nextPage} disabled={currentPage === pages.length - 1}>

              下一页

            </button>

          </div>

        </>

      )}

    </div>

  );

};

export default PagedDocxPreview;

```

方案3：CSS多列分页（简单但有限）

```typescript

// components/CSSPagedPreview.tsx

import React, { useRef, useEffect, useState } from 'react';

import { renderAsync } from 'docx-preview';

const CSSPagedPreview: React.FC<{ document: string | File }> = ({ document }) => {

  const previewRef = useRef<HTMLDivElement>(null);

  const [loading, setLoading] = useState(false);

  useEffect(() => {

    const renderDoc = async () => {

      if (!previewRef.current) return;

      setLoading(true);

      try {

        let arrayBuffer: ArrayBuffer;

        if (typeof document === 'string') {

          const response = await fetch(document);

          arrayBuffer = await response.arrayBuffer();

        } else {

          arrayBuffer = await document.arrayBuffer();

        }

        previewRef.current.innerHTML = '';

        await renderAsync(arrayBuffer, previewRef.current);

      } catch (error) {

        console.error('渲染失败:', error);

      } finally {

        setLoading(false);

      }

    };

    if (document) {

      renderDoc();

    }

  }, [document]);

  return (

    <div className="css-paged-container">

      {loading && <div>加载中...</div>}

      <div

        ref={previewRef}

        className="docx-content"

        style={{

          columnCount: 1, // 可以改为2实现双栏

          columnGap: '20px',

          height: '80vh',

          overflow: 'auto'

        }}

      />

    </div>

  );

};

export default CSSPagedPreview;

```

🎨 配套CSS

```css

.paged-docx-preview {

  max-width: 800px;

  margin: 0 auto;

}

.page-controls {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 10px;

  background: #f5f5f5;

  margin: 10px 0;

}

.page-controls button {

  padding: 8px 16px;

  background: #007bff;

  color: white;

  border: none;

  border-radius: 4px;

  cursor: pointer;

}

.page-controls button:disabled {

  background: #6c757d;

  cursor: not-allowed;

}

.page-container {

  min-height: 600px;

  border: 1px solid #e0e0e0;

  background: white;

}

.docx-page {

  background: white;

  box-shadow: 0 2px 8px rgba(0,0,0,0.1);

}

```

💡 现实建议

如果真正需要分页体验：

1. 后端转换PDF + 前端PDF.js（最佳体验）

2. 使用专业文档服务（如微软Office Online）

3. 接受连续滚动（大多数场景足够）

docx-preview 的优势：

· 快速集成

· 格式保真度高

· 适合内容预览（非打印场景）

所以如果你需要真正的分页，建议考虑PDF方案！