蒙朦 RP 1.2k
2018-11-09 发布

PDF.js实现个性化PDF渲染（文本复制）

> 我肥来啦😄 。看到Redux教程突破3w的浏览量，小窃喜，很高兴自己的文章能够帮助到大家。

这次重返，依然带给大家一个小指南，也是最近工作中遇到的一个小case。

前不久，产品经理提出要在界面上优雅地展示PDF文档，当即就有了两种实现方式：

实现方式一
使用 `embed` 标记来使用浏览器自带的pdf工具。

这种实现方式优缺点都很明显：
优点：自带“打印”，“搜索”，“翻页”等功能，强大且实现方便。
缺点：不同浏览器的pdf工具样式不一，且无法满足个性化需求，比如：禁止打印，下载等。

我们的产品经理是挑剔的😒 ，于是...

实现方式二
使用Mozilla的 `PDF.js` ， 自定义展示PDF。

> 下面我们就细致讲述一下使用 `PDF.js` 过程中遇到的问题。主要包括：

- 基础功能集成
- 使用 `Text-Layers` 渲染

什么是PDF.JS

PDF.js是基于HTML5技术构建的，用于展示可移植文档格式的文件(PDF)，它可以在现代浏览器中使用且无需安装任何第三方插件。

基础功能集成

1️⃣ 引用

首先，引用 `PDF.js` 就遇到了问题，官网中提到通过CDN引用或者下载源码至本地。
而我们并不想污染我们的 `index.html` 并且希望可以对每一个引用的框架有统一的版本管理。于是，我们搜寻到一个包：**pdfjs-dist**。

通过 `npm install pdfjs-dist` ，我们引入了PDF.js。

基础功能有两个必须引用的文件：

- pdf.js
- pdf.worker.js

如果使用CDN的方式，直接引用如下对应文件即可：

- https://mozilla.github.io/pdf...
- https://mozilla.github.io/pdf...

如果使用npm的方式，则在需要使用PDF.js的文件中如下引用:

```
import PDFJS from 'pdfjs-dist';

PDFJS.GlobalWorkerOptions.workerSrc = 'pdfjs-dist/build/pdf.worker.js';
```

这两个文件包含了获取、解析和展示PDF文档的方法，但是解析和渲染PDF需要较长的时间，可能会阻塞其它JS代码的运行。

为解决该问题，pdf.js依赖了HTML5引入的Web Workers——通过从主线程中移除大量CPU操作（如解析和渲染）来提升性能。

PDF.js的API都会返回一个Promise，使得我们可以优雅的处理异步操作。

2️⃣ 使用

首先，我们需要在HTML中添加 `<canvas>` 元素以渲染PDF：

```
<canvas id="pdf-canvas"></canvas>
```

然后添加渲染PDF的js代码：

```
var url = 'Helloworld.pdf';

PDFJS.getDocument(url).then((pdf) => {
    return pdf.getPage(1);
}).then((page) => {
    // 设置展示比例
    var scale = 1.5;
    // 获取pdf尺寸
    var viewport = page.getViewport(scale);
    // 获取需要渲染的元素
    var canvas = document.getElementById('pdf-canvas');
    var context = canvas.getContext('2d');
    canvas.height = viewport.height;
    canvas.width = viewport.width;

    var renderContext = {
        canvasContext: context,
        viewport: viewport
    };

    page.render(renderContext);
});
```

现在，PDF已经成功渲染在界面上了。我们来分析一下使用到的函数：

`getDocument()`：用于异步获取PDf文档，发送多个Ajax请求以块的形式下载文档。它返回一个Promise，该Promise的成功回调传递一个对象，该对象包含PDF文档的信息，该回调中的代码将在完成PDf文档获取时执行。

`getPage()`：用于获取PDF文档中的各个页面。

`getViewport()`：针对提供的展示比例，返回PDf文档的页面尺寸。

`render()`：渲染PDF。

到这里，基本功能告一段落了。
满心欢喜准备上线的时候，产品经理提出了另一个需求：文本复制。
然鹅。。。翻了好几遍官方文档，也没有找到文本复制的方法，并且stackoverflow上有很多类似的问题。
在不断的尝试下，我们发现了 `Text-Layer` 。

使用Text-Layers渲染

PDF.js支持在使用Canvas渲染的PDF页面上渲染文本图层。然而，这个功能需要用到额外的两个文件：`text_layer_builder.js` 和 `text_layer_builder.css` 。我们可以在GitHub的repo中获取到。

如果是使用npm，则需要做如下引用：

```
import { TextLayerBuilder } from 'pdfjs-dist/web/pdf_viewer';
import 'pdfjs-dist/web/pdf_viewer.css';
```

现在，我们开始实现文本复制功能。

首先，创建渲染需要用到DOM节点：

```
<div id="container"></div>
```

`div#container` 为最外层节点，在该div中，我们会为PDF的每个页面创建自己的 `div` ，在每个页面的 `div` 中，都会有 `Canvas` 元素。

接着，我们修改JS代码：

```
var container, pageDiv;

function getPDF(url) {
    PDFJS.getDocument(url).then((pdf) => {
        pdfDoc = pdf;
        container = document.getElementById('container');
        for (var i = 1; i<= pdf.numPages; i++) {
            renderPDF(i);
        }
    })
}

function renderPDF(num) {
    pdf.getPage(num).then((page) => {
        var scale = 1.5;
        var viewport = page.getViewport(scale);
        pageDiv = document.createElement('div');
        pageDiv.setAttribute('id', 'page-' + (page.pageIndex + 1));
        pageDiv.setAttribute('style', 'position: relative');
        container.appendChild(pageDiv);
        var canvas = document.createElement('canvas');
        pageDiv.appendChild(canvas);
        var context = canvas.getContext('2d');
        canvas.height = viewport.height;
```

以上代码只是实现了多页渲染，接下来，开始渲染文本图层。我们需要将 `page.render(renderContext)` 修改为以下代码：

```
page.render(renderContext).then(() => {
    return page.getTextContent();
}).then((textContent) => {
    // 创建文本图层div
    const textLayerDiv = document.createElement('div');
    textLayerDiv.setAttribute('class', 'textLayer');
    // 将文本图层div添加至每页pdf的div中
    pageDiv.appendChild(textLayerDiv);

    // 创建新的TextLayerBuilder实例
    var textLayer = new TextLayerBuilder({
        textLayerDiv: textLayerDiv,
        pageIndex: page.pageIndex,
        viewport: viewport
    });

    textLayer.setTextContent(textContent);

    textLayer.render();
});
```

我们依旧来讲解以下用到的几个关键函数：

`page.render()`： 该函数返回一个当PDF页面成功渲染到界面上时解析的 `promise`， 我们可以使用成功回调来渲染文本图层。

`page.getTextContent()`： 该函数的成功回调会返回PDF页面上的文本片段。

`TextLayerBuilder`： 该类的实例有两个重要的方法。 `setTextContent()` 用于设置 `page.getTextContent()` 函数返回的文本片段； `render()` 用于渲染文本图层。

> Bingo😎 ！ 通过以上改造，文本复制功能就实现了。官方文档上可没有这个小技巧哦。

PDF.js是一个很棒的工具，但无奈文档写的较为精简，需要开发人员不断探索PDF.js的强大功能。

如果这篇文章有帮助到您，记得点赞咯👍 ！

你可能感兴趣的

- **复制文本加上版权信息功能实现** 善良的乌贼 javascript html5
- **网页渲染性能优化** 晨风明悟 性能分析 网页渲染 性能优化
- **纯js 文本复制功能** Z不懂 html5 javascript
- **SegmentFault Hackathon 文艺复兴** SegmentFault 黑客马拉松
- **渲染机制** spolier javascript html5
- **个人文章分类整理** samsara0511 apache vue.js html javascript css
- **使用PDF.JS插件在HTML中预览PDF文件** mydetails javascript html5
- **性能优化(二期)——超大文档渲染优化** scaukk javascript

12 条评论

默认排序 | 时间排序

小无路 · 2018年11月09日

棒棒哒，很好

赞 回复

徐小良啊 · 2018年11月09日

很好，先收藏了

赞 回复

忆故人 · 2018年11月09日

谢谢楼主。之前遇到过这样的需要。没有实现，，

赞 回复

> 😂 我记得你上一条评论是404，我还在等你回答呢
>
> — 蒙朦 作者 · 2018年11月09日

> 额。就是用原生js。直接用那2个cdn，不知道操作那个对象。
>
> — 忆故人 · 2018年11月12日

> 于是我就把他套进去angular里面了.....
>
> — 忆故人 · 2018年11月12日

添加回复 | 显示更多

7nz · 1月10日

楼主，有没有遇到动态切换pdf地址，出现canvas渲染的问题， `Error: Cannot use the same canvas during multiple render() operations. Use different canvas or ensure previous operations were cancelled or completed.`

赞 回复

> 我的渲染方式和这个不同，我参考了https://jsfiddle.net/pdfjs/wa...，只用了一个 `canvas` ，上一页，下一页自定义的，我还加了窗口的resize函数，窗口大小改变的时候重新渲染pdf。
> 当我动态改变pdf地址，再次渲染pdf的时候就会报上面的错误，最后每次渲染之前先把canvas dom节点清除掉，再重新添加，这样就可以了，但是会有一个问题是改变窗口再次渲染的时候会闪动
>
> — **7nz** · 1月15日

添加回复

烛啉尒昱 · 4月4日

楼主我这边遇到 Https下无法加载cdn上的pdf 应该是http的原因 该怎么解决

赞 回复

youngjuning · 4月15日

```
PDFJS.GlobalWorkerOptions.workerSrc = 'node_modules/pdfjs-dist/build/pdf.worker.js'
```

赞 回复

原滋原味 · 6月22日

写得很详细。楼主有没有遇到需要显示word之类的需求

赞 回复

文明社会，理性评论

发布评论