Vue3 自定义打印原理

前言

最近接触到了一个 Vue3 的打印需求，我发现自己虽然从事前端开发已有多年，但对如何实现自定义打印还没有深入研究，一般都是找现成的库来解决问题，借这次的机会研究了一下如何实现自定义打印。

在现在的前端开发中，打印是一个比较常见的功能，无论是在生成报表、下载发票还是其他用途上都会用到。 最基本的打印方式是直接打印整个页面内容，但在实际项目中，我们常常需要更灵活的打印功能，例如选择性打印某个特定部分、提供打印预览功能等。这就产生了许多 JavaScript 打印库，如 `print-js`、`vue-print-nb` 等，这些库不仅简化了打印流程，还提供了丰富的自定义打印配置，满足各种不同的打印需求。

由于当前的项目基于 Vue3 进行开发，我简单调研后发现 `vue-print-nb` 是 Vue 中常用的打印库，同时其提供了适用于 Vue3 的 `vue3-print-nb` 库。然而，在实际使用过程中，我发现该库由于发布较早，且后期未继续维护，因此在支持 Vue3 的一些特性方面有所欠缺，所以我在此基础上进行了优化和改造，来更好地满足需求。

本文将详细探讨如何使用原生 JavaScript 实现自定义打印功能，同时讲解对 `vue3-print-nb` 的改造和优化。

实现自定义打印

实现自定义打印的核心思想是通过将要打印的内容放入一个 `iframe` 或者新窗口中，然后调用 `window.print()` 方法进行打印。然而，需要考虑的问题远不止这些，例如样式还原、表单展示、以及 canvas 的打印等。

实现思路

1. 确定打印区域

首先，需要确定用户希望打印的页面区域, 这通常通过用户传递的一个选择器或直接传入的 DOM 元素来实现：

- **选择器**：通过传递一个 CSS 选择器来标识需要打印的元素。
- **DOM 元素**：直接传递所需打印的 DOM 对象。

2. 克隆打印内容

为了保证打印内容与页面其他部分隔离开来，通常会将需要打印的内容深度克隆。这确保了打印时不会受到动态内容变化的干扰，并且可以对其进行独立处理：

```
/**
 * 拷贝需要打印的节点
 */
function cloneContent(selector) {
  const originalContent = document
        .querySelector(selector)
        .cloneNode(true);

  // 在克隆内容中处理样式和事件，比如删除不需要的节点
  originalContent
        .querySelectorAll('.no-print')
        .forEach((el) => el.remove());

  return originalContent.outerHTML;
}
```

3. 创建新的上下文

为了确保打印内容不受当前页面的影响，通常会创建一个新的窗口或 iframe 并复制打印内容到新的上下文中。两种常见的实现方式是：

- **新窗口**：打开一个新的浏览器窗口或标签页，将克隆内容放置其内，然后触发打印。
- **Iframe**：在当前页面中创建一个隐藏的 `iframe`，将打印内容放入 `iframe` 后，再从 `iframe` 中触发打印。

```
/**
 * 创建一个新的上下文（打印窗口）
 */
function createPrintIframe() {
    const iframe = document.createElement('iframe');
        iframe.id = this.iframeId;
        iframe.src = new Date().getTime().toString();
        iframe.style.display = 'none';
        document.body.appendChild(iframe);
    return iframe;
}

/**
 * 将需要打印的内容写入新的上下文
 */
function write(printWindow, writeContent) {
  const iframeDocument = printWindow.contentDocument;
  iframeDocument.open();
```

```
  iframeDocument.write(`<!DOCTYPE html><html lang='zh'><head>${getHead()}</head>
<body>${writeContent}</body></html>`);
  iframeDocument.close();
}
```

4. 处理打印样式

设计打印库时，首先需要解决的问题就是如何在新的上下文中还原样式，包括页面原有样式和用户自定义样式，以满足复杂业务需求。

- **独立样式**：插入到新的打印文档或上下文中，可以是内联样式或者外部样式表。
- **打印媒体查询**：使用 `@media print` 媒体查询，以定义仅在打印时生效的特殊样式，使得打印和屏幕显示效果独立。

```
/**
 * 获取 head
 */
function getHead() {
  // 获取网页原有的 css 链接
  const links = Array.from(document.querySelectorAll('link'))
        .filter((item) => item.href.includes('.css'))
        .map(
          (item) =>
                `<link type="text/css" rel="stylesheet" href='${item.href}'>`
        )
        .join('');

  // 获取页面中的 style 标签
  const style = Array.from(document.styleSheets).reduce(
        (acc, styleSheet) => {
          const rules = styleSheet.cssRules || styleSheet.rules;
          if (!rules) return acc;
          acc += Array.from(rules).reduce(
                (innerAcc, rule) => innerAcc + rule.cssText,
                ''
          );
          return acc;
        },
        ''
  );
  // ...省略处理用户自定义引入的 css 样式

  // 返回 head
  return `<title>自定义打印示例</title>${links}<style type="text/css">${style}
</style>`;
}
```

5. 触发打印

准备好打印内容和样式后，打印库通过调用 `window.print` 方法来触发打印。打印操作通常包含以下步骤：

- **打印对话框**：调用 `window.print` 来触发系统打印对话框。
- **清除上下文**：打印完成后，关闭或销毁用于打印的临时窗口或 `iframe`，以避免内存泄漏和资源占用。

```
/**
 * 触发打印操作
 */
function triggerPrint(printWindow) {
  const iframeWin = printWindow?.contentWindow;
  iframeWin.addEventListener('load', () => {
        iframeWin.focus();
        iframeWin.print();
        iframe.remove();
  })
}
```

通过上述步骤，我们就可以灵活地控制打印内容和样式，并能提供良好的打印体验和可定制化的功能。

综合示例

通过以上步骤，我们可以实现一个简易的自定义打印功能。

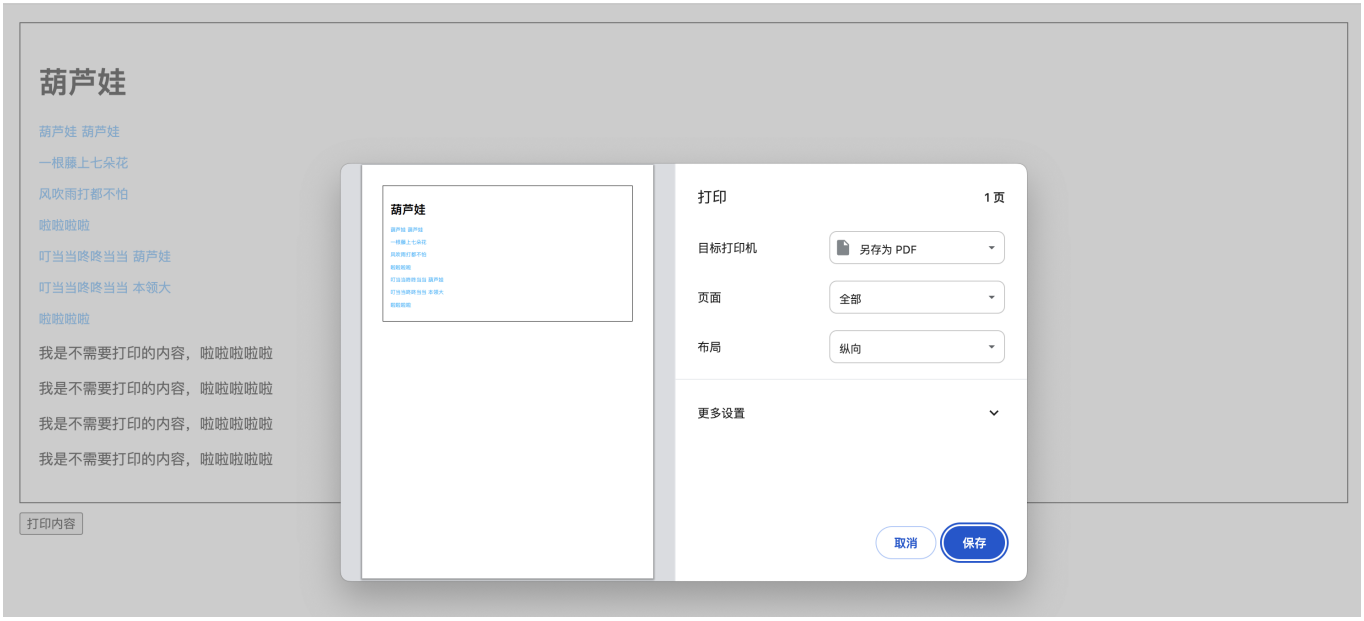
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>自定义打印示例</title>
  <style>
    html,
    body {
      margin: 0;
    }

    body {
      padding: 20px
    }
```

```
    .printable {
      border: 1px solid #000;
      padding: 20px;
    }

    .gc {
      color: #0aff;
      font-size: 14px;
    }
  </style>
</head>

<body>
  <div class="printable" id="printableArea">
    <h1>葫芦娃</h1>
    <p class="gc">葫芦娃 葫芦娃</p>
    <p class="gc">一根藤上七朵花</p>
    <p class="gc">风吹雨打都不怕 </p>
    <p class="gc">啦啦啦啦</p>
    <p class="gc">叮当当咚咚当当 葫芦娃</p>
    <p class="gc">叮当当咚咚当当 本领大</p>
    <p class="gc">啦啦啦啦</p>
    <p class="no-print">我是不需要打印的内容，啦啦啦啦啦啦</p>
    <p class="no-print">我是不需要打印的内容，啦啦啦啦啦啦</p>
    <p class="no-print">我是不需要打印的内容，啦啦啦啦啦啦</p>
    <p class="no-print">我是不需要打印的内容，啦啦啦啦啦啦</p>
  </div>
  <div>
    <button id="printBtn" style="margin-top: 10px">打印内容</button>
  </div>

  <script>
    /**
     * 获取 head
     */
    function getHead() {
      // 获取网页原有的 css 链接
      const links = Array.from(document.querySelectorAll('link'))
        .filter((item) => item.href.includes('.css'))
        .map(
          (item) =>
            `<link type="text/css" rel="stylesheet" href='${item.href}'>`
        )
        .join('');

      // 获取页面中的 style 标签
      const style = Array.from(document.styleSheets).reduce(
        (acc, styleSheet) => {
          const rules = styleSheet.cssRules || styleSheet.rules;
```

```
          if (!rules) return acc;
          acc += Array.from(rules).reduce(
            (innerAcc, rule) => innerAcc + rule.cssText,
            ''
          );
          return acc;
        },
        ''
      );

      // 返回 head
      return `<title>自定义打印示例</title>${links}<style type="text/css">${style}
</style>`;
    }

    /**
     * 创建一个新的上下文（打印窗口）
     */
    function createPrintIframe() {
      const iframe = document.createElement('iframe');
      iframe.src = new Date().getTime().toString();
      iframe.style.display = 'none';
      document.body.appendChild(iframe);
      return iframe;
    }

    /**
     * 拷贝需要打印的节点
     */
    function cloneContent(selector) {
      const originalContent = document
        .querySelector(selector)
        .cloneNode(true);

      // 在克隆内容中处理样式和事件，比如删除不需要的节点
      originalContent
        .querySelectorAll('.no-print')
        .forEach((el) => el.remove());

      return originalContent.outerHTML;
    }

    /**
     * 将需要打印的内容写入新的上下文
     */
    function write(printWindow, writeContent) {
      const iframeDocument = printWindow.contentDocument;
      iframeDocument.open();
      iframeDocument.write(`<!DOCTYPE html><html lang='zh'><head>${getHead()}
```

```
</head><body>${writeContent}</body></html>`);
      iframeDocument.close();
    }

    /**
     * 触发打印操作
     */
    function triggerPrint(printWindow) {
      const iframeWin = printWindow?.contentWindow;
      iframeWin.addEventListener('load', () => {
        iframeWin.focus();
        iframeWin.print();
        iframe.remove();
      })
    }

    /**
     * 打印元素
     */
    function printElement(selector) {
      const clonedContent = cloneContent(selector);
      const printWindow = createPrintIframe();
      write(printWindow, clonedContent);
      triggerPrint(printWindow);
    }

    document.getElementById('printBtn').addEventListener('click', () => {
      printElement('#printableArea');
    });
  </script>
</body>

</html>
```

实现效果如下：



通过上述示例，可以实现一个简易的自定义打印功能。在实际业务中，我们可能需要支持更多复杂的需求，如外部 CSS 的引入，打印 canvas、表单等。

`vue3-print-nb`

`vue3-print-nb` 是一个用于 Vue3 的打印插件，提供了相对丰富的打印功能。但在实际使用过程中，我发现了一些不足之处，例如对 Vue3 setup 函数的支持不够友好，以及不支持手动调用等。

优化和改造

为了更好地支持 Vue3 setup 函数，我对 `vue3-print-nb` 进行了以下改造：

1. 使用 TypeScript 重写了整个库，使得调用传参时体验更佳。
2. 增加了对 `VuePrintNext` 类的导出，允许手动调用打印方法，不再局限于指令调用。
3. 增强 Vue3 setup 支持，允许通过直接导入 `vPrint` 指令进行局部导入使用。
4. 支持设置指定的 CSS 选择器来忽略打印部分内容。
5. 支持通过 CSS 选择器或手动传入 DOM 节点进行局部打印。

此外，还修复了一些已知的 bug：

- 背景色打印失效问题。
- 通过 URL 打印时，有时无法触发打印行为的问题。
- 其他一些小问题。

新打印插件

我已将上述优化和改造封装成一个新的插件：`vue-print-next`，并已上传至 GitHub：https://github.com/Alessandro-Pang/vue-print-next。你可以通过 npm 进行安装和使用。

```
npm install vue-print-next
```

结语

通过本文，我们对自定义打印的实现原理有了比较全面的了解。尽管实现打印功能仍然依赖于 `window.print` API，但通过 `window.open` 或者 `iframe` 巧妙的将所需打印内容进行隔离处理，可以实现灵活的打印功能。

相关链接

- window.print: https://developer.mozilla.org/zh-CN/docs/Web/API/Window/print
- vue3-print-nb: https://github.com/Power-kxLee/vue3-print-nb