软件设计文档

目　　录

1 开发规划

1.1 开发人员

分　　工	人　　员
前端	欧光文、林锦涛
语法解析	庄嘉鑫
生成 pdf	刘继汉、欧光文
功能、文档	黄新伟、赖君秋
界面设计	黄新伟
测试	赖君秋

1.2 开发计划

5.12 完成第一次迭代，包括前端，语法解析
5.14 生成 pdf

1.3 开发语言、环境和工具

开发语言

语言	作用
HTML、CSS	前端
JavaScript	后端

开发环境

环境	人员
Windows 系统	黄新伟、赖君秋
Ubuntu 系统	庄嘉鑫、林锦涛、刘继汉
Mac OS 系统	欧光文

开发工具

工具	作用
Axure RP Pro	设计软件界面
Sublime Text	编写软件代码
IE 浏览器	观察软件效果

2 总体设计

Markdown 是一种可以使用普通文本编辑器编写的标记语言，通过简单的标记语法，它可以使普通文本内容具有一定的格式。这种写作方式最大的特点是用“符号”表示“格式”。例如，使用 word 等传统方式写作的时候，输入了一个标题，那就需要选中这个标题去格式栏中设置格式，而使用 markdown 方式写作的时候，只需要在标题前面加“##”这样的井号标记来表示标题的级别。可以明显的感觉到你输入两个“##”的过程完全不会打断你的写作，这要比单独设置格式方便的多。所以 markdown 写作方式，就是一种在码字的时候，把文章的格式也顺便“码”进去的写作方式。用户可以使用诸如“*”、“#”等简单的标记符号以最小的输入代价生成极富表现力的 md 文档。

Markdown 具有很多优点：

①写作中添加简单符号即完成排版，所见即所得，达到实时效果预览的效果，使用户能够专注于文字而不是排版。

②格式转换方便，Markdown 的文本你可以轻松转换为 pdf 文档等。

③可以保存成纯文本。

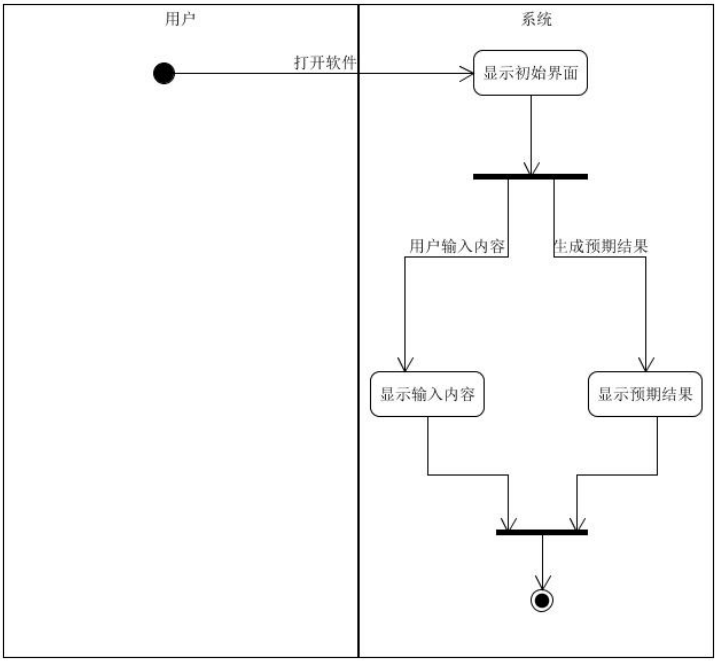
支持 Markdown 的编辑器太多，功能也不完全一致。本项目团队编写的 marker，是使用 markdown 语言的编辑器，目前将提供两个功能：

①实时效果预览。marker 主要界面分为两部分，左半边是编辑区域，用于 markdown 语言写作；右半边是预览区域，能够实时地对左半边编辑区域中的代码进行编译并实时展示。

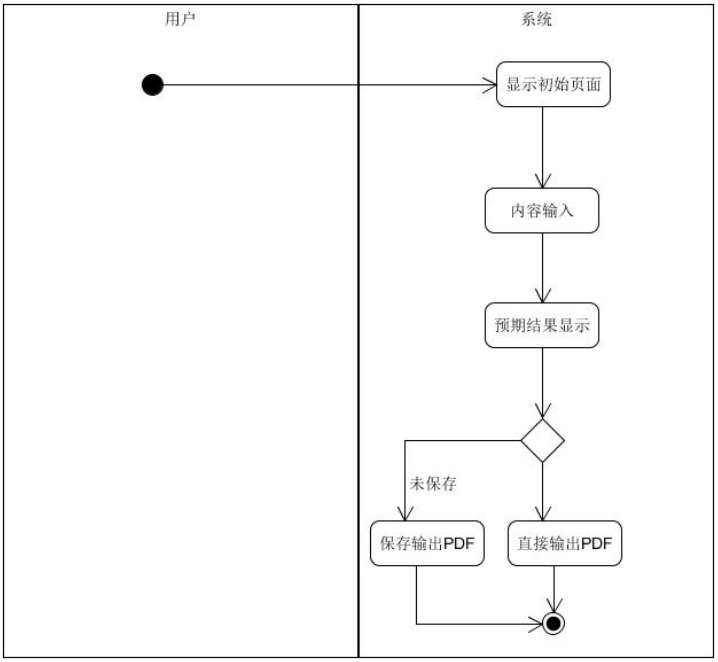
②md 文件转 pdf 文件。将 markdown 语言文本经过编译之后得到的效果转换为 pdf 文件并保存到本地。

2.1 主要界面流程描述

2.1.1 实时效果预览 界面流程

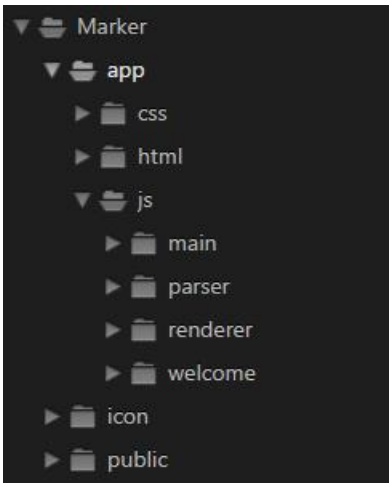


2.1.2 md 文件转 pdf 文件 界面流程



2.2 模块划分

程序主要分为 CSS，HTML，JS，三个模块，JS 又分为 main，parser，renderer，welcome，四个部分。首先是 main 部分，主要是 electron 的主进程的所有 JS 代码。Parser 部分是负责使 markdown 编辑器执行语法解析的功能。Renderer 是负责处理渲染线程的，与 html 目录中 index.html 相关。Welcome 部分是关于点击软件后，一开始弹出欢迎页面的功能，与 html 目录中 welcome.html 相关。icon 文件夹是保存了欢迎页面的图标。Pubilc 保存的是页面上需要用的图片。



3 软件设计技术

Design pattern：运用到了观察者模式，主要是通过在主进程监听渲染进程的通信消息。

```
app.on('ready', createWindow);

// 当应用即将退出时,
// 注销全局快捷键
app.on('will-quit', () => {
  shortCutUtil.unregisterSC();
});

app.on('window-all-closed', () => {
  // 在 OS X 上，通常用户在明确地按下 Cmd + Q 之前
  // 应用会保持活动状态
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', () => {
  // 在 OS X 上，当用户点击dock图标
  // 而没有打开的窗口时
  // 将会自动创建一个窗口
  if (mainWindow === null) {
    createWindow();
  }
```

下面是在渲染进程中监听主进程的通信消息：

```
// 监听快捷键操作
ipcRenderer.on('SC', (event, args) => {
  toolUtil[args](fileIndex);
});

// 监听按行操作
ipcRenderer.on('SE', (event, args) => {
  cursorUtil[args](fileIndex);
});

// 监听字号变化
ipcRenderer.on('FS', (event, args) => {
  fontsizeUtil[args]();
});

// 监听字体变化
ipcRenderer.on('FF', (event, args) => {
  fontfamilyUtil.setFontfamily(args);
});

// 监听行号的现实与隐藏
ipcRenderer.on('LN', (event, args) => {
  linenumberUtil.toggleLinenumber();
});

// 监听file通道的上的信息，调用对应的文件操作函数
ipcRenderer.on('file', (event, args) => {
  if (args[0] === 'focus' || args[0] === 'newFile' || args[0] === 'close') {
    fileIndex = args[1].index;
  }
  fileUtil[args[0]](args[1]);
});
```

4 技术选型理由

Electron:

我们选择了 Electron 是因为 Electron 可以让你使用纯 JavaScript 调用丰富的原生 APIs 来创造桌面应用。你可以把它看作是专注于桌面应用而不是 web 服务器的，io.js 的一个变体。
这不意味着 Electron 是绑定了 GUI 库的 JavaScript。相反，Electron 使用 web 页面作为它的 GUI，所以你能把它看作成一个被 JavaScript 控制的，精简版的 Chromium 浏览器。

在 Electron 里，运行 `package.json` 里 `main` 脚本的进程被称为主进程。在主进程运行的脚本可以以创建 web 页面的形式展示 GUI。

由于 Electron 使用 Chromium 来展示页面，所以 Chromium 的多进程结构也被充分利用。每个 Electron 的页面都在运行着自己的进程，这样的进程我们称之为渲染进程。

在一般浏览器中，网页通常会在沙盒环境下运行，并且不允许访问原生资源。然而，Electron 用户拥有在网页中调用 io.js 的 APIs 的能力，可以与底层操作系统直接交互。

markdown-pdf:

用来将 markdown 文件转成 pdf 格式的一个依赖包。

```
    // 选择文件时点击取消会导致undifined
    if (typeof (filename) !== 'undefined') {
      markdownpdf().from(fileList[index].path).to(filename, function () {
        console.log("Done");
      });
    }
```

node-localstorage:

用来将用户信息存储在 Electron 中，当用户再次打开时，软件就能将上次用户操作的信息读取出来。

```
// 用来获取和窗口相关的所有信息
exports.getWindowState = () => storage.getItem('windowState');

/**
 * [用来设置和窗口有关的所有信息]
 * @param  {[boolean]} isMaxmized [窗口是否最大化]
 * @param  {[object]}  bounds     [窗口位置信息]
 */
exports.setWindowState = (isMaxmized, bounds) => {
  const windowState = {
    isMaxmized,
    bounds,
  };
  storage.setItem('windowState', windowState);
};
```

jquery:

用来操作 html 里面的动操作。

```
//阻止浏览器默认行
$(document).on({
  dragleave: function(e) { // 拖离
    e.preventDefault();
  },
  drop: function(e) { // 拖后放
    e.preventDefault();
  },
  dragenter: function(e) { // 拖进
    e.preventDefault();
  },
  dragover: function(e) { // 拖来拖去
    e.preventDefault();
  }
});

// 文件拖拽上传
$('.editor').on('drop', function(e) {
  e.preventDefault();
  const fileList = e.originalEvent.dataTransfer.files;

  // 如果文件为空则返回
  if (fileList.length === 0) {
    return;
  }
  // 如果不是markdown文件则返回
  if (fileList[0].type !== 'text/markdown') {
    return;
  }
});
```

marked:

用来将 markdown 文件转成 html 的一个依赖包。

```
exports.parser = (value) => {
    // marked 用于解析文档为对应的html
    const marked = require('marked');

    marked.setOptions({
      renderer: new marked.Renderer(),
      gfm: true,
      tables: true,
      breaks: true,
      pedantic: false,
      sanitize: true,
      smartLists: true,
      smartypants: false,
      // 使用hightlight进行高亮
      highlight: function (code) {
          return require('highlight.js').highlightAuto(code).value;
      }
    });

    return marked(value);
};
```

eslint:

在开发时进行 JS 语法编辑规范。

```
"lint": "eslint ./**/*.js"
```