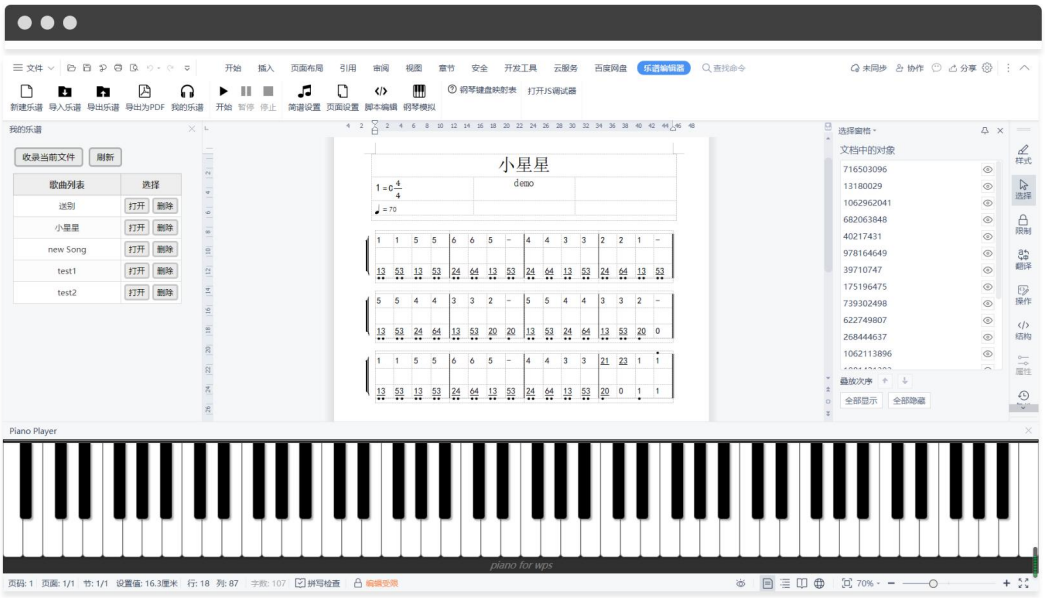
Music Editor

乐谱编辑器设计文档

作者 ： 李祖贤

目录

一、插件介绍



通过 WPS 加载项，开发一套简单的乐谱编辑插件。

目前主要支持的功能

① 简谱制作

② 排版

③ 导出

④ 对正文内容进行播放试听

⑤ 乐谱的本地化管理

⑥ 钢琴键盘弹奏

为音乐爱好者提供简单的乐谱编辑能力，使用离线插件部署模式，拓展 wps 的功能

二、前言

2.1 关于为什么要做音乐插件

以目前来看，WPS 加载项这套开发体系是有一些局限性的，目前定位，主要是以跨平台跨浏览器集成方案为主。

但 JSAPI 只能做集成吗？抱着这种想法我进行了一次尝试。

2.2 插件定位:

目前是独立的乐谱编辑插件，所以自然也是离线模式打包进行发布和安装，这样即使没有服务器也可以正常使用。

2.3 作品的自我评价

开发中是存在一些困难的，因为我发现虽然开发模式类似浏览器开发。但 html 有 css 和 dom 可以去操作。但 wps 似乎无法通过更简易的方式去渲染正文中的各种元素和样式。

如果只是嵌入网页进 WPS，这不叫 WPS 开发。所以我选择了使用表格去进行定位。配合 JS 类的属性，进行 wps 元素的存储与识别，但同时渲染带来的卡顿和开发难度，却远远大于网页开发

2.4 最后

希望 WPS 加载项开发体系被更多的个人开发者所认识。

创造出五彩缤纷的插件世界

三、代码设计与构成

3.1 开发环境

开发工具：Visual Studio Code

开发环境：npm 、node

3.2 安装并运行

进入 /源码/musicEdit 目录依次执行

npm install

npm run serve

此时会弹出一个页面，在弹出的页面中选择 music-editor 进行安装

运行时用到 live-server 工具包（调试中可以随时修改代码且实时更新）

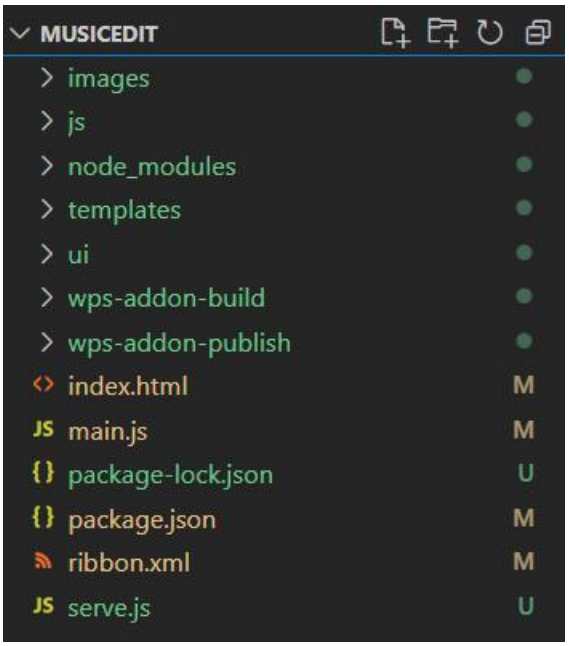
如果插件没有加载可能需要 quickstartoffice restart 一下

3.3 功能演示和使用说明

参考目录下演示 PPT

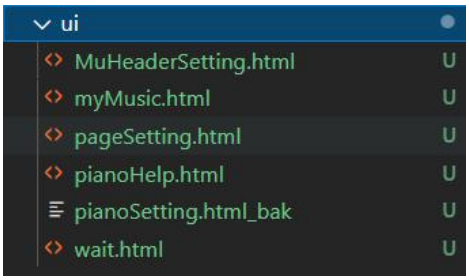
3.4 代码结构

3.4.1 整体结构



Images	图片素材
Js	所有 js 代码
ui	Html 页面 dialog 和 taskpane
Serve.js	Node 脚本 & Live-server 配置

3.4.2 ui 说明



从上至下依次为

MuHeaderSetting.html ---------- 乐谱设置

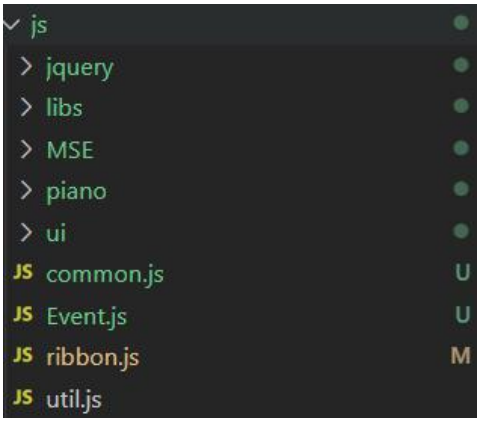
myMusic.html ---------- 我的乐谱

pageSetting.html ---------- 页面设置

pianoHelp.html ---------- 钢琴及键盘对照表

wait.html ---------- 渲染时的等待页面

3.4.3 JS 说明



Libs ---------- 用得到其他的包

MSE ---------- 乐谱编辑代码（主要代码）

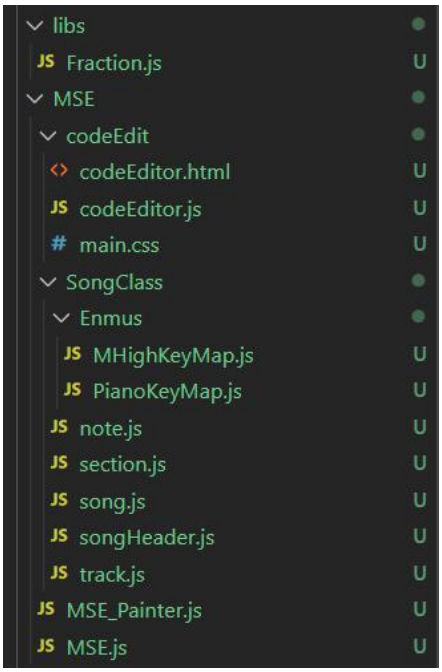
Piano ---------- piano-simulator 钢琴构建后的脚本

ui ---------- html-UI 的对应脚本

common.js ---------- 公用类 包含常用方法

Event.js ---------- 事件监听代码

Ribbon.js ---------- ribbon.xml 对应代码



Libs/Fraction.js ---------- 分数操作类

MSE/codeEdit ---------- 脚本编辑器

SongClass ---------- 歌曲类

Enmus ---------- 枚举 (钢琴与按键等对应枚举)

Note.js ---------- 音符类

Section.js ---------- 小节类

Song.js ---------- 歌曲类

songHeader.js ---------- 歌曲信息类

Track.js ---------- 歌曲音轨类

MSE_Painter.js ---------- 乐谱脚本至 WPS 渲染代码

MSE.js ---------- 主代码封装

关于歌曲类，其中的关系是

song -> tracks -> track -> sections -> section -> notes -> note

歌曲-> 音轨 s-> 音轨 -> 小节 s -> 小节 -> 音符 s -> 音符

3.4.4 类说明

这部分说明源码代码部分有相关注释

0. MSE 类说明

MSE 是 Music Score Editor 的缩写 ， 意为乐谱编辑器

MSE 相当于整个乐谱编辑器的工具类

```
w.MSE = {
    /** 工具状态 **/
    workStatus: false,          //插件工作状态 false闲置 true乐谱编辑状态
    PlayStatus: 0,              //播放状态 0停止 1播放 2暂停
    MScoreUrl: "",              //乐谱文件路径
    midiUrl: "",                //midi文件路径
    Editor: null,               //编辑器
    editorCommand: "",          //编辑器指令
    piano: null,                //钢琴面板
    pianoCommand: "",           //钢琴指令 用于wps调用taskpane  play/pause/stop
    song: null,                 //歌曲类
    JPcode: "",                 //简谱脚本
    JPcodeChangeInfo: "",       //简谱脚本最新修改信息  {version:xxx,CmdNo:LineNo(行号)}
    oldJPcode: "",              //上次简谱脚本
    oldQMax: 0,                 //原先最大脚本行（不包括无效行）
    oldJPcodeChangeInfo: { "version": 0, "CmdNo": 0 },    //简谱脚本上次修改信息
```

1. Song 歌曲类

```
//音乐
function Song(info) {
    this.tracks = [];                                       //音轨中的小节
    this.Header = new fakeSongHeader(info.Header);          //头部信息
    this.TimeSum = 0;                                       //歌曲总时长 毫秒
    this.afterPlayCommand = info.afterPlayCommand;          //播放后执行的命令
    this.afterRenderCommand = info.afterRenderCommand;      //渲染后执行的命令(该音符之后的显示动作)
    if (info.tracks) {
        info.tracks.forEach(t => {
            this.tracks.push(new fakeTrack(t));
        });
    }
}
```

2. songHeader 歌曲信息类

```
//音乐信息
function SongHeader(info) {
    this.B1 = info.B1;                                      //主标题
    this.B2 = info.B2;                                      //副标题
    // this.Z1 = info.Z1;
    this.Z2 = info.Z2;                                      //曲 作者
    this.D = info.D;                                        //调式D
    this.P = info.P;                                        //拍号P
    this.J = info.J;                                        //节拍 (速度 120...)
    this.noteType = info.noteType;                          //拍号P的分母分之1 (X分音符是一拍 1/X)
}
```

2. track 音轨类

```
//音轨类
function track(info) {
    this.sections = [];                                 //音轨中的小节
    this.afterPlayCommand = info.afterPlayCommand;      //播放后执行的命令
    this.afterRenderCommand = info.afterRenderCommand;  //渲染后执行的命令(该音符之后的显示动作)
}
```

2. section 小节类

```
//小节类
function section(info) {
    this.notes = [];                                    //小节中的音符
    this.symbol = "";                                   //标记 { 小节线| 开始[| 结束|] }
    this.total = new Fraction(0, 0);                    //记录本小结总时长，只能通过setNote来改变
    this.nowCellNo = 1;                                 //当前所在格位
    this.afterPlayCommand = info.afterPlayCommand;      //播放后执行的命令
    this.afterRenderCommand = info.afterRenderCommand;  //渲染后执行的命令(该音符之后的显示动作)
}
```

2. Note 音符类

```
//音符类
function note(info) {
    this.noteType = info.noteType;                      //拍号P的分母分之1 (X分音符是一拍 1/X)
    this.noteName = info.noteName;                      //C3 D3 E3 ....
    this.value = info.value;                            //表示该音符是几分音符 [1全音 1/2二分 1/4四分.... 1/32]  减时用"/"表示
    // this.timeLine = info.timeLine;                      //表示该音符的时值 (就是value 如果有附点再多加个附点的时间 3/4 3/8...)
    this.symbol = info.symbol;                          //[1~7 0 -] 谱面显示的数字/字符 默认休止符"-"
    this.high = info.high;                              //[-2,-1,0,1,2] 倍低音/无/倍高音  _JPcode倍高音符号' 倍低音符号,
    this.attDot = info.attDot;                          //是否有附点  _JPcode附点符号.
    this.STmark = info.STmark;                          //Short Time Mark [# $ = ]升音 降音 还原音 空代表没有
    this.afterPlayCommand = info.afterPlayCommand;      //播放后执行的命令
    this.afterRenderCommand = info.afterRenderCommand;  //渲染后执行的命令(该音符之后的显示动作)
    this.addr = info.addr;                              //wps中的地址  track表的小节->section表的note  //[1, [2, 2], 1, [2, 1]]
    this.shapes = [];                                   //shapes  对应的图形 倍高、倍低等
}
```

3.4.5 脚本解析与渲染说明

渲染为 MSE_Painter.js ,其中，每次渲染都需要与上次脚本进行版本比较

在敲打新的音符后，通过**传递捕获**，将新的脚本与老的脚本进行对比，不一致的地方进行解析与重新渲染，如果产生行数变化 (删除多行)，则进行全部渲染。

MSE_Painter.js 变量存放图

```
var varr = {
    codes: "",
    QNo: [],        //单音轨-行号s
    Q1No: [],       //多音轨-音轨1-行号s
    Q2No: [],       //多音轨-音轨2-行号s
    Q3No: [],       //多音轨-音轨3-行号s
    //=======================================
    oldCodes: "",
    oldQNo: [],     //单音轨-行号s_before
    oldQ1No: [],    //多音轨-音轨1-行号s_before
    oldQ2No: [],    //多音轨-音轨2-行号s_before
    oldQ3No: [],    //多音轨-音轨3-行号s_before
    setVarr2Old: function () {
        varr.oldCodes = varr.codes;
        varr.oldQNo = MSE.song.tracks.length > 1 ? varr.Q1No : varr.QNo;
        varr.oldQ1No = varr.Q1No;
        varr.oldQ2No = varr.Q2No;
        varr.oldQ3No = varr.Q3No;
        varr.codes = "";
        varr.QNo = [];
        varr.Q1No = [];
        varr.Q2No = [];
        varr.Q3No = [];
    }
}
```

MSE_Painter.js 脚本解析构想:

解析主要是通过对脚本进行分句，分行，分段，分音符的拆分与解析，识别每个音符有多少个符号，对其进行节拍、音高、时间的判断，然后重新写入 MSE.song 类中

MSE_Painter.js 渲染构想：

渲染方法截图：

```
/**
 * //通过JpCode绘制wps正文
 * @param {string} JPcode 简谱脚本
 * @param {number} CmdNo 如果单绘制脚本某一行,则该参数传入脚本行数,从1开始
 * @param {boolean} force 强制重新渲染，在全部脚本渲染时，不进行上次代码是否一致的比较，全部重新渲染
 */
function drawByJPcode(JPcode, CmdNo, force) {
```

QCode 为已确认为脚本行的单行代码，在使用过程中输入 Q 定义新的一行

脚本编辑器使用说明（目前支持的功能）：

[1234567] 为 do re mi fa...

‘ 为倍高音 ， 为倍低音 ’’ 双倍高 ,, 双倍低

/ 减时线 1/ 为 8 分音符 do 1// 为 16 分音符 do

```
//通过Qcode绘制Section
function drawQ(Qcode, lineNo, TrackNo) {
```

脚本编辑

```
#================MuBody================
Q:5 3/ 5/ 1' - |6 1' 5 - |5 1/ 2/ 3 2/ 1/ |2 - 0 0
```

编辑器截图

```
//通过section绘制note (小节代码,代码所在行序号,音轨序号,小节号)
function drawSection(sectionCode, lineNo, TrackNo, sectionNo) {
```

详细说明见 MSE_Painter.js 代码

3.4.6 多页面传递捕获思路【命令接收器】

其实，不管是 Piano，脚本编辑器 还是 MSE 这些页面都相当于是单独的模块

如何进行数据的传递捕获是个问题

我的方案是用 setTimeOut 与 PluginStorage 进行递归调用。

在这里我的作品设计到了一个叫做**命令接收器**的东西，它专门负责用来接受其他页面发送过来的命令并进行相应操作。

比如说，我给 MSE 工具设置了命令接受器，给 piano 与脚本编辑器设置了命令接收器，那么他们之间在对方进行动作的时候可以及时捕获并且接受。

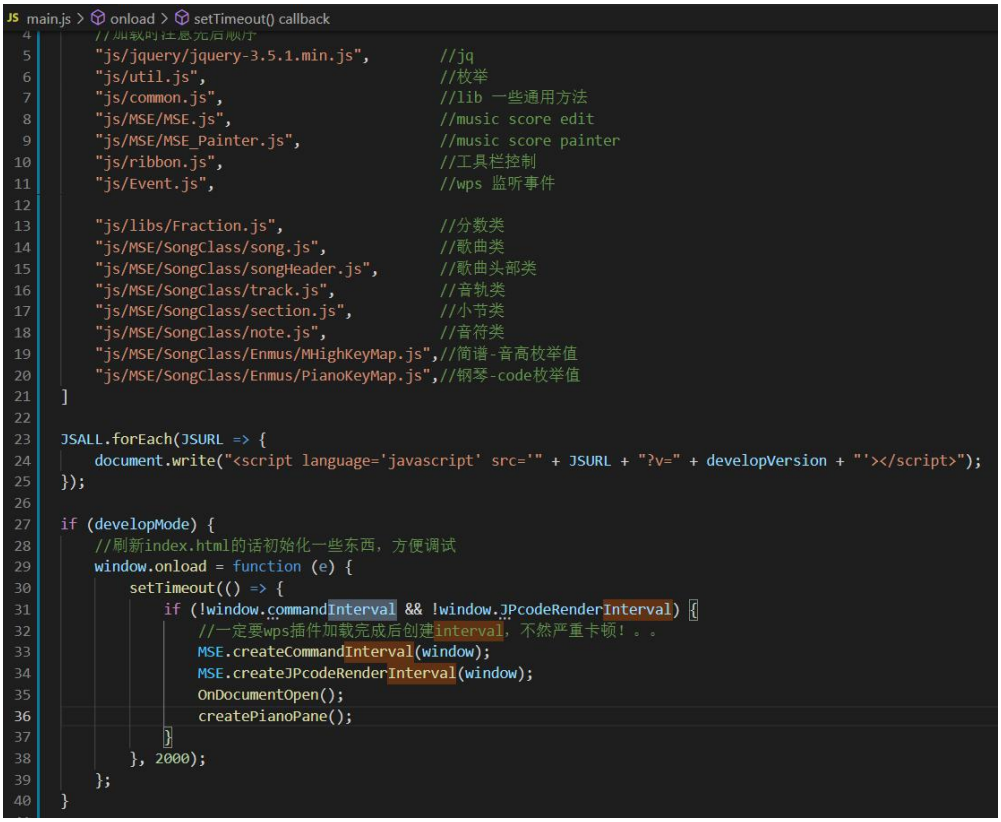
举个使用的例子，在脚本编辑器页面我设置了命令接收器，那么在乐谱收藏夹打开新的乐谱的时候，我就需要向脚本编辑器发送新乐谱的代码。

再举个例子，在脚本编辑器敲打新的代码的时候，我就会给 MSE 工具发送重新渲染的命令。

再再举个例子，当 MSE 调用播放的时候，其实是把 Song 类进行重新解析，计算各个音符的各种信息，解析为 piano 模块自定义的 Json 类型，然后转字符串放入 PluginStorage 中，并对 Piano 发送 “播放信号”，此时 Piano 去获取音乐，进行解析播放。

命令接收器代码截图

创建命令接收器



```
main.js > onload > setTimeout() callback
    "js/jquery/jquery-3.5.1.min.js",        //jq
    "js/util.js",                           //枚举
    "js/common.js",                         //lib 一些通用方法
    "js/MSE/MSE.js",                        //music score edit
    "js/MSE/MSE_Painter.js",                //music score painter
    "js/ribbon.js",                         //工具栏控制
    "js/Event.js",                          //wps 监听事件

    "js/libs/Fraction.js",                  //分数类
    "js/MSE/SongClass/song.js",             //歌曲类
    "js/MSE/SongClass/songHeader.js",       //歌曲头部类
    "js/MSE/SongClass/track.js",            //音轨类
    "js/MSE/SongClass/section.js",          //小节类
    "js/MSE/SongClass/note.js",             //音符类
    "js/MSE/SongClass/Enmus/MHighKeyMap.js",//简谱-音高枚举值
    "js/MSE/SongClass/Enmus/PianoKeyMap.js",//钢琴-code枚举值
]

JSALL.forEach(JSURL => {
    document.write("<script language='javascript' src='" + JSURL + "?v=" + developVersion + "'></script>");
});

if (developMode) {
    //刷新index.html的话初始化一些东西，方便调试
    window.onload = function (e) {
        setTimeout(() => {
            if (!window.commandInterval && !window.JPcodeRenderInterval) {
                //一定要wps插件加载完成后创建interval，不然严重卡顿！。。
                MSE.createCommandInterval(window);
                MSE.createJPcodeRenderInterval(window);
                OnDocumentOpen();
                createPianoPane();
            }
        }, 2000);
    };
}
```

简谱接收器：

例：图中为简谱接收器在获取版本号后进行重新渲染

```
//renderContent 渲染页面 通过JScode的版本号判断如果代码发生了改变重新渲染
createJPcodeRenderInterval: (w) => {
    w.JPcodeRenderInterval = () => {
        setTimeout(() => {
            w.JPcodeRenderInterval();
            if (!MSE.checkIsMuscr()) return;
            MSE.setValue("JPcodeChangeInfo", JSON.parse(wps.PluginStorage.getItem("JPcodeChangeInfo")));
            var codeMemory = wps.PluginStorage.getItem("JPcode");
            if (MSE.JPcodeChangeInfo.version == MSE.oldJPcodeChangeInfo.version)
                return
            // if (!codeMemory || codeMemory == "null")
            //     return
            console.log("JPcodeChangeInfo.version = ", MSE.JPcodeChangeInfo.version);

            //-----开始操作文档
            var app = wps.WpsApplication();
            app.ActiveDocument.Unprotect("passW0rd"); //先解锁然后操作

            MSE.setValue("JPcode", codeMemory);
            MSE.WriteCodeToFile();
            MSE.WriteSongToFile();
            drawByJPcode(MSE.JPcode, MSE.JPcodeChangeInfo.CmdNo, MSE.JPcodeChangeInfo.force);
            MSE.oldJPcode = MSE.JPcode;
            MSE.oldJPcodeChangeInfo = MSE.JPcodeChangeInfo;

            app.ActiveDocument.Protect(3, false, "passW0rd");
        }, 600);
    };
    w.JPcodeRenderInterval();
},
```

MSE 接收器：

图中为 钢琴播放完成后，新建歌曲，重置信息，弹出等待框 的命令

```
//MSEcommand 用来接收其他页面传递的命令
//_0、这里有个小bug window下如果有setInterval，同时初始化taskpane，图标加载就会很卡，且windows下无法移动开发者工具
//_1、为了工具栏更快加载，在main.js中延时触发该函数
//_2、2021-9-8 interval改为timeout方法 自定义interval
createCommandInterval: (w) => {
    w.commandInterval = () => {
        setTimeout(() => {
            w.commandInterval();    //一开始就得调用下次的timeout 别return了
            var command = wps.PluginStorage.getItem("MSEcommand");
            if (!command) return;
            MSE.clearMSEcommand();
            console.log("MSEcommand = ", command);

            var app = wps.WpsApplication();
            app.ActiveDocument.Unprotect("passW0rd"); //先解锁然后操作
            switch (command) {
                case "pianoFinish":
                    MSE.setValue("PlayStatus", 0);
                    wps.ribbonUI.Invalidate();
                    break;
                case "newSong":
                    MSE.resetMSE();
                    MSE.setValue("song", new fakeSong(JSON.parse(MSE.getValue("song"))));
                    MSE.song.Header.setValue2Wps(); //乐谱信息填充
                    //把Song类写入song书签
                    MSE.WriteSongToFile();
                    break;
                case "setSongHeader":
                    MSE.song.setHeader(new fakeSongHeader(JSON.parse(MSE.getValue("songHeader"))));
                    MSE.song.Header.setValue2Wps();
                    MSE.WriteSongToFile();
                    break;
                case "setEditorNull":
                    MSE.Editor = null;
                    break;
                case "alertWait":
                    MSE.alertWaitBox();
```

3.4.7 播放思路

前面已经讲到过，Piano 在接收到播放命令时会去 pluginStorage 中获取最新放脚本。

此脚本是点击播放时 MSE 进行重新渲染，生成的只有音符的数组。并且依据每个音符出现的时间进行排序。通过音符相加时长的相加，得到总体音乐的时长。播放时候的进度条就是这么来的。

钢琴的命令接收器（piano-simulator 中）

```
var commandInterval = () => {
  setTimeout(() => {
    commandInterval();
    var command = wps.PluginStorage.getItem("pianoCommand");
    // console.log("command =", command);
    if (!command) return;
    clearPianoCommand();
    console.log("command =", command);
    switch (command) {
      case "play":
        this.play();
        break;
      case "pause":
        this.pause();
        break;
      case "stop":
        this.stop();
        break;
      case "playMuscr":
        this.playMuscr();
        break;
    }
  }, 369);
};
```

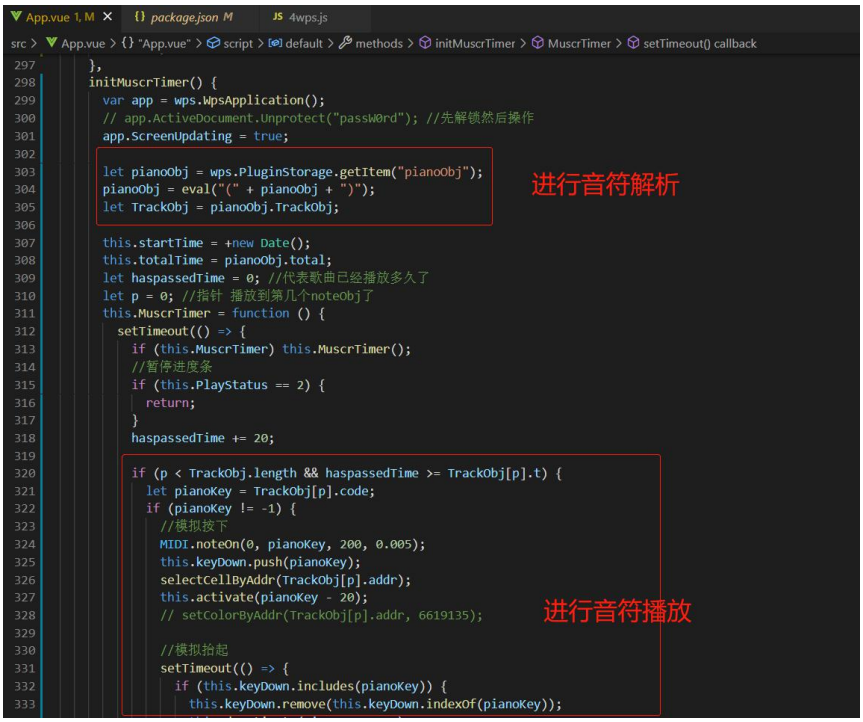
Piano - WPS 播放相关

```
App.vue 1, M   {} package.json M   JS 4wps.js
public > js > libs > JS 4wps.js > ...

// addr 所在表格位置   [1, [2, 2], 1, [2, 1]] 第一个表中2,2格子中的表1，里面的2,1格子的内容
export function setColorByAddr(addr, color) {
    var tab = wps.WpsApplication().ActiveDocument.Tables.Item(addr[0] + 1).Cell(addr[1][0], addr[1][1])
        .Tables.Item(1);
    setColorByColumnsNo(tab, addr[3][1], color);
}

/**
 * @param {object} tab 表格
 * @param {number} colNo 列号
 */
function setColorByColumnsNo(tab, colNo, color) {
    let colCells = tab.Columns.Item(colNo).Cells;
    for (let i = 1; i <= colCells.Count; i++) {
        colCells.Item(i).Shading.BackgroundPatternColor = color;
    }
    tab.Columns.Item(colNo).Select();
}

export function selectCellByAddr(addr) {
    var tab = wps.WpsApplication().ActiveDocument.Tables.Item(addr[0] + 1).Cell(addr[1][0], addr[1][1])
        .Tables.Item(1);
    tab.Columns.Item(addr[3][1]).Select();
}
```



3.4.8 更多

到这里差不多就是整个插件的构成啦，想了解更多就看看源码吧~