**《阮中楠的RPGMV开发笔记》**

作者：阮中楠（B站同名UP主）

1. mv项目的技术拓展
2. 成品游戏的安装包生成与打包

先**部署**文件，部署完成后，再对部署文件进行“**打包**”。这里说的打包并不是简单地压缩文件，而是用专业的软件对其进行整合压缩，最后生成专业的文件[[[1]](#footnote-0)]。做好的文件打包后，解压以后，才可以生成eve格式的游戏打开端口。

1. RPGMV的前沿技术

迈向未来

1. RPGMV 1.7.1

看到好像有些朋友还不知道，我来发一个pixi5的升级补丁吧，直接覆盖即可，此补丁将mv的pixi版本从4.5.4提升到5.3.8，pixi从v5版本用上了webGL2.0，我个人觉得是MZ流畅提升的关键，群里已经有一个“RPGMV 1.7.1”的民间补丁，那个补丁使用的pixi版本是5.3.3，也是可以的，但是存在一点BUG，所以还是更推荐用这个  
我不是作者，这是原作者的链接  
https://github.com/kins-dev/rpgmaker-mv-pixi5

之后强烈建议再升级一下mwjs版本，mv默认的版本已经是远古时代的了，chrome版本太老旧了，这个可以直接去nw.js官网下载，然后替换掉mv目录下的nwjs-win和nwjs-win-test里面的旧版本（注意备份），然后只需要把nw.exe改名为game.exe即可，如果嫌弃nwjs不够稳定，还可以研究一下用electron发布你的游戏，也不是很复杂

1. 基于JavaScript的RPGMV项目插件编写
2. 自定义一个窗口

如何直接做一个窗口类[[[2]](#footnote-1)]？调用重写了很多构造函数。这是无插件指令的写法。

Scene\_Splash.prototype.create 方法、Scene\_Splash.prototype.start 方法、Scene\_Splash.prototype.stop 方法、Scene\_Splash.prototype.terminate 方法、Scene\_Splash.prototype.update 方法等（当然除了构建器部分外，其它方法都是可选的）。下面就在Scene\_Splash类中重写这些方法。

图像：想把图像放到正中间，可以考虑this.centerSprite()方法[[[3]](#footnote-2)]。

1. 在Sprite类及其子类添加图片

一般我们在Sprite类的create方法内写导入图片，我们要先用this的方式来添加一个新的变量作为导入的图片，然后用导入图片的方法导入图片即可。随后Sprite类就会出现这个图片。为了美观，可以在create方法内部顺便定义好图片的坐标值。

为什么要用this的方式添加变量而不是用var的方式呢？作者做了一个实验，发用this的方式定义的新变量才可以正常使用。当前无法解释其原理。

我们在导入图片的时候，还要使用到addChild方法，将图片正式添加到Sprite类中，这样才可以把局部变量变成全局变量。

1. TilingSprite满版精灵类的使用——实现图像的移动

《针对SIAKO.Mobi 14节课——满版图像》

TilingSprite类是pixi源码里的类，称为“满版精灵类”。为了使得导入的图像可以持续的移动，我们要先给图像确定中心，然后才能移动。

用TilingSprite.prototype.move方法来确定图像的位置和大小，其中，大小的参数我们通常会用Graphics.width和Graphics.height的方式。其本质是Graphics静态类调用图像的width和height属性。

我们在TilingSprite对象的所在场景的update方法[[[4]](#footnote-3)]内，通过调整此对象的origin成员变量附带的x，y坐标值，来实现TilingSprite对象在场景内部的坐标移动。我们之所以使用origin成员变量是因为origin在TilingSprite中是专门使用与图像移动的一个坐标控制点。

在网课中，专门提到了move方法和origin成员变量的相互使用，这两个用法必须同时使用。暂时无法解释其必要性。

1. 在已有的菜单栏中创建新的窗口——写一个自己的窗口类

《SIAKO.Mobi15节课——在菜单栏窗口中创建一个新的窗口[[[5]](#footnote-4)]》

主要的算法是，在已知的“窗口场景”的内部里面，再添加一个新的窗口。

在Scene\_Menu.prototype.createCommandWindow方法内部，添加新的窗口。输出数据的方式是使用Window\_Base窗口基类的drawTextEx方法来输出一行字体。可以输出字符串。数据的输出方式也是当做是字符串输出。如果我们要把数值当做字符串输出时，我们可以用String()方法来进行类似于其他语言的“强制类型转换”。输出字体一般写在场景类的refresh方法内部。

网课里面用的是重写Window\_Gold类，最重要的是重写一个窗口类的以下方法：定义、initialize、windowWidth、windowHeight、refresh[[[6]](#footnote-5)]。我们写的窗口类是以Window\_Base作为父类的，而不是选择Window类。现在无法解释。

1. 创建一个新的场景类

《SIAKO.Mobi16节课——新建场景，嵌入窗口》

主要算法：先新建一个场景类，然后再把窗口显示在场景内部。SIAKO.Mobi指出，不可以用addChild方法代替addWindow方法。但是在我们自己的实验中，是可以实现互换的。

我们新建的一个场景类，是用Scene\_MenuBase作为父类的，试了一下用Scene\_Base作为父类，发现背后的场景是全黑的。并不是原来的半透明效果。

试了一下用Stage类作为父类，发现直接报错。说明Stage类并没有具体实现一些方法。

新建的场景类主要重写Scene\_Menu菜单场景、继承于Scene\_MenuBase菜单基类场景。因为我们需要使用到菜单场景自带实现的半透明背景图，所以一般会选择Scene\_MenuBase作为基类。主要重写的方法有：定义、initialize、create、start。如果需要场景刷新，可以看情况选择重写update和refresh方法。

1. 建立自己的游戏启动场景

《SIAKO.Mobi17节课——建立自己的游戏启动场景》

首先要建立一个新的场景，主要重写Scene\_Gameover场景类。重写的方法有：initialize、create、start、update、stop、terminate。

在create方法中、这个场景需要开始执行播放背景音乐，创建启动界面图。故需要在自建的Scene\_Splash场景类添加playSplashBGM和createSplashSecneImage方法。

在playSplashBGM方法中，要先关闭背景音乐。故需要AudioManager.stopBgs()和AudioManager.stopMe()方法。随后开始声明音乐类型。实际上完成播放的核心方法仅有 AudioManager.playBgm方法。AudioManager.stopAll()方法停止全部的音频工作。

createSplashSecneImage方法给Scene\_Splash类添加一个新的图片精灵对象。1：新建一个Sprite类精灵类图片对象。2：用ImageManager.loadSystem方法导入图片。3：用addChild将图片添加进入Scene\_Splash类。

在update方法中，需要监听用户的输入流。因此需要重写isTriggered方法。使之可以监控键盘和屏幕的输入流。update方法需要转移场景到Scene\_Title标题场景类内部。

其余的大部分代码都不需要重写。只需要保留在类中即可。他们会自动执行。

1. 更改一个窗口的背景图片，并设置其位置、大小、透明度等参数

这里会有一个误区：“我们能修改菜单窗口背景图，其他的窗口都不能修改其背景。”这个误区一直“指导”我们来修改Scene类的背景图。我不记得这个误区是不是从SIAKO.Mobi的视频里面学来的，反正我印象深刻。凡是一个Scene都可以通过addChild方法来添加一张图片。注意，我们只是添加了一张图片。Scene可以使用addChild方法，是毋庸置疑的。核心证据是最深层的Stage类实现了PIXI的Container容器类。所以Container类的子类均可以使用addChild方法[[[7]](#footnote-6)]。用这种追根溯源的方式，我们甚至可以看到任何的窗口类都可以使用addChild的证据[[[8]](#footnote-7)]以及精灵类可以使用addChild的证据[[[9]](#footnote-8)]。

我们直接修改的对象是来自Window基类的\_windowBackSprite变量。\_windowBackSprite表示的就是一个窗口的背景，这个背景就是以一个Sprite精灵的方式来描述的。我们只需要给这个精灵对象赋值即可。

但是这里的赋值又会出现一个问题，我们不能new一个新的Sprite来直接代替好这个\_windowBackSprite。而是要在\_windowBackSprite.bitmap里面，添加位图对象。**给精灵的位图添加对象，添加位图，而不是精灵。**目前我无法解释，为什么要这样写，反正程序就是达不到效果。具体的写法可以如下：

var backImage\_Bitmap = ImageManager.loadSystem('2020\_temp');

this.\_windowBackSprite.bitmap = backImage\_Bitmap;

现在一般默认用ImageManager. 图片管理者的方式来导入位图。这个方法是最稳妥的。在上面的教程中已经提及到过。

一般我们是先采取导入图片，再对Sprite对象进行调整。mv的Sprite类继承并重写了来自PIXI的Sprite类。从源码易知，可以更改tint、alpha、scale等参数。scale参数是修改一个Sprite对象大小的核心要素[[[10]](#footnote-9)]。不能直接用width和height的方式来直接修改一个Sprite对象。这个没有效果。

我更愿意用alpha来代替使用opacity。因为opacity的本质是修改一个Sprite的透明度，其实就是封装了alpha变量[[[11]](#footnote-10)]。这个用法是直接调用PIXI的定义的。

***待检验：用setFrame方法来实现导入图片的大小修改。（本方法由QQ ： 2335937510 提供）***

1. 利用$gameActors输出一个角色所具有的技能

严格来说，人物角色和技能是被专门地放在两份.json数据库文件的。我们肯定要找到人物技能和人物之间的关系的。按照这种思路，我们肯定要用人物来输出人物附属的技能。

在DataManager类中，明确说明了$gameActors是Game\_Actors类的成员。

***$gameActors = new Game\_Actors();***

接下来在Game\_Actors类中，我们发现其本质就是包装了Game\_Actor类，其中的Game\_Actors.prototype.actor方法告诉我们了如何使用这个$gameActors来访问具体存放角色数据的方式——在其形参内填写角色的编号。值得一提的是，我们要通过具体的调试才能准确判别角色编号。有可能会出现数组下标为0时，所指向的并不是第一个角色的情况。

***Game\_Actors.prototype.actor = function(actorId) {***

***if ($dataActors[actorId]) {***

***if (!this.\_data[actorId]) {***

***this.\_data[actorId] = new Game\_Actor(actorId);***

***}***

***return this.\_data[actorId]; }***

***return null; };***

在Game\_Actor类中，我们可以使用其Game\_Actor.prototype.skills方法来输出其技能。在这里，$gameActors调用到了$dataSkills。易知，skills方法输出的是这个角色的全部技能，这个技能是以一个数组的形式输出的。

***Game\_Actor.prototype.skills = function() {***

***var list = [];***

***this.\_skills.concat(this.addedSkills()).forEach(function(id) {***

***if (!list.contains($dataSkills[id])) {***

***list.push($dataSkills[id]); }***

***});***

***return list; };***

综上所述，输出一个角色的核心代码写法为：$gameActors.actor(1).skills()[1].name 这里的name表示的是技能名而不是角色名。注意数组下标的选取。易知，这种写法类似于一个二维数组。

值得说明的是，直接调用.json数据库的方式是$gameActors[index]，直接把“$gameActors”当做是一个数组名，访问数组内容，即数据库内部的内容。而不是一个对象来访问其成员方法。

1. 基于JavaScript ES5 语言版本与“开闭原则”的MV代码编写
2. 本章前言与阅读建议

本节写的代码，更多的是写一个类，一个系统，一个超脱于插件的代码群。这里会给出一些写代码的设计技巧。

1. 读者的身份转换声明

此时的读者已经不再是那种依赖于写插件的人了，应该多多少少意识到插件编写方式在软件工程中的不规范性。该考虑插件导入原理和index.html文件中各个.js文件的导入顺序原理了。

此时的读者不应该直纠结于插件的效果，还应该深究各种mv源码运行原理。

1. 用css实现动态变化（待细化）

GameCanvas

使用:hover伪类来实现一个动态变化。使用css的id选择器来针对性的完成变化。

在mv项目中查到的，html元素的id号：

GameCanvas

UpperCanvas

1. MV源码原理理解与工作原理解释
2. DataManager.loadDataFile方法原理解释

我们的json文件是怎么加入到项目中的？其本质是运用了XMLHttpRequest类[[[12]](#footnote-11)]的方法。CSDN大多数的文章都简单地介绍了XMLHttpRequest类的使用方式，DataManager.loadDataFile方法的流程也几乎如出一辙。基本上的流程是：

用XMLHttpRequest.open方法来打开json文件。

用XMLHttpRequest.overrideMimeType方法指定导入文件的类型为json。

用XMLHttpRequest.onload方法来显性地写一个函数，并接受XMLHttpRequest.responseText属性的返回值。

用[XMLHttpRequest.send](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest/send)方法来收尾。

值得说明的是，上述的全部流程都是严格的，先写open，在写send；使用onload方法时，还运用了window全局变量，mv很多的全局变量都被保存与此；json文件的本质是字符串形式的JavaScript对象，要用JSON.parse[[[13]](#footnote-12)]方法来将字符串转换成具体的对象。

1. 人物对话的消息窗口是怎么控制的

Game\_Interpreter.prototype.command101方法设置了人物对话的脸图、和说话内容。

1. 对项目中出现的canvas标签的理解

（学习方式：当前部分的内容以探究为主，尝试实现对不同位置的，已知的canvas标签进行修改。）

1. Graphics图像处理静态类中的canvas

主要是其成员变量\_upperCanvas和\_canvas。\_upperCanvas的本质是上层画布，**而\_canvas暂时理解不了**。

\_upperCanvas上层画布的值要用style的方式来调用。值得注意的是，其本质是HTMLElement.style 属性并返回一个[CSSStyleDeclaration](https://developer.mozilla.org/zh-US/docs/DOM/CSSStyleDeclaration" \o "DOM/CSSStyleDeclaration)对象，不是一个可以修改的canvas。[[[14]](#footnote-13)]。

值得区别的是：

Graphics.\_upperCanvas是canvas；Graphics.\_upperCanvas.style是[CSSStyleDeclaration](https://developer.mozilla.org/zh-US/docs/DOM/CSSStyleDeclaration" \o "DOM/CSSStyleDeclaration)对象。

主要的修改方式示例：Graphics.\_upperCanvas.style.backgroundColor = 'rgb(255,0,0)';

backgroundColor在CSS中的写法是：background-color，CSS和JavaScript之间的样式更改是有区别的[[[15]](#footnote-14)]。

1. Bitmap位图类的canvas

相关的只有\_canvas变量。Bitmap类的本质是canvas标签。canvas便签的写法几乎都被封装了[[[16]](#footnote-15)]。

UP主：赤瞳大白猫：

“有些bitmap的canvas属性是null

bitmap是这样的，如果你提供图片的文件地址，他只是一个浏览器的img标签元素

但如果你一旦准备在上面绘图，他就会自动创建canvas”

关于bitmap本质的东西暂时跳过。有争议。○

1. Sprite精灵类、WindowLayer类的canvas

本质是还是运用了bitmap自带的canvas。基本上没有太大的操作空间。

1. bitmap.x的写法误区原理解释

比如this.\_windowBackSprite.bitmap.y的写法事实上是错误的，在断点调试时，总是查到其变量未定义。bitmap的本质是一个canvas标签。这个标签尚未说明其具体位置。bitmap通常是依赖于sprite的，精灵sprite有坐标值，而bitmap本身没有坐标值。

1. makeCommandList方法所设置的“命令名”和“命令关键字”的保存位置 ？

首先，makeCommandList方法来自于Window\_Command类，起核心作用的是Window\_Command的addCommand方法。

1. window系的opacity变量的本质

这里指的是window基类定义的opacity变量，opacity变量的本质是 \_windowSpriteContainer.alpha ，“窗口精灵容器”的透明度。只要修改了opacity的值，就修改了窗口的框和其背景的透明度[[[17]](#footnote-16)]。

其中，窗口精灵容器\_windowSpriteContainer包括了：\_windowBackSprite和\_windowFrameSprite ，即窗口背景精灵和窗口框精灵。

读者可以探究以下的问题来加深对此内容的理解：为什么Window\_MapName窗口可以让其窗口框及背景为空？使得我们每次看到地图名称时，都看不到窗口的边框和背景？

1. window系的contentsOpacity和contents变量的本质

这两个变量都和\_windowContentsSprite，“窗口内容精灵”相关。contentsOpacity是精灵的透明度、contents是精灵的位图bitmap。

Window\_MapName窗口的背景绘图就是在contents内部绘制的，其渐变的刷新效果也是控制contentsOpacity变量。

1. Window\_Base.\_dimmerSprite变量设计的意义

这个变量设计的意义是，让窗口的背景和其框统一变成由\_dimmerSprite控制的灰色渐变样式。，让dimmerSprite的层级覆盖过this.\_windowContentsSprite窗口内容精灵，进而实现“替换”窗口背景和窗口边框。

我们可以探究以下这个问题：

Window\_MapName.prototype.drawBackground和Window\_Base.prototype.refreshDimmerBitmap的效果区别问题？

首先，Window\_Base.prototype.refreshDimmerBitmap方法修改的是this.\_dimmerSprite.bitmap，

而Window\_MapName.prototype.drawBackground方法修改的是this.contents即this.\_windowContentsSprite。由这二者的区别，我们可以提出：为什么要设置Window\_Base.\_dimmerSprite这个变量？绘制阴影为什么要专门设置该变量？

\_dimmerSprite和其他常见的窗口精灵不同，他被首次定义在Window\_Base.initialize方法内部，是Window\_Base定义了它。

***Window\_Base.prototype.initialize = function(x, y, width, height) {***

***Window.prototype.initialize.call(this);***

***......***

***//模糊精灵 = null***

***this.\_dimmerSprite = null;***

***};***

真正使其加载进入窗口的是Window\_Base.showBackgroundDimmer方法。这个方法不仅加载\_dimmerSprite，而且还实质性地设置了具体的bitmap值，使得窗口出现竖直的灰色渐变。

***Window\_Base.prototype.showBackgroundDimmer = function() {***

***if (!this.\_dimmerSprite) {***

***this.\_dimmerSprite = new Sprite();***

***this.\_dimmerSprite.bitmap = new Bitmap(0, 0);***

***this.addChildToBack(this.\_dimmerSprite);***

***}***

***var bitmap = this.\_dimmerSprite.bitmap;***

***if (bitmap.width !== this.width || bitmap.height !== this.height) {***

***this.refreshDimmerBitmap();***

***}***

***this.\_dimmerSprite.visible = true;***

***this.updateBackgroundDimmer();***

***};***

Window.addChildToBack方法才将\_dimmerSprite正式并归进入窗口内部。他并归的是一个window的\_windowSpriteContainer，即“窗口精灵容器”的后面一位。这个代码让\_dimmerSprite盖过了\_windowSpriteContainer，所以我们才看不到\_windowSpriteContainer所包含的窗口背景及窗口框。

***Window.prototype.addChildToBack = function(child) {***

***var containerIndex = this.children.indexOf(this.\_windowSpriteContainer);***

***return this.addChildAt(child, containerIndex + 1);***

***};***

1. 待整理的原理

今天写代码的时候遇到了很多前所未有的挑战，懂得了一下的经验：

Graphics和SceneManager的初始化问题：

你只有执行了main.js，你才可以使用Graphics的方法，而且只要是用了Graphics，SceneManager也一并启动了。因为Graphics的初始化就是用SceneManager来完成的。因此在写代码报错时，要注意此时是否已经执行到main.js了，如果没有运行到main.js的SceneManager，就不能使用Graphics系列的静态方法。

SceneManager.run方法无条件获取错误的特点：

只要是被SceneManager运行过的代码，自己单独写的throw error就一定会被其catchException方法所捕捉，然后后续输出的e.name是无法被更改的，这就意味着，我们做不了自定义错误类的name属性。

try-catch语句的特性：

真正使程序彻底停下来的，不是throw error，而是try-catch语句。当程序有多个错误时，mv提示界面只显示了一个错误的原因是，try-catch语句会立刻让程序停下来。

1. 其他人的一些随笔说明

Drill\_up：

“窗口和贴图都是同源的，窗口就是一种组合式贴图，只不过窗口被rmmv封装了一层，把特殊文字、窗口皮肤、下一步指针封装到一起了，而且还不能拆出来。这里比较麻烦的是，如果要显示简单文字，贴图可以直接drawtext，但是如果要显示特殊文字，包括多行文本、不同颜色不同大小的字符，都必须贴个窗口，这就显得比较臃肿，虽然不会用上……”

1. 基于ES2015的mv插件开发

首先，这里说的“ES2015”指的是“ES6”，即现代JavaScript，并不是前文多次提到的“ES5”。在这一章节中，我们会广泛地使用ES6的新特性和语法糖来开发mv插件。我建议先了解一下ES6的新特性，再来阅读此部分[[[18]](#footnote-17)]。

1. 类编写的规范

注意这一段代码：

this.name = name || "";

严格来说，这并不是JavaScript的要求，但这几乎是开发者的惯用写法。在对类的属性初始化时，我们用这种手段保证代码一定会初始化而不会出现undefined的情况。

1. 如何用ES6的类语法糖来继承prototype的“函数类”？

想要使用ES6开发插件，必须会遇到如何使用ES6来继承源码的prototype函数类，MDN的class参考给了我们一个具体的示例[[[19]](#footnote-18)]：

function Animal (name) {

this.name = name;

}

Animal.prototype.speak = function () {

console.log(this.name + ' makes a noise.');}

class Dog extends Animal {

speak() {

super.speak();

console.log(this.name + ' barks.');

}}

var d = new Dog('Mitzie');

d.speak();//Mitzie makes a noise. Mitzie barks.

我们可以给出这样的准则：

1. 在class继承的子类中，在构造函数内都使用super来调用父类的属性。否则子类无法使用this来调用父类的属性。
2. 如果重写的方法在功能上和父类相同，就使用super，否则不使用super。这种写法相当于.call(this)。
3. ES6类的继承写法

https://developer.mozilla.org/zh-CN/docs/Learn/JavaScript/Objects/Inheritance#inheritance\_with\_class\_syntax

请阅读上面的英文解释即可。

1. nodejs环境变量配置：

可能是每次Git自动更新，导致环境变量丢失，现在将阮中楠本机的配置环境变量的常用步骤记录于此：

自主检查配置是否成功的cmd指令：

npm -v

node -v

npm config get prefix

npm config get cache

先输入node，再输入require('cluster')

npm install axios

npm install jsdoc

用户变量 NODE\_PATH添加：

E:\NodeJS\_install

系统变量path添加：

E:\NodeJS\_install

在E:\NodeJS\_install目录下创建两个目录，分别是node\_cache和node\_global

用户变量 NODE\_PATH添加：

E:\NodeJS\_install\node\_global\node\_modules

在用户变量path中，将npm的值改成：

E:\NodeJS\_install\node\_global\

1. 值得被保留的一些代码写法

    /\*\*

     \* @author 阮中楠

     \* @method loadSetting\_RuanZhongNan

     \* @deprecated

     \* @description

     \* 导入阮中楠的开发者个人设置。这个方法用于导入阮中楠专属的设置控件。

     \*

     \* 主调方法：Scene\_Boot.prototype.create

     \*

     \* 算法：

     \* 在游戏开始装载时，与 DataManager.loadDatabase()方法并列地导入数据。

     \*

     \* 使用DataManager.loadDataFile的算法来导入。

     \*

     \* 教程：

     \* 主教程：https://blog.csdn.net/ryelqy/article/details/79279273

     \*

     \* 规范示例：https://xhr.spec.whatwg.org/#interface-xmlhttprequest

     \*

     \* API接口：https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest

     \*/

    static loadSetting\_RuanZhongNan() {

        var xhr = new XMLHttpRequest();

        //网址请求 打开( 'GET' , url位置) 这里写的是绝对路径。

        xhr.open('GET', 'js/plugins/customDateFile/setting\_RuanZhongNan.json', true);

        //设置导入数据的数据类型

        xhr.overrideMimeType('application/json');

        //导入数据 这个写法是固定的。

        xhr.onload = function () {

            //我们的全局变量 $settingRZN 都被window全局变量所保存。

            window.$settingRZN = JSON.parse(xhr.responseText);

        };

        //发送请求

        xhr.send(null);

    }

1. 待研究的光追技术

YEP\_GridFreeDoodads.js

FilterController.js

ParticleEmitter.js

https://sigmasuccour.itch.io/false-server

遊戲這裡下載

1. 待学习的mv新版pixi更新包

在官方论坛里搜pixi就可以找到了，本体在itch

1. 待学习的jsdoc2md技术

这个技术可以将jsdoc转变成md。

<https://github.com/jsdoc2md/jsdoc-to-markdown>

<https://github.com/jsdoc2md/jsdoc-to-markdown/blob/master/docs/API.md>

1. abs即时战斗系统的研究。
2. L2d

spine db spriter live2d 骨骼类的mv都有成熟支持啦 可以去官坛找下

1. [] <http://www.k73.com/glzq/217985.html?tdsourcetag=s_pcqq_aiomsg> [↑](#footnote-ref-0)
2. [] 简书.鳗驼螺.【RPG Maker MV插件编程】【实例教程2】制作一个启动画面 <https://www.jianshu.com/p/8b67041f02d5> [↑](#footnote-ref-1)
3. [] 鳗驼螺.4.标题画面 <https://www.jianshu.com/p/527a82a2fd6a> [↑](#footnote-ref-2)
4. [] update方法更新每一帧，而图像移动是更新每一帧的。故图像的更新写在场景类的update方法内。 [↑](#footnote-ref-3)
5. [] SIAKO.Mobi 15节 <https://www.bilibili.com/video/av23706785?p=15> [↑](#footnote-ref-4)
6. [] refresh方法是在更新的时候才调用。和update不一样。 [↑](#footnote-ref-5)
7. [] Scene\_Base.prototype = Object.create(Stage.prototype);说明了任何场景类均实现了Stage类。

   Stage.prototype = Object.create(PIXI.Container.prototype); 说明了Stage类实现了PIXI的容器类。 [↑](#footnote-ref-6)
8. [] Window.prototype = Object.create(PIXI.Container.prototype);说明了Window类实现了PIXI的容器类。 [↑](#footnote-ref-7)
9. [] @extends PIXI.Container 在PIXI源码内，这说明了Sprite类继承了容器类。 [↑](#footnote-ref-8)
10. [] width: {

    get: function () {

    return Math.abs(this.scale.x) \* this.texture.orig.width;

    },

    set: function (value) {

    var sign = utils.sign(this.scale.x) || 1;

    this.scale.x = sign \* value / this.texture.orig.width;

    this.\_width = value;

    }

    }, 说明了Sprite类的宽高属性本质上就是scale和其材质texture的操作。 [↑](#footnote-ref-9)
11. []

    Object.defineProperty(Sprite.prototype, 'opacity', {

    get: function() {

    return this.alpha \* 255;

    },

    set: function(value) {

    this.alpha = value.clamp(0, 255) / 255;

    },

    configurable: true

    }); 说明了opacity的本质就是封装了Sprite的alpha值。 [↑](#footnote-ref-10)
12. [] MDN. XMLHttpRequest <https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest> [↑](#footnote-ref-11)
13. [] 菜鸟教程.JavaScript JSON.parse() <https://www.runoob.com/js/javascript-json-parse.html> [↑](#footnote-ref-12)
14. [] MDN. HTMLElement.style <https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/style> [↑](#footnote-ref-13)
15. [] MDN. CSS Properties Reference <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference> [↑](#footnote-ref-14)
16. [] CSDN.学习HTML5 Canvas这一篇文章就够了 <https://blog.csdn.net/u012468376/article/details/73350998> [↑](#footnote-ref-15)
17. []

    Window.prototype.\_createAllParts = function() {

    ......

    this.\_windowSpriteContainer.addChild(this.\_windowBackSprite);

    this.\_windowSpriteContainer.addChild(this.\_windowFrameSprite);

    ......

    } [↑](#footnote-ref-16)
18. [] <https://babeljs.io/docs/en/learn> [↑](#footnote-ref-17)
19. [] <https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Classes> [↑](#footnote-ref-18)