**《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. Sprite类的形参理解

《针对“ 【SIAKO.Mobi】RPG Maker MV Plugin Scripting 脚本教学【渐进篇】 13节[[[4]](#footnote-3)]”和“ 【RPG Maker MV插件编程】【实例教程4】玩转标题画面 26 above”的导入图片示例的具体写法对比研究——Sprite类形参的写法》

我们按照SIAKO.Mobi的写法，写出了一下的代码，但是运行的时候总是出现“Cannot set property’bitmap’ of undefined”的报错：

this.titleFrontImage = new Sprite();

this.titleFrontImage.bitmap = ImageManager.loadTitle2(titleFrontImage);

this.addChild(this.titleFrontImage);

以下这段代码是可以运行的27 above。

var temp = ImageManager.loadTitle2(titleFrontImageName);

this.temp\_image = new Sprite(temp);

this.addChild(this.temp\_image);

我现在猜测，是在new一个Sprite类对象的时候，形参写错了，我参考了以下的源码得出的猜想：

//在Scene\_Title类的源码中。系统针对Sprite类形参的写法是，另外在new一个Bitmap类对象。其中，Bitmap类对象的参数为Graphics.width, Graphics.height

this.\_gameTitleSprite = new Sprite(new Bitmap(Graphics.width, Graphics.height));

//在Scene\_Title类的源码中。系统源码的在填写Sprite类的形参是直接填写一个图片。

this.\_backSprite1 = new Sprite(ImageManager.loadTitle1($dataSystem.title1Name));

现在更改了一下代码的写法，就通过了：

this.titleFrontImage = new Sprite( ImageManager.loadTitle2(titleFrontImageName) );

this.addChild(this.temp\_image);

我觉得，在new一个Sprite类对象的时候，形参应该要填写好具体的图像。而不是像SIAKO.Mobi那样不填写参数格式。我重新写好了一段代码，发现不是SIAKO.Mobi写法的问题，而是我的写法问题。

**先new一个Bitmap位图对选哪个，用ImageManager.方法导图，然后再把Bitmap位图对象填写形参至Sprite对象内。或者针对于现存的Sprite对象，直接对Sprite.bitmap进行赋值。**

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[[[5]](#footnote-4)]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[[[6]](#footnote-5)]》

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[[[7]](#footnote-6)]。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[[[8]](#footnote-7)]。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[[[9]](#footnote-8)]以及精灵类可以使用addChild的证据[[[10]](#footnote-9)]。

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[[[11]](#footnote-10)]。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[[[12]](#footnote-11)]。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. 本章前言与阅读建议

本节不写代码，只讲一些深层次的理论和理解。

1. 读者的身份转换声明

读者再在此应该积极地思考各种mv源码运行原理。

1. 通论

drill：简单来说rmmv就是一个封装好的盒子，在rmmv里面可以向调用外面的dom结构，但是在dom层，是没法接触到rmmv的内部结构的。

1. PluginManager.loadScript方法的原理解释

PluginManager.loadScript方法是插件管理者的本质。这解释了，我们的插件代码是怎么加入到index.html内部的。

核心原理：

将.js文件变成.html文件内部的一个标签内容，相当于添加类似于如下形式的标签：<script type="text/javascript" src="js/rpg\_windows.js"></script> 。

主要是运用了 var script = document.createElement('script'); 和 document.body.appendChild(script); 这两个核心代码。

1. 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[[[13]](#footnote-12)]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[[[14]](#footnote-13)]方法来将字符串转换成具体的对象。

1. SceneManager类常用方法原理的简要解释

（本部分要填写一些核心的工作方法，还有一些特别的变量。但不要写成SceneManager类的API接口。）

1. SceneManager类的主要功能、大概工作原理及核心执行方法：

SceneManager类是一个静态类，实现各个场景的“增删改查”操作。其中包括：新建、储存、切换、刷新、删除、截屏、退出、等基本操作。

还有另一种理解是，该类的本质是一个用数组这个“数据结构”实现的一个“栈”，对场景实现栈的“先进后出”操作。最典型的是其pop和push方法的实现。

1. SceneManager.\_stack

\_stack变量的本质是一个数组。是最能体现该类本质的一个变量。

1. SceneManager.goto

该方法主要功能是进入到一个场景。一般来说，用这个方法的前提是该场景已经存在。绝大多数情况下，我们使用该方法的目的仅仅只是想进入到另外一个场景内部而已。至于该场景的删除，则不是这个方法所考虑的事情。

该方法切勿和C语言的goto语句联系，二者毫无关系。

1. SceneManager.push

该方法的主要功也能实现场景的跳转，但是其本质是包含了SceneManager.goto方法。一般认为该方法是给数组栈\_stack进行赋值。该方法先完成对数组栈的入栈，再实现场景的跳转。

1. SceneManager.snap

该方法的主要功能是实现

1. SceneManager.update
2. SceneManager.updateMain
3. SceneManager.changeScene
4. SceneManager.updateScene
5. 菜单场景类为什么可以直接退回到地图

给出几个猜测：输入流、场景管理者、场景基类

\_previousClass

1. 针对SceneManager类的转场解释

SceneManager.goto(Scene\_Map);语句可以实现转换场景到指定的场景内部，这个功能可以用来控制场景转换。

1. 解释为什么每次打开Scene\_Menu时，其背景图都是当前的游戏界面以及半透明效果

我们现在进行源码分析，因为Scene\_Menu调用了Scene\_MenuBase的create方法。源码如下：

***Scene\_Menu.prototype.create = function() {***

***Scene\_MenuBase.prototype.create.call(this);***

***this.createCommandWindow();***

***this.createGoldWindow();***

***this.createStatusWindow();***

***};***

Scene\_MenuBase.prototype.create方法用了Scene\_MenuBase对象自带的一个createBackground创建背景方法。

***Scene\_MenuBase.prototype.create = function() {***

***Scene\_Base.prototype.create.call(this);***

***this.createBackground();***

***this.updateActor();***

***this.createWindowLayer();***

***};***

createBackground方法新建了一个Sprite精灵类对象，并用专门的SceneManager.backgroundBitmap方法来截取当前的游戏画面，并作为背景。

***Scene\_MenuBase.prototype.createBackground = function() {***

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

***this.\_backgroundSprite.bitmap = SceneManager.backgroundBitmap();***

***this.addChild(this.\_backgroundSprite);***

***};***

SceneManager.backgroundBitmap静态类方法返回自己的成员变量\_backgroundBitmap。

***SceneManager.backgroundBitmap = function() {***

***return this.\_backgroundBitmap;***

***};***

\_backgroundBitmap成员变量是一个新设定的值，这个值被SceneManager.snap方法赋初值。其中，SceneManager.snapForBackground方法还对这个位图进行了模糊效果，即调用了Bitmap.prototype.blur方法。Bitmap.prototype.blur方法就实现了每次开启菜单栏时，菜单背景都是半透明的效果。

***SceneManager.snapForBackground = function() {***

***this.\_backgroundBitmap = this.snap();***

***this.\_backgroundBitmap.blur();***

***};***

调用底层的Bitmap.snap方法来截屏。

***SceneManager.snap = function() {***

***return Bitmap.snap(this.\_scene);***

***};***

1. 对addChild()方法的理解以及与addWindow()方法的联系，addWindow()方法的必要性说明

在Stage基类和Window基类中，他们都是对PIXI.Container类的实现，在PIXI.Container中，都具有addChild()方法。所以绝大部分的用Stage类和Window类作为超类的子类都可以使用addChild()方法。

事实上，addWindow()方法其本质也是实现addChild()方法。因此，我们可以尝试全部使用addChild方法来代替使用addWindow方法，减少使用新的方法，减少记忆。源码如下：

***Scene\_Base.prototype.addWindow = function(window) {***

***this.\_windowLayer.addChild(window);***

***};***

但是我们最好还是使用addWindow方法。因为addWindow方法是建立在Scene\_Base类的\_windowLayer成员变量，只要控制了\_windowLayer变量，就可以对一个场景内部的全体窗口进行统一的控制了。这就是addWindow的必要性。

1. 场景转换原理的解释

SceneManager.goto方法的原理是在形参内填写一个类名，然后在手动新建一个新的对象，即完成了场景转换。

1. 可选窗口的“确定点击窗口行为”的工作原理解释——关于ok字符串的来龙去脉

Window\_Selectable.prototype.callOkHandler 方法调用了this.callHandler('ok'); 注意ok这个字符串。这个字符串更多是一种指示变量，用来引导一个确定行为。追根溯源，callHandler方法会执行ok字符串所对应的一个句柄函数方法。

那么ok字符串对应的函数句柄在哪里设置的呢？大部分是在场景类中被设置的。

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。[[[15]](#footnote-14)]。

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[[[16]](#footnote-15)]。

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[[[17]](#footnote-16)]。

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[[[18]](#footnote-17)]。

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. 对MV界著名开源框架——Drill，的一系列理解
2. 对Galv系列插件的理解
3. Galv\_QuestLog.js 任务插件 的理解

理解其前台和后台的制作流程。

1. 对Pixi.js的学习与研究，探索pixi与mv代码之间的联系
2. 初始化与本地服务器搭建（Live Server）

RPGMV源码运行的原理是在一个.html网页文件上运行的。现在我们尝试做一个.html文件的方式来导入相关的js脚本文件，进而实现整个游戏[[[19]](#footnote-18)]。

我们在直接打开RPGMV项目自带的index.html文件时，会出现文件导入失败。这是因为“文件协议”和“html协议”是有区别的。解决方式是两种：1：用mv软件来打开游戏文件。2：在自己的电脑里面，搭建本地服务器。我们肯定要选择后者。建立本地服务器。通过查找大量的资料得知，我们要建立的是“本地http://localhost”。

localhost的建立方法有很多，有的是用下载wamp来配置[[[20]](#footnote-19)]，有的是直接在电脑配置IIS[[[21]](#footnote-20)]，还有一种是在VScode中下载插件“Live Server”[[[22]](#footnote-21)]。在安装Live Server插件时，我并没有找到对应要求的修改.json文件，因为找不到。

在使用Live Server时，要注意用VScode的打开文件夹的方式来打开对应的index.html文件。Live Server默认打开名为index的文件。至此，我们才算是搭建好了服务器。

我们现在的目的不仅仅是为了打开RPGMV项目自带的index.html文件，而是写出另外一份导入了pixi框架的html文件。接下来的问题是导入pixi框架。教程里面默认是用外部导入的方式来实现的。我的方式是，导入现成的pixi文件。在这里，我们暂时不下载最新版本的pixi来作为我们的lib[[[23]](#footnote-22)]。

1. 创建画布、渲染器、与舞台

在编写对应的代码时，发现了两种不同的写法，一种需要写PIXI.Application，另一种需要写渲染器 (renderer)(另有说法称“画布”)PIXI.autoDetectRenderer。通过查找mv源码，并没有发现任何代码调用了PIXI.Application，但是有代码调用了PIXI.autoDetectRenderer。查找pixi，发现Application会调用到autoDetectRenderer。换句话说，autoDetectRenderer更加底层。我们选择autoDetectRenderer作为写renderer的方式。

1. 导入图片

接下来是导入图片。导入图片会涉及到几个复杂的概念。图片都是“材质/纹理texture”，材质不能直接体现，要用“精灵”来显示出来。接下来是创建精灵对象。1：用材质来创建精灵。2：用图片直接创建精灵。这些创建精灵的方式都是可以理解为导入图片。导图的方式是用PIXI.loader来导入，再用setup函数，来统一地导入图片。

但是我发现了更加简单清晰的导入图片方式：

var sprite = new PIXI.Sprite.fromImage('assets/image.png');//直接由图片创建[[[24]](#footnote-23)]

这个方式有点问题。现在不得不学习如何用材质的方式、.json的方式来导入图片。但是这个方式决定了我们必须使用图块组的方式来进行导图。

接下来的工作，应该是把时间花在了解mv源码上。

plugin.name = YEP\_CoreEngine

plugin.parameters 本质上是一个数组。就是插件参数。

1. 把mv源码从ES5版本调整到ES6版本

在阅读源码时，我们注意到ES5版本的JavaScript在编写类的语法相当繁琐，我们考虑用一些比较特别的方式让mv变成ES6。

这里使用（）的工具来实现转写。

进度记录：

目前采用的项目：

E:\RPGMV工作域\project\proj-2-change-to-ES6-bySelf

git仓库路径：

https://gitee.com/HechiCollegeComputerAssociation/change-rpgmv-to-es6

开始逐步修改。

更改写法。

模块化开发的代码，仍然要手动的引入script标签。这样才能便于重构与修改。

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[[[25]](#footnote-24)]。

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[[[26]](#footnote-25)]：

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. 重构准则

在模块化开发下，这些代码应该这样重构：  
  
1：成员变量初始化写法的重构  
old： this.initialize.apply(this, arguments);  
new：super();  
  
这行代码的本质是实现变量的成员变量的赋初值，在constructor构造函数内调用super();即可初始化来自父类的成员变量。  
  
2：父类方法调用的重构  
old：Scene\_Base.prototype.create.call(this);  
new：super.create();  
  
这行代码的本质是调用来自父类的方法，并在调用父类方法的时候，绑定子类的this对象，实现子类对象调用父类方法。用类似于Java的super关键字即可调用来自父类的方法。

1. 问题堆栈与sundry：插件开发的几个实际问题？
2. 插件开发问题

那这样的话，就可以去看事件源码，看人物和队伍类了

比如说，怎么用代码显示当前的战斗队伍

怎么用代码给角色的最大生命增加50

怎么用代码给背包增加一件装备，或者减少

写装备强化啊，装备套装啊，装备镶嵌啊插件

WindowLayer {\_events: Events, \_eventsCount: 1, tempDisplayObjectParent: null, transform: TransformStatic, alpha: 1, …}

children

\_paramPlus

项目是怎么导入window.png的？又是如何运用内部图片的？懂了。

1. canvas标签id查找问题

当前的观点是，每一个mv对象都是canvas标签，怎么用JavaScript的方法准确的找到当前canvas标签的id？并让其执行指定的CSS样式？

答案：Graphics.\_upperCanvas的标签id是upperCanvas。

1. 精灵和其附属位图的宽高值是那个代码负责的？

this.\_windowBackSprite.width与this.\_windowBackSprite.bitmap.width的值是一样的。是哪个代码产生这样的结果的？

答案：是Sprite.prototype.\_onBitmapLoad方法。

Sprite.prototype.\_onBitmapLoad = function() {

    if (this.\_frame.width === 0 && this.\_frame.height === 0) {

        this.\_frame.width = this.\_bitmap.width;

        this.\_frame.height = this.\_bitmap.height;

    }

    this.\_refresh();

};

这个方法让精灵和其附属位图的宽高值相等，在导入位图的确认时，框定好这个精灵容器的框架大小。

drill：这个问题其实你没答到关键点，js是用的浅拷贝，也就是指针，width指向了bitmap的width，所以是同一个值

有待完善○

1. 为什么Window\_MapName窗口可以设置其content（bitmap）的颜色渐变并让其窗口框及背景为空？

（开始探究）

查询的内容：

open方法。

1. 为什么Window\_MapName窗口的底层灰色内容不会遮挡到地图的名称？
2. 不同mv启动方式对存档调用的问题

用VScode+Chrome、VScode+360浏览器、rmmv这三种启动方式都遇到了不同形式存档问题。

目前唯独rmmv的启动方式可以调用全部的存档。

问题初步排查：目前怀疑的是Window\_TitleCommand.prototype.makeCommandList方法。因为这个方法会让“继续游戏”这个字段变成灰色。

***Window\_TitleCommand.prototype.makeCommandList = function() {***

***this.addCommand(TextManager.newGame,   'newGame');***

***this.addCommand(TextManager.continue\_, 'continue', this.isContinueEnabled());***

***this.addCommand(TextManager.options,   'options');***

***};***

在这个方法中，就开始牵扯到数据管理者的方法了。

Window\_TitleCommand.prototype.isContinueEnabled = function() {

    return DataManager.isAnySavefileExists();

};

显示灰色的主要原因是下面的isAnySavefileExists方法返回了假值。可能是globalInfo数组根本就什么内容都没有。

DataManager.isAnySavefileExists = function() {

    var globalInfo = this.loadGlobalInfo();

    if (globalInfo) {

        for (var i = 1; i < globalInfo.length; i++) {

            if (this.isThisGameFile(i)) {

                return true;

            }

        }

    }

    return false;

};

通过深入的断点调试，目前到了这个可以的方法：

在存档问题中，其他人的观点：

奋斗小乐：

使用NW.js的时候就使用它的接口保存在本地上。一般来说，浏览器就没有本地文件的操作权限。你网页不保存localStorage，那还想保存哪里。操作本地文件的权限很可怕的，可以直接让你电脑中毒。mz使用了localforage库，你也可以用这个库

赤瞳大白猫：  
 localforage会先检查浏览器是否支持indexDB。不支持就用localstorage

在StorageManager类中被广泛使用的localStorage变量，在localforage.js重写了。localStorage变量用于存档，将文件变成json格式。

出现在StorageManager类的字符串压缩算法方法。

LZString.compressToBase64

从源码中，我们发现decompressFromBase64方法似乎是一个对应的解压方法。这个方法是否运用到了mv中？是的。

现在的解决思路是，自己搭建一个数据储存的过程，将网络数据储存到本地中。

1. 使用被JSDoc的@private标签修饰过的方法，可能降低运行速度的问题
2. 14.1.2小优任务插件，代码尚未运行

报一个bug

VScode+Chrome浏览器的方式打开游戏，会出现$gameSystem.appendListener方法不存在的错误。通过断点调试，发现程序不会去运行以下的程序段：

Game\_System.prototype.initialize = function() {

this.Lagomoro\_Mission\_initialize();

this.\_Lagomoro\_Mission\_Data = {};

};

但是使用rmmv来启动程序时，则不会出现上述的问题。

我搞不懂，为什么mv就不运行被重写的Game\_System.prototype.initialize方法呢。

1. 待添加的问题
2. sarange-project-code-database 萨兰奇项目注意事项
3. JSDoc命令：

在 E:\GitText\sarange-project-code-database\js 文件下启动cmd，并执行以下代码。

jsdoc CodeManager.js -r plugins

jsdoc -r plugins

代码解释：对js文件夹内的CodeManager.js和其目录下的plugins文件夹的全部内容，进行JSDoc文档的输出。

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. 代码阅读

常见的名词：

Graphviz

CodeViz

doxygen

（截至目前，尚未找到合适的代码阅读工具，只有一个JSDoc代码API接口文档生成工具正在使用。）

1. 其他可视化编程的工具、技术、引擎、框架

ECharts

<https://www.runoob.com/echarts/echarts-tutorial.html>

前端游戏框架哪个好

<https://blog.csdn.net/valada/article/details/81639708>

1. sundry

这是巴哈姆特的siakomobi写的书本：《RPG Maker MV游戏制作基本外功篇:从操作到完成游戏一镜到底,马上就会!》这个书本可以说是RPGMV界内的教材。我现在没有。

Siakomobi的作品：《七音图腾篇》（UltraKagura）

“居中要一个非常麻烦的变量：字符宽度，而这个宽度，必须先绘制一次之后，才能得到。”

也就是说，这里无法实现文本的自动换行么？

drill：简单来说rmmv的词法分析器和文本绘制是直接纠缠在一起的，获取宽度必须先经过词法分析器，所以只能先绘制一遍，yep就是这样干的

ok字段是哪里设置的？

E:\NodeJS\_install\node\_cache

npm config set prefix “E:\NodeJS\_install\node\_global”

npm config set cache “E:\NodeJS\_install\node\_cache”

npm config get prefix

npm config get cache

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

E:\NodeJS\_install\node\_global\

require('cluster')

DataManager.loadDataFile的原理与PluginManager.loadScript的区别

fetch.then((response) => { })

XMLHttpRequest

DataManager.loadMapData(mapId);

（问个问题，地形碰撞检测怎么做的？我现在有一个半成品的战斗系统，现在想做一个“炮弹”发射碰撞到地形的功能。你有思路么？）

呃……不如直接用我的弹幕插件？ https://rpg.blue/thread-486840-1-1.html

我的话是用了一个SAT算法来做的碰撞检测。

(我最初自己写了sat算法；后来我为了制作子弹的“反弹”功能用了matter.js库，后来又舍弃了，因为快速的反弹功能我自己会写了，不用借助matter.js了，这库更适合用mv/mz做做愤怒的小鸟；然后我又想用回sat，当时比较急着更新所以不想自己写，就在github里找了库然后进行魔改)。

YEP GALV SRD MOG/DRILL ，还有web官网80多页的发布区的帖子里的所有插件，我通通都很仔细的看过

是不是每一个移动的单位都会被当做是一个事件，所以需要用循环的方式来循环一个事件，才能实现批量刷新怪物的业务？：

仇九：

1.

Game\_CharacterBase有子类Game\_Character，

Game\_Character 有子类Game\_player和Game\_Event。

Game\_Character有对应的显示精灵Sprite\_Character，Game\_player和Game\_Event的对应精灵都是一个Sprite\_Character。

2.

Game\_player的实例是$gamePlayer。

Game\_Event的实例都储存在Game\_Map的\_events属性中，而 Game\_Map的实例是$gameMap.

3.在Scene\_Map中（实例为SceneManager.\_scene）的updateMain中，$gamePlayer和$gameMap会执行对应的update函数，而$gameMap的update函数又会执行\_events中各个事件实例的update函数.

仇九：

至于复制事件的插件：

1.所有的复制事件插件都大同小异，就是先加载对应的地图文件（Map001.json等），然后将其转为js对象，然后获取其中的事件数据，整个储存在$dataMap中，而事件数据在$dataMap.events中以数组的形式储存着各个事件的数据。

在载入地图时就会依照$dataMap.events中的数据，建立每一个事件的Game\_Evnets实例，将其储存在$gameMap.\_events中。

2.而复制事件脚本就是重复这个步骤而已，不过是在当前地图载入其他插件的数据文件（Map001.json等），然后在此地图使用这个事件数据实例化Game\_Evnets，然后也添加到$gameMap.\_events中。一般来说，新的复制的事件id就是当前地图最大事件id+1.

3.我的插件可以在复制了事件后使用函数$gameMap.lastestSpawnEventId()获得刚才复制的新事件的id.

问题：是不是每一个移动的单位都会被当做是一个事件，所以需要用循环的方式来循环一个事件，才能实现批量刷新怪物的业务？

仇九：

“一个移动的单位”的相关东西在Game\_Chharacter中，而Game\_Player（玩家）和Game\_Event（事件）都继承了他的方法。所有的事件都是循环刷新的啊。

问题：将两个事件结合在一起的写法，省略 spawnEvent 的写法。

仇九：

1.要建立一个新的事件当然还是得那样（读取地图数据，读取这个地图数据中某个事件的数据，然后建立Game\_Event的实例）。

2.一个怪物数据总归还是用事件做的方便嘛，如果是要完全做一个新的敌人，不使用插件，那可以写一个Game\_Character的新子类，比如Game\_MapEnemy(不直接使用Game\_Enemy的原因是mv已经使用了)，然后完全根据脚本对其进行控制。

3.如果你想直接根据某个事件数据直接建立新的事件（而不是读取其他地图数据中的事件数据），那你用我的复制事件插件也支持这样做。

首先将新事件的事件数据赋值给$dataDirectEvent（用于传递新事件的事件数据），然后执行$gameMap.spawnEvent(null,null,null,x,y)就可以在当前地图的(x,y)处建立一个新事件，这个新事件的各个数据就依照那个事件数据生成。我做这个功能的主要原因是我以前给我的ARPG系统做“队友战斗”时要用。

一个地图数据的基础格式是这样：

1. 临时代码
2. 待学习的打包技术：

QQ：770436947

然后输npm init

npm install @capacitor/core @capacitor/cli

然后就是npx cap add 加相应平台。

然后把add换成open就可以打开

如果在add的时候出现了错误因为你没有在那个文件夹中放入3w文件夹并且在其中放入index点html。

然后那个工程就已经构建好了，按照那个工具的打包方法就可以打包了

详情参照as和xcoda的使用教程。

<https://www.bilibili.com/read/cv7828113>

<https://ionicframework.com/>

<https://capacitorjs.com/>

drill群 QQ： 273890194：

electron...这东西打包以后确实能唬一下小白，但因为本身只是单纯的打包并没有加密，所以解包也只需一行命令的事

正确的加密方式还是使用默认加密，将密码直接写进JS代码，再用mwjs SDK提供的加密方式将JS代码全部编译成bin字节码，如此一来对于一般的小白已经束手无策了

还不放心的话可以在此基础上再用Enigma Protector（非EnigmaVBox，那个没有加密）对代码和json数据再套一层壳，这样就算是一般大佬也不想去碰你的工程了，浪费时间

1. 待学习的socket.io技术

socket是网络编程技术。

可以找：  
 QQ：770436947、1442417954

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. [] SIAKO.Mobi 13节 <https://www.bilibili.com/video/av23706785?p=13> [↑](#footnote-ref-3)
5. [] 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-4)
6. [] SIAKO.Mobi 15节 <https://www.bilibili.com/video/av23706785?p=15> [↑](#footnote-ref-5)
7. [] refresh方法是在更新的时候才调用。和update不一样。 [↑](#footnote-ref-6)
8. [] 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-7)
9. [] Window.prototype = Object.create(PIXI.Container.prototype);说明了Window类实现了PIXI的容器类。 [↑](#footnote-ref-8)
10. [] @extends PIXI.Container 在PIXI源码内，这说明了Sprite类继承了容器类。 [↑](#footnote-ref-9)
11. [] 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-10)
12. []

    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-11)
13. [] MDN. XMLHttpRequest <https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest> [↑](#footnote-ref-12)
14. [] 菜鸟教程.JavaScript JSON.parse() <https://www.runoob.com/js/javascript-json-parse.html> [↑](#footnote-ref-13)
15. [] MDN. HTMLElement.style <https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/style> [↑](#footnote-ref-14)
16. [] MDN. CSS Properties Reference <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference> [↑](#footnote-ref-15)
17. [] CSDN.学习HTML5 Canvas这一篇文章就够了 <https://blog.csdn.net/u012468376/article/details/73350998> [↑](#footnote-ref-16)
18. []

    Window.prototype.\_createAllParts = function() {

    ......

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

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

    ......

    } [↑](#footnote-ref-17)
19. [] 使用 Pixi.js 创建 HTML5游戏 <https://learnku.com/articles/3838/use-pixijs-to-create-html5-games> [↑](#footnote-ref-18)
20. [] 360新知.用自己的电脑做服务器搭建网站并让互联网访问 <http://xinzhi.wenda.so.com/a/1542122927208741> [↑](#footnote-ref-19)
21. [] 360新知.怎样打开win7的IIS功能 <http://xinzhi.wenda.so.com/a/1518018465610079> [↑](#footnote-ref-20)
22. [] CSDN.vscode下关于Live Server的使用 <https://blog.csdn.net/zhouwei_doris/article/details/80604604> [↑](#footnote-ref-21)
23. [] GitHub.pixi最新版下载地址 <https://github.com/pixijs/pixi.js/releases> [↑](#footnote-ref-22)
24. [] pixi.js教程中文版--基础篇 <https://www.cnblogs.com/songchunmin/p/7789611.html> [↑](#footnote-ref-23)
25. [] <https://babeljs.io/docs/en/learn> [↑](#footnote-ref-24)
26. [] <https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Classes> [↑](#footnote-ref-25)