使用说明

RPGMV脚本API速查

信息

显示文字（Show Text）

一共有4个需要设定的地方：

- 设置脸图（可选）

  如果想给弹出的文本框设置脸图，那么就可以用这个代码。

参数	参数类型/参数内容	默认值	说明	
faceName	字符串，内容就是脸图的名字		就是脸图文件的名字，事件编辑器那里面选择显示文字，然后点击脸图，就会有一个弹窗。弹窗里面就是脸图的文件，这个faceName就是这些文件的名字。	
faceIndex	0到8		一个脸图有8个头像，从零开始数，第一行是0到4，第二行是5到7。写几就是第几个图。	

```
$gameMessage.setFaceImage(faceName, faceIndex);
```

- 设置背景（可选）

参数	参数类型/参数内容	默认值	说明	
background	0：窗口 1：暗淡 2：透明		这个参数就对应着背景下拉框的属性	

```
$gameMessage.setBackground(background);
```

- 窗口位置（可选）
  - positionType：有三个固定的数值
    - 0：顶部
    - 1：中间
    - 2：底部

```
$gameMessage.setPositionType(positionType);
```

- 文本

  这个就是本指令的核心方法，效果就是弹出来文本框。如果没有配置前面的样式，那么会按照默认样式来显示。

  - text：这个就是一个普通的字符串

```
$gameMessage.add(text);
```

示例：

```
$gameMessage.setFaceImage("Actor1",0);
$gameMessage.setBackground(0);
$gameMessage.setPositionType(2);
$gameMessage.add("内容");
```

显示选项（Show Choices）

- 选项框背景

参数	参数类型/参数内容	默认值	说明	
background	0：窗口 1：暗淡 2：透明		窗体的样式	

```
$gameMessage.setChoiceBackground(0);
```

- 窗口位置

  positionType：有三个固定的数值

  - 0：顶部
  - 1：中间
  - 2：底部

```
$gameMessage.setChoicePositionType(1);
```

- 选项本体

参数	参数类型/参数内容	默认值	说明	
choices	数组		数组的内容就是选项的内容，比如["是","否"]。**注意！元素必须是字符串类型。**	
defaultType	0，1，2，，，	0	光标默认移动到第几个选项，从0开始。就是说一进入这个选择页面。	
cancelType	0，1，2，，，	0	取消选项时，默认选择哪个选项。	

```
$gameMessage.setChoices(choices, defaultType, cancelType);
```

- 回调函数

```
$gameMessage.setChoiceCallback(callback);
```

示例：

```
$gameMessage.setChoices([ "我","是","选","项"],0,2);
$gameMessage.setChoiceBackground(1);
$gameMessage.setChoicePositionType(1);
$gameMessage.setChoiceCallback(function(n){
  console.log("你选择的是第"+(n+1)+"项！")
});
```

数值输入处理（Input Number）

参数

- variableId：所要输入进的变量ID，变量ID可以在编辑器里面直接找到。
- maxDigits：输入的位数。你可以发现，在事件编辑器里面，这个是最多8位的，但是用代码的话，可以想写几位就写几位。非常自由。

```
$gameMessage.setNumberInput(6,20);
```

物品选择处理（Select Item）

这个函数可以储存所选物品的物品ID，前提是你必须拥有这个物品。虽然我从来没有用过，但是我想这个功能还是很有用的。

参数

- variableId：要把物品ID存入哪个变量中，变量ID可以在编辑器里面直接找到。

- itemType：物品的类型，一共有四种数值，而且是从1开始的。真搞不懂官方怎么设计的。
    - 1：普通物品
    - 2：重要物品
    - 3：隐藏物品A
    - 4：隐藏物品B

```
$gameMessage.setItemChoice(5,1);
```

显示滚动文字

参数：

- speed：文字滚动的速度，2是默认值

```
$gameMessage.setScroll(2,false);
```

示例：

```
$gameMessage.add("内容1");
$gameMessage.add("内容2");
$gameMessage.add("内容3");
$gameMessage.setScroll(2,false);
```

显示滚动文字（Show Scrolling Text）

参数：

- speed：文字滚动的速度，2是默认值

```
$gameMessage.setScroll(2,false);
```

示例：

```
$gameMessage.add("内容1");
$gameMessage.add("内容2");
$gameMessage.add("内容3");
$gameMessage.setScroll(2,false);
```

游戏进程

开关操作（Control Switches）

```
$gameSwitches.setValue(variableId, value);
```

变量操作（Control Variables）

```
$gameSwitches.setValue(variableId, value);
```

独立开关操作（Control Self Switch）

需要传递四个参数

1. 地图ID，表示第几张地图
2. 变量ID
3. 独立开关名称
4. true还是false

```
$gameSelfSwitches.setValue([4,287,'A'],false)
```

查询值

```
$gameSelfSwitches.value([4,72,'A'])
```

计时器操作（Control Timer）

```
$gameTimer.start(count);//
$gameTimer.stop();
$gameTimer.seconds();//返回秒数
```

流程控制

分支条件（Conditional Branch）

在脚本中建议使用if else来代替

循环（Loop）

在脚本中建议使用for循环替代

跳出循环（Break Loop）

终止事件处理

公共事件

标签（Label）

转到标签（Jump to Label）

注释（Comment）

队伍

增减金币（Change Gold）

实际上源代码中，loseGold直接调用了gainGold(-amount)，就离谱。

```
$gameParty.gainGold(300);//得到300块钱
$gameParty.loseGold(300);//失去300块钱
```

增减物品（Change Items）

实际上源代码还是比较复杂的，我这个把源代码提炼精简了一下，你只管放心用。

参数

- item：需要传递一个物品对象，可以直接传递`$dataItems[1]`这种格式。代表第一个物品，注意这些`$`开头的都是从1开始的。
- amount：得到的数量

示例：

```
$gameParty.gainItem($dataItems[1], 30);//得到30个1号物品
$gameParty.loseItem($dataItems[1], 10);//失去10个1号物品
```

增减武器（Change Weapons）

```
 $gameParty.gainItem($dataWeapons[this._params[0]], value, this._params[4]);
```

增减护甲（Change Armors）

```
$gameParty.gainItem($dataArmors[this._params[0]], value, this._params[4]);
```

队伍管理（Change Party Member）

注意！如果你队伍中已经有了1号角色，那么再调用增加1号角色的代码将不会有效果（你应该庆幸没有报bug）。

```
$gameParty.addActor(1);//为队伍增加1号角色
$gameParty.removeActor(1);//1号角色离队
```

获取领队信息*

```
$gameParty.leader();
```

角色

增减HP（Change HP）

```
Game_Interpreter.prototype.changeHp(target, value, allowDeath)

$gameParty.leader().gainHp(-30)
```

示例：

```
Game_Interpreter.prototype.changeHp($gameParty.leader(),-300,true)
```

增减MP（Change MP）

```
$gameParty.leader().gainMp(-30)
```

增减TP（Change TP）

```
$gameParty.leader().gainTp(-30)
```

更改状态（Change State）

```
   actor.addState(this._params[3]);
actor.removeState(this._params[3]);
```

完全恢复（Recover All）

```
actor.recoverAll();
```

增减经验值（Change EXP）

```
 actor.changeExp(exp, show);//show代表是否展示升级信息，exp代表经验值
```

增减等级（Change Level）

```
 actor.changeLevel(level, show);//同上
```

增减能力值（Change Parameter）

增减技能（Change Skill）

更改装备（Change Equipment）

更改名字（Change Name）

```
$gameParty.leader().setName("起的名字");
$gameParty.leader().setName(prompt("自己起一个名字吧") );
```

更改职业（Change Class）

更改昵称（Change Nickname）

```
actor.setNickname(nickname);
```

更改简介（Change Profile）

移动

场所移动（Transfer Player）

第一种就是直接移动，但是画面不会移动。

```
$gamePlayer.setPosition(23,24)
```

第二种就是编辑器里面那个

```
$gamePlayer.reserveTransfer(mapId, x, y, d, fadeType);
//d代表direction，默认写0.fadetype也写0
$gamePlayer.reserveTransfer(3,33,34,0,0)
```

fadeType:

1. 0：黑
2. 1：白
3. 2：无

设置载具位置（Set Vehicle Location）

设置事件位置（Set Event Location）

滚动地图（Scroll Map）

设置移动路线（Set Movement Route）

```
$gameMap._events[event.id].moveStraight(2);//向下移动

$gameMap._events[event.id].moveStraight(4);//向左移动



this.setThrough(true); //开启穿透

this.setThrough(false); //关闭穿透

this.setTransparent(true); //开启透明状态

this.setTransparent(false);//关闭透明状态

this.turnRandom(); //随机转向

```

```
21
22  this.turnTowardPlayer(); //朝向玩家
23
24  this.turnAwayFromPlayer(); //背向玩家
25
26
27
28  this.moveTowardPlayer();//朝向玩家移动
29
30  this.moveAwayFromPlayer();//远离玩家移动
31
32  this.setMoveFrequency(3); //移动频率
33
34  $gamePlayer.setMoveSpeed(4); //角色移动速度
35  //4是默认速度，要注意，这个速板是指数增加的，一旦变了一点点，速度就有极大变化。所以每次增
    加0.01之类的比较好，如果直接加1速度会太快。
36
37
38
39  this.moveDiagonally(4, 2);//左下移动
40
41
42  this.moveDiagonally(6, 2);//右下移动
43
44
45  this.moveDiagonally();//左上移动
46
47
48  this.moveDiagonally(6, 8);//右上移动
49
50
51
52
53
54
55
56
57  switch (command.code) {
58      case gc.ROUTE_END:
59          this.processRouteEnd();
60          break;
61
62      case gc.ROUTE_MOVE_RIGHT:
63          this.moveStraight(6);//向右移动
64          break;
65      case gc.ROUTE_MOVE_UP:
66          this.moveStraight(8);//向上移动
67          break;
68
69      case gc.ROUTE_MOVE_RANDOM:
70          this.moveRandom();//随机移动
71          break;
72
73      case gc.ROUTE_MOVE_FORWARD:
74          this.moveForward();
75          break;
76      case gc.ROUTE_MOVE_BACKWARD:
77          this.moveBackward();
```

```
            break;
        case gc.ROUTE_JUMP:
            this.jump(params[0], params[1]);
            break;
        case gc.ROUTE_WAIT:
            this._waitCount = params[0] - 1;
            break;
        case gc.ROUTE_TURN_DOWN:
            this.setDirection(2); //设置朝向下方
            break;
        case gc.ROUTE_TURN_LEFT:
            this.setDirection(4);//设置朝向左方
            break;
        case gc.ROUTE_TURN_RIGHT:
            this.setDirection(6);//设置朝向右方
            break;
        case gc.ROUTE_TURN_UP:
            this.setDirection(8);//设置朝向上方
            break;
        case gc.ROUTE_TURN_90D_R:
            this.turnRight90();
            break;
        case gc.ROUTE_TURN_90D_L:
            this.turnLeft90();
            break;
        case gc.ROUTE_TURN_180D:
            this.turn180();
            break;
        case gc.ROUTE_TURN_90D_R_L:
            this.turnRightOrLeft90();
            break;


        case gc.ROUTE_WALK_ANIME_ON:
            this.setWalkAnime(true);
            break;
        case gc.ROUTE_WALK_ANIME_OFF:
            this.setWalkAnime(false);
            break;
        case gc.ROUTE_STEP_ANIME_ON:
            this.setStepAnime(true);
            break;
        case gc.ROUTE_STEP_ANIME_OFF:
            this.setStepAnime(false);
            break;
        case gc.ROUTE_DIR_FIX_ON:
            this.setDirectionFix(true);
            break;
        case gc.ROUTE_DIR_FIX_OFF:
            this.setDirectionFix(false);
            break;

        case gc.ROUTE_CHANGE_IMAGE:
            this.setImage(params[0], params[1]);
            break;
        case gc.ROUTE_CHANGE_OPACITY:
            this.setOpacity(params[0]);
            break;
```

```
    case gc.ROUTE_CHANGE_BLEND_MODE:
        this.setBlendMode(params[0]);
        break;
    case gc.ROUTE_PLAY_SE:
        AudioManager.playSe(params[0]);
        break;

    }
```

载具乘降（Getting On and Off Vehicles）

人物

在这里特指地图上那个人物的动画。要和角色区分开。角色是记录角色数据信息的，而人物则反映了在画面上的移动速度、动画等信息。

更改透明状态

更改队列行进

集合队列成员

显示动画（Show Animation）

```
$gamePlayer.requestAnimation(1);
```

显示气泡图标（Show Balloon Icon）

```
$gamePlayer.requestBalloon(1) //显示惊讶
```

要注意理论上来说这个id可以无限大

有多大取决于系统中Ballon.png有多高，默认720像素，每一个小方块48像素。默认有10个，其图片还预留了5个扩展位置，开发者可以自己画，如果不够了只需要向下拓展即可。

例如多扩展48*5像素，那么就可以多画5个气泡，最多可以访问到20号气泡。

暂时消除事件

```
$gameMap.eraseEvent(this._eventId);
```

设置人物位置*

注意！这个设置不会使画面一起移动。如果想要画面也一起移动，建议使用移动中的"场所移动"。

```
$gamePlayer.setPosition(3,6)
```

查看队伍朝向*

MV中朝向被定义为4个数字

- 左：4
- 右：6
- 上：8
- 下：2

其实，，这个和小键盘的方向是一一对应的。

```
$gamePlayer.direction() //查看队伍方向
```

图片

显示图片（Show Picture)

```
$gameScreen.showPicture(this._params[0], this._params[1], this._params[2],
        x, y, this._params[6], this._params[7], this._params[8],
this._params[9]);
```

移动图片（Move Picture)

```
$gameScreen.movePicture(this._params[0], this._params[2], x, y,
this._params[6],
        this._params[7], this._params[8], this._params[9], this._params[10]);
```

旋转图片（Rotate Picture)

```
$gameScreen.rotatePicture(this._params[0], this._params[1]);
```

更改图片色调（Tint Picture）

```
$gameScreen.tintPicture(this._params[0], this._params[1], this._params[2]);
```

消除图片（Erase Picture）

```
$gameScreen.erasePicture(this._params[0]);
```

清除所有图片*

```
$gameScreen.clearPictures();
```

计时

等待

画面

淡入淡出

```
$gameScreen.startFadeOut(duration) //淡出
$gameScreen.startFadeIn(duration) //淡入
```

更改画面色调

```
$gameScreen.startTint(tone, duration);
```

这个tone同样是一个颜色。数组各项分别为RGB+灰度。

```
//下面是官方的预设值：
$gameScreen.startTint([0, 0, 0, 0], 60); //正常
$gameScreen.startTint([-68, -68, -68, 0], 60); //黑暗
$gameScreen.startTint([34, -34, -68, 170], 60); //茶色
$gameScreen.startTint([68, -34, -34, 0], 60); //黄昏
$gameScreen.startTint([-68, -68, 0, 68], 60); //夜晚

//下面是我自己总结的
$gameScreen.startTint([50, 50, 50, 0], 60); //正午
$gameScreen.startTint([-120, -100, 0, 68], 60);//深夜
```

闪烁画面

```
$gameScreen.startFlash(color, duration);
```

颜色使用的是RGB+强度，调用时使用数组

```
$gameScreen.startFlash([255, 0, 0, 128], 8);
```

震动屏幕

默认5，5，60

```
$gameScreen.startShake(power, speed, duration);
```

设置天气（Set Weather Effect）

```
$gameScreen.changeWeather(this._params[0], this._params[1], this._params[2]);
```

音频&视频

播放BGM（Play BGM）

注意，如果之前已经有bgm的话，这个会把之前的给挤掉。

```
var bgm = new Object();
bgm.name = "对峙";//bgm的名称
bgm.volume = 100;//响度
bgm.pitch = 100;//音调
bgm.pan = 0;//偏移
AudioManager.playBgm(bgm);
//AudioManager.playBgm(bgm,pos);//pos可以缺省
```

淡出BGM（Fadeout BGM）

参数为秒数

```
AudioManager.fadeOutBgm(duration);
```

保存BGM（Save BGM）

```
$gameSystem.saveBgm();
```

还原BGM（Resume BGM）

```
$gameSystem.replayBgm();
```

播放BGS（Play BGS）

```
 AudioManager.playBgs(this._params[0]);
```

淡出BGS（Fadeout BGS）

```
AudioManager.fadeOutBgs(this._params[0]);
```

播放ME（Play ME）

```
AudioManager.playMe(this._params[0]);
```

播放SE（Play SE）

```
var se  = new Object();
se.name = "Bell3";//se的名称
se.volume = 100;//响度
se.pitch = 100;//音调
se.pan = 0;//偏移
AudioManager.playSe(se);//播放音效
```

经过我的感觉，一些音名对照表如下。

C	D	E	F	G	A	B
40	45	50	52	58	65	75

停止SE（Stop SE）

播放影像（Play Movie）

```
Graphics.playVideo('movies/' + name + ext);
```

播放系统音效*

这个所谓的系统，并不是说windows系统，而是说MV数据库里面那个系统。这个音效的顺序和MV系统里面的音保持一致。

```
SoundManager.playCursor()//光标
SoundManager.playOk()//确定
SoundManager.playCancel()//取消
SoundManager.playBuzzer()//无效
SoundManager.playEquip()//装备
SoundManager.playSave()//存档
SoundManager.playLoad() //读档
SoundManager.playBattleStart() //战斗开始
SoundManager.playEscape() //逃跑
SoundManager.playEnemyAttack() //敌人攻击
SoundManager.playEnemyDamage() //敌人受伤
SoundManager.playEnemyCollapse() //敌人消失
SoundManager.playBossCollapse1() //Boss消失1
SoundManager.playBossCollapse2() //Boss消失2
SoundManager.playActorDamage() //角色受伤
SoundManager.playActorCollapse() //角色无法战斗
SoundManager.playRecovery() //恢复
SoundManager.playMiss() //未命中
SoundManager.playEvasion() //回避
SoundManager.playMagicEvasion() //魔法回避
SoundManager.playReflection() //魔法反射
SoundManager.playShop() //商店
SoundManager.playUseItem() //使用物品
SoundManager.playUseSkill() //使用技能
```

场景控制

战斗处理（Battle Processing）

商店处理（Shop Processing）

名字输入处理（Name Input Processing）

打开菜单画面（Open Menu Screen）

```
SceneManager.push(Scene_Menu);
```

打开存档画面（Open Save Screen）

```
SceneManager.push(Scene_Save);
```

游戏结束（Game Over）

```
SceneManager.goto(Scene_Gameover);
```

返回标题画面（Return to Title Screen）

```
SceneManager.goto(Scene_Title);
```

系统设置

更改战斗BGM（Change Battle BGM）

```
 $gameSystem.setBattleBgm(this._params[0]);
```

更改胜利ME（Change Victory ME）

```
 $gameSystem.setVictoryMe(this._params[0]);
```

更改战败ME（Change Defeat ME）

```
$gameSystem.setDefeatMe(this._params[0]);
```

更改载具BGM（Change Vehicle BGM）

```
var vehicle = $gameMap.vehicle(this._params[0]);
if (vehicle) {
    vehicle.setBgm(this._params[1]);
}
```

启用/禁用存档（Change Save Access）

```
$gameSystem.disableSave();
$gameSystem.enableSave();
```

启用/禁用菜单（Change Menu Access）

```
$gameSystem.disableMenu();
$gameSystem.enableMenu();
```

启用/禁用遇敌

启用/禁用整队

更改窗口颜色（Change Window Color）

```
$gameSystem.setWindowTone(this._params[0]);
```

更改角色图像

更改载具图像

游戏时间*

```
$gameSystem.playtimeText() //游戏时间，用时分秒表示
$gameSystem.playtime() //游戏时间，用秒表示
```

显示帧数*

```
Graphics.showFps();
```

关闭游戏*

这个是直接暴力关闭窗口，不会有任何弹窗！！！慎用!

```
window.close()
```

地图

启用/禁用显示地图名称（Change Map Name Display）

```
$gameMap.enableNameDisplay();
$gameMap.disableNameDisplay();
```

更改地图图块（Change Tileset）

更改战斗背景（Change Battle Back）

更改远景（Change Parallax）

```
$gameMap.changeParallax(this._params[0], this._params[1],
        this._params[2], this._params[3], this._params[4]);
```

获取指定位置的信息（Get Location Info）

- 变量

```
1
```

-

地图尺寸*

```
$gameMap.height()
$gameMap.width()
```

战斗

增减敌人HP（Change Enemy HP）

增减敌人MP（Change Enemy MP）

增减敌人TP（Change Enemy TP）

更改敌人状态（Change Enemy State）

敌人完全恢复（Enemy Recover All）

敌人出现（Enemy Appear）

敌人变身（Enemy Transform）

显示战斗动画（Show Battle Animation）

强制战斗行动（Force Action）

中断战斗（Abort Battle）

高级

脚本（Script）

请使用eval()代替

插件指令（Plugin Command）

```
function 调用插件指令(插件指令){
    var args =插件指令.split(" ");
    var command = args.shift();
    Game_Interpreter.prototype.pluginCommand(command, args);
}
//使用时，直接把需要的插件指令复制到形式参数里面就可以了
//比如说
调用插件指令（"我的插件 参数"）

```

动画*

播放动画

直接可以用事件来播放动画，动画会作用在其身上。

里面的参数只需要传递动画的编号即可

```
$gameMap._events[11].requestAnimation(1)
```

事件*

获取事件注释

```
//这个方法可以直接获得注释的内容
$dataMap.events[eventID].note

//如果注释是以对象形式写的，那么可以用meta属性进行访问
//<name:莲华><age:13>
$dataMap.events[eventID].meta.name
$dataMap.events[eventID].meta.age
```

获取指定位置的事件编号

如果指定的位置没有事件，那么返回0。

```
$gameMap.eventIdXy(x,y);
```

运行事件

可以直接用脚本来开启事件

```
$gameMap._events[11].start()
```

访问事件对象

在地图上，直接在对象的事件指令里面创建一个脚本，里面写上下面这个，就可以打印出该事件对象。

```
console.log($gameMap._events[this._eventId])
```

如果不是写在事件编辑器的话，可以指定事件ID来进行访问。

```
console.log($gameMap._events[11])
```

也可以直接根据坐标来访问

```
$gameMap.eventsXy(31, 33)
```

事件对象的移动速度

标准速度是4，也就是人物的速度。

```
$gameMap._events[11]._moveSpeed
```

鼠标*

鼠标移动

这个函数也比较特殊，并不是只要你移动了鼠标就能被检测到，而是说你必须一直按着并且移动才会true。这个其实是为触摸板设计的。

```
TouchInput.isMoved() //是否按下鼠标左键同时移动了鼠标
```

鼠标左键是否正在被按下

```
TouchInput.isPressed()  //鼠标左键是否正在被按下
```

是否切换为鼠标左键

```
TouchInput.isTriggered() //鼠标左键刚刚是否按下
```

鼠标左键是否连续按下

```
TouchInput.isRepeated() //鼠标左键是否连续点击，或者一直处于按下状态
TouchInput.isLongPressed() //鼠标左键是否一直按下
```

要注意的就是这个`isRepeated()`和`isLongPressed()`，这两个比较像，但是还是有很大区别的。如果你鼠标一直狂点，那么`isRepeated()`会返回true，但是`isLongPressed()`不会。而长按鼠标左键，两个都会返回true。

鼠标左键释放

```
TouchInput.isReleased() //鼠标左键是否释放
```

鼠标右键点击

```
TouchInput.isCancelled() //鼠标右键是否点击
```

鼠标位置

```
TouchInput.x
TouchInput.y
```

键盘*

增加WASD控制方向

```
Input.keyMapper = {
    9: 'tab',
    13: 'ok',
    16: 'shift',
    17: 'control',
    18: 'control',
    27: 'escape',
    32: 'ok',
    33: 'pageup',
    34: 'pagedown',
    37: 'left',
    38: 'up',
    39: 'right',
    40: 'down',
    45: 'escape',
    81: 'pageup',
    87: 'pagedown',
    88: 'escape',
    90: 'ok',
    96: 'escape',
    98: 'down',
```

```
    100: 'left',
    102: 'right',
    104: 'up',
    120: 'debug',
    //WASD的按键
    65: 'left',     // 左箭头键
    87: 'up',       // 上箭头键
    68: 'right',    // 右箭头键
    83: 'down'     // 下箭头键
};
```

检测键盘事件

下面这些函数要传递一个参数，也就是要检测的按键的名字，刚刚那个映射表里面都写了，直接复制就行。（注意把引号也带上）

```
Input.isPressed('ok')  //按键是否正在被按下
Input.isTriggered() //按键刚刚是否按下
Input.isRepeated() //按键是否连续点击，或者一直处于按下状态
Input.isLongPressed() //按键是否一直按下
```

这个和上面的鼠标一模一样，不多做解释。

存档*

载入存档

-1：载入配置文件

0：载入第一个文件

```
StorageManager.load(savefileId);//返回一个json文件
```

载入全局配置

```
DataManager.loadGlobalInfo();//返回一个数组，从1开始。
```

武器与装备*

查看武器信息

```
$dataWeapons[$gameParty.leader()._equips[0]._itemId].name
```

获取装备信息*

```
$gameParty.leader()._equips//获取领队的装备数据
```

这个会返回一个数组，里面有5个元素，分别对应游戏里面的装备信息

- 0：武器
- 1：盾牌
- 2：头部
- 3：身体
- 4：装饰品

但是这个的信息非常简陋，里面只存放了武器和装备的编号而已，并没有真正地存放物品信息。所以如果想获取真正的数据，还得去访问$dataWeapons。

```
$dataWeapons[$gameParty.leader()._equips[0]._itemId]
```

这样子，先获取玩家的装备编号，再进行查询。然后就好了。

这个$dataWeapons的数据如下：

属性	说明
animationId	动画编号
description	说明
etypeId	
iconIndex	图标编号
id	武器的编号
maxItem	最大持有量
meta	注释的JSON
name	武器名字
note	备注
params	武器数据的数组，从0到7分别是[0:最大HP，1:最大MP，2:攻击力，3:防御力，4:魔法攻击，5:魔法防御，6:敏捷，7:幸运]
price	价格
traits	特性，比如追加能力什么的
wtypeId	武器类型的编号

另外，装备里面的备注是可以直接用对象的形式访问的。

```
$dataWeapons[$gameParty.leader()._equips[0]._itemId].meta.你写的标签属性
```

nw.js脚本API速查

窗口

> `Window` 是DOM中 顶层 `window`对象的包装类 . 可扩展操作以及接收窗口事件 .
>
> 每个 `Window`都是EventEmitter类实例 , 使用 `Window.on(...)`可响应窗口事件 .

获取当前窗口

有两种等价写法

```
var win = require('nw.gui').Window.get();
var win2 = nw.Window.get();
```

但是实际上`win===win2`，因为窗口是一个非常重要的对象，所以这个是一个单例，不管是谁，获取到的对象都是一个。

打开新窗口

一共三个参数，其中，第二个是配置信息，详情参考配置中的窗口子字段。

```
nw.Window.open('test.html', {}, function (new_win) {
    //新窗口创建时的回调函数
    new_win.on('focus', function () {
        console.log('点击了新窗口！');
    });
});
```

窗口的位置

属性

窗口相对于显示器的位置

```
nw.Window.get().x
nw.Window.get().y
```

通过坐标移动

```
nw.Window.get().moveTo(x, y)//移动窗口到指定位置,将窗口的左上角移动到指定的坐标
nw.Window.get().moveBy(x, y)//移动窗口到相对于当前窗口左上角的横纵偏移,比如窗口上移10px
```

通过默认值移动

`position` String -指定的窗口位置,可选值: `null`(不固定), `center`(屏幕居中), `mouse`(鼠标所在位置)

```
nw.Window.get().setPosition('mouse')//移动窗口到鼠标所在位置
```

窗口的大小

属性

```
nw.Window.get().width
nw.Window.get().height
```

设置大小

```
nw.Window.get().resizeTo(width, height)//重设窗口尺寸为 width 和 height
nw.Window.get().resizeBy(width, height)//调整窗口尺寸为相对于当前窗口尺寸的width 和 height,比如窗口加宽10px
```

设置最大/最小值

```
nw.Window.get().setMaximumSize(width, height)//设置窗口的最大宽高
nw.Window.get().setMinimumSize(width, height)//设置窗口的最小宽高
```

是否允许玩家调整大小

```
nw.Window.get().setResizable(true)
```

聚焦窗口

就是让窗口保持最前面,

```
nw.Window.get().focus()
```

永久置顶

```
nw.Window.get().setAlwaysOnTop(false) //是否允许窗口总是置顶(在其余窗口上面)
```

控制台窗口（仅SDK模式下可用）

打开/关闭控制台

```
require('nw.gui').Window.get().showDevTools();//打开
require('nw.gui').Window.get().closeDevTools(); //关闭
```

是否打开了控制台

```
nw.Window.get().isDevToolsOpen()
```

窗口最大化/最小化

```
nw.Window.get().enterFullscreen();//全屏
nw.Window.get().leaveFullscreen() //退出全屏

nw.Window.get().minimize();//最小化

nw.Window.get().maximize()//最大化
用途:Linux和Window中最大化窗口 ， Mac中放大窗口

win.minimize()
用途:Windows/Mac中最小化窗口 ， Linux中图标化窗口
```

隐藏窗口

```
nw.Window.get().hide()
```

窗口事件和监听

最小化/最大化时

```
// 监听最小化事件
nw.Window.get().on('minimize', function() {
  console.log('最小化了！');
});

// 移除最小化监听事件
nw.Window.get().removeAllListeners('minimize');

```

```
// 监听最大化事件
nw.Window.get().on('maximize', function() {
  console.log('最大化了！');
});

// 移除最大化监听事件
nw.Window.get().removeAllListeners('maximize');
```

窗口关闭时

```
nw.Window.get().on('close', function () {
    this.hide(); // 关闭窗口
    this.close(true); // 真正关闭窗口进程，注意一旦里面写false，就会无限循环
});

nw.Window.get().removeAllListeners('close');
```

窗口获得/失去焦点时

```
//获取焦点事件
nw.Window.get().on('focus', function () {

});

nw.Window.get().removeAllListeners('focus');
```

```
//失去焦点事件
nw.Window.get().on('blur', function () {

});

nw.Window.get().removeAllListeners('blur');
```

全屏时

```
//失去焦点事件
nw.Window.get().on('enter-fullscreen', function () {

});

nw.Window.get().removeAllListeners('enter-fullscreen');
```

调试工具被关闭时

```
//阻止游戏关闭
nw.Window.get().on('devtools-closed', function () {
    this.hide(); // 关闭窗口
    this.close(true); // 真正关闭窗口进程，注意一旦里面写false，就会无限循环
});


nw.Window.get().removeAllListeners('devtools-closed');
```

打包

1. 将所需的文件压缩为zip文件，注意rar不行！
2. 把zip文件的后缀名改为nw
3. 把这个文件放到nw.js环境同级目录下
4. 在nw.js的目录下打开命令行，输入下面的命令

```
copy /b nw.exe+app.nw app.exe
```

注意，nw.exe就是咱nw环境的入口，名字可能会变。后面那个就是咱刚才打包的文件。名字也是随意的。

最后那个app.exe就是咱打包之后的文件。

全局对象

```
global.a =1;
```

package.json配置速查

示例

```
{
    /**指定程序的起始页面。*/
    "main": "index.html",

    /**字符串必须是小写字母或者数字，可以包含.或者_或者-不允许带空格。name必须全局唯一。*/
    "name": "demo",

    /**程序描述*/
    "description": "demo app of node-webkit",

    /**程序版本号*/
    "version": "0.1.0",

    /**关键字*/
    "keywords": ["demo","node-webkit"],

    /**bool值，如果设置为false，将禁用webkit的node支持。*/
    "nodejs": true,
    /**
```

```
    * 指定一个node.js文件，当程序启动时，该文件会被运行，启动时间要早于node-webkit加载html的时间。
    * 它在node上下文中运行，可以用它来实现类似后台线程的功能。
    * （不需要可注释不用）
    */
    "node-main": "js/node.js",


    /**
    * bool值。默认情况下，如果将node-webkit程序打包发布，那么只能启动一个该应用的实例。
    * 如果你希望允许同时启动多个实例，将该值设置为false。
    */
    "single-instance": true,

    /**窗口属性设置 */
    "window": {
        /**字符串，设置默认title。*/
        "title": "demo",
        /**窗口的icon。*/
        "icon": "link.png",
        /**bool值。是否显示导航栏。*/
        "toolbar": false,
        /**bool值。是否允许调整窗口大小。*/
        "resizable": true,
        /**是否全屏*/
        "fullscreen": false,
        /**是否在win任务栏显示图标*/
        "show_in_taskbar": true,
        /**bool值。如果设置为false，程序将无边框显示。*/
        "frame": true,
        /**字符串。窗口打开时的位置，可以设置为“null”、“center”或者“mouse”。*/
        "position": "center",
        /**主窗口的的宽度。*/
        "width": 800,
        /**主窗口的的高度。*/
        "height": 670,
        /**窗口的最小宽度。*/
        "min_width": 400,
        /**窗口的最小高度。*/
        "min_height": 335,
        /**窗口显示的最大宽度，可不设。*/
        "max_width": 800,
        /**窗口显示的最大高度，可不设。*/
        "max_height": 670,
        /**bool值，如果设置为false，启动时窗口不可见。*/
        "show": true,
        /**是否在任务栏显示图标。*/
        "show_in_taskbar":true,
        /**
         * bool值。是否使用kiosk模式。如果使用kiosk模式，
         * 应用程序将全屏显示，并且阻止用户离开应用。
         * */
        "kiosk": false
    },

    /**webkit设置*/
    "webkit": {
```

```
        /**bool值，是否加载插件，如flash，默认值为false。*/
        "plugin": true,
        /**bool值，是否加载Java applets，默认为false。*/
        "java": false,
        /**bool值，是否启用页面缓存，默认为false。*/
        "page-cache": false
    }
}
```

RPGMaker原生配置

```
{
    "name": "",
    "main": "index.html",
    "js-flags": "--expose-gc",
    "window": {
        "title": "",
        "toolbar": false,
        "width": 816,
        "height": 624,
        "icon": "icon/icon.png"
    }
}

```

常用代码模板速查

插件模板

```
/*:
 * @plugindesc 对插件的描述
 *
 * @author 作者
 *
 * @param 插件的第一个参数
 * @desc 描述
 * @default 默认值
 *
 * @param 插件的第一个参数
 * @desc 描述
 * @default 默认值
 *
 *
 *
 * @help
 * 下面这些是对脚本的说明，格式没有强制要求
```

```
 * 但是至少应该把使用方法和插件指令列出来。
 *
 */

//这个一般都是你的名字，但是没有强制要求，
//只要是一个对象就可以
var Yan = Yan || {};
//下面这个就是来注册插件，格式是固定的
//注意名字要和文件名一样
Yan.Parameters = PluginManager.parameters('你插件的名字');

//下面这个就是插件的参数，
Yan.Parameters.参数1 = String(Yan.Parameters['参数'] || '默认值');
Yan.Parameters.参数2 = String(Yan.Parameters['参数2'] || '默认值');

//下面这个格式也是固定的
//注意，要是不需要插件指令的话，可以不写
var _Game_Interpreter_pluginCommand =
Game_Interpreter.prototype.pluginCommand;
Game_Interpreter.prototype.pluginCommand = function (command, args) {
  _Game_Interpreter_pluginCommand.call(this, command, args);
  //设计你的插件指令，command就是你插件指令的名字，当然这个可以不止一个，可以用else来创
建多个指令
  if (command === '插件指令名字') {
    //参数0就是你的第一个参数
    //比如说插件指令为command 第一个参数为one
    //那么指令就是这样调用的  command one
    switch (args[0]) {
      case '某':
       //代码
        break;
      case '某某某':
       //代码
        break;
    }
  }
};
```

修改存档最大数量

直接修改数字。

```
DataManager.maxSavefiles = function() {
    return 你想要的数字;
};
```

禁用鼠标

```
TouchInput._onMouseDown = function(event) {
    // 什么都不要写
};
```

自定义按键映射

在游戏开发中经常出现需要自己添加一个新按钮或者快捷键的需求。

实现自定义映射其实很简单，我们首先需要知道JS里面如何给一个对象增加属性。其中就有一个**对象键值对的赋值**。不懂的可以看我的JS基础。

在脚本中，左边写所映射按键的键码，右边写你给这个键起的keyName。

```
Input.keyMapper[65] = "A";
```

这样就把A，这个按键绑定到MV里面了。

具体的键码可以参考附录1。

队伍移动

让玩家可以根据按键进行移动。

不建议直接使用，但是这个很适合重写。

```
Game_Player.prototype.moveByInput = function(){

}
```

自定义存档

存档时的信息储存

```
DataManager.makeSavefileInfo = function() {
    var info = {};
    info.globalId   = this._globalId;
    info.title      = $dataSystem.gameTitle;
    info.characters = $gameParty.charactersForSavefile();
    info.faces      = $gameParty.facesForSavefile();
    info.playtime   = $gameSystem.playtimeText();
    info.timestamp  = Date.now();
    return info;
};
```

窗体重写常用模板

自定义菜单指令

```
(() => {

    //1.自己的自定义窗体
    function Window_Test() {
        this.initialize.apply(this, arguments);
    }

    //2.继承一个窗体类，根据不同需求可以自行改动
    Window_Test.prototype = Object.create(Window_Command.prototype);

    //3.把构造函数指向自己
    Window_Test.prototype.constructor = Window_Test;

    //4.写初始化函数
    Window_Test.prototype.initialize = function (x, y) {
        Window_Command.prototype.initialize.call(this, x, y)//xy代表位置
        this.refresh();//刷新窗口
        this.activate();//激活
    }

    //窗口类的核心方法，自定义代码就在这里写
    Window_Test.prototype.makeCommandList = function () {
        this.addCommand("自定义的指令", '指令名', true);//指令名是代码内部调用的，可以通过setHandler来使用
        //自定义代码，都在这里写。
    }


    //右侧的说明区域
    function 自定义HELP(){
        this.initialize.apply(this, arguments);
    }
    自定义HELP.prototype = Object.create(Window_Base.prototype);
    自定义HELP.prototype.constructor = 自定义HELP;

    自定义HELP.prototype.initialize = function () {
        var x = 250;
        var y = 0;
        var width = 500;
        var height = this.fittingHeight(2);//行宽
        Window_Base.prototype.initialize.call(this, x, y, width, height);
        this._text = '';
    }

    //窗口类的核心方法，自定义代码就在这里写
    自定义HELP.prototype.setInfo = function(text){
        this.contents.clear();
        this._text =text;
        this.drawTextEx(this._text, this.textPadding(), 0);
    }



    //这个就是点击指令后的场景
    function Scene_Test() {
```

```
        this.initialize.apply(this, arguments)
    }

    Scene_Test.prototype = Object.create(Scene_MenuBase.prototype);//自行查看窗体所在的场景
    Scene_Test.prototype.constructor = Scene_Test;
    Scene_Test.prototype.initialize = function () {
        Scene_MenuBase.prototype.initialize.call(this)
    }

    Scene_Test.prototype.create = function () {
        Scene_MenuBase.prototype.create.call(this)
        this.createTestWindow();
        this.createHelpWindow();
        this.bindCommands();
    }

    Scene_Test.prototype.start = function () {
        Scene_MenuBase.prototype.start.call(this)
        this.windowTest.refresh();
    }

    Scene_Test.prototype.createHelpWindow =function(){
        this._helpWindow = new 自定义HELP();
        this._helpWindow.setInfo("hello world");
        this.addWindow(this._helpWindow);
    }

    Scene_Test.prototype.createTestWindow = function () {

        var x = 0;
        var y = 0;
        // console.log(xx,yy)
        this.windowTest = new Window_Test(x, y)
        this.addWindow(this.windowTest);//根据需求，可以添加很多个窗口

    }

    Scene_Test.prototype.bindCommands = function () {
        this.windowTest.setHandler('ok', this.windowOK.bind(this));
        this.windowTest.setHandler('cancel', this.windowCancel.bind(this));
    }

    //当点击时触发
    Scene_Test.prototype.windowOK = function () {
        console.log("ok")
        this.windowTest.activate();

        // 监听按键，改变说明区域的文字
        this._helpWindow.setInfo("我是自定义的指令哟");

    }

    //当返回时除法
    Scene_Test.prototype.windowCancel = function () {
        console.log("cancel")
        SceneManager.pop();//返回上一级
    }
```

```
112
113
114
115
116     Window_MenuCommand.prototype.addOriginalCommands = function () {//这个方法是专门给我们提供的，可以用来重写。
117         //在菜单指令中添加指令，但是要注意，如果只写了这个，是没有效果的，
118         //必须在createCommandWindow中，给test注册响应函数才能点进去
119         this.addCommand('指令名', 'test', true);
120     }
121
122
123     Scene_Menu.prototype.commandTest = function () {
124         SceneManager.push(Scene_Test);
125     }
126
127
128     Scene_Menu.prototype.createCommandWindow = function () {
129
130         this._commandWindow = new Window_MenuCommand(0, 0);
131         this._commandWindow.setHandler('item', this.commandItem.bind(this));
132         this._commandWindow.setHandler('skill', this.commandPersonal.bind(this));
133         this._commandWindow.setHandler('equip', this.commandPersonal.bind(this));
134         this._commandWindow.setHandler('status', this.commandPersonal.bind(this));
135         this._commandWindow.setHandler('formation', this.commandFormation.bind(this));
136         this._commandWindow.setHandler('options', this.commandOptions.bind(this));
137         this._commandWindow.setHandler('save', this.commandSave.bind(this));
138         this._commandWindow.setHandler('gameEnd', this.commandGameEnd.bind(this));
139         this._commandWindow.setHandler('cancel', this.popScene.bind(this));
140         this._commandWindow.setHandler('test', this.commandTest.bind(this));
141         this.addWindow(this._commandWindow);
142     }
143
144 })()
```

状态界面

```
1 (function () {
2
3     //1.创建一个函数，名字就是你自定义窗体的名字
4
5     function 窗体名() {
6         this.initialize.apply(this, arguments);
7     }
8     //2.继承Window_Base类
9     窗体名.prototype = Object.create(Window_Base.prototype);
```

```
    窗体名.prototype.constructor = 窗体名;

    //3.初始化窗体大小
    var statusWindow = {
        x: 240,
        y: 0,
        width: 576,
        height: 624
    };
    窗体名.prototype.initialize = function () {
        Window_Base.prototype.initialize.call(this, statusWindow.x,
statusWindow.y,
            statusWindow.width, statusWindow.height);
    };

    //4.写刷新函数，系统会自动调用这个函数
    窗体名.prototype.refresh = function () {
        this.contents.clear();
        this.drawText("hello world", 144, 0, 200);
    };



    //5.将窗口布局到游戏里面
    Scene_Menu.prototype.createStatusWindow = function () {
        this._statusWindow = new 窗体名();
        this.addWindow(this._statusWindow);
    };

})();
```

地图界面

```
(function () {
  //1.创建一个函数，名字就是你自定义窗体的名字
  function 窗体名() {
    this.initialize.apply(this, arguments);
  }
  //2.继承Window_Base类
  窗体名.prototype = Object.create(Window_Base.prototype);
  窗体名.prototype.constructor = 窗体名;

  //3.初始化窗体大小
  窗体名.prototype.initialize = function () {
    Window_Base.prototype.initialize.call(this, 0, 0, 100, 100);//分别代表位置
和长宽
    this.drawText('莲华', 5, 5, 100, 'left');
  };

  //4.写创建窗体函数
  Scene_Base.prototype.创建窗体 = function () {
    this.addWindow(new 窗体名());
  };

```

```
21
22    //5.创建窗体
23    var _Scene_Map_createDisplayObjects =
Scene_Map.prototype.createDisplayObjects;
24    Scene_Map.prototype.createDisplayObjects = function () {
25      _Scene_Map_createDisplayObjects.call(this);
26      this.创建窗体();
27    };
28
29  })();
```

主界面新窗体

```
1  //构造函数体
2  function Window_PlayerCount() {
3      this.initialize.apply(this, arguments);
4  }
5  //继承自Window_Base
6  Window_PlayerCount.prototype = Object.create(Window_Base.prototype);
7  //设定构造函数
8  Window_PlayerCount.prototype.constructor = Window_PlayerCount;
9  //初始化
10 Window_PlayerCount.prototype.initialize = function (x, y) {
11     var width = 240;
12     var height = this.fittingHeight(1);
13     Window_Base.prototype.initialize.call(this, x, y, width, height);
14     this.drawText('团伙中有' + $gameParty.members().length + '人', 0, 0,
200);
15 };
16 //重写Scene_Menu，加入我们自定窗口
17 Scene_Menu.prototype.create = function () {
18     Scene_MenuBase.prototype.create.call(this);
19     this.createCommandWindow();
20     this.createGoldWindow();
21     this.createStatusWindow();
22     //加入我们自己的窗口
23     var win = new Window_PlayerCount(0, this._commandWindow.height);
24     this.addWindow(win);
25 };
```

自定义精灵类

```
1  //我们对系统提供的精灵类进行复制，创建一个"自定义精灵类"
2  function 自定义精灵类() {
3      this.initialize.apply(this, arguments);
4  }
5  自定义精灵类.prototype = Object.create(Sprite.prototype);
6  自定义精灵类.prototype.constructor = 自定义精灵类;
```

```
自定义精灵类.prototype.initialize = function () {
    Sprite.prototype.initialize.call(this);
    //以上为固定写法，不需要弄清楚
    //以下是设置这个精灵类的共同属性，所有这个对象构造的精灵都具有的属性，类比C++中的类
    /*设置框架，即精灵所处矩形范围，精灵的活动区域被限制在此框架内，四个参数分别为左上角的X,Y的坐标，宽度（横），长度（竖）
    需要注意的是X和Y一般是负数，如果大于0图片就会往屏幕外移动*/
    this.setFrame(0, 0, 1000, 1500);

    //运行创造子精灵的函数
    this.createAll();

    //绘制位图（bitMap）的文字函数
    this.drawText();

    //move作用和setFrame是一样的，两个数值相当于把框架沿x轴y轴移动多少单位
    this.move(20, 10);
};

自定义精灵类.prototype.createAll = function () {
    //使用图片来建立精灵
    this._clock = new Sprite(ImageManager.loadBitmap('img/pictures/', "SF_Actor3_8", true));
    //使用系统提供的位图对象建立精灵,这里使用的是一个空位图，具体的文字会用drawText函数上描绘
    this._cont = new Sprite(new Bitmap(380, 418));        //定义位图的宽和高

    //anchor，范围0-1，设置精设置绘制精灵的起始坐标，0.5可以令其居中显示
    this._clock.anchor.x = this._clock.anchor.y = 0.5;
    /*这里的setFrame是精灵中的精灵活动范围，低级精灵不能超过其容器活动范围，不然无法显示
    _clock精灵属于mySprite类，其活动范围不能超过他的类，如果不setFrame，系统将会给一个默认值*/
    this._clock.setFrame(0, 0, 100, 100);
    this._clock.move(32, 42);
    this.addChild(this._clock);
    this.addChild(this._cont);
};

//这里定义了一个函数，是用于对前面构建_cont精灵所用位图进行描绘的
自定义精灵类.prototype.drawText = function () {
    this._cont.bitmap.fontSize = 30;    //设置字型大小
    this._cont.bitmap.textColor ="rgb(255,255,255)"||'red';  //设置字体颜色，可用RGB表示法或者英文字符
    //'' +: 根据JavaScript语法规则，可以将后面的数据强制转化为字符串，这里$spriteTest.Parameters.text原本就是字符串，这样做法是为了防止出错
    var text = "测试文本 ";  //设置文本
    this._cont.bitmap.drawText(text,0, 88, 136, 44);
}


SUBupdate = Scene_Map.prototype.update;
Scene_Map.prototype.update = function () {
    SUBupdate.call(this);
    this.myMapSprite = new 自定义精灵类();
    //前面已经说过，在构建对象的函数里面已经设置了对象初始值，这里就不需要再设置了
    this.addChild(this.myMapSprite);
    //用addChild函数将精灵加入场景中
};
```

文件和文件格式速查

audio

文件分类

存放音频素材，下分4个子类

- bgm（background music）：背景音乐
- bgs（background sound）：背景音效:
- me（music effect）：音乐效果
- se（sound effect）：声音效果

音频分类

- m4a文件：用于手机端，加密后变成rpgmvm文件。
- ogg文件：用于PC端，加密后形成rpgmvo文件。

data

文件分类

- Actor.json——角色数据
- Animations.json——动画模块
- Armor.json——装备数据
- Classes.json——职业数据
- CommonEvents.json——公共事件数据
- Enemies.json——敌人数据
- Items.json——道具数据
- MapXXX.json——各地图的详细信息（包括事件
- MapInfos.json——各地图的大致信息
- Skills.json——技能数据
- States.json——状态数据
- System.json——系统、类型、用语
- Tileset.json——图块组模块
- Troop.json——敌群数据
- Weapons.json——武器数据

fonts

游戏的字体文件

1. 去根目录下fonts文件夹中
2. 打开gamefont.css文件，这个里面就存着字体的信息。

```
@font-face {
    font-family: GameFont;
    /*url里面就是字体文件的路径*/
    src: url("mplus-1m-regular.ttf");
}

.IIV::-webkit-media-controls-play-button,
video::-webkit-media-controls-start-playback-button {
    opacity: 0;
    pointer-events: none;
    width: 5px;
}
```

3. 把自己下载的ttf文件放入fonts文件夹中，把url里面的名字换成你下载的就行了

顺便一说，fonts文件都放在了C:\Windows\Fonts中，有需要可以自取。

icon

游戏的图标文件

img

游戏中所有的图片素材

js

游戏源代码，包括官方的代码和插件代码

movies

游戏中的视频文件

素材格式和尺寸

图片

素材类型	大小
立绘	高度为600px
CG	816*624px
脸图	144*144px
一个图块大小	48*48px
标题图片	816*624px

常用屏幕分辨率

- PC：16:9

  1920*1080

  1600*900

文件目录

- animations：动画特效
- battlebacks1：战斗背景图，一般都是地面或者战斗近景图
- battlebacks2：战斗的远景图，远景图可以省略，近景图必须指定。
- characters：角色的行走图，或者其他物品的动作图，大小为48*48

  在文件名前面加上半角符号"$"，那么该文件就只能容纳一个角色元。
- enemies：敌人的战斗图，也就是战斗时看到的立绘
- faces：各种角色的头像，144*144
- parallaxes：远景图，尺寸没有限制
- pictures：这些图片可以通过事件指令中的显示图片指令显示在游戏中，图片尺寸大小不限。
- titles1：标题的背景图
- titles2：标题背景图的边框，图片的大小为 816x624 ，Titles1 包含标题画面的背景，titles2 则主要是边框，使用它们可以组成标题画面。

附录

键码映射表

- 字母与数字

按键	键码	按键	键码	按键	键码	按键	键码
A	65	J	74	S	83	1	49
B	66	K	75	T	84	2	50
C	67	L	76	U	85	3	51
D	68	M	77	V	86	4	52
E	69	N	78	W	87	5	53
F	70	O	79	X	88	6	54
G	71	P	80	Y	89	7	55
H	72	Q	81	Z	90	8	56
I	73	R	82	0	48	9	57

- 小键盘和功能键

按键	键码	按键	键码	按键	键码	按键	键码
0	96	8	104	F1	112	F7	118
1	97	9	105	F2	113	F8	119
2	98	*	106	F3	114	F9	120
3	99	+	107	F4	115	F10	121
4	100	Enter	108	F5	116	F11	122
5	101	-	109	F6	117	F12	123
6	102	.	110				
7	103	/	111				

- 控制键

按键	键码	按键	键码	按键	键码	按键	键码
BackSpace	8	Esc	27	Right Arrow	39	-_	189
Tab	9	Spacebar	32	Dw Arrow	40	.>	190
Clear	12	Page Up	33	Insert	45	/?	191
Enter	13	Page Down	34	Delete	46	`~	192
Shift	16	End	35	Num Lock	144	[{	219
Control	17	Home	36	;:	186	|	220
Alt	18	Left Arrow	37	=+	187	]}	221
Cape Lock	20	Up Arrow	38	,<	188	'"	222

- 多媒体键

按键	键码
音量加	175
音量减	174
停止	179
静音	173
浏览器	172
邮件	180
搜索	170
收藏	171

转义字符速查

控制字符	功能
\V[n]	替换为第n个变量的值。
\N[n]	替换为第 n个角色的名字。
\P[n]	替换为第n个队伍成员。
\G	替换为货币单位。
\C[n]	后方的文字显示为第n号颜色。
\I[n]	绘制第n个图标。
\{	将字体大小增加一级。
\}	将字体大小减小一级。
\\	替换为反斜杠字符。
\$	打开金钱窗口。
\.	等待1/4秒。
\\	反斜杠
\!	等待按键输入。
\>	立刻显示一行内剩余的文字。
\<	取消立刻显示文字的效果。
\^	显示文本后不等待输入。

RPGMV的键盘映射

定义在rpg_core中。Input.keyMapper

MV里面键盘的按键名字和真正的键盘名字是**不一样**的，一个名字会对应多个实际按钮，所以要特别注意。

keyName	实际按钮
'tab'	tab
'ok'	enter、空格、Z
'shift'	shift
'control'	ctrl、alt
'escape'	esc、小键盘0、insert、X
'pageup'	pageup、Q
'pagedown'	pagedown、W
'left'	方向键左、小键盘4
'up'	方向键上、小键盘8
'right'	方向键右、小键盘6
'down'	方向键下、小键盘2
'debug'	F9

类继承表

场景的继承关系

- Scene_Base：所有场景的基类
    - Scene_Boot：用于初始化整个游戏
    - Scene_Title：标题界面的场景
    - Scene_Map：地图界面的场景
    - Scene_MenuBase：所有菜单类型的基类，也就是游戏中点右键那个菜单
        - Scene_Menu：主菜单的场景

          调用窗口：`Window_MenuCommand`、`Window_Gold`、`Window_MenuStatus`
        - Scene_ItemBase：技能界面和物品界面的基类
            - Scene_Item：物品界面

              调用窗口：Window_Help
            - Scene_Skill：技能界面
        - Scene_Equip：装备界面
        - Scene_Status：状态界面
        - Scene_Options：选项界面
        - Scene_File：储存和读取界面的基类
            - Scene_Save：储存界面

        - Scene_Load：读取界面
    - Scene_GameEnd：游戏结束界面（点击结束游戏时那个界面）
    - Scene_Shop：商店界面
    - Scene_Name：姓名输入界面
    - Scene_Debug：Debug界面
  - Scene_Battle：战斗场景
  - Scene_Gameover：游戏结束场景

窗体的继承关系

- Window_Base：所有窗体的基类
  - Window_Selectable：可以用鼠标点击和滑轮滑动的的窗体
    - Window_Command：选择指令的父类窗口
      - Window_HorzCommand：The command window for the horizontal selection format.
        - Window_ItemCategory： The window for selecting a category of items on the item and shop screens.
        - Window_EquipCommand： The window for selecting a command on the equipment screen.
        - Window_ShopCommand：展示选择买卖的商店窗口
      - Window_MenuCommand：主菜单可选指令的窗口，也就是鼠标右键之后，左边那一栏
      - Window_SkillType： The window for selecting a skill type on the skill screen.
      - Window_Options：The window for changing various settings on the options screen.
      - Window_ChoiceList：对应着事件中`显示选项`的窗口。
      - Window_PartyCommand： The window for selecting whether to fight or escape on the battle screen.
      - Window_ActorCommand：The window for selecting an actor's action on the battle screen.
      - Window_TitleCommand： The window for selecting New Game/Continue on the title screen.
      - Window_GameEnd：The window for selecting "Go to Title" on the game end screen.
    - Window_MenuStatus：展示角色成员状态的窗口
      - Window_MenuActor：The window for selecting a target actor on the item and skill screens.
    - Window_ItemList：The window for selecting an item on the item screen.
      - Window_EquipItem：The window for selecting an equipment item on the equipment screen.
      - Window_ShopSell：he window for selecting an item to sell on the shop screen.

        - Window_EventItem：对应着事件中`物品选择处理`的窗口
        - Window_BattleItem：The window for selecting an item to use on the battle screen.
    - Window_SkillList：The window for selecting a skill on the skill screen.
        - Window_BattleSkill：The window for selecting a skill to use on the battle screen.
    - Window_EquipSlot：The window for selecting an equipment slot on the equipment screen.
    - Window_Status：The window for displaying full status on the status screen.
    - Window_SavefileList：The window for selecting a save file on the save and load screens.
    - Window_ShopBuy：The window for selecting an item to buy on the shop screen.
    - Window_ShopNumber： The window for inputting quantity of items to buy or sell on the shop

      // screen.
    - Window_NameInput：选择玩家姓名的窗口，里面全都是各种字母还有50音。
    - Window_NumberInput：对应着事件指令`数值输入处理`的窗口。
    - Window_BattleLog： The window for displaying battle progress. No frame is displayed, but it is

      // handled as a window for convenience.
    - Window_BattleStatus： The window for displaying the status of party members on the battle screen.
    - Window_BattleEnemy：The window for selecting a target enemy on the battle screen.
    - Window_DebugRange：The window for selecting a block of switches/variables on the debug screen.
    - Window_DebugEdit：The window for displaying switches and variables on the debug screen.
- Window_Help：描述选择物品的窗口
- Window_Gold：展示队伍金币数量的窗口
- Window_SkillStatus：The window for displaying the skill user's status on the skill screen.
- Window_EquipStatus：The window for displaying parameter changes on the equipment screen.
- Window_ShopStatus：The window for displaying number of items in possession and the actor's

  // equipment on the shop screen.
- Window_NameEdit：编辑角色姓名的窗口，特指上面那个。
- Window_Message：展示文字信息的窗口
- Window_ScrollText：The window for displaying scrolling text. No frame is displayed, but it

// is handled as a window for convenience.

- Window_MapName：展示地图名字的窗口

精灵的继承关系

- Sprite_Base：具有显示动画功能的精灵类
    - Sprite_Character：显示角色的精灵
    - Sprite_Battler：Sprite_Actor 和 Sprite_Enemy 的父类
        - Sprite_Actor：显示主角的精灵
        - Sprite_Enemy：显示敌人
    - Sprite_StateOverlay：The sprite for displaying an overlay image for a state.
    - Sprite_Weapon：显示武器攻击图像的精灵
    - Sprite_Balloon：显示气泡图标的精灵
- Sprite_Button：显示按钮的精灵
- Sprite_Animation：播放动画的精灵
- Sprite_Damage：显示弹出式伤害的精灵
- Sprite_StateIcon：展示状态图标的精灵
- Sprite_Picture：显示图片的精灵
- Sprite_Timer：显示时间的精灵
- Sprite_Destination：显示输入区域的精灵
- Spriteset_Base：Spriteset_Map 和 Spriteset_Battle 的父类
    - Spriteset_Map：在地图屏幕上的一组精灵
    - Spriteset_Battle：战斗屏幕的一组精灵

游戏全局对象类继承关系

- Game_Temp：用于并不会保存到存档里面的临时数据
- Game_System：系统数据
- Game_Message：就是用来显示文本或者选择的类
- Game_Switches：开关的类
- Game_Variables：变量的类
- Game_SelfSwitches：独立开关的类
- Game_Screen：用于游戏屏幕特效的类，；例如色调更改或者闪屏等
- Game_Picture：显示图片的类
- Game_Item：
- Game_Action：
- Game_ActionResult：
- Game_BattlerBase：
    - Game_Battler

        - Game_Actor
        - Game_Enemy
- Game_Actors：游戏主角们数组的封装类
- Game_Unit： Game_Party 和Game_Troop的父类
    - Game_Party ： 游戏队伍的类，里面包含着金币和道具等数据
    - Game_Troop：
- Game_Map：
- Game_CommonEvent：
- Game_CharacterBase：
    - Game_Character： Game_Player, Game_Follower, GameVehicle和Game_Event的父类
        - Game_Player
        - Game_Follower
        - GameVehicle
        - Game_Event
- Game_Interpreter：

作者信息

姓名：闫辰祥

QQ：1796655849

QQ交流群：529245311

B站教程：https://www.bilibili.com/video/BV1Vb4y1q717?spm_id_from=333.999.0.0