## 简报

### 任务

|  |
| --- |
| 任务目标：在任意界面中放一个GIF贴图。  任务帮助：当前目录下，提供了一个插件模板：Drill\_SimpleCourseD2，D3，你需要在模板中编写相关代码，来熟悉基本结构。  任务提示：从这节课讲解的内容 比较散 。  记住下面几个课程关键词：**数组、js对象、$gameVariable、$gameTemp、自定义场景 。**  课程将围绕这些知识点展开。 |

### 基本意识

1.所有底层和插件都是相通的，可以直接调用或覆写。也就是说你的插件函数名如果乱起名，有几率覆盖掉底层函数。

尽量要注意函数名不要起得太简单了哦。

2.所有脚本，都是基于ES5的js格式。

所有脚本没有import指令、没有const、没有let。没有箭头函数。

（你可以写并且能跑通，但是要注意有些旧机器环境是最多支持到ES5的，最好稳定性优先。）

ES5的特性：<https://www.w3school.com.cn/js/js_object_es5.asp>

关于稳定的函数写法，可以参考工具箱的”基本函数查询表.docx”。

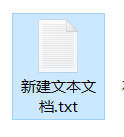
## 开始课程（上）

### 开始写脚本

下面开始写脚本，先把下列步骤完整过一遍。

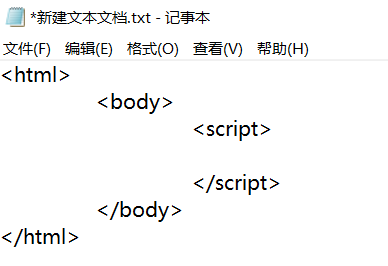
#### 新建一个文本文档

新建一个文本文档。



#### 打开文本文档，输入标签

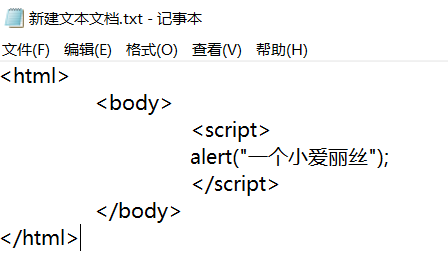
按下图分别输入下面三个标签：html、body、script。



#### 编写js脚本

这里加个脚本：alert("一个小爱丽丝");

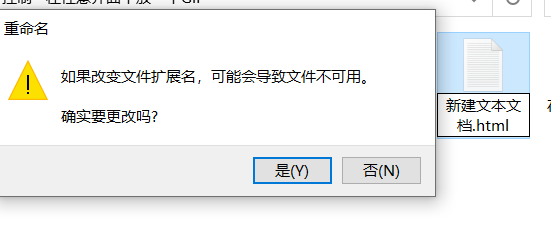
然后保存。



#### 修改文件后缀名

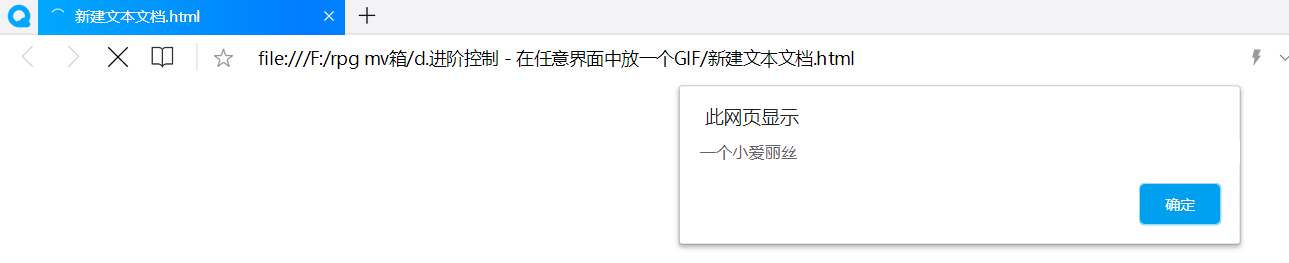
把文件后缀名”.txt”改为”.html”。

（如果看不见后缀名，去百度一下“显示文件的后缀名”）



#### 运行文件

双击运行文件，你电脑的默认浏览器会打开这个文件，并执行了js脚本：



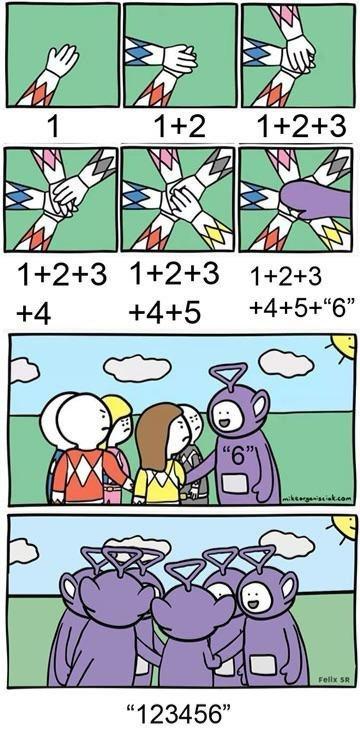
以上就是网页js（dom js）的运行方式，编辑几个简单的标签，即可让你的浏览器运行你写的js脚本。

但这里要注意的是，网页中提供的js环境十分有限，不包含rmmv的底层函数库。

### 详解 - 数组的定义

**1）论字符串的破坏性**

这节课的讲解从一个表情包开始：



聪明的你赶紧写一个脚本试试，来验证一下这个表情包在js语言中是否正确吧！

|  |
| --- |
| alert(1+2+3+4+5+"6"); |

这个答案，按照脚本的流程，作者我简单试了一下：



结果似乎让所有人都失望，因为我们大部分人都 猜错 了。

而我们的误判也在于这个字符串的破坏性，因为实际结果完全绕开了我们的预期结果。

因此，我们应尽量避免这种 奇怪的写法 以及调用方法，因为它带来了各种不确定性。

**2）数组的定义**

Js语言中，一个数组的定义如下：

|  |
| --- |
| var aaa\_tank = []; |

这里注意一下，是方括号。不要错误写成花括号了，花括号的会定义json格式的对象，在部分情况下，一样能够用起来像 ”数组” 一样，但注意本质上完全不是数组。

一个数组具有基本的 增删改查 四个基本功能：

|  |
| --- |
| aaa\_tank.unshift( 4 ); //增 - 头部添加  aaa\_tank.push( 5 ); //增 - 尾部添加  aaa\_tank.splice(i,1); //删  aaa\_tank[0] = 1; //改  aaa\_tank[1] = 2; //改  aaa\_tank[Number(bbb)] = 3; //改  var data = aaa\_tank[0]; //查 |

数组的结构在”基本函数查询表.docx”中也有介绍。

关于数组的更多详细内容这里就不赘述了，如果你对数组了解不足，可以去网上看看视频教程补习一下。

**3）****数组长度的变化**

在Js语言中，可以支持下面的特殊写法：

|  |
| --- |
| var aaa\_tank = [];  aaa\_tank[4] = 3;  alert(aaa\_tank.length); |

代码 强制越界 赋值了数组，那么数组的长度会根据情况自动扩充，将数组长度扩为5。

（代码中的 aaa\_tank[0], aaa\_tank[1], aaa\_tank[2], aaa\_tank[3] 获取到的值为null。）

### 详解 - 遍历

**1）遍历数组**

一个标准数组的遍历如下：

（注意，如果你对for in、foreach、map这些高级函数了解不深，那么尽量不要写。）

（就用最原始的for，就写谁都看的懂的循环结构。）

|  |
| --- |
| $gameSystem.\_drill\_BBa\_changingTank = [];  //...  for( var j = 0; j < $gameSystem.\_drill\_BBa\_changingTank.length; j++ ){  var temp = $gameSystem.\_drill\_BBa\_changingTank[j];  //...  } |

**2）数组求和**

如果我需要将数组中 2-10 的元素求和，那么写法如下：

|  |
| --- |
| var aaa\_tank = [1,2,3,4,5,6,7,8,9,10,11,12];  var result = 0;  for( var j = 2; j <= 10; j++ ){  result += aaa\_tank[j];  }  alert(result); |

如果换成 变量的 2-10 的元素求和，那么写法如下：

|  |
| --- |
| var result = 0;  for( var j = 2; j <= 10; j++ ){  result += $gameVariables.value( j );  }  alert(result); |

**在最初的课程中，有提及一个很严重的误区，现在不妨再提一下：**

|  |
| --- |
| **误区**：  由于js的var可以是任意类型，这就使得对脚本一知半解的朋友会尝试在脚本中塞入其他类型的数据。比如：  $gameVariables.setValue(22,”一串字符串”);  var bb = $gameVariables.value(22);  通过这种方式，参数bb是可以获取到字符串的。但是记住，千万不能这样写脚本。  这样做有潜在的风险，一个只装整数的容器，塞入了字符串、数组等其他的东西，就好比巧克力豆的罐子中混入了图钉。（这种操作在c语言和java语言中是分分钟判死刑的，会报语法错误） |

你已经猜到了，此操作就等同于 “来了个天线宝宝” 。

这里我们不妨试着重现之前的 天线宝宝表情包 所讲述的问题：

在数字数组中强行插入了一个不合理的字符串对象，那么它们求和后，会得到什么？

（往数组中插入不同类型的操作在js语言中不会报错，但是在其他严格的编程语言会提示语法错误。）

|  |
| --- |
| var aaa\_tank = [1,2,3,4,5,6,7,8,9,10,11,12];  aaa\_tank[5] = "6";  var result = 0;  for( var j = 2; j <= 10; j++ ){  result += aaa\_tank[j];  }  alert(result); |

这里的结果是什么呢？自己上手试试看！

写插件初期时，会遇到很多“明明写对了程序，但是运行结果极其奇怪”的情况，这些都是由于没有注意到的类型或者空指针细节所造成的。

另外需要一提的是，这里如果插入的是一个null。

|  |
| --- |
| var aaa\_tank = [1,2,3,4,5,6,7,8,9,10,11,12];  aaa\_tank[5] = null;  var result = 0;  for( var j = 2; j <= 10; j++ ){  result += aaa\_tank[j];  }  alert(result); |

这里的结果值，输出为 NaN 。

后期在写插件过程中，null值经常会被误加进来，从而影响整体的结果。

如果你打印某个变量显示了NaN，那么多半都是误加了null的问题。

**3）清退数组**

通常清空数组，直接赋值空数组即可。

|  |
| --- |
| aaa\_tank = []; |

但是有时，需要删掉数组中指定的几个元素而不是全部。这时应该用下面的写法：

注意，必须用递减的for循环，然后一个个splice掉。

|  |
| --- |
| for( var i = aaa\_tank.length-1; i>=0; i--){  var data = aaa\_tank[i];  if( ... ){  //...  this.\_drill\_ERA\_animTank.splice(i,1); //递减遍历去除时，不会影响数组下标  }  } |

清退贴图中的子贴图时，也需要通过同样的方式removeChild：

|  |
| --- |
| for( var i = this.\_drill\_ERA\_animTank.length-1; i>=0; i--){  var sprite = this.\_drill\_ERA\_animTank[i];  this.\_drill\_map\_ui\_board.removeChild(sprite);  this.\_drill\_ERA\_animTank.splice(i,1); //递减遍历去除时，不会影响数组下标  } |

### 详解 - “变量类”剖析

回到rmmv的设定，开关 和 变量 就像两个功能类型单一的数组。

而事实上也的确是这样的，开关类 中存放布尔，变量类 中存放整数。

现在我们打开“rpg\_objects.js”脚本，搜索：“Game\_Variables”。



可以看到 变量类 的定义：（白色部分是作者我加的注释）

|  |
| --- |
| //==============================  // \* 变量 - 定义  //==============================  function Game\_Variables(){  this.initialize.apply(this, arguments);  }  //==============================  // \* 变量 - 初始化  //==============================  Game\_Variables.prototype.initialize = function(){  this.clear();  };  //==============================  // \* 变量 - 清理全部  //==============================  Game\_Variables.prototype.clear = function(){ this.\_data = []; }; |

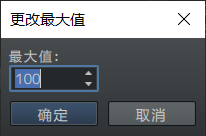
|  |
| --- |
| //==============================  // \* 变量 - 获取值  //==============================  Game\_Variables.prototype.value = function( variableId ){  return this.\_data[variableId] || 0;  };  //==============================  // \* 变量 - 设置值  //==============================  Game\_Variables.prototype.setValue = function( variableId, value ){  if( variableId > 0 && variableId < $dataSystem.variables.length ){  if( typeof value === 'number' ){  value = Math.floor(value);  }  this.\_data[variableId] = value;  this.onChange();  }  };  //==============================  // \* 变量 - 刷新地图事件  //==============================  Game\_Variables.prototype.onChange = function(){  $gameMap.requestRefresh();  }; |

变量初始化函数中，只调用了clear函数，而clear函数，只定义了一个 \_data 数组。

从this.\_data 的设置可以看出，Game\_Variables 变量类的定义，本质上就是一个数组。

在setValue赋值时，variableId对应了数组的索引编号。而数组下标0时，是跳过的。

需要注意的是，这个类中，并没有”最大值”的参数定义，这也是js特殊写法 [数组长度的变化](#数组长度的变化) 决定的。



另外，需要一提的是，当变量的数据变化时，变量会向 $gameMap地图对象 发送一个“请求刷新”的信号。

这个信号在 地图界面 每帧会监听一次，**变量/开关 改变事件页的原理，就是基于此信号触发的。**

|  |
| --- |
| //==============================  // \* 变量 - 刷新地图事件  //==============================  Game\_Variables.prototype.onChange = function(){  $gameMap.requestRefresh();  }; |

另外，$gameVariables的定义在 “rpg\_manager.js”脚本中。



在脚本中搜索 ” $gameVariables”，

可以看到常见的 $gameSystem 和 $gameTemp定义。

注意这些定义都是写在最外层的，也就是说被作为全局变量使用，可以在函数任意位置调用。

|  |
| --- |
| //==============================  // \* 数据管理器 - 容器定义  //==============================  var $gameTemp = null;  var $gameSystem = null;  var $gameScreen = null;  var $gameTimer = null;  var $gameMessage = null;  var $gameSwitches = null;  var $gameVariables = null;  var $gameSelfSwitches = null;  var $gameActors = null;  var $gameParty = null;  var $gameTroop = null;  var $gameMap = null;  var $gamePlayer = null;  //==============================  // \* 数据管理器 - 容器初始化  //==============================  DataManager.createGameObjects = function() {  $gameTemp = new Game\_Temp();  $gameSystem = new Game\_System();  $gameScreen = new Game\_Screen();  $gameTimer = new Game\_Timer();  $gameMessage = new Game\_Message();  $gameSwitches = new Game\_Switches();  $gameVariables = new Game\_Variables();  $gameSelfSwitches = new Game\_SelfSwitches();  $gameActors = new Game\_Actors();  $gameParty = new Game\_Party();  $gameTroop = new Game\_Troop();  $gameMap = new Game\_Map();  $gamePlayer = new Game\_Player();  }; |

另外，仿造此定义，也有两个核心插件基于此写法。（你可以打开目标插件去看看结构）

Drill\_CoreOfNumberArray 系统 - 变量数组核心

Drill\_CoreOfString 系统 - 字符串核心

由于插件需要针对特定的功能，这两个插件中脚本的写法过程比较复杂，稍微了解结构和原理即可。

## 开始课程（中）

### 开始回顾

刚才我们讲解了数组的结构。

这个部分，我们先回顾一下c课程的知识，顺带进一步了解一些知识。

注意，不要跳过哦，跟着我一起过一遍，加深印象。

#### 1）贴图的定义

**贴图（Sprite）**：是pixi库中提供的Sprite类，所有在游戏中显示的画面，都是通过贴图控制显示的。贴图还有一些别名，比如精灵、图片等。

**资源（Bitmap）**：是rpg\_core中的私有封装类，能够存储资源图片的数据，并提供一些绘制功能。主要用于隔离pixi的sprite中的texture材质渲染结构。

作者我总结了下列常用的贴图属性：

|  |  |  |
| --- | --- | --- |
| 属性名 | 含义 | 功能 |
| .x  .y | 坐标xy | 贴图在父类中的相对坐标位置。  如果有多级父类，那么相对位置会被无限叠加嵌套。所有sprite基本都是根据自身相对位置来定的，很少有绝对位置的说法。 |
| .bitmap | 资源对象 | 通过ImageManager获得的资源对象。  可以反复赋值不出问题且不影响性能，因为赋值的是对象指针。  Gif的实现原理基于此，存储一个bitmap容器，然后每帧按规律变bitmap。 |
| .anchor.x  .anchor.y | 中心锚点xy | 锚点决定图片的起始位置。  (0.0,0.0)表示贴图左上角，(0.5,0.5)表示贴图中心。  锚点影响 缩放/旋转 效果比较多。事件贴图的锚点都为(0.5,1.0)正下方。 |
| .blendMode | 混合模式 | 只有0正常、1叠加、2乘积、3滤色 四种混合模式可用。  0和2是常用颜色模式，内部硬性赋值比较好，容易让其它人理解。  比如纯色滤镜就是通过混合模式2实现的，蓝色会过滤出蓝色的光，过滤方式与红绿蓝底色有关。若直接摆出混合模式0123会让其它人较难理解。 |
| .opacity | 透明度 | Rmmv的封装属性，范围为 0 - 255。  原Pixi库中为alpha，范围为0 - 1.0。  如果opacity接受了一个undefined或者NaN值，则rmmv会报“clamp”错误，这里留意一下，因为这个错误比较容易出现。 |
| .visible | 显示 | 如果为false，则图片不渲染。  如果为true，则图片渲染。  从图形层面来看，不渲染比渲染要节省很多性能，但是千万不要滥用.visible=false，因为用多了你会经常因为图片不显示而半天找不到问题。 |
| .rotation | 旋转 | 贴图围绕锚点旋转。  单位为弧度，2 \* Math.PI为一周（2\* 3.14）。  正数顺时针，负数逆时针。（以0朝右为基准，则1.57朝下，-1.57朝上）  如果你让一个贴图单纯自旋转，比如魔法圈，是没问题的。  旋转令人疼的地方，是与事件的东南西北朝向对应过渡的问题，极坐标与直角象限比较难对应。 |
| .SCC1le.x  .SCC1le.y | 缩放 | 贴图围绕锚点缩放。  1.0为原比例缩放，2.0为两倍大小，0.5为一半大小。  锚点非常容易影响缩放的视觉效果，注意控制。 |
| .skew.x  .skew.y | 斜切 | 0.0表示正常，x 1.0时，形成向右上倾斜平行四边形，-1.0时左上倾斜。  功能不常用。 |

其中.zIndex图片层级顺序是自写的额外属性，底层中并不存在这个属性。

#### 2）继承与覆写

插件之间是有顺序关系的，核心插件尽量往前放，扩展插件尽量往后放。

那么为什么插件的顺序会造成那么大的影响呢？因为继承/覆写顺序不一样，会造成不同的效果。

继承也存在顺序问题，因为先继承的对象会被后继承的对象包裹。

其中，功能A是可以操作功能B中的属性的，因为A在后面，包裹了B。

|  |
| --- |
| //=============================================================================  // \* 插件 – 大功能B  //=============================================================================  var \_drill\_SCC1\_pluginCommand = Game\_Interpreter.prototype.pluginCommand;  Game\_Interpreter.prototype.pluginCommand = function(command, args) {  \_drill\_SCC1\_pluginCommand.call(this, command, args);  //...  };  //=============================================================================  // \* 插件 – 大功能A  //=============================================================================  var \_drill\_SCC1\_pluginCommand2 = Game\_Interpreter.prototype.pluginCommand;  Game\_Interpreter.prototype.pluginCommand = function(command, args) {  \_drill\_SCC1\_pluginCommand2.call(this, command, args);  //...  }; |

如果A需要 操作B的属性或函数，那么就可以看做 A -> B 的需求关系。如果没有B，那么A的功能就不能实现。这就出现了 **基于/作用于/扩展于** 的关系。

### 开始写插件

回顾了前面所学内容后，下面开始写插件，先把下列步骤完整过一遍。

#### 打开D2脚本

打开脚本。



#### 脚本复制到：资源文件参数部分

将下列脚本复制或者手敲到D2插件的指定位置。

|  |
| --- |
| \*  \* @param 资源-GIF  \* @desc png图片资源组，多张构成gif。  \* @require 1  \* @dir img/Course\_\_D/  \* @type file[]  \* @default []  \*  \* @param 帧间隔  \* @type number  \* @min 1  \* @desc gif每帧播放间隔时间，单位帧。（1秒60帧）  \* @default 4  \*  \* @param 是否倒放  \* @type boolean  \* @on 倒放  \* @off 不倒放  \* @desc true - 倒放，false - 不倒放  \* @default false  \* |

#### 脚本复制到：静态数据部分

将下列脚本复制或者手敲到D2插件的指定位置。

|  |
| --- |
| if( DrillUp.parameters["资源-GIF"] != "" &&  DrillUp.parameters["资源-GIF"] != undefined ){  DrillUp.g\_SCD2\_srcList = JSON.parse( DrillUp.parameters["资源-GIF"] );  }else{  DrillUp.g\_SCD2\_srcList = [];  }  DrillUp.g\_SCD2\_interval = Number(DrillUp.parameters["帧间隔"] || 4);  DrillUp.g\_SCD2\_back\_run = String(DrillUp.parameters["是否倒放"] || "true") === "true"; |

#### 脚本复制到：贴图创建部分

将下列脚本复制或者手敲到D2插件的指定位置。

|  |
| --- |
| //==============================  // \* 贴图 - 创建  //==============================  var \_drill\_SCD2\_layer\_createAllWindows = Scene\_Battle.prototype.createAllWindows;  Scene\_Battle.prototype.createAllWindows = function() {  \_drill\_SCD2\_layer\_createAllWindows.call(this); //rmmv对话框 < 最顶层    // > 创建贴图  var temp\_sprite = new Sprite();  temp\_sprite.x = 100;  temp\_sprite.y = 100;  temp\_sprite['\_drill\_time'] = 0; //计时器  temp\_sprite['\_drill\_bitmapTank'] = []; //资源对象列表（注意，这里的数据存放了bitmap对象，所以与sprite一样随时会被销毁）  for(var j = 0; j < DrillUp.g\_SCD2\_srcList.length ; j++){  var src\_str = DrillUp.g\_SCD2\_srcList[j];  var src\_bitmap = ImageManager.load\_CourseD( src\_str );  temp\_sprite['\_drill\_bitmapTank'].push( src\_bitmap );  }  temp\_sprite.bitmap = temp\_sprite['\_drill\_bitmapTank'][0]; //（先赋值第一张图像）    this.addChild( temp\_sprite ); //（添加到场景中）  this.\_drill\_SCD2\_sprite = temp\_sprite;    } |

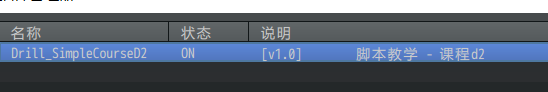
#### 脚本复制到：贴图帧刷新部分

将下列脚本复制或者手敲到D2插件的指定位置。

|  |
| --- |
| //==============================  // \* 贴图 - 帧刷新  //==============================  var \_drill\_SCD2\_update = Scene\_Battle.prototype.update;  Scene\_Battle.prototype.update = function() {  \_drill\_SCD2\_update.call(this);    if( this.\_drill\_SCD2\_sprite != undefined ){    // > 时间+1  this.\_drill\_SCD2\_sprite['\_drill\_time'] += 1;    // > 开始播放  var bitmap\_list = this.\_drill\_SCD2\_sprite['\_drill\_bitmapTank'];  var time = this.\_drill\_SCD2\_sprite['\_drill\_time'];  time = time / DrillUp.g\_SCD2\_interval; //（每隔n帧）  time = time % bitmap\_list.length; //（循环播放）  if( DrillUp.g\_SCD2\_back\_run ){ //（倒放设置）  time = bitmap\_list.length - 1 - time; //  }  time = Math.floor(time); //（小数取整）  this.\_drill\_SCD2\_sprite.bitmap = bitmap\_list[time];  }  }; |

#### 加入插件

把Drill\_SimpleCourseD2插件加入到工程中，之前课程的插件全去掉。



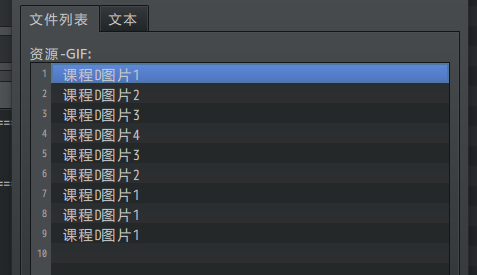
#### 放置图片

将课程所给的图片，放到你新建工程目录下 "\img\Course\_\_D" 文件夹下（注意是两个下划线）。



#### 配置图片

配置数张图片到GIF配置中。（重复的图片用于表示小爱丽丝眨眼睛的过程）

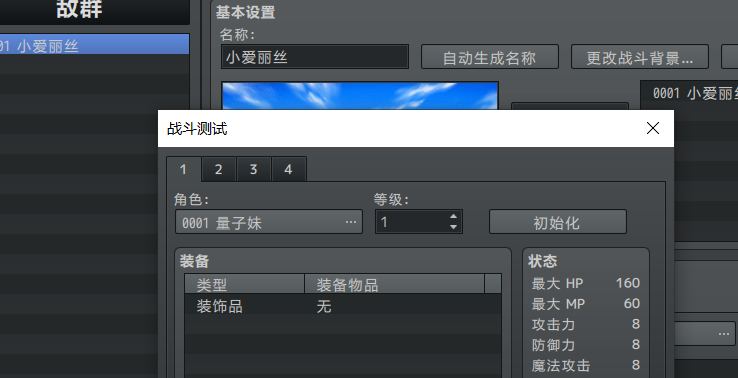


#### 功能测试

测试时，按F8，查看插件是否成功载入。如果你的插件拼写错误，控制台会出现相关报错。



触发一场战斗，就可以看到小爱丽丝出现，并且在最顶层的位置，能遮挡对话框。





ヽ(\*。>Д<)o゜在你完成上述流程之后，接下来我们开始分析一下操作过程中的细节吧。

### 详解 - 对象与JSON

在脚本中，你经常会见到这样的结构：

|  |
| --- |
| var temp\_sprite = new Sprite();  temp\_sprite.x = 100;  temp\_sprite.y = 100;  var temp\_data = {};  temp\_data.x = 100;  temp\_data.y = 100; |

这个”temp\_sprite”经常会称呼为 ”对象”， 其下的”x”和”y”经常被称呼为 **”属性”或者”成员”**。

而”temp\_data”经常会被称呼为 ”数据”，其下的”x”和”y” 经常被称呼为 **”参数”或者”临时变量”**。

这里的”对象” 的称呼与js语言本身有一些区别。

因为从原理上说，Js语言的任何类型都是对象。

而rmmv中，对象的称呼与纯数据是相互区别的。

rmmv称呼的定义：

|  |
| --- |
| **Object对象：**指贴图、窗口、有图像数据的实体对象。  对象的数据量非常大，一般无法使用JSON.stringify( ) 的方法显示所有数据。  **Data纯数据：**指字符串、数字等基本类型的数据，包括数组和json数据。  data纯数据可以保存进存档。 |

对象是不能放存档的，只能通过其他方式绕开。

因此，禁止在$gameSystem里面存放obj对象，这会把存档炸了，无法读取。

（存档底层就是 JSON数据 与 js对象 转字符串，部分js对象本身不具备转换并存储的功能，所以出错）

如果你必须要将使得对象能在全局中调用，你可以放入$gameTemp。

|  |
| --- |
| **称呼说明：**  从用法和格式上而言，由于JSON数据和js对象几乎是一模一样的，所以实际应用中，大家都傻傻分不清楚。  如果统一叫 ”js对象” 反而感觉很别扭，因为js中都是对象，指代不明。  而如果我说这是一个json串，那么你的脑海里一定会复现json的经典格式，便能理解这个对象的格式是json的格式。因此，为了方便理解，有时会称呼为json格式。 |

Js对象的常用写法如下：

|  |
| --- |
| var data = {'x':0,'y':0 };  var data = {};  data['x'] = 0;  data['y'] = 0;  var data = {};  data.x = 0;  data.y = 0; |

这三种定义方式一模一样，都是在data数据中定义了x参数和y参数。

作者我这里的代码中有时候会在三种模式之间切换使用，用于区分一些特殊的值。

比如，代码创建贴图时，为了方便区分temp\_sprite的**obj对象本身的属性** 和 **使用时临时定义的参数**。

这里就分开为两种写法：

|  |
| --- |
| // > 创建贴图  var temp\_sprite = new Sprite();  temp\_sprite.x = 100;  temp\_sprite.y = 100;  temp\_sprite['\_drill\_time'] = 0; //计时器  temp\_sprite['\_drill\_bitmapTank'] = []; //资源对象列表 |

另外需要说明的是：

**贴图sprite 和 资源bitmap 都是不可分割的实体对象，不能放在$gameSystem中存储。**

但由于所有js对象都可以像json那样随时挂自己的参数，于是就诞生了这种写法。

另外，如果你没有定义这个参数，而直接用此方法去获取，将得到undefined。

### 详解 - $gameTemp与全局变量

在任意一个js对象中，都可以挂自己用的临时参数。

全局变量$gameTemp也是如此。

上节课中，我们用到了这样一种写法：

|  |
| --- |
| if (command === ">放置贴图") {  $gameTemp.\_drill\_SCC2\_switch = true;  } |

通过在全局变量下 挂”\_drill\_SCC2\_switch”参数，可以定义一个在任何地方都可以使用的开关。

这个参数跨越了插件指令，在帧刷新中被捕获到，并执行了贴图创建。

|  |
| --- |
| //==============================  // \* 帧刷新 - 创建一个贴图  //==============================  Scene\_Map.prototype.drill\_SCC2\_updateCreateSprite = function() {  if( $gameTemp.\_drill\_SCC2\_switch == true ){  $gameTemp.\_drill\_SCC2\_switch = false;  //...  }  } |

不过，使用这种方式捕获时，需要注意两点：

1. 通过标记捕获，不能立即执行。插件指令执行后，必须要在下一帧，才能执行。

2. "\_drill\_SCC2\_switch"参数在挂上之前，获取到值的是 undefined，并非false。

这里我们不妨去看看$gameTemp是怎么定义的。

之前在$gameVariables的定义中，已经介绍到了全局变量的定义。

$gameTemp的定义也在这里。

这些 变量定义 都写在最外层，在js文件加载时，这些参数就被初始化了。

|  |
| --- |
| //==============================  // \* 数据管理器 - 容器定义  //==============================  var $gameTemp = null;  var $gameSystem = null;  var $gameScreen = null;  var $gameTimer = null;  var $gameMessage = null;  var $gameSwitches = null;  var $gameVariables = null;  var $gameSelfSwitches = null;  var $gameActors = null;  var $gameParty = null;  var $gameTroop = null;  var $gameMap = null;  var $gamePlayer = null; |

**全局变量：**是指你可以在任何函数位置调用到的变量。

相对而言，局部变量，只能在指定的函数中，才能调用，否则为undefined。0

|  |
| --- |
| var Imported = Imported || {}; //导入识别类  Imported.Drill\_SimpleCourseB3 = true; //导入的插件标记 |

插件中最外层定义的Imported变量、DrillUp变量，也都是全局变量。

这些变量和$gameTemp的功能一模一样，都充当了 临时全局变量 的功能。

但这里为了统一写法，插件中的 临时全局变量 都会在$gameTemp中挂定义，单独定义出来不合适。

在$gameTemp中挂定义，可以让其他脚本开发者一眼就能理解你参数的作用范围。

|  |
| --- |
| 这里你可能会产生疑问，这里的 $gameSystem 和$gameTemp都是在这里被同时定义的，那么为什么$gameTemp只是临时的，而$gameSystem可以存储呢？ |

答案在下面的函数中：

|  |
| --- |
| //==============================  // \* 存档文件 - 保存存档 - 数据获取  //==============================  DataManager.makeSaveContents = function() {  // （存档数据不含： $gameTemp, $gameMessage, $gameTroop）  var contents = {};  contents.system = $gameSystem;  contents.screen = $gameScreen;  contents.timer = $gameTimer;  contents.switches = $gameSwitches;  contents.variables = $gameVariables;  contents.selfSwitches = $gameSelfSwitches;  contents.actors = $gameActors;  contents.party = $gameParty;  contents.map = $gameMap;  contents.player = $gamePlayer;  return contents;  };  //==============================  // \* 存档文件 - 载入存档 - 数据赋值  //==============================  DataManager.extractSaveContents = function(contents) {  $gameSystem = contents.system;  $gameScreen = contents.screen;  $gameTimer = contents.timer;  $gameSwitches = contents.switches;  $gameVariables = contents.variables;  $gameSelfSwitches = contents.selfSwitches;  $gameActors = contents.actors;  $gameParty = contents.party;  $gameMap = contents.map;  $gamePlayer = contents.player;  }; |

保存、载入时，每个要存储的全局变量，都对应到了一个大的json数据容器中。

而$gameTemp是被排除在外的，

因此$gameTemp中是可以放obj对象的，而$gameSystem要转化字符串，所以不能放对象。

### 详解 - bitmap轮播

说到动图与GIF，你可以先试试思考下面的问题：

|  |
| --- |
| 在电脑上会动的GIF图，打印到纸上后就不会动了，请问怎么解决？ - 知乎  <https://www.zhihu.com/question/307127950> |

动图之所以动，是因为不同的时间段会播放不同的图片，动图必须要有**时间轴**的控制。

单纯的GIF文件是不能被打开的，它需要一个播放器来展开其中的图像，这个播放器可能是浏览器，也可能是window自带的图像播放器。**千万不要产生 ”gif文件能够直接贴在贴图上播放” 的幻觉。**

因此，为了实现动态播放，这里设计了两个控制GIF播放的参数：

|  |
| --- |
| // > 创建贴图  var temp\_sprite = new Sprite();  temp\_sprite.x = 100;  temp\_sprite.y = 100;  temp\_sprite['\_drill\_time'] = 0; //计时器  temp\_sprite['\_drill\_bitmapTank'] = []; //资源对象列表 |

将bitmap数组存放在临时容器’\_drill\_bitmapTank’中，

随后通过计时器，每隔几帧修改贴图的bitmap对象属性来实现切换：

|  |
| --- |
| // > 时间+1  this.\_drill\_SCD2\_sprite['\_drill\_time'] += 1;    // > 开始播放  var bitmap\_list = this.\_drill\_SCD2\_sprite['\_drill\_bitmapTank'];  var time = this.\_drill\_SCD2\_sprite['\_drill\_time'];  time = time / DrillUp.g\_SCD2\_interval; //（每隔n帧）  time = time % bitmap\_list.length; //（循环播放）  if( DrillUp.g\_SCD2\_back\_run ){ //（倒放设置）  time = bitmap\_list.length - 1 - time; //  }  time = Math.floor(time); //（小数取整）  this.\_drill\_SCD2\_sprite.bitmap = bitmap\_list[time]; |

这种bitmap轮播的逻辑结构是固定的，切换数组的bitmap索引，实现轮播，相关的你可以看看插件：

Drill\_LayerGif 地图 - 多层地图GIF

Drill\_BattleGif 战斗 - 多层战斗GIF

Drill\_MenuGif 主菜单 - 多层菜单GIF

这些插件的GIF播放原理都是一样的。

### 详解 - 场景与图层

在编写继承方法时，我们反复看到了 场景Scene的定义，以及sprite图层的注释，这里简要说明一下。

**1）场景**

场景分为三种：**战斗**、**地图**、**菜单**。

这种划分基于 功能 定义的，因为一个战斗场景的体量，要远远大于一个菜单场景。

所有场景都是继承于 Scene\_Base 类。

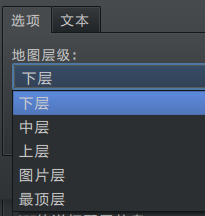
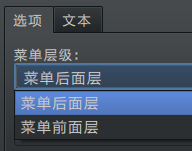


菜单、地图、战斗 的界面继承关系如下：

|  |
| --- |
| 场景基类【Scene\_Base】  └─菜单界面基类【Scene\_MenuBase】  └─各项菜单界面……  └─地图场景【Scene\_Map】  └─战斗场景【Scene\_Battle】 |

**2）图层**

在使用 drill装饰类 插件时，你会发现 战斗界面分成了4个层级，地图界面5个层级，而菜单界面只有2个层级。

这是因为这三类场景，有大图层的划分结构：

**正因为包含了大图层的划分，所以单纯靠改变zIndex并不能完美规划所有贴图的层级。**

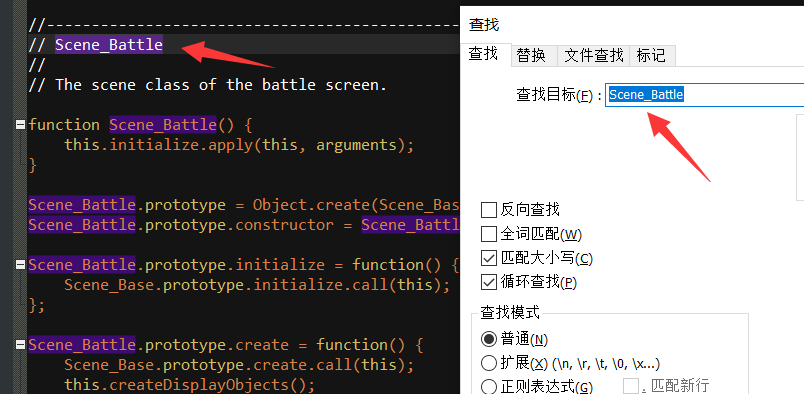
|  |
| --- |
| 菜单界面基类【Scene\_MenuBase】  > 没有图层  地图场景【Scene\_Map】  > 大图层【Spriteset\_Map】 （衍生出 地图活动镜头 插件）  > createLowerLayer() 下层控制函数（远景、图块、事件等）  > createUpperLayer() 上层控制函数（窗口、对话框、计时器等）  战斗场景【Scene\_Battle】  > 大图层【Spriteset\_Battle】 （衍生出 战斗活动镜头 插件）  > createLowerLayer() 下层控制函数（战斗背景、敌人等）  > createUpperLayer() 上层控制函数（窗口、对话框、计时器等） |

由于菜单没有划分图层，所以很多大图层的功能都不能在菜单中使用。

这也是为什么菜单界面无法显示 对话框 的原因。

这些类这里只做简单的介绍，你可以打开 “rpg\_sprites” 和 “rpg\_scenes” 脚本去看看上述的类，

留个初步的印象。



这里只简单介绍了一下 战斗界面的图层和地图界面的图层，后期会详细分析图层的层次结构。

## 开始课程（下）

### 开始写插件

经过了前面课程的放置贴图操作，这里我们再对插件稍微改进一下，把下列步骤完整过一遍。

#### 打开D3脚本

打开脚本。



#### 脚本复制到：资源文件参数部分

将下列脚本复制或者手敲到D3插件的指定位置。

|  |
| --- |
| \*  \* @param 资源-GIF  \* @desc png图片资源组，多张构成gif。  \* @require 1  \* @dir img/Course\_\_D/  \* @type file[]  \* @default []  \*  \* @param 帧间隔  \* @type number  \* @min 1  \* @desc gif每帧播放间隔时间，单位帧。（1秒60帧）  \* @default 4  \*  \* @param 是否倒放  \* @type boolean  \* @on 倒放  \* @off 不倒放  \* @desc true - 倒放，false - 不倒放  \* @default false  \*  \* @param 平移-GIF X  \* @desc x轴方向平移，单位像素。0为贴最左边。  \* @default 100  \*  \* @param 平移-GIF Y  \* @desc y轴方向平移，单位像素。0为贴最上面。  \* @default 100  \* |

#### 脚本复制到：静态数据部分

将下列脚本复制或者手敲到D3插件的指定位置。

|  |
| --- |
| if( DrillUp.parameters["资源-GIF"] != "" &&  DrillUp.parameters["资源-GIF"] != undefined ){  DrillUp.g\_SCD3\_srcList = JSON.parse( DrillUp.parameters["资源-GIF"] );  }else{  DrillUp.g\_SCD3\_srcList = [];  }  DrillUp.g\_SCD3\_interval = Number(DrillUp.parameters["帧间隔"] || 4);  DrillUp.g\_SCD3\_back\_run = String(DrillUp.parameters["是否倒放"] || "true") === "true";  DrillUp.g\_SCD3\_x = Number(DrillUp.parameters["平移-GIF X"] || 0);  DrillUp.g\_SCD3\_y = Number(DrillUp.parameters["平移-GIF Y"] || 0); |

#### 脚本复制到：插件指令部分

将下列脚本复制或者手敲到D3插件的指定位置。

|  |
| --- |
| if( command === ">菜单面板D3" ){  if(args.length == 2){  var type = String(args[1]);  if( type == "打开面板" ){ //打开菜单  SceneManager.push(Scene\_Drill\_SCD3);  }  }  } |

#### 脚本复制到：界面定义部分

将下列脚本复制或者手敲到D3插件的指定位置。

Scene\_MenuBase是一个标准的界面父类，继承后，可以拥有菜单的基本功能，还可以被 装饰插件 装饰。

|  |
| --- |
| //==============================  // \* 面板 - 定义  //==============================  function Scene\_Drill\_SCD3() {  this.initialize.apply(this, arguments);  }  Scene\_Drill\_SCD3.prototype = Object.create(Scene\_MenuBase.prototype);  Scene\_Drill\_SCD3.prototype.constructor = Scene\_Drill\_SCD3;  //==============================  // \* 面板 - 初始化  //==============================  Scene\_Drill\_SCD3.prototype.initialize = function() {  Scene\_MenuBase.prototype.initialize.call(this);  }; |

#### 脚本复制到：界面创建部分

将下列脚本复制或者手敲到D3插件的指定位置。

|  |
| --- |
| //==============================  // \* 面板 - 创建  //==============================  Scene\_Drill\_SCD3.prototype.create = function() {  Scene\_MenuBase.prototype.create.call(this);    this.drill\_createSprite(); //GIF贴图  }  //==============================  // \* 创建 - GIF贴图  //==============================  Scene\_Drill\_SCD3.prototype.drill\_createSprite = function() {    var temp\_sprite = new Sprite();  temp\_sprite.x = DrillUp.g\_SCD3\_x;  temp\_sprite.y = DrillUp.g\_SCD3\_y;  temp\_sprite['\_drill\_time'] = 0; //计时器  temp\_sprite['\_drill\_bitmapTank'] = []; //资源对象列表（注意，这里的数据存放了bitmap对象，所以与sprite一样随时会被销毁）  for(var j = 0; j < DrillUp.g\_SCD3\_srcList.length ; j++){  var src\_str = DrillUp.g\_SCD3\_srcList[j];  var src\_bitmap = ImageManager.load\_CourseD( src\_str );  temp\_sprite['\_drill\_bitmapTank'].push( src\_bitmap );  }  temp\_sprite.bitmap = temp\_sprite['\_drill\_bitmapTank'][0]; //（先赋值第一张图像）    this.addChild( temp\_sprite ); //（添加到场景中）  this.\_drill\_SCD3\_sprite = temp\_sprite;  } |

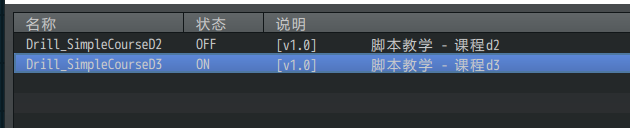
#### 脚本复制到：界面帧刷新部分

将下列脚本复制或者手敲到D3插件的指定位置。

|  |
| --- |
| //==============================  // \* 面板 - 帧刷新  //==============================  Scene\_Drill\_SCD3.prototype.update = function() {  Scene\_MenuBase.prototype.update.call(this);    this.drill\_updateSprite(); //帧刷新 - GIF贴图  this.drill\_updateQuit(); //帧刷新 - 退出监听  }  //==============================  // \* 帧刷新 - GIF贴图  //==============================  Scene\_Drill\_SCD3.prototype.drill\_updateSprite = function() {  if( this.\_drill\_SCD3\_sprite == undefined ){ return; }    // > 时间+1  this.\_drill\_SCD3\_sprite['\_drill\_time'] += 1;    // > 开始播放  var bitmap\_list = this.\_drill\_SCD3\_sprite['\_drill\_bitmapTank'];  var time = this.\_drill\_SCD3\_sprite['\_drill\_time'];  time = time / DrillUp.g\_SCD3\_interval; //（每隔n帧）  time = time % bitmap\_list.length; //（循环播放）  if( DrillUp.g\_SCD3\_back\_run ){ //（倒放设置）  time = bitmap\_list.length - 1 - time; //  }  time = Math.floor(time); //（去掉小数情况）  this.\_drill\_SCD3\_sprite.bitmap = bitmap\_list[time];  }  //==============================  // \* 帧刷新 - 退出监听  //==============================  Scene\_Drill\_SCD3.prototype.drill\_updateQuit = function() {    // > 按键退出  if( TouchInput.isCancelled() || Input.isTriggered("cancel") ) {  SoundManager.playCursor();  SceneManager.pop();  };    } |

#### 加入插件

把Drill\_SimpleCourseD3插件加入到工程中，之前课程的插件关闭。



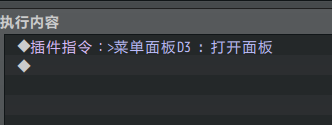
#### 放置图片

将课程所给的图片，放到你新建工程目录下 "\img\Course\_\_D" 文件夹下（注意是两个下划线）。



#### 建立事件

添加一个小爱丽丝，执行插件指令。

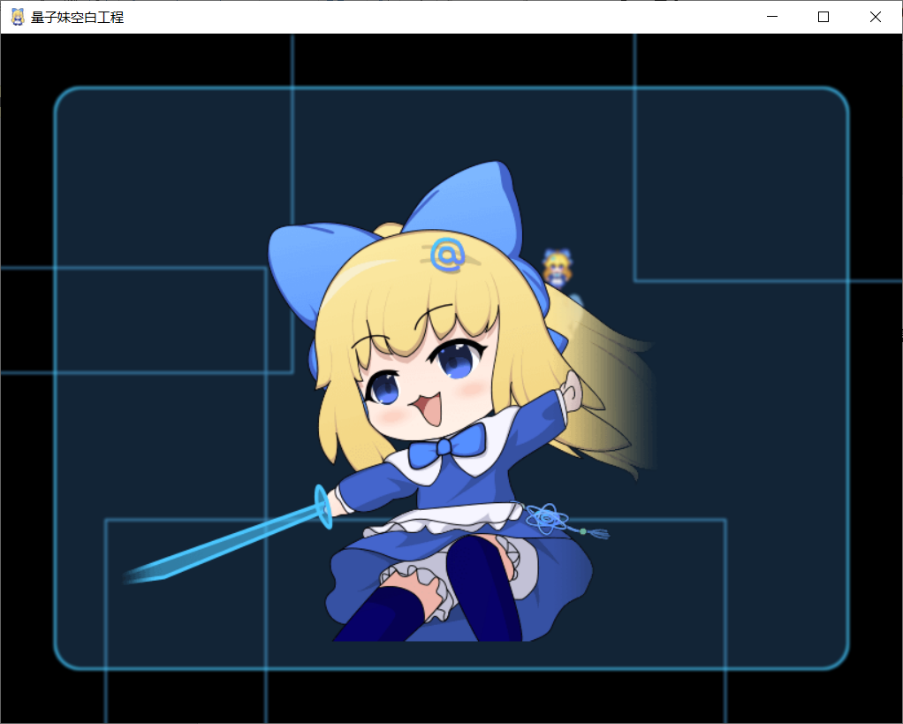
 

#### 功能测试

触发添加的小爱丽丝，可以看到下图。

地图背景被模糊化（因为进入了自定义的菜单界面，默认用模糊底图），

需要按取消键才能离开界面。



ヽ(\*。>Д<)o゜在你完成上述流程之后，接下来我们开始分析一下操作过程中的细节吧。

### 详解 - 自定义场景

如果要自定义一个菜单场景，通常遵循下面的模板即可：

继承Scene\_MenuBase菜单基类，包含 初始化、创建、帧刷新 三项基本的函数。

|  |
| --- |
| //==============================  // \* 面板 - 定义  //==============================  function Scene\_Drill\_SCD3() {  this.initialize.apply(this, arguments);  }  Scene\_Drill\_SCD3.prototype = Object.create(Scene\_MenuBase.prototype);  Scene\_Drill\_SCD3.prototype.constructor = Scene\_Drill\_SCD3;  //==============================  // \* 面板 - 初始化  //==============================  Scene\_Drill\_SCD3.prototype.initialize = function() {  Scene\_MenuBase.prototype.initialize.call(this);  //...  };  //==============================  // \* 面板 - 创建  //==============================  Scene\_Drill\_SCD3.prototype.create = function() {  Scene\_MenuBase.prototype.create.call(this);  //...  }  //==============================  // \* 面板 - 帧刷新  //==============================  Scene\_Drill\_SCD3.prototype.update = function() {  Scene\_MenuBase.prototype.update.call(this);  //...  } |

插件 Drill\_SceneEmpty 全自定义空面板 ，就是一个继承了菜单面板，却没添加任何窗口的面板。

关于Scene\_MenuBase菜单界面的底层定义，后面课程会慢慢剖析。

这里需要注意一点的是，当你进入了一个菜单界面后，需要考虑玩家在界面中操作的情况。

比如玩家按取消键，便能离开面板。（如果你不写这个，玩家就会一直卡在你的自定义菜单了。）

|  |
| --- |
| //==============================  // \* 帧刷新 - 退出监听  //==============================  Scene\_Drill\_SCD3.prototype.drill\_updateQuit = function() {    // > 按键退出  if( TouchInput.isCancelled() || Input.isTriggered("cancel") ) {  SoundManager.playCursor();  SceneManager.pop();  };    } |

而这种 按键->关闭面板 的过程。就是一个简易的 流程。

就像每个全自定义面板文档中反复提及的：

**流程是程序内部无法改变的固定业务逻辑结构，你可以换界面、换外皮，但是无法改变流程，除非新写插件。**

继承了一个场景后，除了给你一个光秃秃的背景，是真的一点操作流程都没有的，

你必须手敲代码，把 鼠标监听、键盘监听、窗口先后顺序 等触发一个个加上。

这里有几个默认常用的 输入设备 监听函数：

（注意，这些监听函数必须放在 帧刷新 中实时监听。）

|  |
| --- |
| if( TouchInput.isPressed() ){} // 鼠标按下确定键（持续）  if( TouchInput.isTriggered() ){} // 鼠标单击确定键（一帧）  if( TouchInput.isRepeated() ){} // 鼠标双击确定键  if( TouchInput.isCancelled() ){} // 鼠标按取消键  if( Input.isPressed("ok") ){} // 键盘按下取消键（持续）  if( Input.isTriggered("ok") ){} // 键盘单击取消键（持续）  if( Input.isRepeated("ok") ){} // 键盘双击取消键  if( Input.isPressed("cancel") ){} // 键盘按下取消键（持续）  if( Input.isTriggered("cancel") ){} // 键盘单击取消键（持续）  if( Input.isRepeated("cancel") ){} // 键盘双击取消键 |

注意，默认的函数在部分情况下可能并不够用。

如果你需要更完整的触发监听函数，

可以去看看插件 Drill\_CoreOfInput 系统 - 输入设备核心 提供的接口。

|  |
| --- |
| if( TouchInput.drill\_isWheelUp() ){ } // 滚轮向上[一帧]  if( TouchInput.drill\_isWheelDown() ){ } // 滚轮向下[一帧]  if( TouchInput.drill\_isLeftPressed() ){ } // 左键按下[持续]  if( TouchInput.drill\_isLeftTriggerd() ){ } // 左键按下[一帧]  if( TouchInput.drill\_isLeftReleased() ){ } // 左键释放[一帧]  if( TouchInput.drill\_isLeftDoubled() ){ } // 左键双击[一帧]  //... |

### 详解 - 场景与图层II

现在我们知道了菜单场景，有 初始化、创建、帧刷新 三个基本函数，但这三个函数的执行时机是如何的呢？是否还有其他基类的函数？

这里我们不妨去看看底层文件：“rpg\_scene.js”

场景基类Scene\_Base 的定义。

|  |
| --- |
| //==============================  // \* 场景基类 - 定义  //==============================  function Scene\_Base() {  this.initialize.apply(this, arguments);  }  Scene\_Base.prototype = Object.create(Stage.prototype);  Scene\_Base.prototype.constructor = Scene\_Base;  //==============================  // \* 场景基类 - 初始化  //==============================  Scene\_Base.prototype.initialize = function() {  Stage.prototype.initialize.call(this);  this.\_active = false;  this.\_fadeSign = 0;  this.\_fadeDuration = 0;  this.\_fadeSprite = null;  this.\_imageReservationId = Utils.generateRuntimeId();  };  //==============================  // \* 场景基类 - 创建  //==============================  Scene\_Base.prototype.create = function() {  };  //==============================  // \* 场景基类 - 帧刷新  //==============================  Scene\_Base.prototype.update = function() {  this.updateFade();  this.updateChildren();  };  //==============================  // \* 场景基类 - 开始运行  //==============================  Scene\_Base.prototype.start = function() {  this.\_active = true;  };  //==============================  // \* 场景基类 - 结束运行  //==============================  Scene\_Base.prototype.stop = function() {  this.\_active = false;  };  //==============================  // \* 场景基类 - 析构函数  // 说明： 退出该场景/切换场景 前，执行的函数。  //==============================  Scene\_Base.prototype.terminate = function() {  }; |

场景基类 给予了一全套流程方法，这里的运行的顺序如下：

**初始化 -> 创建 -> 帧刷新 -> ”加载中” -> 开始运行 -> 结束运行 -> “切换场景时黑屏” -> 析构函数**

这里在继承时需要注意下面几点：

1. 创建(create)，比 帧刷新(update) 执行要早。

2. 初始化(initialize)、创建(create)、帧刷新(update) 是常用继承函数，其他的函数一般少见。

3. 在场景创建后，将进入短暂的”加载中”界面，这个时候 帧刷新(update) 是在持续执行的，而 开始运行(start) 在等待加载，加载全部资源完成才会执行。

4. 如果你执行了 切换场景 指令，那么 结束运行(stop) 函数会被立即执行。等完全离开场景后，会执行 析构函数(terminate)。

## 课程小结

下面来总结一下课程的全部内容，内容密度有点大，如果你对下面的知识点仍然感到不好把握，可以回去看看，或者自己试验试验。

1）**基本意识**：所有底层函数和插件都是用ES5的兼容写法。

2）**数组的定义**：js数组中如果直接使用aaa[5]=10赋值，数组的长度可能会自适应变化。定义一个数组后，一定要确保数组内的类型都是一致的，否则会出现天线宝宝乱入的错误结果。

3）**遍历**：尽量使用最原始的for循环结构，最容易理解。数组一定要统一数据类型，否则求和的结果不稳定。

4）**”变量类”剖析**：rmmv的变量类Game\_Variables，本质上就是一个数组。变量值改变时，会发送刷新地图的信号。

5）**对象与JSON**：这里称呼的“对象”都是不可分割的结构，而json是可以划分并显示出字符串的结构。a[‘x’]和a.x 的定义效果是一样的，使用不同写法可以区分 对象自身的属性 和 临时使用的属性。

6）**$gameTemp**：建议把所有临时使用的全局变量，都挂到$gameTemp全局变量下，方便其它人理解。

7）**全局变量**：全局变量即写在最外层的定义，在任何函数中都可以直接引用。

8）**bitmap轮播**：动图必须有时间轴的定义，需要你自己写计时器控制图像播放。

9）**场景与图层**：场景分为菜单、地图、战斗，其中菜单场景都应该继承一个菜单基类Scene\_MenuBase。地图界面中，有专门的地图图层类；战斗界面中，有专门的战斗图层类。

10）**自定义场景**：通过继承Scene\_MenuBase菜单基类，实现 初始化、创建、帧刷新 三项基本的函数，就可以自定义一个菜单界面。但是需要注意，新的菜单界面没有任何 流程，需要手动添加完整的业务逻辑。

11）**场景与图层II**：场景基类不但有 初始化、创建、帧刷新 三个基本函数，还有开始运行、结束运行、析构函数等接口。

|  |
| --- |
| 课后作业：写一个自定义场景的插件，插件中可以配置一个 整体布局（静态贴图）和两个GIF贴图。  作业提示：注意，你要写三个部件，一个静态贴图，两个GIF贴图；编写时如何划分函数，如何命名区分贴图，是很关键的，一定要熟练哦！ |

如果你能完成这个作业，就说明你已经有场景和贴图的基础啦！

|  |
| --- |
| **挖的坑**：  关于Scene\_MenuBase菜单界面的底层定义，后面课程会慢慢剖析。  这里只简单介绍了一下 战斗界面的图层和地图界面的图层，后期会详细分析图层的层次结构。 |