## 概述

### 背景

现在网络上随便百度都能找到大量let、const、箭头函数等代码。

而这些内容都是es6的特性，不能在es5的环境中支持。同时也为了防止以后百度出现es789什么的乱七八糟的新功能混淆，这里需要说明一下稳定的es5函数写法。

2020/6 ，ES11发布了。我们这ES5已经成古董了……新版又加了一堆特性，似乎也都用不上……

**至少，es5环境可以支持毕竟古老的浏览器，包括xp系统，相对比更高版本要稳定。**

所以，尽量先用这里的格式，不建议用百度找的不明js写法。至少在插件中，下面的用法是经过反复试验有效的。

## 语言

### 简单区分

Rmmv固定使用js语言（这里的js以ES5为标准），typescript暂时和js划分在一起，毕竟同源。如果你是从其他语言转到这里，有必要了解一下。

另外，截止2020/6，ruby语言已经快要死透了，现在用matlab语言的人都比用ruby的人多，如果你是从xp、va或vx转来的，赶紧弃了ruby吧。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Javascript/Typescript | C++/C# | Java | Python | Ruby |
| 简介 | 一门语言，  少了node.js或html5，  什么都干不了。 | 一门语言，  少了vs编译器，  什么都干不了。 | 一门语言，  少了虚拟机jvm或dvm，  什么都干不了。 | 一门语言，  少了python库，  什么都干不了。 | 一门语言，  少了rmvx，rmva，rmxp  什么都干不了……（误） |
| 缩进 | 随意缩进 | 随意缩进 | 随意缩进 | 严格缩进 | 随意缩进 |
| 分号 | 可加，可不加 | 必须加 | 必须加 | 可加，可不加 | 可加，可不加 |
| 花括号 | 函数代码块，  var a = {} 定义json | 函数、内部类代码块，  int\* a = new int[3]{1, 2, 3}; 定义数组 | 函数、内部类代码块，  int a[] = new int[]{1, 2, 3}; 定义数组 | 与函数无关（函数用冒号），  a = {} 定义 字典 | 与函数无关  b = {1 => 2} 定义 散列 |
| 常量 | js ES5木有常量定义，  ts用const a = 0； | #define DRILL 0  const int a =0; | public static final int a = 0; | 木有常量定义 | 木有常量定义 |
| 变量 | var a;  写int a会炸。 | int a; float a; double a;  bool a; …… | int a; float a; double a;  boolean a; …… | a = 1  a = "a" | a = 1  a = "a" |
| 注释 | // 单行注释  /\* 多行注释 \*/ | // 单行注释  /\* 多行注释 \*/ | // 单行注释  /\* 多行注释 \*/ | # 单行注释  ’ ’ ’ 多行注释 ’ ’ ’ | # 单行注释  =begin  多行注释  =end |
| if | if( aaa != 1 ){ . . . }  else if( aaa == 1 ){ . . . }  if( aaa !== 1 ){ . . . }  else if( aaa === 1 ){ . . . } | if( aaa != 1 ){ }  else if( aaa == 1 ){ } | if( aaa != 1 ){ . . . }  else if( aaa == 1 ){ . . . } | if not aaa == 1 :  . . .  elif aaa == 1 :  . . . | if aaa != 1  . . .  elsif aaa == 1  . . . |
| for | var a = [1,2,3];  for(var i=0; i<a.length; i++){  . . .  } | int len = 3;  int\* a = new int[len]{1, 2, 3};  for( int i=0; i < len ; i++ ){  . . .  } | int a[] = new int[]{1, 2, 3};  for(int i=0;i<a.length(); i++){  . . .  } | a = [1,2,3]  for i in range(len(a)):  . . . | arr = (1..10).to\_a  for i in arr do  . . .  end |
| 布尔 | true和false字母必须小写 | true和false字母必须小写 | true和false字母必须小写 | True和False  首字母必须大写 | true和false字母必须小写 |
| 引号 | 单引号和双引号都表示字符串。 | 单引号’ ’表示字符。  双引号””表示字符串。 | 单引号’ ’表示字符。  双引号””表示字符串。 | 单引号和双引号都表示字符串。 | 单引号和双引号都表示字符串。 |
| 类成员 | this.xxx  this["xxx"] | this.xxx  this->xxx | this.xxx | this.xxx | @xxx |

### 注释颜色

根据之前提及的js特性：

1. 字符串的单引号和双引号都是字符串。
2. js单行注释和多行注释没有区别，只是写法不一样。

根据上述的特性，个人在编辑器里面将上述四中内容染上了不同的显眼颜色。并且用于划分类型。如下。

|  |
| --- |
| /\* 【rmmv默认注释，全绿】  \* -----------------------------------------------------------------------------  \* ----更新日志  \* [v1.0]  \* 完成插件ヽ(\*。>Д<)o゜  \*/ |

|  |
| --- |
| //=============================================================================  // \* 插件指令【大分段】  //=============================================================================  var \_drill\_EBV\_pluginCommand = Game\_Interpreter.prototype.pluginCommand;  Game\_Interpreter.prototype.pluginCommand = function(command, args) {  \_drill\_EBV\_pluginCommand.call(this, command, args);  if (command === ">事件缓存变量") {  /\*-----------------自定义区域【中分段】------------------\*/  if(args.length == 8){  var unit = String(args[1]);  if( unit == "本事件" ){  // > 事件控制【小分段】  var e\_id = this.\_eventId;  // ...  }  }  }  } |

|  |
| --- |
| this.\_drill\_ESE = {};  this.\_drill\_ESE['redraw\_command'] = false; //重画指令  this.\_drill\_ESE['shatter\_command'] = false; //碎片指令  this.\_drill\_ESE['shatter\_id'] = -1; //当前碎片样式id  this.\_drill\_ESE['shatter\_data'] = {}; //当前碎片样式数据  this.\_drill\_ESE['shatter\_converted'] = false; //反向弹道 |

区分的理由如下：

1. rmmv注释，全程字符串，没法区分。所以只要和代码颜色不一样就可以了，那么就鲜绿色吧。

2. 单行和多行注释常用于给不同的代码分段，

图中定义了大分段（//====）、中分段（/\*-----\*/）、小分段（// > 步骤），因为颜色鲜明，所以很容易注意到注释内容。

3. js的字符串用的非常多，但是大多都是用于['xxxx']指向成员的。所以一般成员用的字符串，使用单引号，且颜色用深绿色，不显眼。另外，成员的英文名并不好理解，而更多的是需要关注中文注释，所以，每个成员都基本用单引号+中文注释。

4. rmmv用到字符串的时候，比较多的是bitmap资源路径、插件指令切割，为了让字符串更加显眼，所以也设置成鲜绿色。

ㄟ(ʅ⊙ω⊙) 其实你不用刻意模仿，按照你的习惯来，这里我只是提及一下我看到的代码是什么样的，因为鲜明的颜色容易引起注意，仅此而已。

比起其他编辑器花里胡哨的颜色，我的编辑器里面是完全没有 蓝色 的。

因为蓝色 = 量子妹 = 游戏界面 = 画画。代码和画画通过颜色分成两个维度，至少不会让我感到不安。

另外，代码耗血，画画耗魔，魔法肯定是蓝色的，哈哈(\*ˉ﹃ˉ) 。

## js写法

### Object（对象）

这里的”对象”称呼与js语言本身有一些区别，这里提及一下。

Js语言的任何类型都是对象。

这里rmmv中对象的称呼与纯数据相互区分。

rmmv称呼的定义：

**Object对象：**指贴图、窗口、有图像数据的实体对象。

**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。

二者区别：

object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纯数据可以保存进存档。



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

对象是不能放存档的，只能通过其他方式绕开。

如果你必须要将使得对象能在全局中调用，你可以放入$gameTemp。

在drill插件中，并没有对对象的命名做明确划分。

不过，常见的 对象 命名为：temp\_sprite、temp\_window、event、temp\_obj等。

常见的 纯数据 命名为：temp\_data、data、this.\_drill\_data。

### 数组（容器）

在drill插件中，数组命名会习惯性地添加一个后缀”list”或”tank”(表示容器)，比如”\_drill\_SCA\_spriteTank”或”\_drill\_SCA\_dataList”用于区分数组与json。

定义数组：

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

判断是否为数组：

如果aaa\_tank是字符串，那么length也会存在。你也可以用Array.isArray( aaa\_tank )来判断数组。

|  |
| --- |
| if( aaa\_tank.length != undefined ){ } |

判断是否为空数组：

|  |
| --- |
| if( aaa\_tank.length === 0 ){ } |

判断是否为非空数组：

|  |
| --- |
| if( aaa\_tank.length > 0 ){ } |

数组赋值：

注意，数组里面只能写数字，不要错写成arr['aaa']字符串了，写成字符串不会报错，但是读到的数据会出问题。如果你加的是一个变量，建议先转一下数字。

|  |
| --- |
| aaa\_tank[0] = 1;  aaa\_tank[1] = 2;  aaa\_tank[Number(bbb)] = 3;  aaa\_tank.unshift( 4 ); //头部插入  aaa\_tank.push( 5 ); //尾部插入 |

for遍历：

这里建议所有数组都使用下面的标准格式，一眼就能看明白，不建议数组到处用for in、for of、foreach写法。

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

退数组：

通常 整体销毁spriteTank 或 根据条件删除sprite 时经常用，注意数组必须是递减遍历。

|  |
| --- |
| 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  } |

map、foreach用法：

ES5是支持map和foreach的，但是要注意后面要跟个 ”, this);” 回调函数。这种写法单独存在还行，而如果出现了反复嵌套，建议直接分新函数来写。

|  |
| --- |
| var aaa\_tank = ccc\_tank.map(  function(value,index,array){  return String(value).toUpperCase();  }  , this ); |

数组拼接数组：

|  |
| --- |
| area\_tank = area\_tank.concat( aaa\_tank ); |

### Json（数据）

在drill插件中，json命名一般会统称data，然后data下列出各种子成员。

如果变量没有”data”名字，一般可能是obj对象。

定义json：

|  |
| --- |
| var data = {}; |

json赋值：

注意，json最好里面只能写字符串，不要错写成data[0]了，写成数字也不会报错，但是后期读代码会很麻烦。另外，成员尽量用单引号，不要用双引号。如果你加的是一个变量，建议先转一下成字符串。

|  |
| --- |
| data['cancel'] = 1;  data['ok'] = 2;  data[String(aaa)] = 3; |

获取json的全部键：（keys是一个数组）

|  |
| --- |
| var key\_list = Object.keys( data ); |

json遍历：

建议只有json遍历时，才用for in 写法。或者转数组，然后进行正常数组遍历。

|  |
| --- |
| for(var key in this.\_drill\_AnimExData\_data ){  var temp = this.\_drill\_AnimExData\_data[key];  }  // 或者  var keys = Object.keys(this.\_drill\_AnimExData\_data);  for(var i = 0; i<keys.length; i++){  var temp = this.\_drill\_AnimExData\_data[keys[i]];  } |

### 类

定义类(单独物体，无继承父类)：

|  |
| --- |
| //==============================  // \* 时间条物体 - 定义  //==============================  function Drill\_GOBTB\_GameTimeBar() {  this.initialize.apply(this, arguments);  }; |

定义类(继承父类的贴图)：

贴图继承有initialize和update两个基本函数，继承写法如下。

|  |
| --- |
| //==============================  // \* 时间条贴图 - 定义  //==============================  function Drill\_GOBTB\_TimeBarSprite() {  this.initialize.apply(this, arguments);  };  Drill\_GOBTB\_TimeBarSprite.prototype = Object.create(Sprite\_Base.prototype);  Drill\_GOBTB\_TimeBarSprite.prototype.constructor = Drill\_GOBTB\_TimeBarSprite;  //==============================  // \* 初始化  //==============================  Drill\_GOBTB\_TimeBarSprite.prototype.initialize = function() {  Sprite\_Base.prototype.initialize.call(this);  // . . .  };  //==============================  // \* 帧刷新  //==============================  Drill\_GOBTB\_TimeBarSprite.prototype.update = function() {  Sprite\_Base.prototype.update.call(this);  // . . .  }; |

判断基本类型：

通过typeof。注意 等于与恒等于，常用 等于”==”。

|  |
| --- |
| if( typeof(\_drill\_newEvent\_event) == "undefined"){ }  if( typeof(\_drill\_newEvent\_event) === undefined ){ } |

判断当前类的名称：

|  |
| --- |
| if( this.constructor.name == "Scene\_Battle" ){ } |

判断子类继承：

|  |
| --- |
| if( this instanceof Drill\_Sprite\_LRR ){ } |