

THE NVL MAKER

新手教程

(五)

存取界面

一、工程设定

之前已经在系统设定有关的教程里大致讲了下 Config。
里面存取相关的东西主要就是这些。

全局设定

档案与通过记录

菜单设定

文字层设定

字体设定

图形与音声设定

历史记录

存档文件设置

存档目录

文件前缀

data

档案格式

普通

存档总数

10

☐ 在存档中保存宏信息

缩略图

☒ 存档时保存缩略图

缩略图宽度

160

色彩模式

24位色位图

特殊存档模式

☐ 只读模式

☐ 自由存档

资料ID

00000000-0000-0000-0000-000000000000

通过记录

☒ 自动记录已读标签

记录模式

手动记录

记录数里

5

自动标签 (AutoLabel) 模式

☒ 使用自动标签 (按行存档)

标签模式

临时标签模式

档案总数：
假设你预计存储界面一页显示 5 个档案，一共打算 5 页，那么这里就设定 25，是很简单的计算。

保存缩略图：
推荐不用去动它，就一直勾着。

图片宽度：
具体的大小会显示在界面编辑里。
假如是使用悬停效果，那么值可以设大一点，如果是每个按钮上都要放小截图，那当然是小一点的好。

二、存储、读取图形设定

因为存取部分要的元素比较多，所以把所有图形有关的都单独出来，界面编辑器里只负责设定排版位置。因此有的时候可能需要在几个设定中间来回切换以查看效果了。

三、界面编辑器

（1）存档按钮数量、位置设定

每个按钮对应的就是一个存档位置。每页的档案个数可以在这里设定：

档案设定

每页档案数

3

设定

档案1

x

320

y

120

档案2

x

320

y

270

档案3

x

320

y

420

当输入数值以后点“设定”，增加或减少每页的存档按钮数量。
接着返回上层界面，就可以拖动新的存档按钮的位置了。
不过注意，每页的档案数肯定不能超过工程设定里的“档案总数”。要是设定了太大的值，编辑器会自动调整。

通过拖动调整新的存档按钮位置：



（2）存档按钮上的信息显示



在“内容设定”里可以调整存档按钮上显示的信息。

勾选必要的信息并设定在按钮上的相对位置，就可以显示不同的文字或截图。

（3）新档标记

设定以后，玩家最后一次存档的栏位附近会显示一张图片标记。

（4）悬停效果



所谓“悬停效果”，是指当鼠标移动到某个存档按钮上时，会进一步在画面其他地方显示这个存档的详细信息。而当鼠标移开时，信息就被隐藏。

支持显示的内容和存档按钮上可以显示的内容差不多。

四、继续自己写系统

相信上次的系统设定里加入“恢复默认值”按钮的体验并不是很吓人。

这次讲的依然是通过修改脚本来增加编辑器里没有的功能，准备好了吗？

不过，因为这次有存储、读取两个系统，所以一共要改两个文件。

当然就是 macro 文件夹下的 save.ks 和 load.ks 了。

（1）宏

首先来说说之前一直提的“宏”吧。

所谓“宏（macro）”，作用是把一直重复用到的几个指令合并成一个新的指令，然后使用这个新的指令来节省工作量。

在 KAG 脚本里，定义了[macro name=宏名称]和[endmacro]中间的内容后，就可以用[宏名称]来调用指令。

比如说

```
[macro name=hero]
```

```
[emb exp="f.姓"]
```

```
[emb exp="f.名"]
```

```
[endmacro]
```

那么每次使用[hero]就代表在对话里显示了 f.姓的值和 f.名的值。

可能一开始写脚本的时候并不会感到不用宏有什么问题，不过，游戏的某个特定效果总是要频繁出现，哪怕重复的部分只有两三行也相当烦人了。如果能用一行来解决，总是比两行要快一些。

还有一个好处就在于，当你需要修改这个效果的时候——如果用的是宏，只要修改一个地方，如果你在每个地方都复制粘贴了这几行代码，就准备全文搜索吧……

总的来说，用 KRKR 的有三种人……

(2) 普通青年&牛逼青年&213 青年

	脚本一	脚本二	介绍
普通青年的脚本	<pre>[iscript] //函数 Function abc() { Xyz; } [endscript] ;宏 [macro name=def] [ghi] [jkl] [opq] [rst] [endmacro] ;----- [return]</pre>	<pre>;定义宏 [call storage="脚本一"] ;XX 界面 *start [rclick] [eval exp="abc()"] [def] [s] ;返回游戏 *return [return]</pre>	<p>使用 KS 的时候比用 TJS 的时候更多。函数和宏放在一个脚本，实际的执行逻辑放在另外一个脚本里。</p> <p>总的来说中规中矩。</p>
牛逼青年的脚本	<pre>Class abc{ Var ghi; Var jkl; function abc(..) function finalize(...) Property xyz{ }</pre>	<pre>[button exp="f.ui=new abc()"]</pre>	<p>除了剧情脚本，几乎找不到 KS 脚本的存在。大部分功能都封装成 TJS 的类。调用的时候直接用一个 TJS 式搞定。</p>
213 青年的脚本	<pre>*start [rclick] [ghi] [jkl] [opq] [jump target=*label3] *label1 [rst] [ghi] [jkl] [opq] [jump target=*start] （……因为太长了所以接右边）</pre>	<pre>*label2 [rst] [ghi] [jkl] [opq] [jump target=*label4] *label3 [rst] [s] *label4 [ghi] [jkl] [return]</pre>	<p>没有第二个脚本所有的东西都写在一起。凡是需要重复使用的，都靠复制粘贴。大量使用跳转蹦到奇怪的地方，搞到最后自己也不知道到底执行逻辑是什么样。而且，完全没有注释……=_=</p>

(3) 确定要添加内容的位置

THE NVL Maker 作者属于标准的普通青年,所以游戏模板工程基本就是个宏和函数的大集合。
存取系统相关的函数和宏,放在 macro_sl.ks 里。
从而保证 save 和 load 的脚本内容依然非常简单。

```
;-----  
;非常懒惰的 save 显示  
;-----  
*start  
[locksnapshot]  
[tempsave]  
;-----  
*window  
;载入配置文件  
[iscript]  
f.config_sl=Scripts.evalStorage("uisave.tjs");  
[endscript]  
[history enabled="false"]  
  
*firstpage  
[locklink]  
[rclick enabled="true" jump="true" storage="save.ks" target=*返回]  
[backlay]  
[image layer=14 page=back storage=&"f.config_sl.bgd" left=0 top=0 visible="true"]  
  
;隐藏系统按钮层  
[hidesysbutton page="back"]  
;用 message4 描绘  
[current layer="message4" page="back"]  
[layopt layer="message4" visible="true" page="back" left=0 top=0]  
[er]  
[eval exp="drawslbutton('back')"]  
;这里是我们添加内容的地方  
  
[trans method="crossfade" time=500]  
[wt]  
[s]  
;-----  
*刷新画面  
[current layer="message4"]  
[er]  
[eval exp="drawslbutton()"]  
;这里是我们添加内容的地方
```



```
[s]
```

```
;-----
```

(以下省略)

(4) 加入“页数选择”按钮

THE NVL Maker 的界面编辑器里提供了“向上翻页”和“向下翻页”两个按钮。可以在不同的存档页中间切换。但是如果存档页比较多，从第一页切到第五页等等，可能就要多点很多次按钮。玩家也并不是很清楚一共有多少页存档。

所以有的时候，会隐藏翻页按钮，而直接加入页数选择按钮。

假设存档一共有 5 页，那么就在界面上添加【1】~【5】的 5 个数字按钮。

这里的按钮图片我们就先借用一下 image 文件夹里的图形数字吧。

我想在系统设定 (option.ks) 里，[locate]和[button]的使用，各位还是记得的？

这回也不过是设定按钮，只不过一个变成了 5 个。

首先在刚打开界面的时候进行描绘。

```
;用 message4 描绘
```

```
[current layer="message4" page="back"]
```

```
[layopt layer="message4" visible="true" page="back" left=0 top=0]
```

```
[er]
```

```
[eval exp="drawslbutton('back')"]
```

```
;这里是我们要添加内容的地方
```

```
[locate x=50 y=150]
```

```
[button normal="num-1"]
```

```
[locate x=50 y=190]
```

```
[button normal="num-2"]
```

```
[locate x=50 y=230]
```

```
[button normal="num-3"]
```

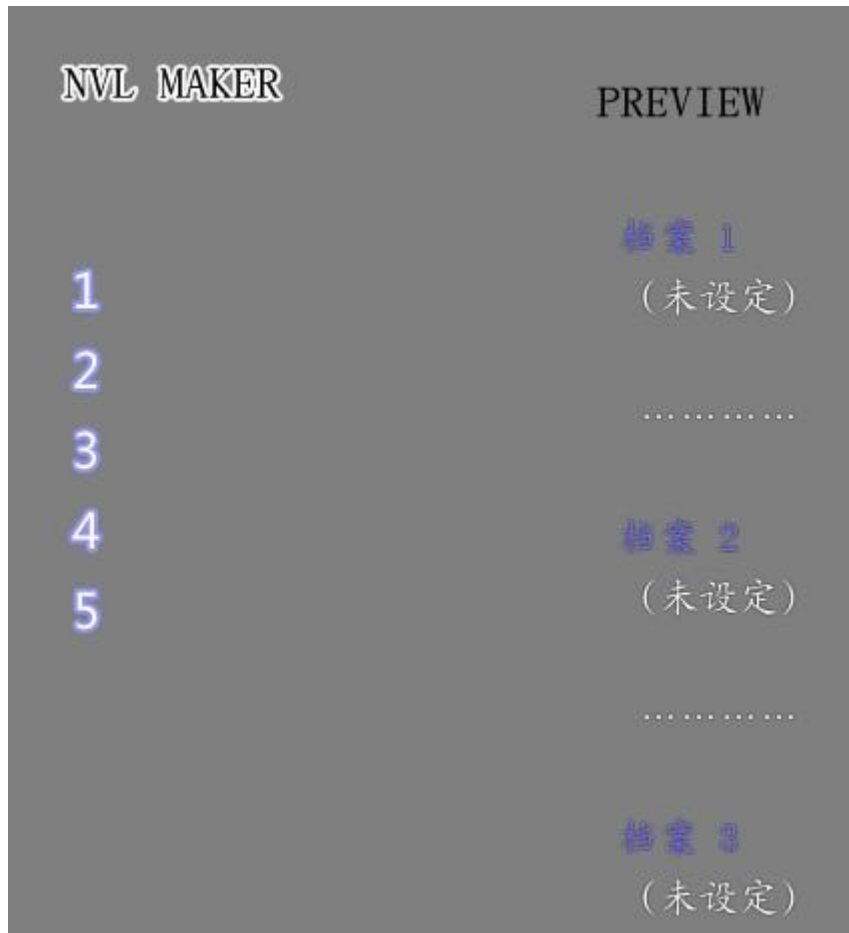
```
[locate x=50 y=270]
```

```
[button normal="num-4"]
```

```
[locate x=50 y=310]
```

```
[button normal="num-5"]
```

现在测试一下：

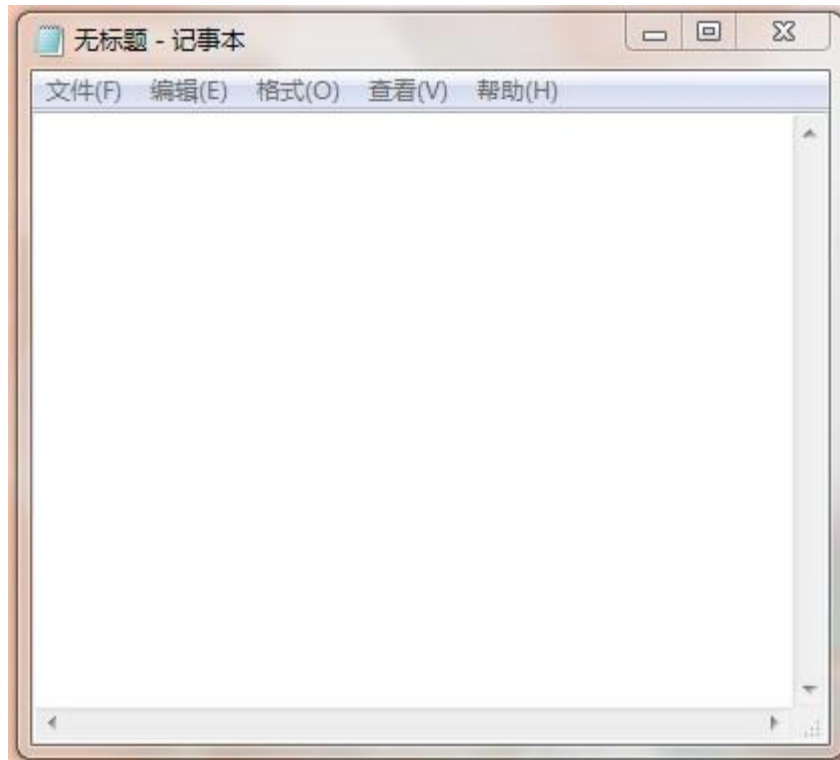


左边显示了 5 个按钮。似乎是成功了。
接下来，为了避免成为 213 青年，把它封装成宏。

（5）封装

所谓“封装”，这里的意思就是定义一个宏。
而当系统读过这个宏定义以后，你就可以在以后的脚本里使用这个定义了。
假设把这个宏的名字叫做[存取系统_翻页按钮]。
接下来找个地方，加入自己的宏定义吧？
要做的事情是，
一，新建一个自己的脚本用来保存宏定义
二，把宏定义写进脚本
三，让系统读入这个脚本一次

首先，打开记事本，



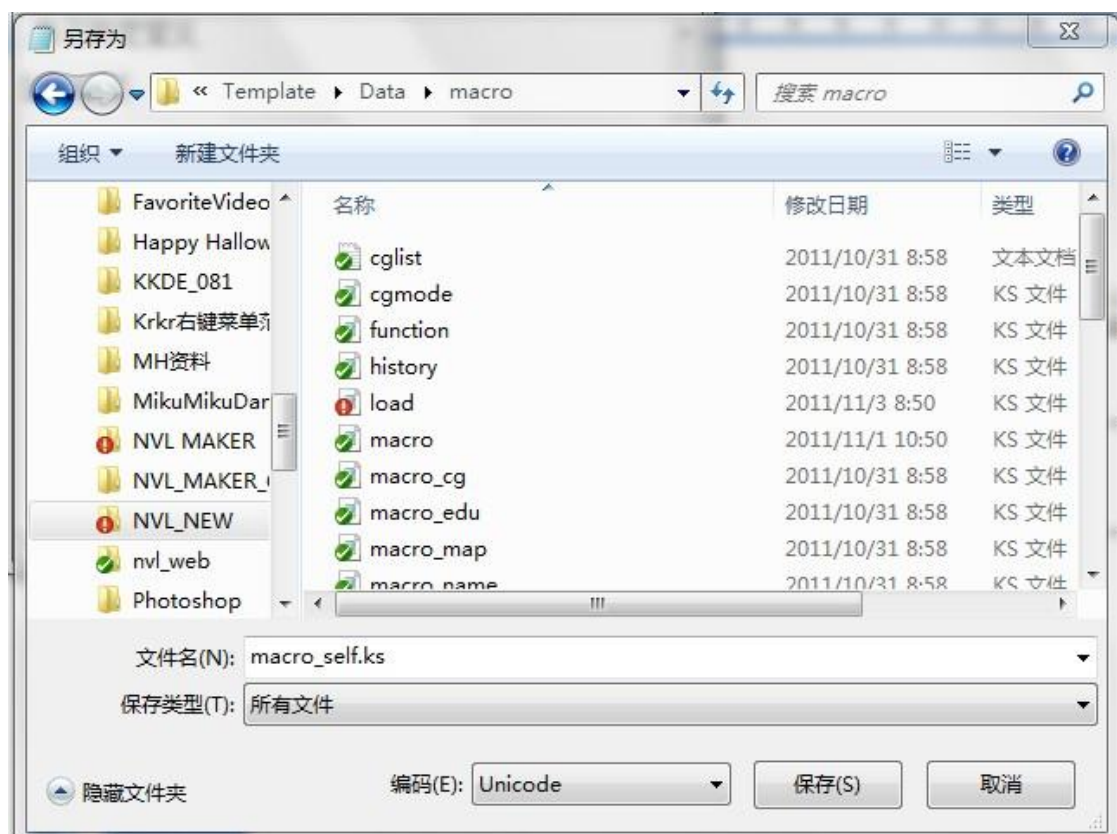
填进去以下内容：

;自己的宏定义

[return]

然后选择保存。

不要忘记[return]哦。



这里有两个地方要注意，第一个是保存类型选择“所有文件”，然后填写文件名 macro_self.ks。

第二个则是这里：

编码(E): Unicode

，必须要设定成 Unicode。

假如你的 windows 系统不提供自动转存 Unicode 格式的话，可以用 THE NVL Maker 的“脚本编辑-新建脚本”功能，存下一个空白脚本，再打开进行编辑。

将所有的脚本存成 Unicode（Unicode-LE）编码可以保证你的游戏在任何语言版本的操作系统下都可以正常执行而不报错，所以非常重要。

接下来，继续修改脚本的内容：

```

;自己的宏定义
[macro name=存取系统_翻页按钮]

[endmacro]
[return]

```

现在直接把刚刚填写的按钮位置和设定直接剪切过来贴到这里。变成这样：

```

;自己的宏定义
[macro name=存取系统_翻页按钮]
[locate x=50 y=150]
[button normal="num-1"]
[locate x=50 y=190]

```

```
[button normal="num-2"]  
[locate x=50 y=230]  
[button normal="num-3"]  
[locate x=50 y=270]  
[button normal="num-4"]  
[locate x=50 y=310]  
[button normal="num-5"]  
[endmacro]  
  
[return]
```

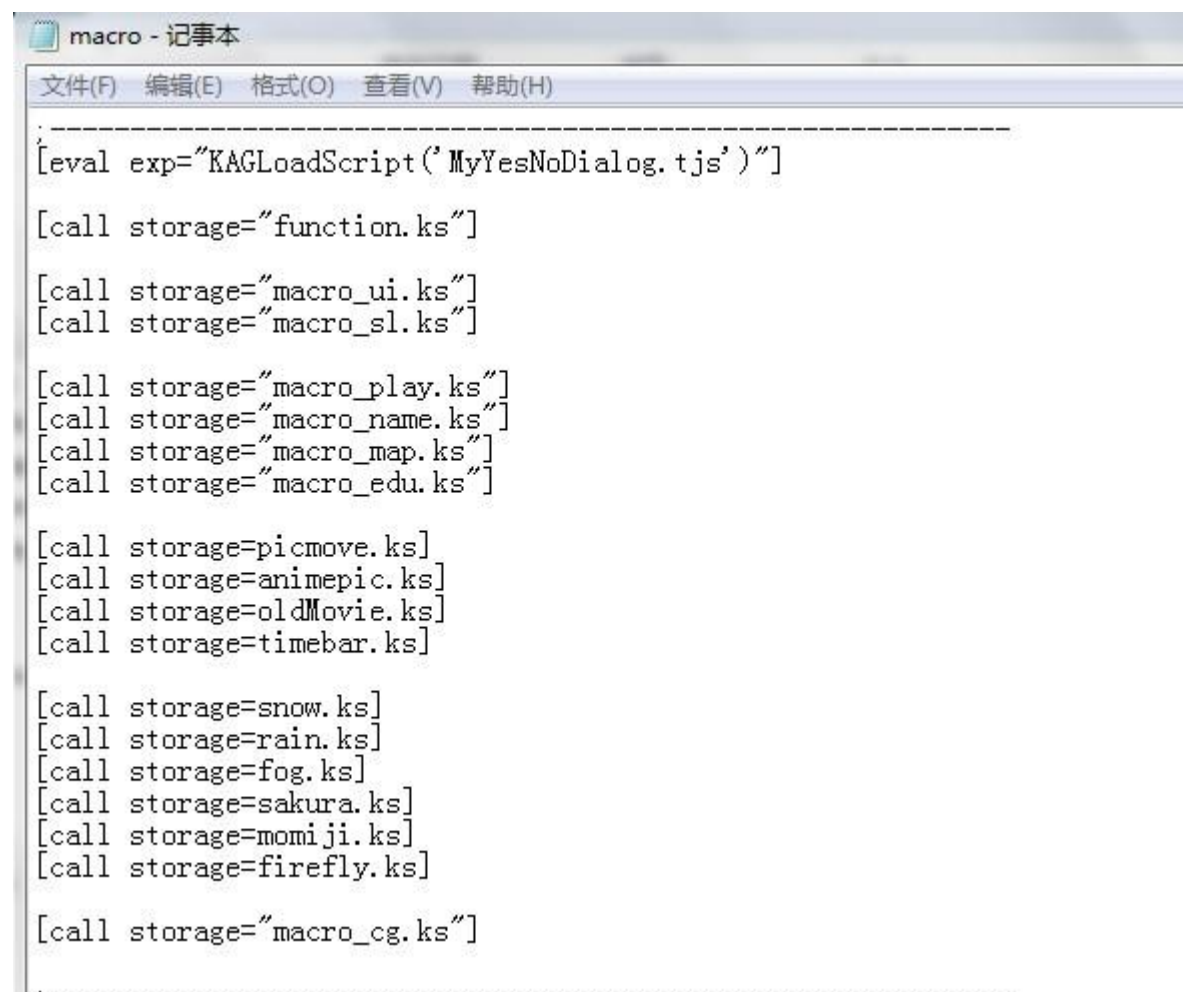
定义完毕。

现在，新建的这个脚本从来没被系统读取过，因此这个宏虽然写完了，却还没人知道。

我们需要在某个已经存在的脚本里“呼叫”一次这个脚本。

这里 macro.ks 是最好选择，因为 THE NVL Maker 工程所有的宏脚本都是在这里读取的。

打开 macro.ks



```
macro - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
.  
[eval exp="KAGLoadScript('MyYesNoDialog.tjs')"]  
[call storage="function.ks"]  
[call storage="macro_ui.ks"]  
[call storage="macro_sl.ks"]  
  
[call storage="macro_play.ks"]  
[call storage="macro_name.ks"]  
[call storage="macro_map.ks"]  
[call storage="macro_edu.ks"]  
  
[call storage=picmove.ks]  
[call storage=animepic.ks]  
[call storage=oldMovie.ks]  
[call storage=timebar.ks]  
  
[call storage=snow.ks]  
[call storage=rain.ks]  
[call storage=fog.ks]  
[call storage=sakura.ks]  
[call storage=momiji.ks]  
[call storage=firefly.ks]  
  
[call storage="macro_cg.ks"]  
.
```

在[call storage="macro_cg.ks"]下面加入一句

[call storage="macro_self.ks"]。

保存，关闭。——现在你在任何地方使用[存取系统_翻页按钮]，就等于建立了这五个按钮。

（6）开始使用宏

之前已经确定了 save.ks 里有两个需要添加按钮的地方，而 load.ks 里也同样。
现在这四个地方都修改成这样。

```
.....  
[eval exp="drawslbutton('back')"]  
;这里是我们要添加内容的地方  
[存取系统_翻页按钮]  
.....
```

没有错，只要一行！四个地方都只要一行！

亲爱的观众朋友们，你还在等什么呢，现在打电话订购宏，还赠送函数……（咦）

（7）继续修改宏的内容

定义了宏以后最大的好处就是这个了，只要修改宏的内容，四个地方的效果都会改变。

接下来当然是为按钮添加点下以后的操作了。

之前在制作选择的时候有说过，可以让按钮点下之后跳转到同一个地方，同时设定一个数值。

这里所有的翻页按钮，都是跳转到*刷新画面，而修改的数值就是“现在翻到第几页”。

在 THE NVL Maker 的存取系统里，“sf.最近存储页”代表的就是存取系统现在翻到的页数。

点下按钮 1，就设定 1，点下 2，就设定 2，以此类推。（当然你要保证你在工程设定里的确设定的存档总数够分成这么多页的……默认是 12 个，这里至少需要 15 个。）

因此我们现在可以把 macro_self.ks 里的[macro name=存取系统_翻页按钮]这个宏改成下面这样：

```
[macro name=存取系统_翻页按钮]  
[locate x=50 y=150]  
[button normal="num-1" target=*刷新画面 exp="sf.最近存储页=1"]  
[locate x=50 y=190]  
[button normal="num-2" target=*刷新画面 exp="sf.最近存储页=2"]  
[locate x=50 y=230]  
[button normal="num-3" target=*刷新画面 exp="sf.最近存储页=3"]  
[locate x=50 y=270]  
[button normal="num-4" target=*刷新画面 exp="sf.最近存储页=4"]  
[locate x=50 y=310]  
[button normal="num-5" target=*刷新画面 exp="sf.最近存储页=5"]  
[endmacro]
```

如果你有两个不同状态的数字图片，还可以用 cond 配合数字图片，用不同按钮来表示当前翻到哪一页。例如：

```
[button normal="num-1" over="num-1_over" target=*刷新画面 exp="sf.最近存储页=1"  
cond="sf.最近存储页!=1"]  
[button normal="num-1_over" target=*刷新画面 exp="sf.最近存储页=1" cond="sf.最近存储页  
==1"]
```

五、从翻页按钮改成页数显示

其实单纯的“向上翻页”和“向下翻页”也不错，但是有时候并不知道现在翻到的是哪一页也有点麻烦。

假如可以直接在画面上有个地方显示现在翻到的页数，不是比做很多个按钮更简单吗。

所以现在干脆去掉这些按钮，再修改一下宏的内容。

（1）在一个宏定义里使用其他宏

一旦宏被定义以后，就可以像普通的 KAG 指令一样使用，所以它同样也可以被用在其他宏里。所谓偷懒，就是这样简单……

这次可以利用到养成部分的宏里的一个，就是“显示图片数字”，在 macro_edu.ks 里。名字叫[drawnum]。

现在，macro_self 里的宏变成了这样：

```
[macro name=存取系统_翻页按钮]
[draw_num num=&"sf.最近存储页" pic="num-" x=50 y=150 layer=15 page=%page|back]
[endmacro]
```

先解释一下，layer15 是存储、读取界面用来描绘缩略图等等的专用图层，因此这里只要将图片数字描绘在层 15 上，就可以显示出来了。

吉里吉里中，层是很重要的概念。

图片需要显示在图片层上，而文字则需要显示在文字层（也可以叫做对话框层、消息层或者 message 层）上。

最后这些图层、文字层按照一定顺序叠加起来，就变成了完整的画面。

而 pic 则是设定了图片数字的前缀名，这里默认是 num-，代表图片 num-0~num-9，你可以设定自己的数字图片。

x 和 y 自然是显示在层上的坐标了。

至于 num，这里是用来显示数字的。但是我们需要它显示的是一个变数的值，而不是单纯的数字。所以使用了&"变数名"来表示“取得这个变数的值”。

&是 KAG 里独有的对变数取值的符号。在 TJS 式和 TJS 代码里有其他的写法，请不要照搬误用。

至于最后这个 page=%page|back 是什么意思先不管它，首先来测试一下。



上次打开的时候是翻到第五页，所以这次系统自动翻到了第五页，并且正常显示了页码。

现在可以按一下“preview”和“next”……

……页码数字不见了。

但是假如关闭这个界面再打开，数字会再次显示出来。

为什么只有第一次打开才有效呢？

（2）page 的“表”和“里”

之前在选择的制一章里，有讲过下面的内容。

文字和图片，设定在“表页”的时候，可以立即显示。

而设定在“里页”的时候，可以通过“执行切换”和“等待切换”来做出各种各样的效果。（比如刚刚的选择按钮，就是渐入显示的。）

“里页”可以看做是舞台的“后台”，当一切准备都做好以后，才通过“切换”拉开幕布，把效果显示到前台来。

而“表页”就是立竿见影的了。

每个图层、文字层都有“表页”和“里页”，可以单独设定。

为什么之前的按钮宏就没有必要指定 page 呢，原因就是在存取系统之前的代码里，已经用过[`current layer=message4 (page="fore")`]和[`current layer="message4" page="back"`]

来指定按钮是显示在 message4 的“表”或者“里”了。这是只有 message 层才有的福利。

图片层的话，就只好一层一层手动指定了……

存取界面第一次打开的时候，所有的内容也是渐入显示的，当翻页的时候，就变成瞬间显示了。第一次打开正常，翻页不正常，一定是瞬间显示的部分出了问题。

试着通过修改代码来解决这个问题吧。

（3）在宏里使用参数

宏就像普通指令一样，有名字，还有具体的参数内容。

这些参数也是在宏定义里设定的。

比如说：

```
[macro name=特殊文字]
[font size=%size color=0xFF0000]
[endmacro]
```

这样当使用[特殊文字]的时候，不但可以显示红色字体，还可以同时用 size 指定这个特殊文字的大小。比如说[特殊文字 size=20]

%用来指定宏的参数名，只会在宏定义里出现，请不要在其他地方误用。

之前提到的 page=%page 就是用来设定叫 page 的参数的，这个参数指定了页码数字应该显示在表页还是里页的。

后面的|back 代表不填写参数时候的默认值。

因为默认的是“里页（back）”，所以什么参数都不填写的时候，数字就被描绘在这里。如果想要瞬间显示，就需要指定为描绘在“表页（fore）”。

所以，save.ks 和 load.ks 里，各有一个地方需要做出小小的修正，就是加上红字部分的参数。

```
*刷新画面
[current layer="message4"]
[er]
[eval exp="drawslbutton()"]
[存取系统_翻页按钮 page=fore]
[s]
```

保存测试。

