zKok 4.0 教學課程
PDF 版

zkok 第一課

(此為舊版本的 help 的教學，可能與新版本(zkok4)有小許出入，但不失作為學習 zkok 的入門台階) ^.^

ZRobot 執行代碼的編寫並不一定複雜，如果你讓它做的事和以前的機器人一樣，那它同樣簡單。

下面是個簡單的例子，是供戰巫練技用的。

```
#%ZRobot.dll
%scene begin
  10::invoke on xxxxx
      # 執行 invoke,  再等 10 秒
  10::cast fireball on xxxxx
      # 執行 cast,  再等 10 秒
  10::recharge
      # 執行 recharge，再等 10 秒
      # 等完 10 秒，重複執行。
```

如果我們只是做些時間驅動的機器人執行代碼，可以按下面的格式來理解

```
#%ZRobot.dll
%scene begin
    # 這一行是必需的
  <執行時間>::<GKK 命令>
  <執行時間>::<GKK 命令>
  ....
  <執行時間>::<GKK 命令>
```

在執行完最後一個命令後，機器人將重複執行下去。

<執行時間>是一個時間描述，一般以秒為單位來描述，更具體的描述方法請大家參考 ZKok General Text File FORMat

在這裡特別需要注意，這裡的執行時間，是指後面 GKK 命令的所要花費的執行時間。 即先發命令，再等，而不是先等，再發命令。 這是唯一要注意的.

下面介紹兩種將其它機器人執行代碼移值過來的方法

KeyText 向 Zkok 移植的方法

在使用 KeyText 做 GKK 機器人時，文檔格式通常是

```
{loop}
{pause ???}<GKK 命令>
{pause ???}<GKK 命令>
{pause ???}<GKK 命令>
...
```

改成 ZRobot 機順人代碼的方法是:
去掉{loop}，用"%scene begin"替代
先忽略第一個等待時間 {pause ???}
每行的<GKK 命令>和下一行的{pause ???}合在一起，轉成 ZRobot 的行.
最後一個<GKK 命令>與第一個{pauase ???}合在一起，轉成 ZRobot 的最後一行.
如果某一行開始沒有{pause ???}，你可以理解成{pause 0}
時間的描述，兩者是一至的.

下面是一個例子:

```
{loop}
{pause 5.0}say hi
{pause 10.0}say 555
say 666          <--- 這一行沒有{pause ???}，把它看成是{pause 0}
{pause 20.0}say bye
```

轉成 ZRobot 變成

```
%scene begin
  10.0::say hi
  0::say 555
  20.0::say 666
  5.0::say bye
```

從 GkkMaster 向 Zkok 移植的方法

如果不想做修改的話，不必移植代碼，在 ZKok 中使用 ZMaster 即可。移植的目的是進一步加入控制。

網上，大家在介紹 GkkMaster 的內容時，通常用下面的格式
<時間> <GKK 命令> <次數>
<時間> <GKK 命令> <次數>
<時間> <GKK 命令> <次數>
...

如果次數都是 1 的話，轉換方法和上面介紹的 KeyText 差不多。如果不是 1，就複雜一些。

將 GkkMaster 每行轉到 ZRobot 的轉換規則如下:
先添寫 (<次數> - 1) 次 ZRobot 行，使用的<時間>是當前行的. (如果次數是一，跳過此處)
<時間>::<GKK 命令>
再添寫一行，<GKK 命令>是當前行的，<時間>是下一行的時間.(如果是最後一行，下一行是指第一行)
<下一行的時間>::<當前行的 GKK 命令>

例如:
3 say hi 5 <--- 這是當前行
5 say bye 1 <--- 這是下一行
轉成
3::say hi
3::say hi
3::say hi
3::say hi
以上是 5-1 = 4 次
5::say hi

體會轉換的差異，就會發現，ZRobot 是先發(命令)後等. 而其它的機器人是 先等後發。
到這裡，你應該能把其它的機器人代碼移過來了吧.如果不能? 那你就別編了，用別人做的。如果你對原來的機器人比較滿意了的話，也可以到此為止了。如果你不滿意原來的機器人，那就繼續閱讀吧

zkok 第二課

OK，你已經會編寫靠時間驅動的機器人了吧，是不是覺得只靠時間，效率不好？
舉一個例，最典型的．冥想，發出去命令，從失敗的立即結束，到恢復幾百點 MP 的長時冥想，怎麼設時間也不好。但如果利用冥想返饋的文本信息，效率就非常高了。

在講解之前，先分析一下，通常情況玩家對一個命令的結果可能的幾種反應
(1) 不能做，等等，能做了再發一次．(如上次施法沒結束)
(2) 失敗了，那再來一次．
(3) 失敗了，算了吧，法力也耗了，還是做下一步，恢復法力吧．
(4) 成功了．等動作結束吧，結束了，做下一步．
(5) 其它． 其它的內容就多了，通常也是機器人無法做的．這裡也就不考慮了．

為命令加上行為控制
下面以冥想為例講解．

步驟一．分析你發送的命令，會有多少種反應

比如當你發冥想的時候，可能有的情況:
(1) 你正忙著呢．<---- 上個動作還沒完呢．
(2) 你正在戰鬥中，無法冥想．
(3) 你為精神狀態不好，不能冥想
(4) 你開始冥想了，一堆垃圾信息傳了過來，最後，告訴你，你冥想結束了．

步驟二．確定對應每種情況，你想怎麼做．

(1) 你正忙著呢．上個動作還沒完呢．
當然是等幾秒，再冥想了．
(2) 你正在戰鬥中，無法冥想．
等戰鬥結束了再冥想．做法上可能有區別，這裡用的是
等待"你得到了(????)點經驗"這句消息出來後，再冥想．
(3) 你為精神狀態不好，不能冥想
可能有多種作法，比如， 不冥想了，做下一步吧。 這裡採用的是
等出現"你的精神恢復正常,可以冥想"的消息時，再開始冥想．
(4) 你開始冥想了，一堆垃圾信息傳了過來，最後，告訴你，你冥想結束了．
一收到"冥想結束"的消息，立即開始下一個動作吧．

步驟三．把上面的分析結果，編成 ZRobot 的一組代碼．

分析結果轉成的執行代碼，就是一個行為控制段。
首先要給這組內容起個名字．比如是為冥想寫的，可以叫 meditate_bc。將每個要匹配的文本，要做的控制動作等合成一行代碼，結果就出來了。把這些代碼加到原來的代碼上，位置在前在後無所謂

```
%bc meditate_bc
  retry:1:你正忙著呢。
            # 等一秒，再重發當前命令
  retry::你得到了( %* )點經驗。
            # 戰鬥結束了，立即重發當前命令
            # 這個匹配描述使用的模式匹配，在下一節有詳細說明
  retry::你的精神恢復正常,可以冥想了。
            # 可以冥想了，立即重發當前命令
  next::你深深地吸了一口氣，從深沈的冥思中醒來 ！
            # 結束冥想，立即進入下一步。
            # 這是二轉 ID 深冥的顯示。如果沒二轉，要注意結束的信息不一樣。
```

步驟四．為你原來的冥想命令添加行為控制

原來的機器人：

```
%scene begin
  .....
  50::meditate
  ....
```

變成

```
  ...
  100:meditate_bc:meditate
  ...
```

原來的%scene begin 段，再加上新的一些%bc ???段，就構成了新的執行代碼。

注意事項
分析過程一定要完整，考慮到任何能出現的情況。例如，一個法術失敗率較高的時候，總重做的話，就會出現「你的法力不夠!!」的信息。這些是不是考慮全了。
　我在做代碼時出現一種情況，就屬於考慮不全。使用戰巫雙重施法，已經確保雙施成功了時，我就開始等兩次施法的信息，因為施的法術是攻擊法術，有時就會出現這種情況，第一次施法把 NPC 給打死了，儘管雙施成功，可沒人可打，於是出現了"這裡沒有這個人"的信息。開始沒想到，時不時的就會停頓好一會兒。命令的延遲時間不能太短，當使用行為控制時，我們可以把這個時間理解為，如果在多長時間內沒有得到有效信息(能匹配的)，就產生 TIME OUT，無條件轉到下一個命令。如果太短，而網又慢的話，非常容易出現 TIME OUT。

在上面的介紹中，出現了兩個行為控制命令
  retry　重做當前命令
  next　開始做下一個命令

zkok 第三課

在最開始添加行為控制時，用戶可能就很容易碰到一些不容易注意到的問題而導致行為控制不起做用。

1．匹配描述與實際 GKK 信息不一至

例如：你寫了下面一行

retry:1:你正忙著呢。

以你的想法，是希望因為上一次動作沒結束，就等一秒再做，想法是正確的，可上面的描述根本不起作用。因為 GKK 的顯示信息不是"你正忙著呢。"，而是"你正忙著呢。"。其它常發生的類似情況就是後面的標點符號，全角和半角是有差異的。

對於上面的情況，只能靠大家的認真比較了。不過這裡介紹一個方法，可以很大程度的減少這種情況。

* 使用 ZKok．在角色子窗口的顯示區，要描述的內容上雙擊鼠標。該行顯示內容就會在下面的輸入行中顯示出來。

* 在輸入行，選中全部內容。按"CTRL + C"

* 在你的代碼編輯器(文本編輯器)中，按"CTRL + V"。

直接把 GKK 的顯示信息 COPY 過來，當然出錯的可能性就小多了。

2．過長信息的匹配描述

當一行的文本內容超過 62(63? 64?)時，GKK 就會把它分成多行顯示。並從第二行起，在每行的頭部加上">>"以示區別。在 ZKok 中，會將這些行再次合併成一行(合併時">>"被刪掉)傳送給 ZRobot。(只是傳遞給 ZRobot，實際顯示和 GKK 一至)。

對於這樣的匹配描述，用戶要注意，應該是完整的一行，而且要去掉">>"。

注意，這裡所說的合併，是指信息過長而導致的強制拆分。GKK 在一次輸出一段多行內容時，也會從第二行起添加">>"，但這個">>"只是提示大家，上面那幾行內容是一體的，並沒有強制拆行。這時候 ZKok 就不會對其進行合併。因此在輸入這樣的匹配描述時，行頭的">>"是不能少的。

模式匹配，只讓一行中的關鍵字起作用

你的角色和一個 NPC 打，你希望在打死 NPC 後，開始下一個動作，能利用的信息就只有"你得到了(???)點經驗。"了，可得到的經驗值是變的。

輸入一整行好煩，其實只要頭幾個字符相同就行了。

NPC 的名字各不一樣，但又夾在顯示信息中.....

如果你遇到上面幾種情況，就可以利用"%*"替換那些變化的內容。

例如：

next::你得到了(%*)點經驗。

next::你喃喃念到%*

next::你手裡的火球向%*的臉上飛去....

zkok 第四課

這裡介紹一下我製作 ZRobot 的基本思路，希望對大家製作 ZRobot 執行代碼有所幫助。

原來的 GKK 機器人，能做的事就是 發一個命令，等一段時間，再發一個命令，再等...，一遍做完了，再重新開始。這時候，機器人的設計有兩個內容:
(1) 你要懂得某職業練功/打經驗的方法.
(2) 確定各動作的執行時間，並且要去掉方法中的智能判斷部份.
例如，某法術容易失效，那就連續多發幾次，保證成功. 這就是一種去掉判斷的實例
在最初設計 ZRobot 時，發命令->等->發命令->等-> ... -> 到頭就重新開始. 這種思維形式仍被保留下來了，當然也必需保留!!!(原因不言自明)。這部份內容，在 ZRobot 中，被稱做"正文段"。ZRobot 仍以正文段為主，圍著正文段設計代碼. 或許是因為仍以這種方式為主的原因吧，有些用過 ZMUD 的用戶覺得 ZRobot 的代碼難寫，其實並不是如此。

用只靠時間驅動機器人的玩家應該很清楚，使用/編代碼中會遇到很多問題而難以解決。
(1) 因為服務器變慢，網速變慢，導致上個動作沒結束就進入下一行? 你把時間調長，可在網快的時候又不划算。
(2) 有些法術容易失敗。
(3) 有些命令本來執行時間就是變化的. 如 meditate(冥想)，不得不設成延遲最大時間。效率較低。
(4) HP 不夠了怎麼辦，MP 不夠了怎麼辦，如果是真人玩，很好辦的一件事，這裡不得不仔細計算。

ZRobot 就是要解決上面問題的。設想真人在遇到上面的情況時，通常會做的事有:
(1) 重發一次命令(即重做)
(2) 命令結束，不用再等了，開始下一步動作吧.
(3) 冥想，或買藥吃藥，HP/MP 夠用了之後，再回頭接著做。
當然還有別的動作，但通常就是這些吧。因為目前的機器人主要是用來練技能，打經驗的，選擇上面三種情況之一來做，對大部份的練功機器人來講是足夠了。
行為控制段就是根據上面的情況設想出來的. 在行為控制部份，並不直接描述該向 GKK 發送什麼樣的命令，而是利用 GKK 返回的命令執行信息，試圖改變正文段的延遲時間，執行順序等來達到較好的運行效率。利用的方法就是將 GKK 顯示信息與在行為控制段中各行的匹配描述相比較(匹配)。

ZRobot 的設計思想在最開始就這麼定了下來，而且一直遵守著這個思路。
利用正文段描述發送命令序列及每個命令的充分運行時間。
利用行為控制段來根據 GKK 信息調整，改變正文段命令序列的執行時間和執行順序。
隨著需求的增多，行為控制段和正文段的結構發生一些變化，功能也增多，但設計 ZRobot 的中心思想和基本規則還是沒變的。

最開始的設計中，一個正文段只使用一個行為控制段。很快的，問題就來了，有些消息會產生不同的動作結果(這也是行為控制依賴於正文段的另一個原因)。有時，這些信息導致 ZRobot 要做的事是一樣的，但有時要做的事就不一樣了。
例如，法牧系常發生的，"你的精神恢復正常，可以冥想了。"。 如果你正處於執行冥想動作期間，因為剛開始執行時，精神狀態不好不能冥想，正等著這行信息，好立即重發冥想命令呢。但如果已經冥想結束，正在另一個施法過程中，遇到了這行信息，就不能發送冥想命令了，必需忽略才行！
於是，在 ZRobot 中，正文段的每個命令都可以有它自己的行為控制段了。針對每一個命令編寫特定的行為控制段，表面看是增加複雜度，其實是變得相對簡單了。用戶在編寫代碼的過程中，只要專注於該命令的各種顯示信息就行了，哪個信息是失敗的信息，不同原因的失敗該如何處理? 哪個信息是成功的信息. 哪個信息將會導致以後信息的分枝(即差異)。不只如此，當有時對信息的先後順序也需要控制的時候，一個 GKK 命令可能會用到多個行為控制段。

有些人這時會想，如果所有職業，所有法術，命令的行為控制段都有人分析出來，形成一個行為控制段庫，以後每做一個機器人，大概只要想執行序列了吧？！想好命序執行順序，然後把用到的各命令對應的行為控制庫含進來..... 大概就是如此，當然絕對含進來也不一定正確，有時，重做？還是做下條？會有差異，但把每個命令能產生的情況分析出來，絕對會讓編碼工作變得相對簡單許多。

理解了 ZRobot 的思路，製作 ZRobot 機器人代碼的簡單方法也就很清晰了。

1. 確定一個機器人的執行序列. (這裡編哪種機器人代碼都得面對的)
2. 為序列中每個步驟(即每個命令) 設計它的結束依據/重做依據.
3. 每個命令的行為控制依據構成一個(或幾個)行為控制段.
4. 把這個序列描述，再添加上所有對應這些命令的行為控制段.

簡單的 ZRobot 機器人代碼編寫方法到此已經就夠了。後面的內容更像是介紹某些編碼技巧。

zkok 第五課

在計算機語言中，變量的作用並不比控制命令弱。只有通過變量的使用，才能讓代碼更小巧，更靈活。

在 ZRobot 中，允許有 10 個變量，變量名為"0" - "9"。ZRobot 並沒有象計算機語言那樣，採用整型變量，字符串變量或浮點數變量這種分法，而是由用戶直接描述該變量所有可能出現的值！(當然，有順序關係)。這種方法是考慮到在 ZRobot 中，變量應用的目的而決定的。畢竟，ZRobot 代碼不是一種計算機語言。
有關變量的定義和描述，參考變量段的格式說明。

變量的使用是通過正文段的替換和行為控制段的信息匹配來實現功能的，有關使用方法請參考 替換 和匹配
變量的用法非常多，可以說，ZRobot 代碼編寫的技巧，全在變量的使用上。下面只是簡單介紹幾個例子。

1．限制匹配的內容
如果 GM 忽然問你是不是機器人？而你又沒在屏幕前，該怎麼辦，最簡單的趕緊斷線吧，如果能摸清他問的內容，還可以適當用機器人應付應付。當然，你不能用"%*"，否則只要有人問你，你就會立即斷線，這並不是你所希望的。

```
%variable 9
  dela
  kings
  xxkx
  ...
%bc begin_bc
  ....
  shutdown::%+(%x9)告訴你 : %*機器人%*
  ....
或者
  ...
  call:answer_gm:%b%+(%x9)告訴你 : %*機器人%*
  ...

  %scene answer_gm
    ::tell %v9 %0 大人，我正在忙，你不要亂猜啦。
    ::@return
```

2．依次對周圍的 NPC 進行動作

```
%variable 0
  womana
  womanb
  womanc
  ......
  womanx
%scene begin
  50:invoke_bc:invoke on %v0
  ::%i0
  50:recharge_bc:recharge
  50:recharge_bc:recharge
```

3. 隨機說一些話
```
  %variable 8
    :我怎麼會是機器人呢？
    :你不要瞎猜了。
    :就算是，也沒影響別人呀。
    :...

  %scene answer_gm
    ::tell %v9 %0 大人，%X8
    ::@return
```

變量段

每個變量段定一個變量，ZRobot 只允許有 10 個變量，且變量名是固定的，即從"0"，"1"，... "9"。

用戶在變量段中設定該變量的所有可能的值。每個變量值有兩個內容，即左值和右值。左值和右值的配合通常是用於正文段與行為控制段的配合。其它情況一般只需要一種值即可。另，左值通常是一個不含空格的值，而右值則隨意一些。

```
# 變量段開始
%variable <var_name>
  <left_value> <right_value>
  <left_value> <right_value>
  ...
  <left_value> <right_value>
# 變量段結束
```

<var_name>
只能是"0"到「9」，如果是其它的值，該段將會忽略掉。

<left_value>
定義一個變量值的左值

<right_value>
定義一個變量值的右值

運行中的變量
當開始運行時，所有變量指向第一個值。
用戶可以指定變量使用下一個值，或回到第一個值。或者隨機產生一個數值 X，將該變量值變成第 X 個。
如果在行為控制段中指定對該變量的模式匹配，那麼在成功後，該變量的值也指向匹配的值。
變量的取值是全局共用的。
對於將某變量的變量值變成下一個的操作，如果當前變量值是最後一個，下一個的含義就是指向第一個值。
變量的用法

變量的使用是通過替換和匹配來體現的。請參閱相關的內容。

替換

正文段的發送命令和重複表達式會用到替換功能。替換的關鍵字在命令文本串中都是以"%"開始。

%%
替換成"%"

%0-%9
根據上一次匹配結果替換其中的模式匹配內容。

%n
替換成角色的英文名

%N
替換成角色的中文名

%K
替換成角色的 NICK 名

%Y
替換成角色的所屬城幫

%C
替換成角色的職業

%h
替換成角色的當前生命值

%H
替換成角色的最大生命值

%m
替換成角色的當前法力值

%M
替換成角色的最大法力值

%g
替換成角色的當有身上擁有的金錢數

%pc/%PC
替換成角色所在的位置

%(x,y)
將指定的相對位置轉換成 GKK 中的位置描述

%e0-%e9
替換成環境變量值。有關環境變量，請參考機器人屬性設定

%p0-%p9
根據變量的當前左值，轉換成 GKK 的位置描述值

%P0-%P9
根據變量的當前右值，轉換成 GKK 的位置描述值

%v0-%v9
替換成變量的左值

%V0-%V9
替換成變量的右值

%z0-%z9
讓變量的值指定第一個。返回值是空值

%i0-%i9
讓變量的值指向下一個。返回值是空值

%I0-%I9
讓變量的值指向下一個。如果回零，返回值是"1"，否則返回值是"0"

%o0-%o9
替換成變量當前值的位置

%x0-%x9
隨機從變量中取一個左值替換

%X0-%X9
隨機從變量中取一個右值替換

匹配

在行為控制段中的匹配描述中，可以使用下面的匹配關鍵字來適應變化的顯示內容

%%
替換匹配，替換成"%"進行匹配。

%0-%9
替換匹配，根據上一次匹配結果替換其中的模式匹配內容進行匹配。

%n
替換匹配，用角色的英文名進行匹配。

%N
替換匹配，用角色的中文名進行匹配。

%K
替換匹配，用角色的 NICK 名進行匹配。

%Y
替換匹配，用角色的所屬城邦中文名進行匹配。

%C
替換匹配，用角色的職業進行匹配。

%e0-%e9
替換匹配，用環境變量值進行匹配。

%v0-%v9
替換匹配，用變量的當前左值進行匹配。

%V0-%V9
替換匹配，用變量的當前右值進行匹配。

%i0-%i9
空值匹配，當匹配成功後，該變量的值指向下一個。

%z0-%z9
空值匹配，當匹配成功後，該變量的值指向第一個。

%b
空值匹配，如果匹配成功，所有的模式匹配結果將被記錄下來，由%0-%9 來使用

%x0-%x9
模式匹配，當被匹配內容與變量中的某個左值相同時，表示匹配成功。如果整個信息匹配成功，變量值指向匹配的內容。

%X0-%X9
模式匹配，當被匹配內容與變量中的某個右值相同時，表示匹配成功。如果整個信息匹配成功，變量值指向匹配的內容。

%*
模式匹配，任意的字符串都被認可。

%+
模式匹配，任意的字符串都被認可，但至少要包含一個字符。

%$
模式匹配，任意 a-z，A-Z，0-9，空格和"_"的組合被認可，但至少要包含一個。

%#
模式匹配，任意 0-9 的組合被認可，但至少要包含一個。

zkok 第六課

每個正文段，可以對應一個行為控制段，在該正文段的運行期間，該行為控制都有效。另外，對於每個命令，在其執行期間可以附加一個行為控制段，用於該命令的專用信息處理。可有些情況，如果信息的先後順序不同而導致處理不同，就需要特殊處理了。

以雙重施法為例，同樣的施法信息是成雙出現的，如何判斷命令的真正結束？ZRobot 沒有通過變量計數的功能。所以方法就是在命令的執行過程中替換行為控制段。

具體代碼如下：

```
%scene begin
  .....
  100:cast1_bc:cast ??? on ???
  .....

%bc cast1_bc
  retry::你上一個動作沒有完成，不能施法.
  next::<施法失敗的各種信息>
  ...      <---- 上面這些是原的了，下面兩行才是新內容.
  change:cast2_bc:你想起動雙重施法，但失敗了。
  change:cast2_bc:<施展該法術的顯示信息>
%bc cast2_bc
  next::這裡沒有這個人。
      # 第一個施法結果導致 NPC 死忙，會出現這種結果
  next::<施展該法術的顯示信息>
      # 如果該信息顯示後距法術結束還有一段時間，可以在 next 後兩個冒號之間，加入延遲時間，
如:next:3:.......
```

有人會想，要是寫戰巫的多重施放，是不是設成幾，就得寫幾個%bc 段．當然，如果你非得用這種方法，確實要寫幾個%bc 段。但是.....你為什麼不用"你的身體恢正常了"這句信息來匹配呢?

zkok 第七課

在 ZRobot 的執行代碼中，可以存在多個正文段。不一定只"begin"一個。這裡介紹幾種利用多個正文段的用法。

1. 如果你的某個 ID 有兩個機器人代碼，一個是練技能的，一個是打經驗的.

* 你可以把這幾個代碼合在一起. 當然,在合的時候要注意各段名稱的衝突!!!!，至少把原來幾個%scene begin 段分別改成

```
%scene jingyan
%scene jineng
```

* 增加一個新的 begin 正文段和新的 begin 行為控制段.

```
%bc new_begin_bc
  goto:jinyan:你說道 : 1
  goto:jineng:你說道 : 2

%scene begin begin_bc
  20::say 你是想打經驗(say 1)還是想練技能(say 2)
```

這樣，在啟動機器人之後，你可以敲

```
say 1   <- 開始打經驗.
say 2   <- 開始練技能.
```

2. 只有在法力不夠時才冥想

* 做一個專門的冥想正文段

```
%scene meditate_seg
  100:heal_bc:cast double_heal on
      # 也可以用強力恢得
  100:meditate_bc:meditate
%bc meditate_bc
  return:retry:你的精神正好，不需要冥想
  ...
```

* 為當前運行的正文段加入一行行為控制

```
%scene begin begin_bc
...
%bc begin_bc
  ...
  call:meditate_seg:你的法力不夠!!
  ...
```

這樣，只有在施法法力不夠時，才開始冥想，一直冥想到法力全滿。

3. 分枝跳轉

```
    ...
    100:med2_bc:meditate
    ...
  %bc med2_bc
    goto:kill_npc:你正在戰鬥中, 無法冥想。
    next::你的精神狀態。。。。。。
    ......
```

4. 對周圍的每個 NPC 都做一系列同樣的操作
   通常這個功能需要與變量結合使用

```
    ...
    ::@call for_npc
    ::%i0
    ::@call for_npc
    ::%i0
    ....
  %scene for_npc
    50:stun_bc:cast stun on %v0
    50:soul_bc:cast soul_steal on %v0
    ::@return
```

在上面的例子中, %bc 段出現了幾個新命令:

```
  call
  return
  goto
```

%scene 段也出現了兩個特殊命令

```
  @call
  @return
```

zkok 第八課

通常情況，正文段不負責流程(序列)的變更，這個工作是由行為控制段來做的。儘管功能劃分很清晰，但有時，利用行為控制段來描述比較麻煩，這通常體現在與變量的結合使用上，用戶希望根據變量值的個數來做為循環次數的依據。此外，行為控制是以被動接收 GKK 顯示信息為主的，而變量的變化，角色 HP/MP 的變化這些信息是非 GKK 文本顯示的。

ZRobot 表達式主要用在控制正文段的循環次數

下面舉個例子，機器人每隔五秒說一次 hi，共說五次，然後再說一次 bye．等十秒後循環

```
%scene sayhi::(%I0 == 0)
  5::say hi
%scane begin
  ::%z0
  ::@call sayhi
  10::say bye
%variable 0
  a
  b
  c
  d
  e
```

下面的例子是，當角色的法力值大於 300 時，依次對周圍的施展深恢，否則就用一次心爆。

```
%variable 0
  mana
  manb
  ...
  manx
%scene begin
  ::@call drnpc
  50:mind_bc:cast mind_blast on %v0
%scene drnpc::(%m > 300)
  50:dr_bc:cast deep_recover on %v0

%bc dr_bc
  next::%i0<成功的消息>
  next::%i0<NPC 不存在的消息>
  next::%i0<NPC 已經有深恢的消息>
  next::<法術失敗的消息>
  retry::<正忙的消息>
  ...
```

注意：正文段被調用時，不判斷表達式。只有在正文段結束時判斷是否重複執行。
表達式的運算符沒有優先級的設定，如果有必要，請使用()來實現優先級功能。

下面列出了 ZRobot 能處理的運算符。 其中只有"="和"!="允許兩邊的值是字符串，其它情況必需是數值。

+ 整數加
- 整數減
* 整數乘
/ 整數除
%% 取模
& 與。 這裡不是按位與，要記住。
| 或。 這裡不是按位或，要記住。
^ 異或。 這裡不是按位異與，要記住。
! 取非，單目運算符
< 小於
<= 小於或等於
> 大於
>= 大於或等於
= 相等/相同
!= 不等/不同
( ) 提高內部優先級。

zkok 第九課

在設計代碼時，最頭疼的大概就是如何指定 NPC。如果你如果你不是城主. 或者，你經常要變換練功位置. 或者....

實際上，GKK 提供了通過位置來指定的方法(而且主要是使用這種方法)。很多編寫機器人的玩家也在使用這種方法，只是在編寫上太麻煩，讓人都大了，而且，編完了位置就不能變。
在 ZROBOT 中，通過位置指定的方未能就變得容易多了.

下面先介紹一下 GKK 的屏幕，其可放置物品，NPC 的點看下面的圖。

```
0  0  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9  10

@     @     @     *     *     *     *     *     @     @     @ <-- 0
   @     @     @     *     *     *     *     @     @     @    <-- 1
*     *     *     *     *     *     *     *     *     *     * <-- 2
   *     *     *     *     *     *     *     *     *     *    <-- 3
*     *     *     *     *     *     *     *     *     *     * <-- 4
   *     *     *     *     *     *     *     *     *     *    <-- 5
*     *     *     *     *     *     *     *     *     *     * <-- 6
   *     *     *     *     *     *     *     *     *     *    <-- 7
*     *     *     *     *     *     *     *     *     *     * <-- 8
   *     *     *     *     *     *     *     *     *     *    <-- 9
*     *     *     *     *     #     *     *     *     *     * <-- 10
   *     *     *     *     *     *     *     *     *     *    <-- 11
*     *     *     *     *     *     *     *     *     *     * <-- 12
   *     *     *     *     *     *     *     *     *     *    <-- 13
*     *     *     *     *     *     *     *     *     *     * <-- 14
   *     *     *     *     *     *     *     *     *     *    <-- 15
*     *     *     *     *     *     *     *     *     *     * <-- 16
   *     *     *     *     *     *     *     *     *     *    <-- 17
*     *     *     *     *     *     *     *     *     *     * <-- 18
   @     @     @     *     *     *     *     @     @     @    <-- 19
@     @     @     *     *     *     *     *     @     @     @ <-- 20
```

是玩家當前所在的位置
* 是能放置 NPC/物品的位置，其中較常用的就是四個角落(用@標識的)

在 ZROBOT 發送的命令中(%scene 中的命令)，可以用下面的格式描述位置
%(橫向坐標,縱向坐標)

例如，你描述的發送命令可以是

```
50:fire_bc:cast fireball on %(0,0)
50:fire_bc:cast fireball on %(0,1)
50:fire_bc:cast fireball on %(1,0)
```

如果你的位置序列是在變量中定義的，使用下面的方法

```
%variable 5
  (0,0)
  (0,1)
  (1,0)
  (1,1)
  (2,0)
  (2,1)
  (0,18)
  (0,17)
  (1,18)
  (1,17)
  ...
  (10,20)

%scene fightall::(%I5 == 0)
  20:soul_steal_bc:cast soul_steal on %p5
```

上面的例子是對 24 個 NPC 依次執行吸魂．全部吸魂後，返回．

其中%p5 表示用變量的左值進行位置變換．相應的 %P5 表示用右值進行位置變換．

這樣，做一個機器人代碼，到任何城都好用了.....

zkok 第十課

正文段

正文段是向 KOK 發送命令和設定延時的主體。其基本思想是採用「發命令」，等一段時間，再發命令，再等的循環機器。

正文段的格式

```
# 段開始
%scene <scene_name> <scene_bc> <expression>
  <time> <cmd_bc> <command>
  <time> <cmd_bc> <command>
  ...
  <time> <cmd_bc> <command>

# 段結束
```

<scene_name>
  定義了該段的名稱，不能為空單詞！！！
<scene_bc>
  每個正文段可以選擇一個行為控制段來控制該段的執行。
  該內容可以為空，如果為空，表示不需要行為控制。
<expression>
  如果這部份為空，表示該 scene 一直執行，直到有行為控制來改變。
  如果不為空，則運算該表達式，如果結果不為 0，則繼續運行該正文段，否則執行返回操作

<time>
  是一個時間描述，表示該行命令執行要花費的時間，內容可以為空，表示執行的命令不花費時間。
<cmd_bc>
  每個%scene 段定義了一個行為控制段，是該正文段共用的。另外的，當執行該命令時，可附加一個行為控制段。這個行為控制段的內容只在該命令執行期間有效。通常其內容只與該命令有關。
  這部份內容可為空
<command>
  具體發送給 KOK 的執行命令。這個內容先要經過替換後才傳送給 GKK。另外有一些特殊命令不傳送給 GKK，而直接由 ZRobot 來執行。這種命令都是以"@"打頭的。 另，如果命令是以「#」可頭的，則不發送命令。
  這部份內容可為空

注意，其內容不是等<time>秒後，再執行命令，而是執行命令，再等<timer>秒後，執行一個步驟。

特殊命令介紹

@call <scene_name>
  轉子，壓棧當前運行狀態，跳到指定的正文段。

@return/@pop [ignore|retry|next|<time>|keep]
  彈棧。根據參數確定返回方式
    空
      如果是由正文段的@call 產生的轉子，等效於@return next
      如果是由行為控制段的 call 產生的轉子，等效於@return retry
    ignore
      只彈棧不返回。繼續運行@return/@pop 後面的命令
    retry
      彈棧返回，重新執行當前命令
    next
      彈棧返回，開始執行下一條命令。
    <time>
      除了延時外，維持調用前的狀態。
    keep
      維持調用前的狀態。

@goto <scene_name>
  立即跳轉到指定的正文段

@skip <expression>
  表達式值為假，進到下一條
  表達式值不為 0，跳過下一條

@prompt <message>
  彈出提示對話框，顯示<message>

@log <message>
  將<message>記錄到 LOG 文件中

@echo <message>
  返回一個消息，該消息即不在 GKK，也不在 ZKOK 中顯示，但被行為控制段獲得並處理。

命令/表達式的替換

要發送的命令和要進行運算的表達式，會先進行替換操作後才真正發送或運算。原始內容中所有"%"及後面關鍵字都會被新內容替換。其目的是減少正文段的編寫內容。

替換功能和行為控制段的匹配以及變量段關係很大，大家可以在瞭解行為控制段和變量段之後一起瞭解有關替換和匹配的內容。

目前所能替換的內容，可參考替換關鍵字介紹。

表達式

表達式可以進行一些基本的整數運算和邏輯(布爾)運算，各運算符沒有優先機！！！要改變優先順序，請使用()。

ZRobot 支持的運算符有:

```
+, -, *, /
& | ^
< <= > >= = !=
( )
```

zkok 第十一課

行為控制段

行為控制段是區別於其它現有 KOK 機器人的重要內容，因為 ZKOK 能夠跟蹤 KOK 的顯示信息，這樣 ZRobot 就可以根據這些顯示變換執行的情況。

注意，行為控制段只是改變執行的狀態和順序，具體發送給ＫＯＫ那些命令，仍然要在正文段中定義。

行為控制段格式

```
# 行為控制段開始
%bc <bc_name>
  <ctrl_command> <ctrl_argument> <match_message>
  <ctrl_command> <ctrl_argument> <match_message>
  ...
  <ctrl_command> <ctrl_argument> <match_message>
# 行為控制段結束
```

<bc_name>
指定該行為控制段的名稱，不能為空

<ctrl_command>
行為控制命令，當顯示信息與<match_message>匹配成功時，將執行<ctrl_command>

<ctrl_argument>
有些行為控制命令需要一個參數，大部份情況可以為空。

<match_message>
匹配描述。

行為控制命令

retry
  重做當前運作
  參數是一個時間，表示過多長時間後再重做，可以為空。
  在延遲時間內，取消了當前命令的附加行為控制段。這時候只有%scene 共用的基本行為控制段有效。

next
  結束當前動作，開始執行下一步。
  參數是一個時間，表示過多長時間後再執行，可以為空。
  在延遲時間內，取消了當前命令的附加行為控制段。但下一個命令的附加行為控制段並不被應用。這時候只有%scene 共用的基本行為控制段有效。

delay
  重置當前命令的執行延遲時間。
  參數是新的延遲時間。
  不修改原有的附加行為控制段。

again
  重新運行當前的正文段，忽略該正文段的運行條件。
  參數是延遲時間。
  在延遲時間內取消當前命令的附加行為控制段。這時候只有%scene 共用的基本行為控制段有效。

end
  結束當前的正文段，根據重複表達式，決定是重新開始運行當前段，還是返回
  參數是延遲時間。
  在延遲時間內取消當前命令的附加行為控制段。這時候只有%scene 共用的基本行為控制段有效。

goto
  停止當前的%scene，轉而開始指定的%scene。
  參數是指定的新的%scene 段名，該參數不能為空。

restart
  復位所有狀態，重新開始機器人的執行。
  參數是一個%scene 段 ID，如果為空，則從缺省的"begin"段開始執行。否則直接跳轉到指定正文段。
  注意，錯誤的段名會導致機器人停止運行.

call
  將當前狀態壓棧，停止當前的%scene，轉而開始指定的正文段。
  參數是指定的新的正文段名，該參數不能為空，空會導致機器人停止運行。
  注意，壓棧只能嵌套 10 次，再壓的話，將停止機器人的執行。

return/pop
  彈棧，如果棧已經為空，將停止機器人的執行。
  參數有下面幾種:
    空: 如果此前的 call 是由正文段的@call 產生的，此時該命令相當於 return next；如果此前的 call 是由行為控制段的 call 控制命令產生的，此時該命令相當於 return retry。
    ignore: 只是彈棧，但不恢復原狀態。
    retry: 恢復壓棧前的狀態，重做壓棧前正在進行的動作。
    next: 恢復壓棧前的狀態，結束壓棧前正在正行的動作，進入下一個動作
    <time>: 恢得壓棧前的狀態，修改延遲時間
    keep: 恢得壓棧前的狀態。

change
  替換當前命令的附加行為控制段，直接命令結束/重做/或另一個 change 命令
  參數是一個%bc 段名，該參數不能為空

ignore
  過濾掉該行信息，不再由機器人對該信息繼續判斷。
  參數被忽略

pause
  暫停機器人的運行
  參數被忽略

stop
  停止機器人的運行
  參數被忽略

shutdown
  強制中止 GKK 的運行
  參數被忽略

prompt
  彈出對話框，顯示匹配的消息
  參數被忽略

log
  將匹配的消息存貯的 LOG 文件中
  參數忽略

信息的匹配

  對於一些要匹配的簡單信息，在行為控制段中直接寫出來就行了。但很多情況下，匹配的內容絕不是對兩個字符串簡單的相等判斷。而編寫機器人時，也只是希望匹配描述只寫出一行信息中的關鍵內容即可。例如，你希望在有人 TELL 你時，要做一些事。這時，"XXX 對你說 ： ***"就千變萬化了。用戶不可能列出所有的玩家中文名，這根本是不可能的。
  因此在 ZKok 中，把匹配分成了四種情況:
固有匹配，就是兩邊的內容直接相同啦，最原始的功能。
模式匹配，當很多種情況都符合條件時，需要採用模式匹配。例如：只要是一個數字就符合要求。
替換匹配，儘管要匹配的內容會不時發生變化，但在匹配時該內容是固定的唯一的，這種匹配就是替換匹配。
空值匹配，並不實際參與匹配內容的相同判斷，但如果整行信息匹配成功，就會執行相應的命令。如變量值加一等等。
  請參考匹配關鍵字介紹。

zkok 第十二課

移植 ZRobot 2.0 的機器人代碼

由於結構的變化和功能的增加，ZRobot 機器人代碼的格式也發生了一定的變化，部份 ZRobot 2.0 的代碼將無法使用。
如果你使用了 ZRobot 2.0 提供的初始化段的幾項功能，在移植到 3.0 時，必需進行必要的代碼修改。

變更內容

hp_low/hp_restore 被 trap 功能替代
mp_low/mp_restore 被 trap 功能替代
public_bc 被%public_bc 功能替代
表達式

修改方法

原文件的格式為

```
#%ZRobot.dll
  hp_low:???
  hp_restore:???
  mp_low:???
  mp_restore:???
  public_bc:<全局共用行為控制段名>
...
%bc <全局共用行為控制段名>
  ...
...
%scene begin:<begin 段用行為控制段名>:...
  ...
```

在改成 3.0 的格式時，可按下面的方法

```
#%ZRobot.dll
%public_bc
  ... (全司供用的行為控制段內容)
...
%scene begin
  trap hplow <hp_low_value>,<hp_restore_value>
  trap mplow <mp_low_value>,<mp_restore_value>
  goto raw_begin
%scene raw_begin:<begin 段用行為控制段名>:...
  ...
```

表達式的差異

因為功能的增加，2.0 的表達式也出現了一些差異．下面的表達式內容需要修改
'='
在 2.0 中，其內容和'=='是相同的，用於相等判斷，在 3.0 中，增加了賦值功能，這樣'='就只能用於賦值了．
'%pc','%px','%py'
因為3.0的變量增加到26個，這樣上面中的c,x,y現在只能表示變量了，所以要使用原功能，請換成 "%sc", "%sx"或"%sy"

zkok 第十三課

ZRobot 3.0 的新特性

1. 取消了初始化段，相關的內容請參見<<移植 ZRobot 2.0 的機器人代碼>>

2. 增加了全局行為控制段標識，除了段標識的差異外，內容和行為控制段完全一樣。

```
%public_bc
  <段內容>
```

3. 新的 ZRobot 已經基本變成了一種語言，每個正文段可以認為對應一個子過程。
在正文段中，每個有效行的內容分成三種類型：
GKK 命令行
控制命令行
標號行
注意：不能將多個命令/標號寫在同一行上！！！

GKK 命令行即原 2.0 格式的正文段文本行。格式為 "時間:行為控制段名:發送命令"。
控制命令用來控制程序的執行，包括判斷，循環，跳轉等等，考慮兼容性的問題，原來的在 GKK 命令行中"@"命令仍被支持，在代碼讀入時將轉換成控制命令，該行的時間和行為控制段內容將會被忽略。
以一個沒有空格和':'，不以數字和'#'開頭的字符串加一個':'，構成一個標號行，標號名本身不含最後的':'

4. 在原變量的基礎上，3.0 把變量增加到 26 個，變量名用'a'到'z'標識或'A'-'Z'標識。為了兼容原來的格式，原變量 0-9 表示仍然有效，不過'0'-'9'分別對應'a'-'j'，即%v0 和%va 描述的是同一個變量。
如果使用數字做變量名，通常是由前一個字符的大小寫來決定使用左值，還是右值，而如果使用'a'-'z'或'A'-'Z'，則由變量名的大小寫來決定使用左值，還是右值。

```
%v2 變量 c(2)的左值
%V2 變量 c(2)的右值
%vc 變量 c(2)的左值
%Vc 變量 c(2)的左值
%vC 變量 c(2)的右值
%VC 變量 c(2)的右值
```

5. 表達式的功能增強了。不過和 2.0 在格式上有小部份的差異。
數值常量
由 0-9 構成
字符串常量
以字符 ' 或 " 開始，並以同樣的字符結束，中間不能包含回車
數值變量
以字符'a-z'或'A-Z'打頭，只包含'a-z','A-Z','0-9'的字符串。(只有第一個字符用來標識變量，其餘的字符只起到註釋作用)
字符串變量
以字符'$'開頭，只包含'a-z','A-Z'的字符串。(只有緊跟'$'之後的第一個字符用來標識變量，其餘的字符只起到註釋作用)
臨時變量
'$0' - '$4'，每個棧結點允許有五個臨時變量，只能進行數值運算。
替換變量
見附錄
操作符
見附錄

I．控制命令一覽表

判斷
  if (表達式) 只能在正文段中使用
    ...
  elif (表達式) 只能在正文段中使用
    ...
  else 只能在正文段中使用
    ...
  endif

循環：
  do (表達式) # 表達式可忽略 只能在正文段中使用
    ...
  loop (表達式) # 表達式可忽略 只能在正文段中使用

  foreach 變量名 只能在正文段中使用
    ...
  loop (表達式) # 表達式可忽略 只能在正文段中使用

  break
  continue

again
  不判斷正文段的結束條件，直接重複。
end
  跳到正文段的末尾，根據正文段條件來決定執行。

ifgoto (表達式) 標號
  條件滿足跳轉到指定標號

goto 標號或正文段名
  跳轉到指定標號或指定正文段

jump <%scene_name>
  跳轉到指定正文段，不查找標號，速度要快過 goto

call <%scene_name>
  轉子到指定正文段。跳轉的正文段要注意有 return

return(pop) [retry|next|ignore|<time>]
  返回，要與 call 配對！！！

echo <info> 只能在正文段中使用
  讓 ZRobot 處理匹配一條不存在的消息

log <info>
  記錄消息
show <info>
  在 ZTerm 中顯示消息

prompt
  彈出提示窗口，顯示消息。

shutdown
  關閉 GKK
run
  繼續運行機器人
stop
  停止機器人
pause
  暫停機器人
priority 只能在正文段中使用
  恢復消息匹配規則，先使用 全局 bc 段，再使用正文段指定的 bc 段。最後匹配命令指定的 bc 段。
priority inverse 只能在正文段中使用
  改變匹配順序，先匹配命令指定的 bc 段，之後是正文段指定的 bc 段。最後是全局 bc 段。
trap
  trap hplow <hp_low_value>, <hp_restore_value>
  trap mplow <mp_low_value>, <mp_restore_value>
  trap position <xpos>, <ypos>
  trap time <time>
  如果在 hplow, mplow, position, time 後面沒有跟參數，表示關閉該項陷阱。

exp (表達式)
  運算表達式

retry <等待時間> 只能在行為控制段中使用
  重複當前命令
next <等待時間> 只能在行為控制段中使用
  停止等待，執行下一個命令
change <行為控制段名> 只能在行為控制段中使用
  重設/修改當前 GKK 命令的行為控制段
delay <等待時間>
  重設當前 GKK 命令的等待時間

II．表達式操作符一覽表

賦值

= 賦值左邊必需是一個變量。

下標指定

. '.'左邊必需是一個 variable 定義的變量，右邊是一個數值。

布爾運算

!
==
!=
<
<=
>
>=
&&
||
^^

字符串運算

+
<<
>>

數學運算

+
-: 可做單目運算符，數值取負
*
/
%: 算餘數，做單目運算符時，表示替換變量
<<
>>
&
|
^: 異或，可做單目運算符，數值取反

III．表達式替換變量一覽表

(?表示一個變量字符，可以是'0-9','a-z','A-Z')
%n　字符串值，角色 ID
%N　字符串值，角色中文名
%k　字符串值，角色職業
%K　字符串值，角色 NICK NAME
%y　字符串值，角色所屬組織
%Y　字符串值，角色所在城邦
%sx　數值，返回角色本身的 X 坐標
%sy　數值，返回角色本身的 Y 坐標
%sc　數值，返回角色本身的坐標
%m
%M
%h
%H
%g　數值，角色手中金錢數量
%E　數值，角色經驗值
%a0-%a9/%0-%9
%o?　數值，%variable 定義變量的當前指針。
%O?　數值，%variable 定義變量的值數。
%v?
%V?
%p?
%P?
%i?　數值，返回加一後的值
%I?　數值，適用於枚舉變量(%variable 定義的)，如已經指向最後一個變量，不加，返回值為 1，否則變量指針加一，返回值為 0
%d?　數值，值減一，返回減一後的值
%D?　數值，如值已經為 0 或指向第一個變量，不減，返回值為 1，否則變量減一或指針減一，返回值為 0
%z?/%Z? 數值，變量值或變量指針清 0

zkok 第十四課

ZKok 鼠標操作

目前 ZKok 可以支持鼠標操作了. 考慮到在代碼編寫上的方便性, 所以沒有使用絕對地址, 而是使用了 GKK 的地址描述方式和 ZKok 地址變換中的相對地址.

能使用的鼠標操作如下:

@&click <參數> : 點擊某個物品, 如櫃台, 魔力爐等等.
@&look <參數> : 查看某個人, 如點小販.
@&kill <參數> : 攻擊某個人.
@&go <參數> : 走到某個位置
@&run <參數> : 跑到某個位置

<參數>的描述方式有三種
(1) 相對位置: (x,y), 有關相對位置請參考 ZKok 的位置變換
(2) 絕對位置: [x,y], 用羅盤看到的位置, 主要適用於@&go/@&run
(3) NPC/物品名: 名稱, 可以是中文名,也可以是英文名.需要注意的是如果重名, 指定的位置不是可預測的.

特殊 GKK 命令, 由 ZKok 變換後發送給 GKK 服務器

@'1 - @'8 :
  相當於 F1 - F8
@"1 - @"8 :
  相當於 Shift F1 - Shift F8
@%0:
  關閉 GKK 的對話框
@%1 - @%9 :
  有對話框彈出時, 第一個按鈕對應 1, 第二個按鈕對應 2...
  例如用 finger/who 等彈出的對話框,
    1 對應"上一頁"
    2 對應"下一頁"
    3 對應"結束"
    0 對應"取消"
@$???:
  如果彈出的 GKK 對話框是一個選擇對話框,如點小販彈出的對話框.
    ???為一個數字, 表示選擇第???項. 0 表示不選擇.
  如果彈出的 GKK 對話框是一個需要輸入單行內容的對話框,如技能投資.
    ???為要輸入的內容

zkok 第十五課

為何我的 3.0 代碼在 4.0 中無法正常使用

如果你的代碼無法在 4.0 中正常運行，可能是以下幾個原因
1．是否是「么/麼」,「?/著」等在作怪，這是 GB/BIG5 內碼變換問題。
2．代碼中是否有相關對話框的判斷？如果有，判斷的文字在 3/4 中是有差別的。
3．有些描述不能匹配，試一試在匹配描述的頭/尾部加上\"%*\"

==========

@$指令在 ZKok4.0 中怎麼不能用了？

在 ZKok4.0 中，@$指令和@%指令被合併到一起了。@$後面的內容作為@%的可選參數
比如，先輸入 research 10，彈出對話框。就可以用下面的命令來輸入：
@%1 10000
而原來的指令(ZKok 3.0 中)則是
@$10000
@%1
列表的命令也是如此，後面跟的參數是選擇第幾個或選擇的內容。
比如，在點擊雜貨店後(假設羅盤在第二項)輸入：

@%1 羅盤
或
@%1 2
當然在編寫代碼時要注意，'@%'要寫成'@%%'

zkok 命令

多數的用法在 zkok 幫助中有，只有個別的沒有。

1 重啟後的連接

shutdown <數>
比如：
shutdown 300
就是五分鐘後重連。注意數不能為 0

"shutdown"(無參數)或"shutdown 0"
則表示中止 GKK 的運行，不再重連。

%public_bc
restart::@restart
shutdown:600:萬王之王的世界將在 1 分?後重新啟動。

利用 restart 還可以不由 begin 開始運行。
如
restart:second_begin:@restart

2 if 應用

當 hp 大於 400
if (%h > 400)
....(這裡是你要執行的指令)
....
endif
當 mp 大於 1000 就不執行下一條命令
if (%m < 1000)
...(這裡?你的 mp 少於 1000 時便執行的指令，多過 1000 便不執行)
endif

3 替換的應用

收人要求站在特定的位置:
%public_bc
call:banish:%b%+(%$)告訴你 : %*開我%*
call:banish:%b【換成你的城市名】%+(%$)%*開我%*
call:banish:%b【換成你的城市名】%+(%$)%*開除%*
call:accept:%*向你申請戶籍 !

%scene banish
::banish %1
return next
%scene accept
20:test_bc:accept ?????? 問號就是指定的位置不要說你不懂
return next

要解釋一下:
call:banish:%b%+(%$)告訴你 : %*開我%*
其中:%b 表示把匹配的內容用%0~%9 替換
%+為第一個匹配,替換為%0
%$為第二個匹配,替換為%1
前一個%*為第三個匹配,替換為%2
後一個%*為第四個匹配,替換為%3

4 關於小寫替換

用例子說明一下：

%scene begin
100:wait_bc:

%bc wait_bc
call:pot:%b%*(%*)告訴你：加潛能
call:pot:%b%*(%*)回答你：加潛能

%scene pot
50:pot_bc:Cast pote??? on %{%1%}
return retry

在%{和%}之間的任何%替換內容中的大寫字母都會被替換成小寫。
注意下面的區別
%{%VA%}: 替換成變量 a 的右值, 如果值中有大寫, 則替換成小寫
%{VA%}：VA 不是%替換值, 所以維持原狀。

5 trap 的應用

trap 可以加在任一正文段中，設置之後一直有效。
例如：
```
%scene begin_trap_hp
trap hp 200, 300
return
```
遇到下面的情況取消對 hp 的 trap
```
restart
```
或
```
trap hp
```
(後面沒有參數)

因為幫助文檔不詳，這個問題好多人問，我今天剛剛研究出來，貼給大家看，呵呵。
另外，自動恢復法力（吸魂等）的原理是一樣的，只要改成 mplow 即可。
```
%public_bc
call:drink:@hplow

%scene begin
trap hplow 350,400

%scene drink
0.1:drink_bc:drink potion
return

%bc drink_bc
stop::What?
```

6 run 的運算

```
@&run
```
ZKok 有更新，終於加上了字符串運算(早就要加，一直懶得寫)，這樣可以實現了。
注意表達式要用()括起來
```
50:run_bc:("@&run [" + (%sx - 3) + "," + (%sy - 3) + "]")
```

7 關於變量的使用
"a - z" 26 個變量，都可以使用，如果變量沒有被%variable 定義，直接賦值，否則用變量的值數量取模

如果變量沒有被%variable 定義
```
(a = 10) %oa: 10, %va: 10
($a = "abc") %oa:???, %va: abc
```

如果
```
%variable a
xxx
yyy
abc
(a = 10) %oa: 1, %va = yyy (10 %3 = 1)
($a = "abc") %oa:2, %va=abc
($a = "sadf") %oa:??? %va:???
```

8 ZRobot 3.0 的行為控制段

行為控制段也可以增加條件判斷

```
%bc stun_bc
retry::你上一個動作....
...
on (%m > 300)
retry::%i0 你...
on (%m <= 300)
next::%i0 你...

%scene begin
50:stun_bc:cast stun on %v0
50:storm_bc:cast mind_storm on %v0
```

9 ZRobot 3.0 的循環

1. do/loop

```
%scene ...
...
do 表達式
... # 循環內容
loop 表達式
...
```

說明: do / loop 後面跟的表達式結果不為 0 時, 繼續循環.
表達式可以省略, 表示條件永為真.

2. foreach/loop

```
%variable x
...
%scene ...
...
foreach x
... # 循環內容
loop
...
```

使用 foreach/loop, 就不用主動使用變量的加一操作了.
foreach 後面跟變量名

3. 在循環中可以用 break 或 continue 來控制
break 和 continue 也可以放在行為控制段中.

```
%bc xxx_bc
break::你的身體狀況不好....
```

10 進行坐標判斷

```
do ((%sx != 67) || (%sy != 108))
1::
loop
```

如果要跑到[50,100]，可採用的幾種方法：

1. 在[50,100]處設置地面信息
2. 用%sx，%sy，%sc 等來判斷

```
::@&run [50,100]
do
    1::
loop ((%sx != 50) || (%sy != 100))
```

3. 使用 trap

```
%scene ...
    trap position (50,100)
    50:run_bc:@&run [50,100]

%bc run_bc
    next::@position
```

11 改額點的命令：

```
setup dex -n
setup con n
```

n=你的額點數

dex=敏捷，con=體質，int=智慧，str=力量

12 自啟動功能

C:\StartKok.txt
那麼只要運行
....\ZTerm.exe C:\StartKok.txt
即可，(可以做成個快捷方式)
自啟動文件的格式
帳號名:帳號密碼:角色名:角色密碼:服務器
帳號名:帳號密碼:角色名:角色密碼:服務器
...
帳號名:帳號密碼:角色名:角色密碼:服務器

本次說的是變量
變量在 zkok 有許多種(幾種?我也不太清楚,你們看看 zkok 的幫助吧!)
這裏我只是講兩種(應該可以讓新手寫一些可以自己用的代碼!我也是菜鳥啊!)
1.用變量段的變量
2.數值變量
(以下一段抄自幫助)
每個變量段定一個變量, ZRobot 只允許有 26 個變量, 變量名可以是"a"-"z"("A"-"Z", "0"-"9")。

變量段格式
%variable 變量名
變量左值 變量右值 #變量段可以對同一個變量定義多個左值和右值的變量組
變量左值 變量右值 #
. . #變量段沒有結束標誌
. . #變量段一般只用左值,對於特殊的地方纔用右值
. . #
. . #
變量左值 變量右值 #

數值變量
數值變量沒有變量段
使用時要用括號括起來,單獨一行就可以.
比如 (a=1)
(a=a+1)
(a=a+2)

運行中的變量
當一個變量有變量段時,第一次使用變量,變量的值指向變量的第一行.
如: %variable a
hi HI
bb BB
cc CC
dd DD
ee EE
第一次使用變量 a 時的左值時 hi,右值是 HI,要使變量的值指向下一行,就要用到其他的一些控制變量段的命令了!

當一個變量為數值變量時,第一次使用變量,變量的值為 0.

變量段的操作指令(不一定說的全,但應該夠用了)
%ia 變量段 a 的內容向下移動一行,當為最後一行時,變量內容就會移動到第一行.
%da 變量段 a 的內容向上移動一行,當為第一行時,變量內容就會移動到最後一行.
%va 變量段 a 的目前內容.
%oa 變量段 a 的目前內容的行數.
%za 變量段 a 的內容返回到第一行.
看下面的代碼!

```
%variable a #初始化變量段 a
hi HI
bb BB
cc CC
dd DD
ee EE
%scene begin #gkk 顯示
::say %va #你說道:hi
::say %vA #你說道:HI
::say %oa #你說道:0
::%ia #變量段 a 內容下移一行
::say %va #你說道:bb
::say %vA #你說道:BB
::say %oa #你說道:1
::%ia #變量段 a 內容下移一行
::say %va #你說道:cc
::say %vA #你說道:CC
::say %oa #你說道:2
::%da #變量段 a 內容上移一行
::say %va #你說道:bb
::say %vA #你說道:BB
::say %oa #你說道:1
::%za #變量段 a 內容返回第一行
::say %va #你說道:hi
::say %vA #你說道:HI
::say %oa #你說道:0
stop
```

從上邊可以看到:變量 a 左值的引用是用小寫的變量名 a,變量 a 右值的引用是用大寫的變量名 A.

數值變量的使用
數值變量的命令
%ib 數值變量 b 加一,相當於 b=b+1
%db 數值變量 b 減一,相當於 b=b-1
%ob 數值變量 b 的值.
%zb 數值變量 b 值清 0,相當於 b=0
看代碼 #gkk 顯示

```
%scene begin #第一次用數值變量 b 時,數值變量 b 的值為 0
::say %ob #你說道:0
::(b=b+1) #數值變量 b 的值加一
::say %ob #你說道:1
::%ib #數值變量 b 的值加一
::say %ob #你說道:2
::%db #數值變量 b 的值減一
::say %ob #你說道:1
::%zb #數值變量 b 的值清 0
::say %ob #你說道:0
```

看完明以上的白了吧!(說錯的請指出來,謝謝觀看,指導)

前言：我們使用機器人或者外掛，目的無非是練級和打錢，zkok 是專為 zkk 設計的，所有也是最好的，這要感謝 jet2k。
第一課：zkok 的安裝運行，幫助文件已經很清楚，如果有不明白的請提出來，下面以牧系練功代碼為例，一步一步的解釋。
%variable s
這是變量段，每個變量段定一個變量，ZRobot 只允許有 26 個變量，變量名可以是"a"-"z"("A"-"Z","0"-"9")，這裡是定義變量 s。
(0,2)
(0,3)
(1,2)
(1,3)
(2,2)
(2,3)
(7,3)
(8,2)
(8,3)
(9,2)
(9,3)
(10,2)
(0,20)
(0,19)
(1,20)
(1,19)
(2,20)
(2,19)
(7,19)
(8,20)
(8,19)
(9,20)
(9,19)
(10,20)
以上是 npc 的相對坐標，zkk 的相對坐標是以你為中心，所能看到的屏幕分成 10x20 的方格，左上為（0,0）右下為（10,20）注意必須是小括號，如果你是大臣打開網格就明白了。
%variable a
[72,152]
[52,150]
以上是定義變量 a 為用羅盤測出的坐標，必須用中括號，這裡是兩個坑的中心坐標（就是站在骷髏頭上的坐標）
%scene begin
#每一個代碼必須有的，也是從這裡開始運行代碼的。
::%ia
變量 a 內的值加 1，例如，原來變量 a 是[72,152]，執行這個命令後成為[52,150]
20::@&run %va
@&run 是跑的指令，後面可以是坐標和相對坐標，%va 是變量 a 的內容，20 是執行@&run [52,150]命令 20 秒，就是向第 2 個坑跑 20 秒，跑不到？就改成 30 秒
goto kill
goto 跳轉到指定標號或指定正文段，這裡是轉到%scene kill
%scene kill

定義正文段 kill，zkok 可以存在多個正文段
1:kill_bc:@&kill (0,2)
1:kill_bc:@&kill (0,3)
1:kill_bc:@&kill (1,2)
1:kill_bc:@&kill (1,3)
1:kill_bc:@&kill (2,2)
1:kill_bc:@&kill (2,3)
1:kill_bc:@&kill (7,3)
1:kill_bc:@&kill (8,2)
1:kill_bc:@&kill (8,3)
1:kill_bc:@&kill (9,2)
1:kill_bc:@&kill (9,3)
1:kill_bc:@&kill (10,2)
1:kill_bc:@&kill (0,20)
1:kill_bc:@&kill (0,19)
1:kill_bc:@&kill (1,20)
1:kill_bc:@&kill (1,19)
1:kill_bc:@&kill (2,20)
1:kill_bc:@&kill (2,19)
1:kill_bc:@&kill (7,19)
1:kill_bc:@&kill (8,20)
1:kill_bc:@&kill (8,19)
1:kill_bc:@&kill (9,20)
1:kill_bc:@&kill (9,19)
1:kill_bc:@&kill (10,20)
@&kill (0,2)是鏢相對坐標（0,2）處的 npc，kill_bc 是根據行為控制段%bc kill_bc 的內容，來決定鏢完了作什麼
100:holy_word_bc:cast holy_word on
放聖言，然後停 100 秒後執行下一個命令，如果 holy_word_bc 跟對應的行為控制段%bc holy_word_bc 匹配的話，執行匹配的命令。

%bc kill_bc
定義行為控制段 kill_bc
goto:begin:這裡沒有任何生物。
跟前面的鏢 npc 的命令對應，如果鏢 npc，屏幕顯示：這裡沒有任何生物。那麼執行 goto begin

%bc holy_word_bc
定義行為控制段 holy_word_bc，跟放聖言對應
retry::你正忙著呢 ！
當放聖言的指令發出後，屏幕顯示：你正忙著呢 !重新執行放聖言的指令
retry::但是什麼事也沒發生。
當放聖言的指令發出後，屏幕顯示：但是什麼事也沒發生。重新執行放聖言的指令
retry::你的法力不夠 !!
retry::你上一個動作沒有完成，不能施法!
retry::你在胸口畫出一個神聖的印記，結束了你的聖禱。
goto:run:你現在不在戰鬥中。
當放聖言的指令發出後，屏幕顯示：你現在不在戰鬥中。跳轉到 run，這裡的作用是用聖言殺死 npc 後跳轉到淨化段。

%scene purify
#定義 purify
foreach s
foreach 和 loop 配合使用，執行它們中間的命令，遇到 loop ，foreach 後面跟著的變量加 1
100:purify_bc:purify %ps
淨化相對坐標處的屍體
loop

%bc purify_bc
next::你面無表情地將%*的屍體埋葬，並安撫他的亡靈，眼中流露出難得一見的慈悲。
淨化指令發出後，屏幕顯示：你面無表情地將%*的屍體埋葬，並安撫他的亡靈，眼中流露出難得一見的慈悲。執行下一個命令，就是淨化下一個。（%*可以代替任意字符）
next::一個朦朧的靈魂從你眼前飄過，向你微笑後，散入無盡的天空。
next::你現在還不能處理這具屍體。
goto:begin:這裡並沒有需要淨化的靈魂。
淨化玩了，重新開始。

想解釋的明白一點，由於本人表達能力有限，只能這樣了，沒有說清楚的請貼出來。