粗学 mush

写在前面

先用的 zmud，后学的 mush，粗略学了一点，还在入门阶段，仅仅停留在会使用阶段，对入门阶段的各种需求和苦闷了解较多，所以写点东西给想学习 mush 却觉得很难无法入门的人看。

zmud 毕竟是十几年年前的东西了，对比不停更新的 mush 要差一些也正常。zugsoft 最新力作 cmud 却不是很给力，有着这样那样的问题，我试用结果是放弃了 cmud，最后转学了 mush。

对比 zmud，zmud 能做到的 mush 基本都能做到，mush 能做到的，zmud 却未必。

mush 的优势 1.速度，2.自由，3.扩展性更强，4.实现复杂功能时候简单

zmud 的优势 1.易用，2.按钮，3.更多的人会用

教学帖里面很多内容都是我自己的看法以及实现办法，本人水平很烂，我写的这东西，高手们就不要看了。没学过的可以参考下。

本人在学习过程中屡次受到水鬼(shuigui)，错错(xkxpker)，本粥(labaz)，studyman 等人的指导，更是少不了多次琢磨 lzkd，littleknife，maper 等前辈的教程和例程，特此感谢下。

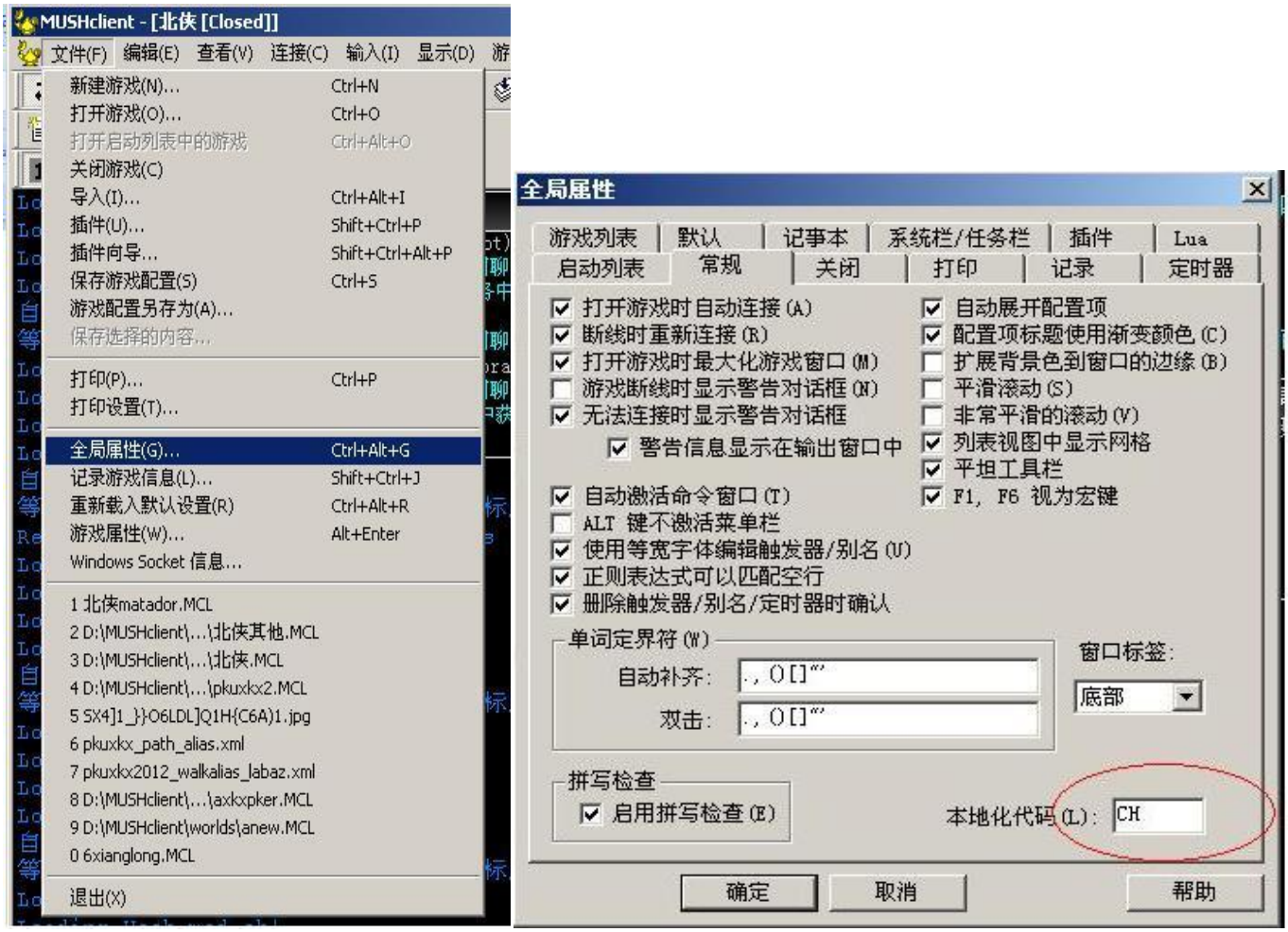
一.从最基本的开始

安装

没啥特别的，除了一点：千万安装到非中文目录，否则各种错误。另外版本最好用高级一点的，不要低于 4.73

基本设定

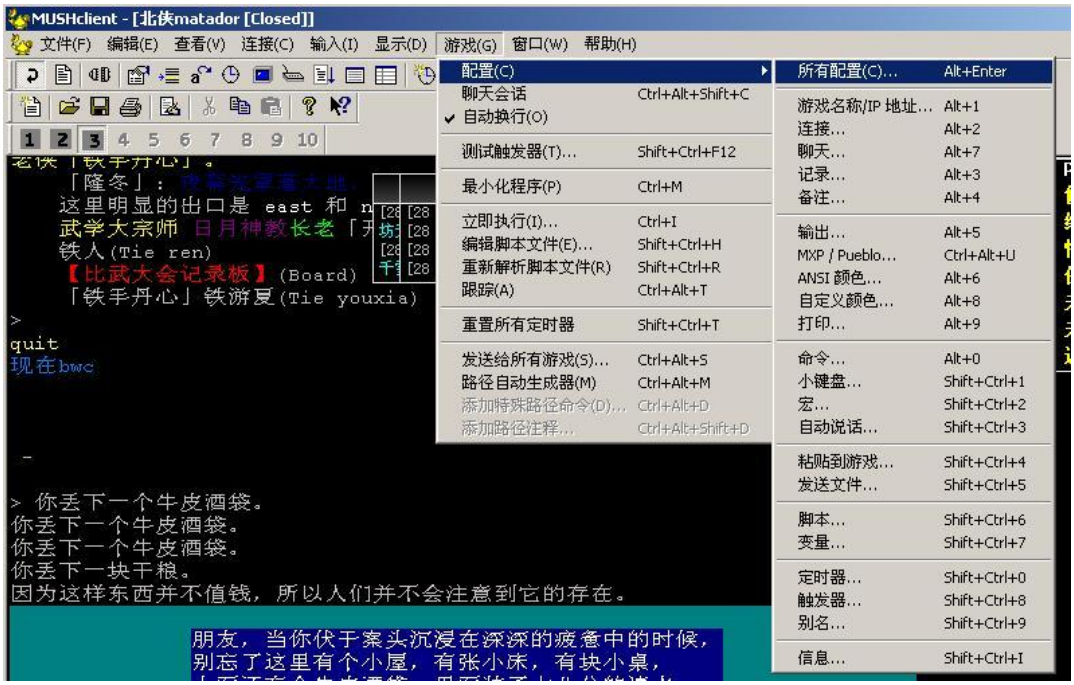
主要是 3 个地方：如果是 E 文版本的，也在同样位置，找到位置改了即可。这是第一个，按照红圈中修改即可：

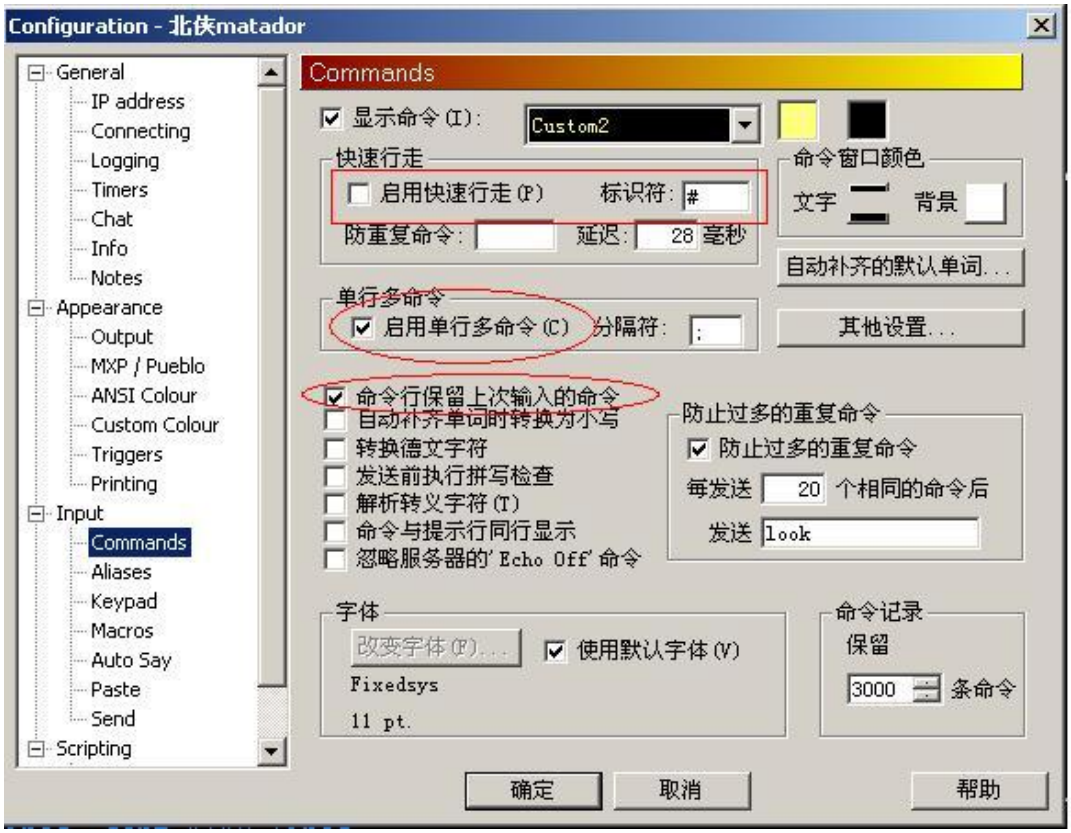

MUSHclient - [北侠 [Closed]]
文件(F) 编辑(E) 查看(V) 连接(C) 输入(I) 显示(D)
新建游戏(N)... Ctrl+N
打开游戏(O)... Ctrl+O
打开启动列表中的游戏 Ctrl+Alt+O
关闭游戏(C)
导入(I)... Ctrl+Alt+I
插件(U)... Shift+Ctrl+P
插件向导... Shift+Ctrl+Alt+P
保存游戏配置(S) Ctrl+S
游戏配置另存为(A)...
保存选择的内容...
打印(P)... Ctrl+P
打印设置(T)...
全局属性(G)... Ctrl+Alt+G
记录游戏信息(L)... Shift+Ctrl+J
重新载入默认设置(R) Ctrl+Alt+R
游戏属性(W)... Alt+Enter
Windows Socket 信息...
1 北侠matador.MCL
2 D:\MUSHclient\...\北侠其他.MCL
3 D:\MUSHclient\...\北侠.MCL
4 D:\MUSHclient\...\pkuxkx2.MCL
5 SX4]1_}}O6LDL]Q1H{C6A)1.jpg
6 pkuxkx_path_alias.xml
7 pkuxkx2012_walkalias_labaz.xml
8 D:\MUSHclient\...\axkxpker.MCL
9 D:\MUSHclient\worlds\anew.MCL
0 6xianglong.MCL
退出(X)
全局属性
游戏列表 默认 记事本 系统栏/任务栏 插件 Lua
启动列表 常规 关闭 打印 记录 定时器
打开游戏时自动连接 (A)
断线时重新连接 (R)
打开游戏时最大化游戏窗口 (M)
游戏断线时显示警告对话框 (N)
无法连接时显示警告对话框
警告信息显示在输出窗口中
自动激活命令窗口 (T)
ALT 键不激活菜单栏
使用等宽字体编辑触发器/别名 (U)
正则表达式可以匹配空行
删除触发器/别名/定时器时确认
自动展开配置项
配置项标题使用渐变颜色 (C)
扩展背景色到窗口的边缘 (B)
平滑滚动 (S)
非常平滑的滚动 (V)
列表视图中显示网格
平坦工具栏
F1, F6 视为宏键
单词定界符 (W)
自动补齐: .,()[]"'
双击: .,()[]"'
窗口标签:
底部
拼写检查
启用拼写检查 (E)
本地化代码 (L): CH
确定
取消
帮助

第二个地方


全局属性
启动列表 常规 关闭 打印 记录 定时器
游戏列表 默认 记事本 系统栏/任务栏 插件 Lua
Lua 初始化代码 (L) (每个 Lua 脚本会话开始时都会执行):
-- Put Lua initialization code (eg. sandbox) here.
-- Possible sandbox, and security tips:
http://www.gammon.com.au/security
编辑 (E)...
允许载入 DLL 文件 (D)
确定
取消
帮助

第三个地方：





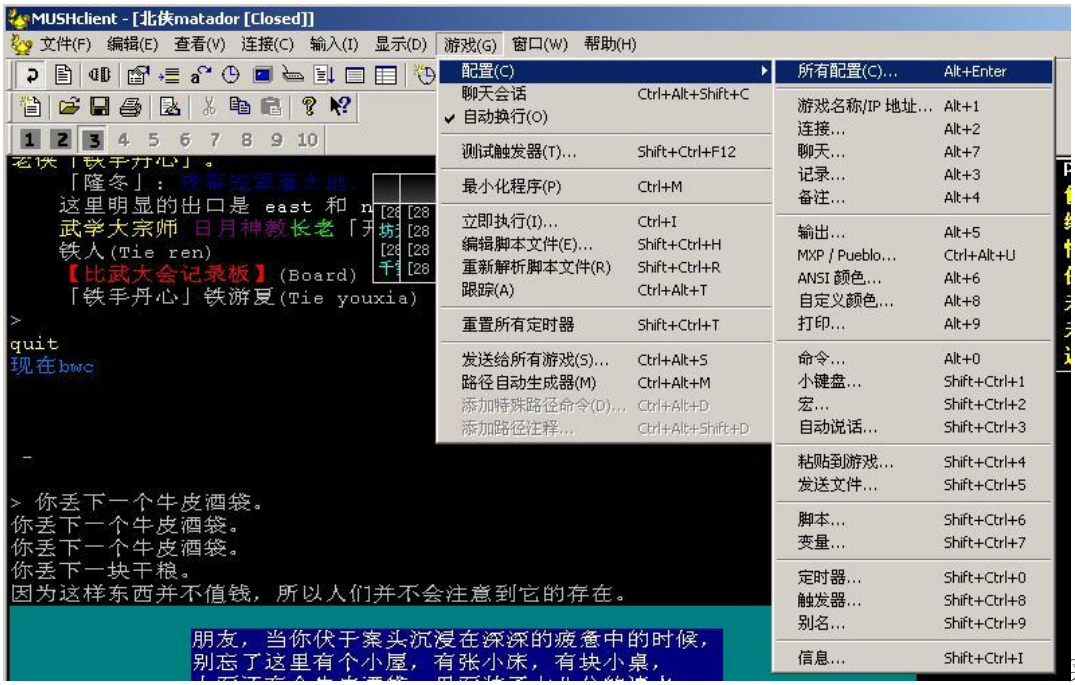
红色方框里面的启动快速行走，可以点上也可以不点，如果习惯了#5 n 这种命令格式可以点上，我#挪作了其他功能触发所以给取消了

登陆以及账号设定

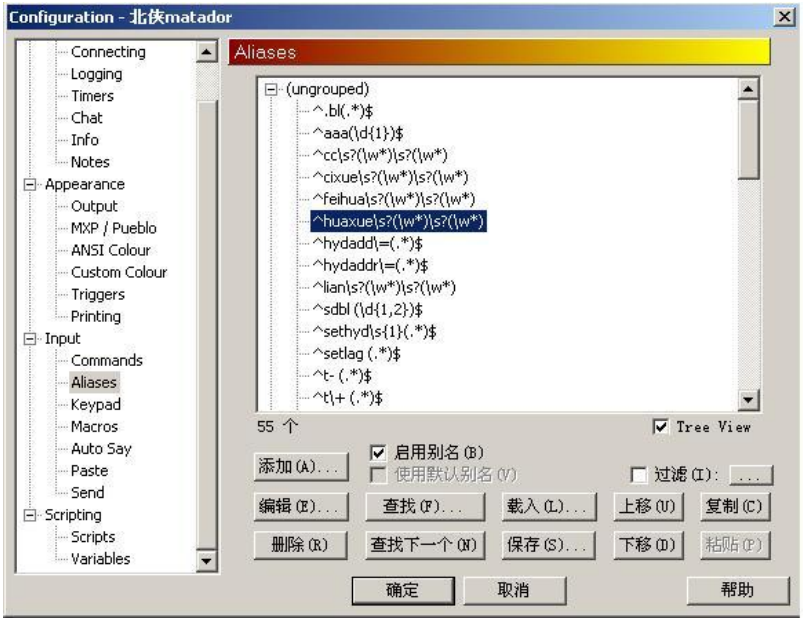

Configuration - 北侠matador
General
IP address
Connecting
Logging
Timers
Chat
Info
Notes
Appearance
Output
MXP / Pueblo
ANSI Colour
Custom Colour
Triggers
Printing
Input
Commands
Aliases
Keypad
Macros
Auto Say
Paste
Send
Scripting
IP address
游戏名称(N): 北侠
游戏地址和端口号
TCP/IP 地址(T): pkuxkx.net
端口号(P): 8080
清除 IP 缓存
代理服务器(O)
无
地址(A):
端口: 1080
代理服务器密码(W)...
关闭游戏时自动保存游戏配置(S)
正在寻找好的 MUD 游戏吗? 点击下面的链接可以查看建议玩耍的 MUD 游戏列表.
http://www.gammon.com.au/links/muds.htm
发现了错误? 有好的建议? 请反馈到这里:
http://www.gammon.com.au/forum/?bbtopic_id=5
确定
取消
帮助


Configuration - 北侠matador
General
IP address
Connecting
Logging
Timers
Chat
Info
Notes
Appearance
Output
MXP / Pueblo
ANSI Colour
Custom Colour
Triggers
Printing
Input
Commands
Aliases
Keypad
Macros
Auto Say
Paste
Send
Scripting
Connecting
脚色名字和密码
名字(N): xxxxx
密码(P): ******
自动连接(C): Diku-style (名字, 密码)
连接后自动发送的命令(T):
(0 行)
如果你想在上面的命令中使用角色的名字和密码, 你可以用 %name% 和 %password% 代替.
确定
取消
帮助

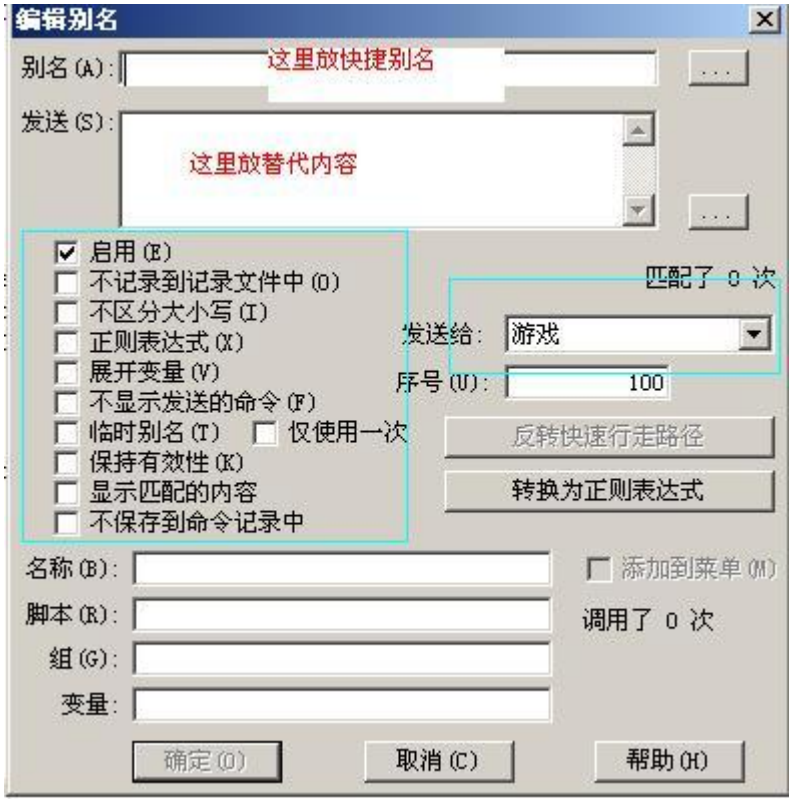
快捷键（alias）

快捷设置在 

菜单下的



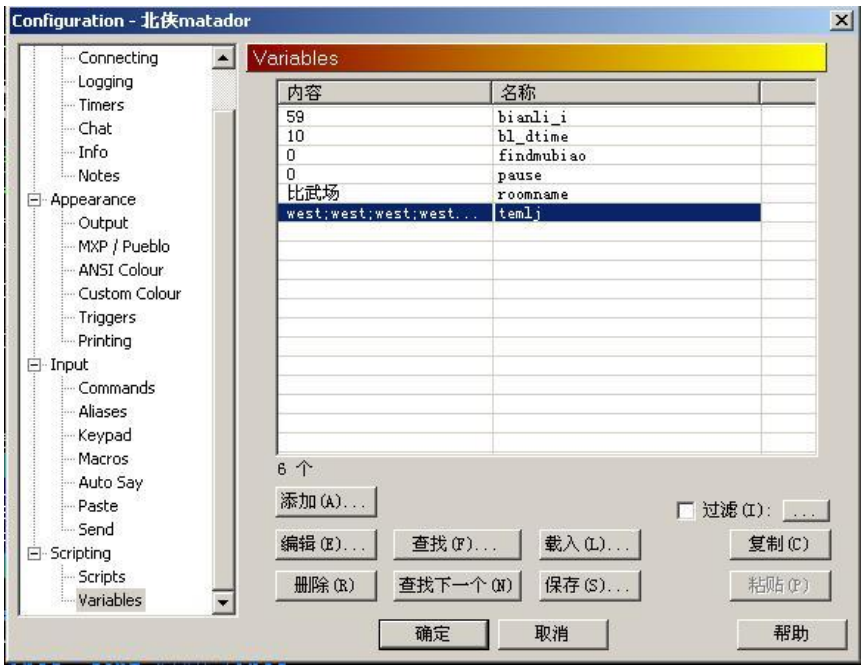
使用方法：添加（add）,界面如下



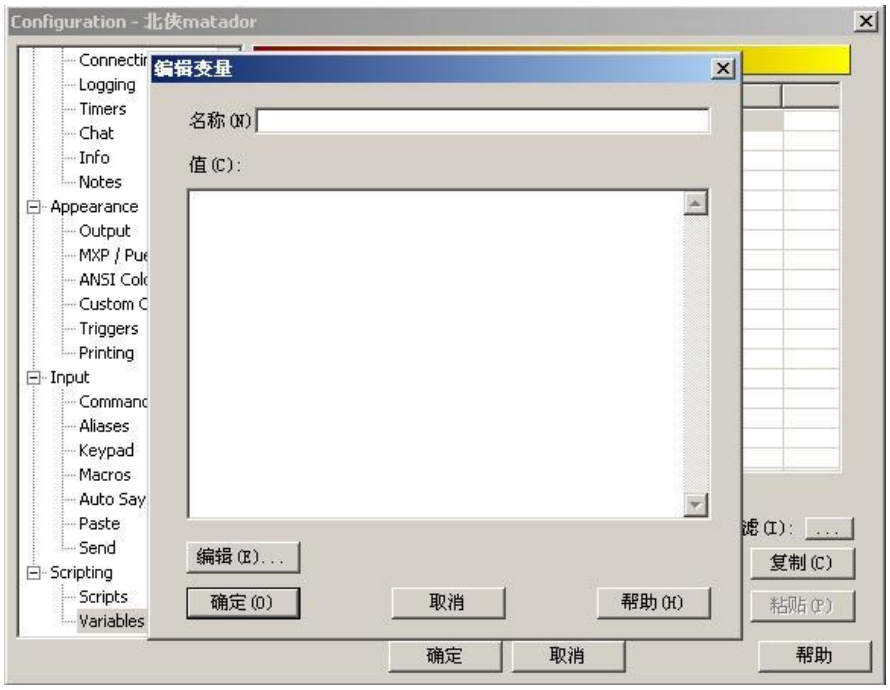
和 zmud 一样，真正不同是两个蓝框里面的东西，等我们下面再说。
你可以在名称栏里面敲入名称，这样可以在其他运用中来关闭打开修改这个 alias。

变量（variable）

在这里配置：



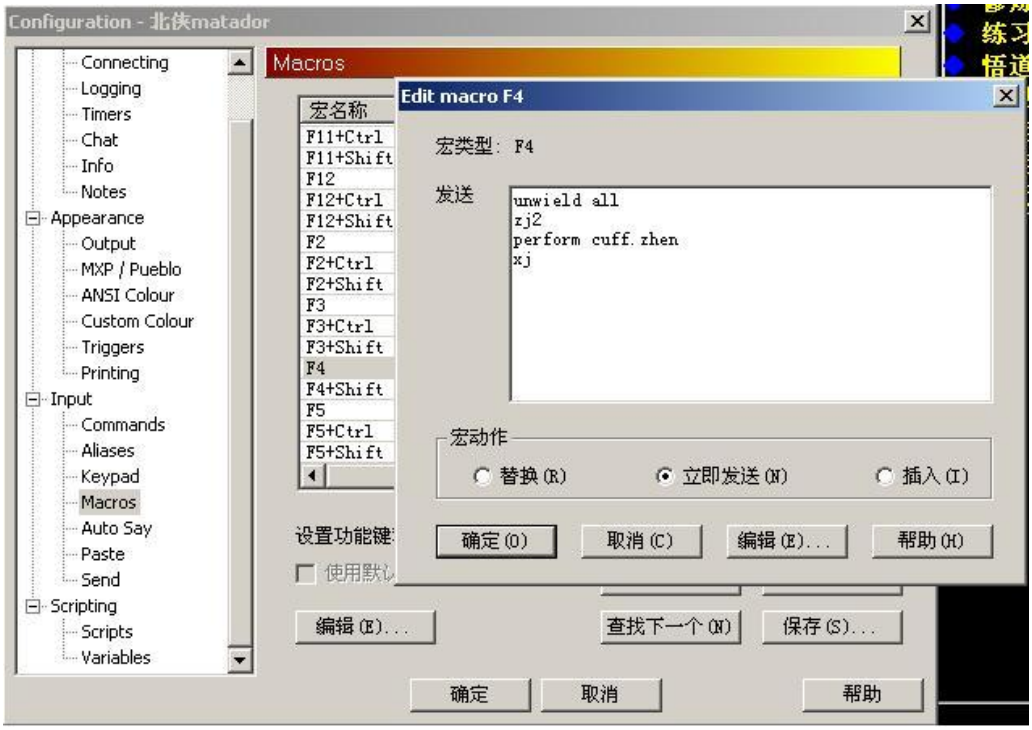
使用添加：



使用起来和 zmud 没啥 2 样，名称里面放变量名，值里面随便放啥，中英文均可，这个变量可以在 alias 和 trigger 里面引用。也可以被脚本在脚本程序里面引用，方法各有不同，我们后面再说。

宏键（macro）

在这里配置：



输出功能较为简单，可以直接输出，可以替换，可以插入，我们一般选择立即发送。不过宏键的直接输出是输出向命令解析器（后面我会详细介绍这个东西），所以可以使用 alias，用 alias 调用各种东西。

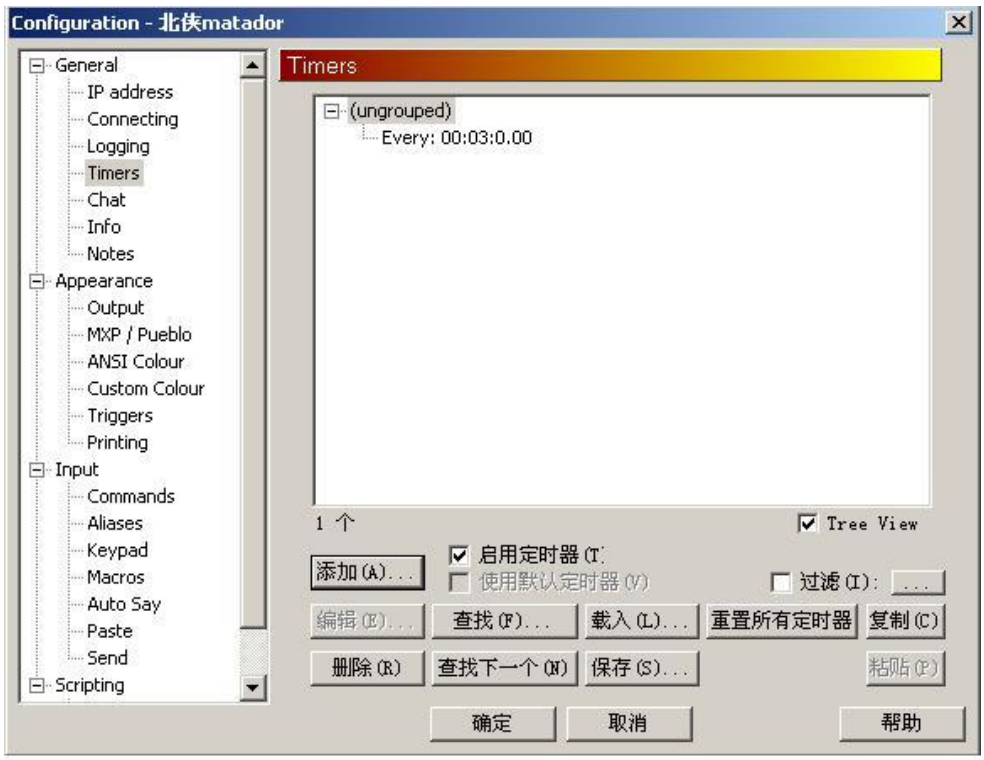
实例：我定义 F5 的动作为 pfmbusy,在 alias 定义中将 pfmbusy 定义指向脚本中函数 pfmbusy()，调用一个函数实现比较复杂的功能。

按钮（button）

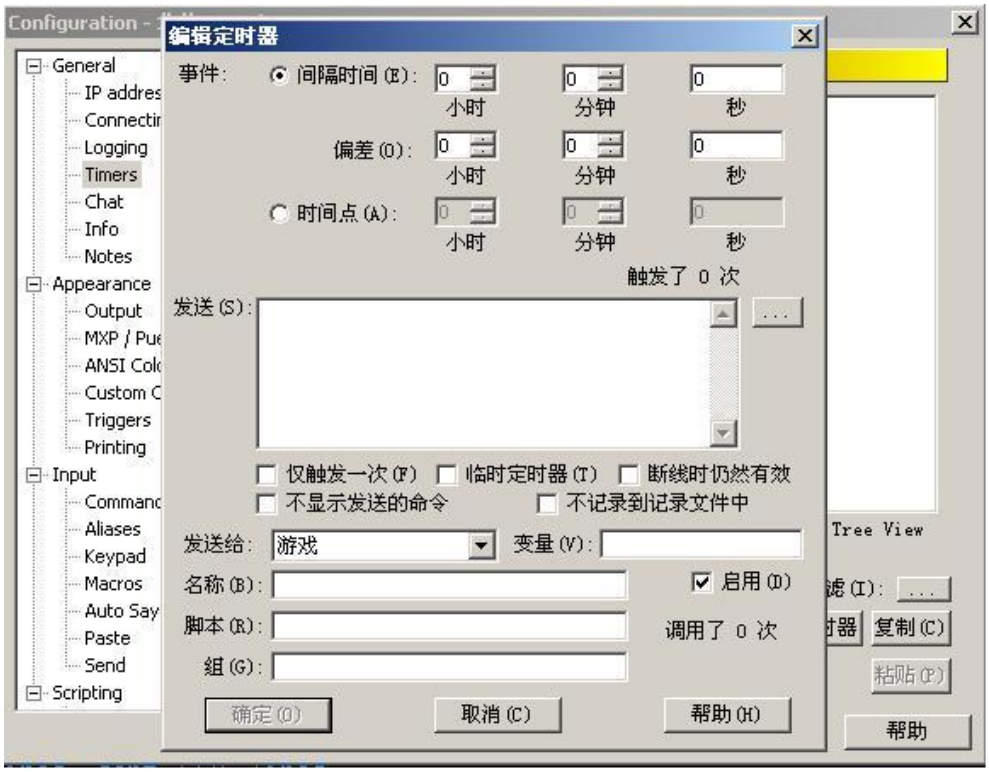
本身没有自带按钮功能，需要按钮功能需要按钮插件才行。如下图：

定时器（timer）

在这里配置：



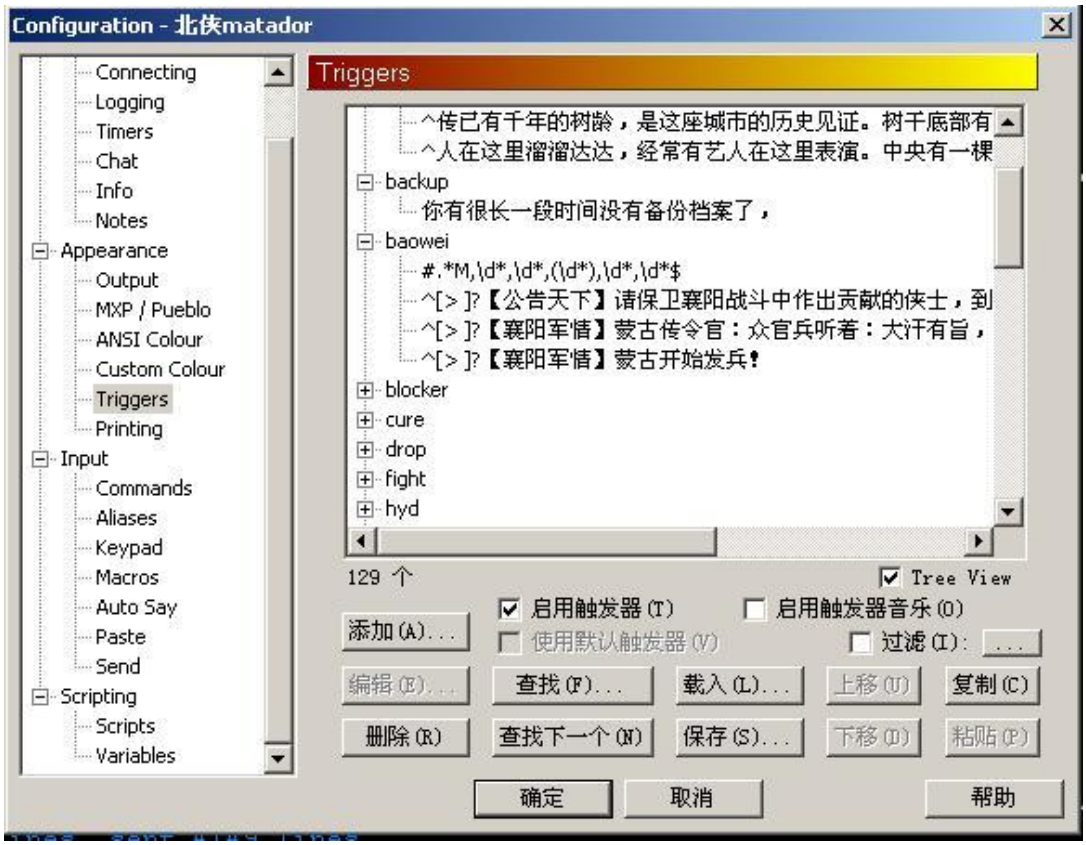
按 add 添加就可以设置使用了



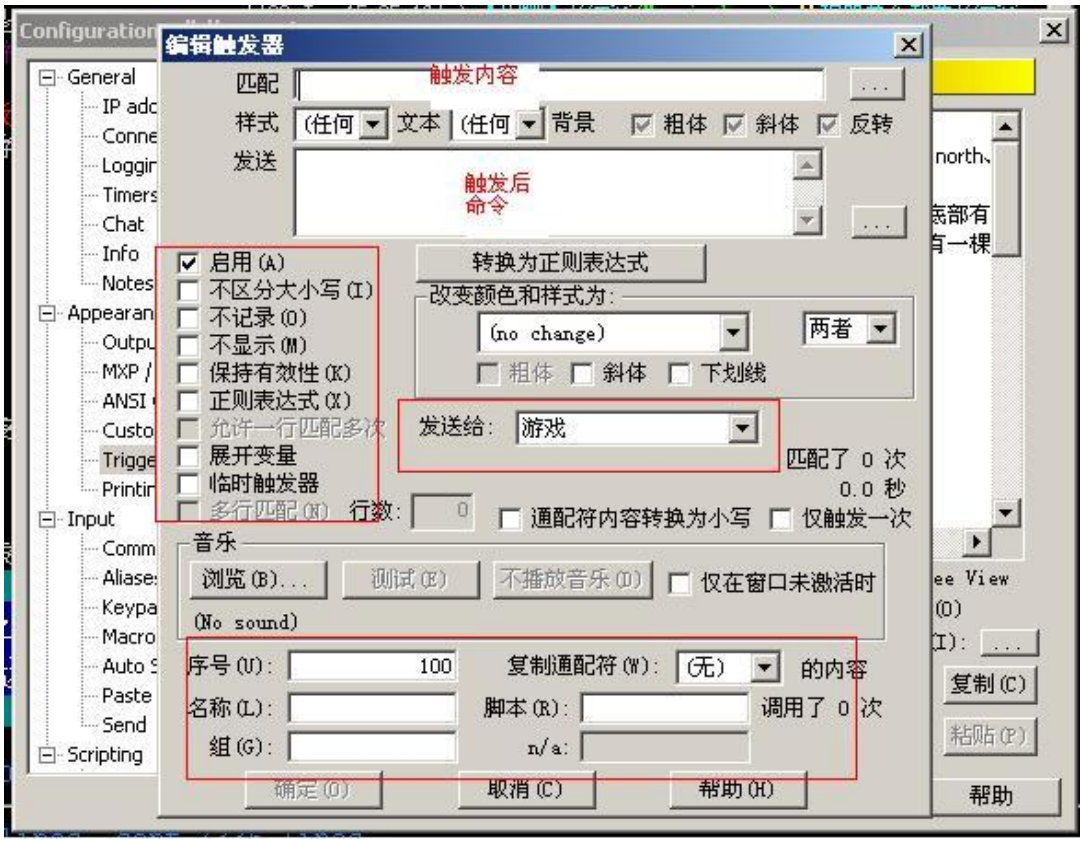
十分强大的 timer 系统，可以定义 N 个 timer，输出指向各种都有。
你可以在名称栏里面敲入名称，这样可以在其他运用中来关闭打开修改这个 timer。

触发（tigger）

在这个位置设置，界面如下：



Add 添加一个新的触发:



真正要紧的是红框内的内容。
你可以在名称栏里面敲入名称，这样可以在其他运用中来关闭打开修改这个 trigger。

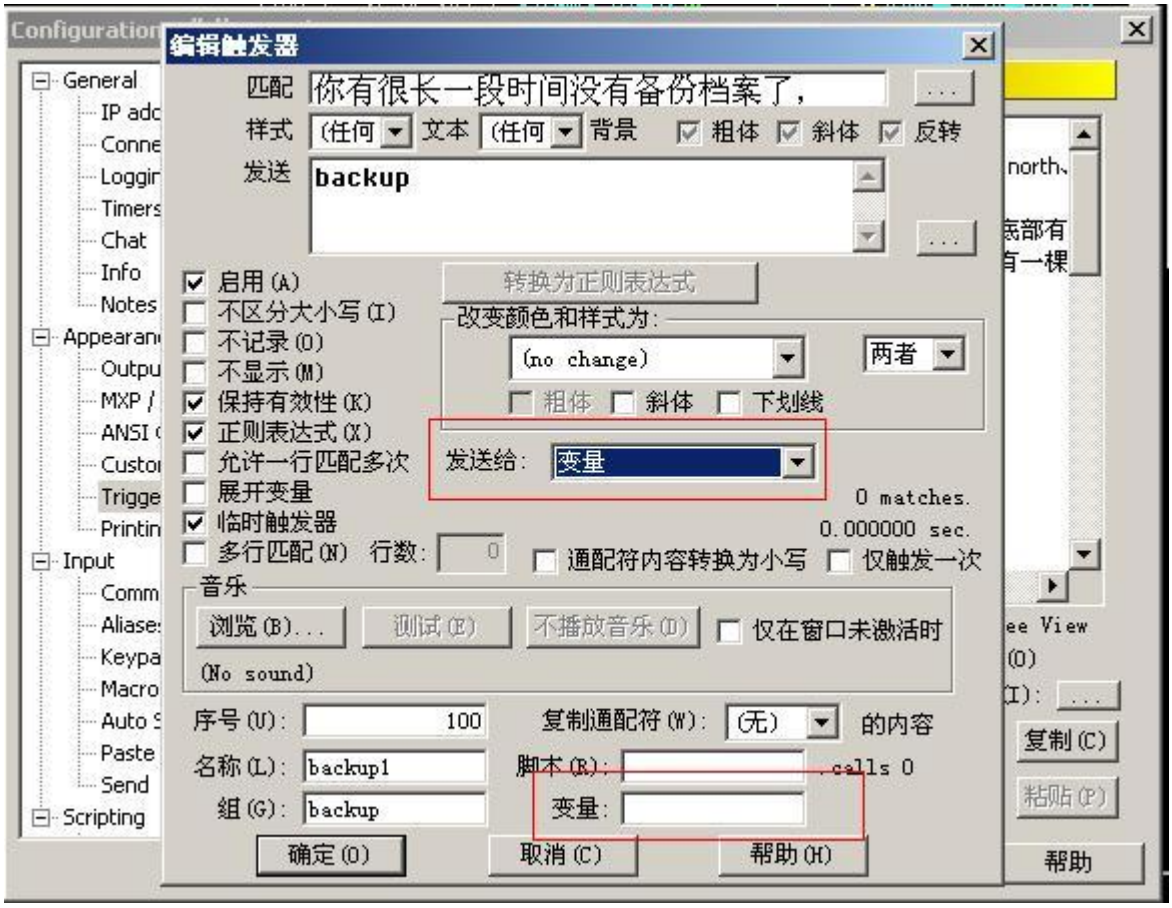
输出指向

看到这里的人，估计都会发现一个 mush 不同于 zmud 的地方 ，**“发送给”**，在 trigger，alias，timer 里面都有。如下图：

这个是什么意思呢：指的就是你上的发送内容如上图中的 backup 这个命令发送到哪个地方去。

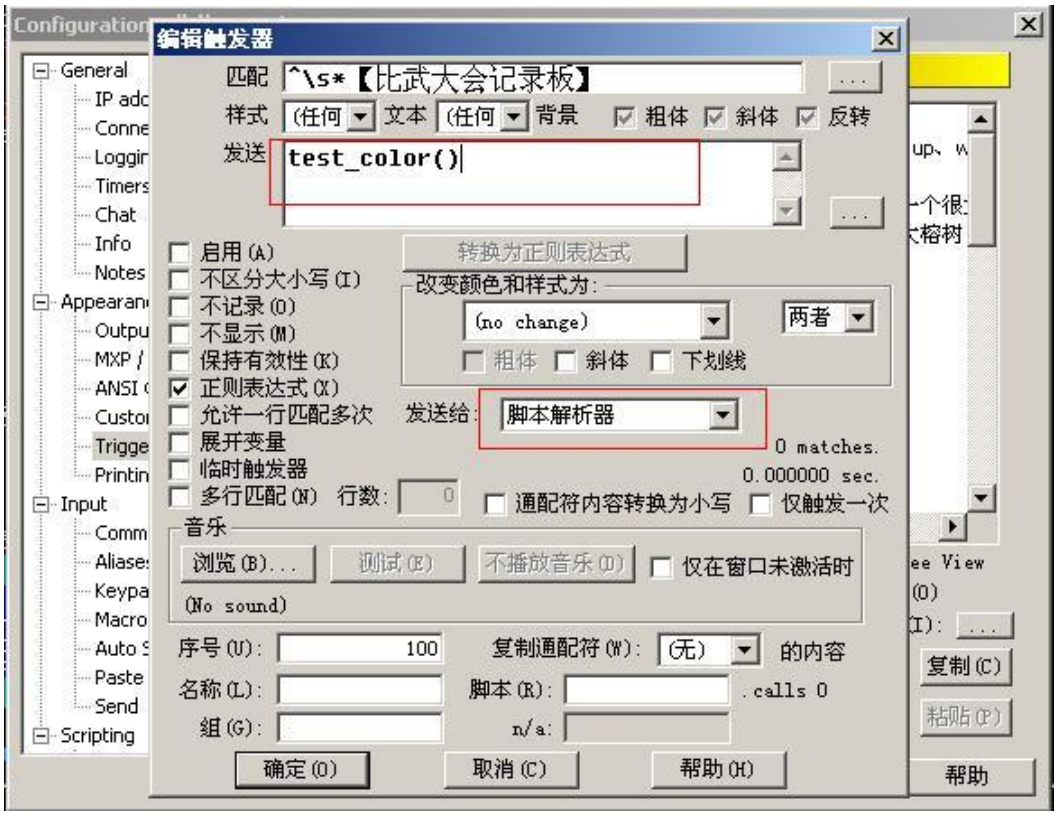
我大概说下几个有用的：1.）**游戏：**指的是这个命令直接发送到游戏，按照字符发送过去了，注意：发送到游戏这里使用你设置好的 alias 是不行的；

2.）**变量：**指这个值直接被送到了上面 variable 设置界面里面设置的变量里面，发送给变量还需要在下面填入变量名，如下图；



3.）**命令解析器：**这个应该是最有用的发送目标之一，发送出的命令需要先在游戏的内部命令系统里面帮忙解析一遍，才发送给游戏。与直接发送给游戏的不同在于：这里发送的内容可以是你前面设置好的 alias，一行多命令等。所以，即使是直接命令，一般也会被送到命令解析器来过一遍而不是直接送游戏；

4.）**脚本解析器：**这也是最有用的发送目的地之一，使用这个发送目标的时候，在发送里面的内容一般是脚本里面的自定义函数名或者 mush 自带的函数名字，如下图：

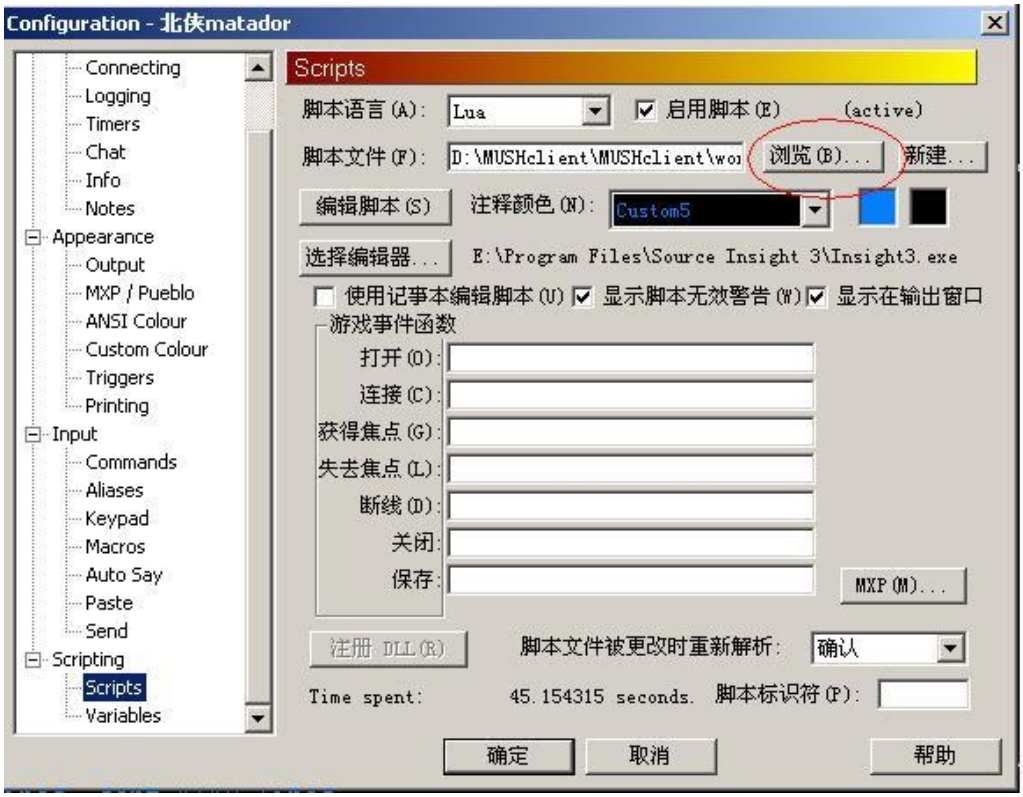


test_color()是我在脚本里面自定义的一个函数。

二.稍进一步的东西

脚本 （script）

脚本的设定在：



点浏览，找到你的脚本文件即可。如果没有就建立一个，按照自己习惯的语言新建一个，点新建按钮即可。

脚本语言支持 lua，vbscript，jscirpt， phython， PerlScript。我只会一点点 lua 语法，所以这里仅用 lua 做示例。

脚本的作用：

脚本可以完成一些用普通触发，alias，timer，variable 完成起来有点麻烦甚至困难的东西，将机器人的编写完全转化为写程序，可以说他是 mush 为什么这么强大的根源所在。下面是一个示例：

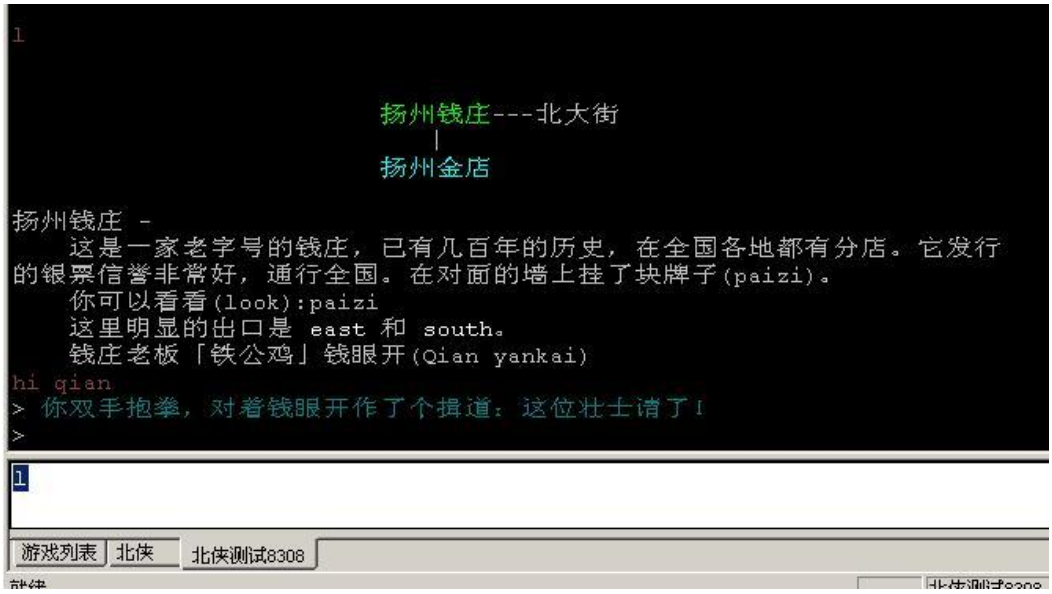
这是一个没用使用正则的脚本例子：

看左侧红圈里面的正则没有勾选，但是还是发送给了脚本解析器。下面是我的脚本程序：

```
zhaohu=function()
    Execute("hi qian")
End
```

Execute 是 mush 内部函数，表示将括号内的字符送到命令解析器

这是执行情况：



也有人说，既然只发了这么简单的命令何必用脚本呢？是的，这么简单可以不使用脚本，不过 mud 里面经常会有讨厌的>这个符号，下面我们会开始说复杂一点的运用----配合正则来讲，不要犯难，不难的。

基本的正则

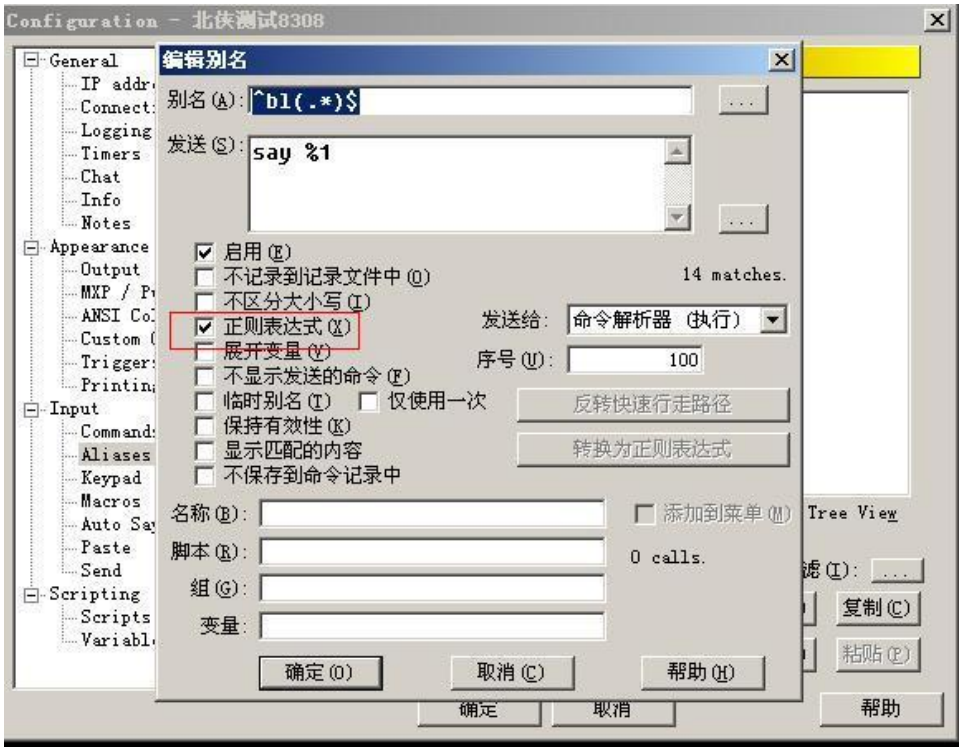
正则说的高深，其实就是 zmud 里面的通配符，也就是用一些约定俗成的符号去通配一些特定的字符，空白等等。
说明
\d
任意一个数字，相当于[0-9]，即 0~9 中的任意一个
\w
任意一个字母或数字或下划线，相当于[a-zA-Z0-9_]
\s
任意空白字符，相当于[\r\n\f\t\v]
\D
任意一个非数字字符，\d 取反，相当于[^0-9]
\W
\w 取反，相当于[^a-zA-Z0-9_]
\S
任意非空白字符，\s 取反，相当于[^ \r\n\f\t\v]
.
匹配除了换行符 \n 以外的任意一个字符
^
匹配字符串开始的位置，不匹配任何字符
$
匹配字符串结束的位置，不匹配任何字符
\b
匹配单词边界，不匹配任何字符
\n
表示换行 （在多行触发里面必须用到），不匹配字符
\r
表示回车，一般在文本文件中使用
一个注意点：以上的通配符仅代表通配一次，举个例：\d 仅仅通配一个一位数，\d\d 通配一个两位数
*
重复零次或更多次
+
重复一次或更多次
?
重复零次或一次
{n}
重复 n 次
{n,}
重复 n 次或更多次
{n,m}
重复 n 到 m 次

*?
重复任意次，但尽可能少重复
+?
重复 1 次或更多次，但尽可能少重复
??
重复 0 次或 1 次，但尽可能少重复
{n,m}?

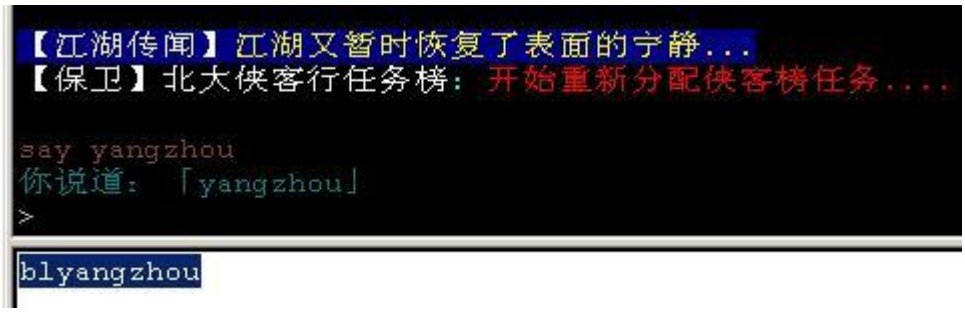
重复 n 到 m 次，但尽可能少重复
{n,}?
重复 n 次以上，但尽可能少重复

[^a-z!@#$%\^&*()\\/.，<>]+
汉字匹配，注意中文标点符号！
北侠中一般用：\S[^a-z，。()]+　来匹配汉字。
北侠中文 ID 允许用 E 文，所以匹配时候要注意。
这里必须要掌握的正则：".","\s","\w","\S","\d","$","^","\n","*","+","?"
举例：

1.）alias 配合正则

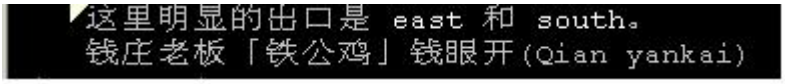


一个注意点：红框里面的正则表达式必须勾选上。
我这里用了^,.*,$ 这几个常用通配符。
^和$代表开头和结束，和 zmud 里面类似。其实这里$在这里是没用的，放在这里做个示例即可。
.*和 zmud 的*是一样的，通配一切除了回车换行。
通配内容的提取，一切默认设置，那么 mush 是和 zmud 一样的，都是使用%1-%99 来提取，另外，想要提取的内容必须用()把它包起来。
执行情况如下：

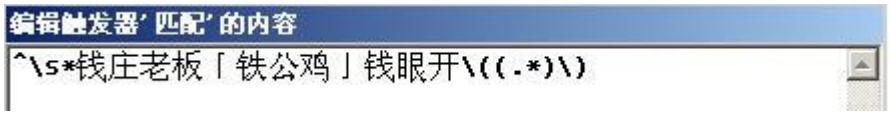


2.）tirgger 配合正则

这是个实际场景：



我写一个触发匹配如下：



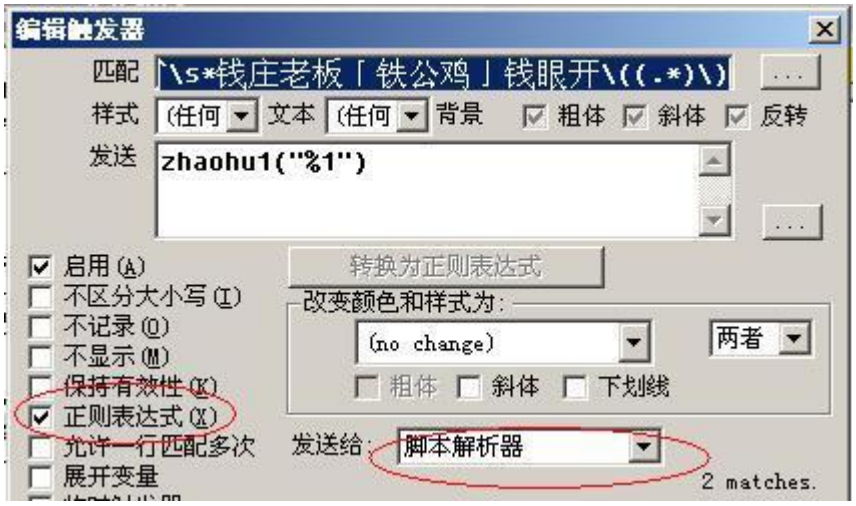
^这个符号匹配顶格，表示一行起始。

\s*通配所有任意空格或者无空格。

\(用来表示这里原文中就有一个(，而不是用来提取内容时候的特定字符(,后面的\)也是一样的情况。以后遇到触发文本中有(),"","'之类的都要加上\用来匹配，和 zmud 中间的~的作用是一样的。

（.*）表示我要在这里提取内容，使用了一个.*,表示只要是括号里面的内容，不管空格，中英文，数字特殊符号，不管长度多少都可以。

这个是触发后的执行部分，是一个脚本里面的函数名：



注意红圈里面的选项内容。

这儿和上面有点不同的地方在于传递了一个内容进脚本，让脚本内的函数可以对这个内容进行操作。注意这里的“%1”，不加这个双引号直接用%1，会有什么效果，大家可以尝试一下。

下面是 zhaohu()和 zhaohu1()的代码

```
zhaohu=function()
    Execute("hi qian")
end

zhaohu1=function(npcid)
Note(npcid)
    local l=string.lower(npcid)
    Execute("ask "..l.." about all")
end
```

npcid 就是上面传递进来的内容

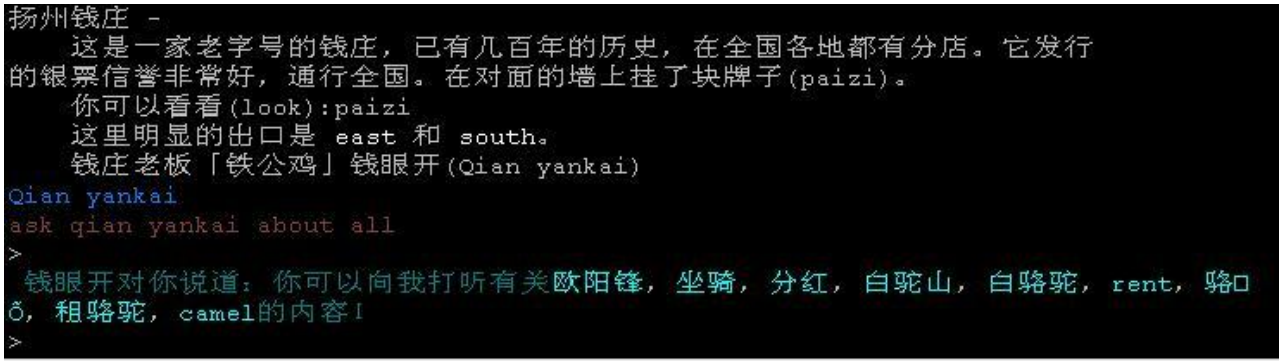
Note 是一个内部函数，主要作用是将括号内的字符在输出窗口显示出来

local l 表示 l 是一个 zhaohu1 函数内部的局部变量，也就是仅在这个函数内部才有效，在其他地方就不认这个 l 了。

string.lower 是 lua 的字符操作命令，作用是将后面的字符变成小写，作用类似于 zmud 中的%lower。

Execute 是 mush 内部函数，表示将括号内的字符送到命令解析器。

执行效果如下：



那行蓝色的 Qian yankai 是 Note(npcid)这句的效果
下面这行 ask qian yankai about all 是下面 2 句的效果

脚本使用过程中需要注意的地方：

1. 脚本内部全局变量与 mush 界面下定义的 variable 是不一样的，不是一回事情。
   对 mush 定义的变量要进行操作需要使用 GetVariable 和 SetVariable 两个 mush 内部函数才行。
2. 引用内容时候的""与不带""是有区别的，具体区别请大家自行尝试。

一些常用脚本内部函数或者命令：

Execute---将后面的字符送到命令解析器。
DoAfterSpecial----间隔时间后将命令发送到指定输出地点（可以是游戏，命令解析器或者脚本）
SetVariable---设定 mush 定义的变量
GetVariable---取 mush 定义的变量
AddTriggerEx---添加一个临时 trigger
SetTriggerOption---对某个触发进行一些配置，一般和 AddTriggerEx 配合使用
EnableTriggerGroup---打开或者关闭某个触发组
AddTimer---添加一个 timer
SetTimerOption---对某个 timer 进行配置
EnableTime---打开或者关闭某个 timer
以上函数在 mush 的 help 下的函数列表里面都有，请自行查找使用方法。

Note/Print----打印想要打印的内容，一般用来输出一些调试信息或者提醒信息
string.find---查找想要的内容，有则输出真，否则输出假
string.format---按照格式输出
string.lower---变小写
string.sub---删除字符串中的内容
tostring---数字改成字符
tonumber---字符变成数字
table.insert---在列表中插入内容
table.remove---在列表中删除内容
以上内容是 lua 语言的内置命令，大致用处我已经说了，但是具体用法请查 lua 语法书或者看别人的例程吧。

多行触发

举个实例，看完你就会了，如下：

$ 表示第一行结束，\n 表示换行。这是触发部分，还需要勾选一个地方，如下：

注意后面的行数，多行是几行就写几行，这是 2 行就写 2

颜色触发：

这里直接调用 maper 前辈的一个例子吧：
抓取劫匪颜色的代码，mushclient，lua 语言。
以下发的是测试 trigger，具体应用请自行研究。
建一个 trigger：

```
<triggers>
  <trigger
   enabled="y"
   match="^劫匪.+"
   regexp="y"
   send_to="12"
   sequence="100"
  >
  <send>robber_color()</send>
  </trigger>
</triggers>
```

lua 脚本代码：

```
function robber_color (name, line, wildcards, styles)
        local line_num = GetLinesInBufferCount()
        local styles_num = GetLineInfo(line_num,11)
        for i = 1,styles_num do
            if GetStyleInfo(line_num,i,1) == "劫匪" then
                jiefei_color = GetStyleInfo(line_num,i,14)
                jiefei_color_ch = RGBColourToName (jiefei_color)
                 print("劫匪的颜色是:"..jiefei_color_ch..", 代码是:"..jiefei_color)
                  break
             end
```

```
        end
end
```

经过测试完全可以，有个注意点：必须把同一颜色块的内容都放到 if GetStyleInfo(line_num,i,1) == "劫匪" then 这一句的判定里面才行，比如 if GetStyleInfo(line_num,i,1) == "劫" then 这个判定是不成立的，必须=="劫匪"才行

引用变量

Mush 的引用变量和 zmud 一样，使用@特殊字符

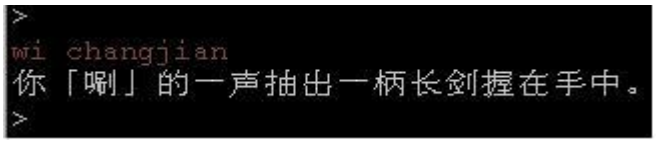
举例：

先定义一个变量 weapon1 为 changjian，如下图：

我们想要在 alias 里面使用这个 weapon 变量，这么做：

在 weapon1 前面加@表示引用 mush 定义变量，然后勾选展开变量就行了。

执行后就是这样：

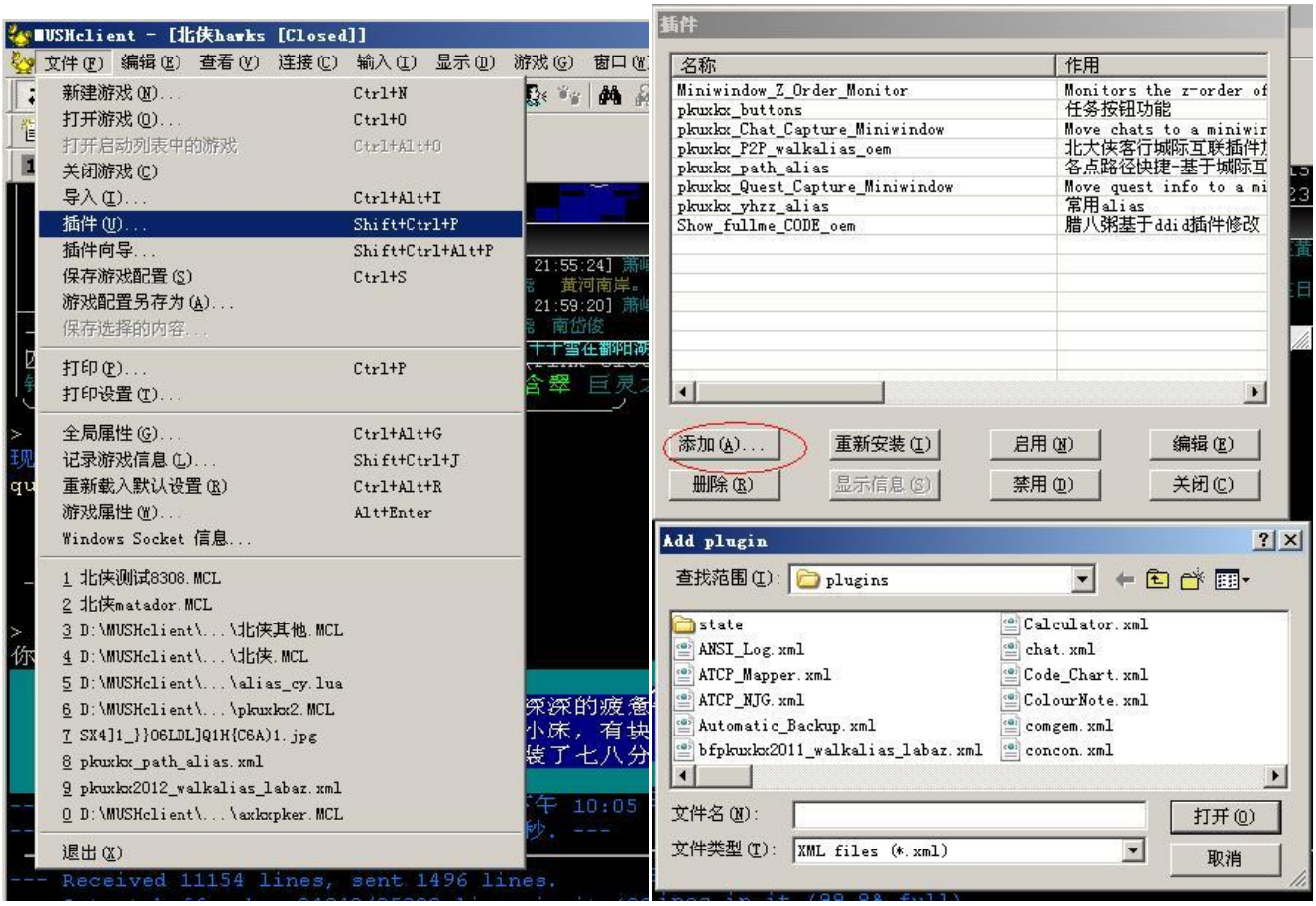


三.插件

插件就是封包好的一些触发，alias，lua 脚本的集合，用来实现比较单一目的的功能。
Mush 支持 xml 格式的插件。对于不想多深入了解插件的人来说，插件会用就行了，没必要深入探讨。

插件的用法：

在这里导入：

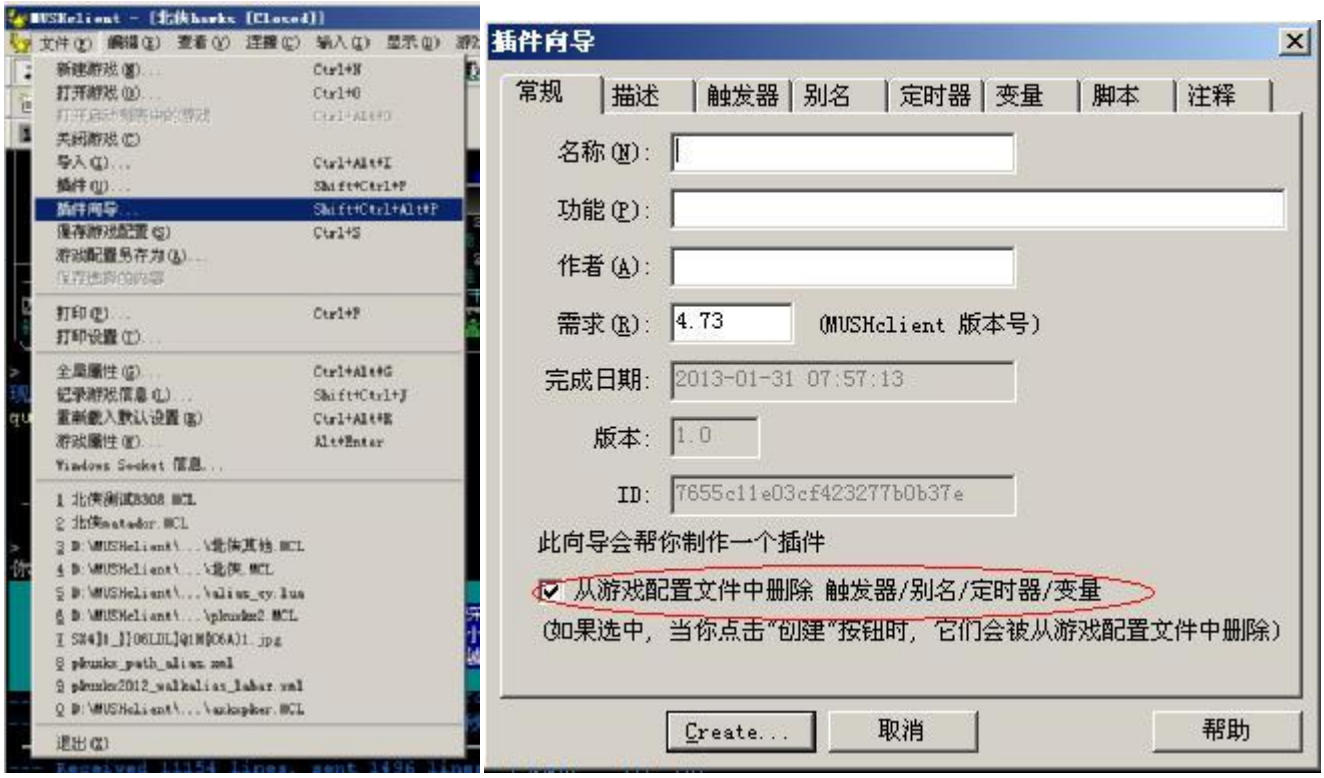


按添加就可以插入你想要的插件了---一定要是 xml 格式的。
至于其他的重新安装，启用禁用，删除就不需要我多介绍了。

插件的好处：

插件的好处是可以使得你的主程序或者说最重要脚本以及其他触发，alias，变量界面显得很干净，而且可以使得程序的可移植性变得相当好，随便一个人拿来就可以用。

插件的生成：



输入名称，功能，作者，描述，注释，后面的触发器，定时器，别名，变量，脚本自己都去看看就知道咋回事了。
注意红圈里面的，自己选择是否点掉。

插件之间的调用：

插件的函数名与全局变量很可能影响到主程序里面的同名函数与变量，所以应该避免这种情况的发生。
在本插件中调用其他插件一般使用：
CallPlugin()这个 mush 自带函数，4.55 版本的 mush 以后支持在脚本里面调用插件
在本插件用调用其他插件的变量，使用：
GetPluginVariable()函数；
想要禁用或者打开某个插件，使用：
EnablePlugin()函数。
插件之间互动，一般用 alias 来实现。
A 插件中 Execute 或者 DoAfter 插入 alias。
在 B 插件中对这个 Alias 做解析，可以很轻松的搞定互动。

目前最有用的插件：

Fullme 插件：Show_Fullme_CODE_oem.xml 作者：DDID,labaz
城际互联插件：pkuxkx_p2p_WalkAlias_oem.xml 作者：littleknife,labaz
Chat 内容采集插件：pkuxkx_Chat_Capture_Miniwindow.xml 作者：Fiendish

四.协程

协程可以这么理解，在你的主线程以外再开一个线程帮你做点主线程里面不方便做的别的事情。

协程的使用准备：

1.需要在主线程中被 require
2.需要在使用前 make(也就是生产）
以 wait 协程实例：
在脚本开始时候需要 require "wait.lua"
在使用的时候需要 wait.make{function()
---这里加入你需要调用协程帮忙的程序
}
3.协程使用的一些注意点：
1.）协程在被调用使用后，如果调用程序结束，那么这个协程就结束了，也就是说重复使用需要重复调用（使用办法使调用程序自身循环不在此类）
2.）协程虽然自我结束了，但是可能是 mush 本身的内存管理问题，导致协程开辟的内存空间并没有被完全释放，导致如果重复调用就会堆积很多内存资源被吃掉，所以不要太过频繁的创建协程。
4.一个很有用的协程：wait
这是个时间类的协程。
wait 有 2 个比较有用的函数：
1.）wait.time(n) n 最小单位为 0.1，我们可以通过他来实现 zmud 中间的#wa 功能，基本用法一样，wait.time(1)代表等待 1 秒，wait.time(0.2)表示等待 0.2 秒
2.)wait.regexp(trigger，timeout，flag)(wait.match 与 regexp 类似),这个功能主要用来等待触发，比如你做了一个什么动作，要等一个什么结果才做下面的事情，那么这个功能极其有用。而且内置了 timeout 功能，比如动作后结果 5 秒还不出来，就会跳出这个等待，你可以接着干别的事情了，或者针对这个结果做一些别的判断。
实例：
tell hash 你是个机器人!!!
local back=wait.regexp(".*你才是机器人.*|.*全家都是机器人.*"，20)
if back==nil then Execute(“chat* robot hash”)
elseif string.find(back,"机器人") then Execute(“chat 还真不是机器人;chat* fear”)
end
如果害喜 20s 没有回答，那就是机器人，哈哈

五.mush 对时间的掌控

有几个办法：

1. DoAfter（包括 DoAfterSpecial）：

先来看 2 个例子：
DoAfter (5, "eat food")---5 秒后执行 eat food 这个命令
DoAfterSpecial(0.1,”wield sword;perform sword.chan;unwieldy all”,10)，0.1 秒后将 wield sword;perform sword.chan;unwieldy all 这一组命令发送到命令解析器(10 就是定义这个的)，如果是 12 那就是发送到脚本
2 个要点：1.）时间的分辨率是 0.1 秒
2.）发送目前可以选择，是发送给命令解析器，直接游戏还是脚本都可以自己配置
有个注意点：
DoAfter 的延后执行是排在命令序列里面，而不是在这里等待，也就是和 zmud 的#wa 不一样，更类似于 zmud 的#alarm。举个例子：
DoAfter (1, "eat food")
Execute(“get ganliang”)
这个会先执行 get ganliang,过 1s 后再执行 eat food
如果你需要按照时间顺序执行一系列命令，那么就需要做好时间增量，然后在固定时间加上这个增量。举例：

DoAfterSpecial(1,”break door”,10)
DoAfterSpecial(2,”eneter”,10)
DoAfterSpecial(3,”get all”,10)
DoAfterSpecial(4,”out,10)

2. 协程 wait

协程 wait 是一个强大的时间工具，如果不是协程的一些固有毛病，我恨不得在绝大部分需要掌控时间的时候用它来实现。

第四大部分协程里面已经介绍了 2 个函数：

1.) wait.time(n)，分辨率为 0.1 秒。可以实现类似 zmud 中间#wa 的作用，也就是在这里停着等待。
2.) wait.regexp/wait.match，自带 timeout 的触发工具，时间分辨率也为 0.1 秒，这个功能对任务过程控制和防止各种误触发相当有用。

3. Timer

Mush 的 timer 可以有很多个，还支持开关以及建立临时，删除等功能，所以极其强大。
Timer 的时间分辨率为 0.01 秒，你的所有时间控制都可用 timer 做的很淋漓尽致。
下面是几个对操控 timer 很有用的函数：
AddTimer()
SetTimerOption()这两个一般配合使用，来建立一个临时 timer
DeleteTimer("timer 名字")---顾名思义，删除
ResetTimer(“timer 名字”)---重置（一般是重置时间）
EnableTimer(“timer 名字”)---允许或者关闭

4. os.date

这是一个取系统时间的函数，
取回来杂用随便你。
---注意：这个函数取回来的是你计算机时间，并非游戏时间，使用前请对表

六.一个固定路径遍历实例：

1. 示例的作用：

这是一个普通路径+动作遍历的示例，作用为将你需要走的所有地方都走一遍；程序在算法上没有任何需要讲解的地方，就是讲一些实现方法讲一下而已。
介绍了 2 种方法：
方法 1：基于触发---这种方法的好处是，可以自动规避服务器延迟，但是在某些场合需要做额外处理来防止出错
方法 2：基于 timer---自带防止出错，但是对抗服务器延迟方面不行

2.一些准备工作：

a.)建立 3 个 mush 层面的 variable（变量）：

bianli_i---存当前普通遍历的步数
bl_dtime---存遍历的延迟时间
bl_dstep---存遍历延迟步数
这 3 个参数设置成 mush 层面的，主要是为了查看方便

b.)先定义几个全局变量(lua 层面)：如下

idself="xyzxyz"---自身 id
blpath={}---存当前遍历路径，1 维数组
blpathover={}---存已经走过的路径，1 维数组
findtarget=0---是否找到目标标志位，1:已经找到 0:尚未找到
yz1="d;out;enter;up;n;se;e;w;nw;w;s;n;e;e;u;enter;out;d;w;n;w;l northwest;u;n;n;s;s;l enter;d;e;e;u;u;d;e;w;w;e;d;w;n;e;w;n;nw;nw;nw;w;nw;se;e;ne;ne;ne;ne;ne;e;e;s;s;e;e;s;s;s;s;s;s;s;s"
---这是一个扬州遍历的区域路径，zmud 习惯的或者说普通 mud alias 习惯的
AddAlias("trigger1", "^t- (.*)$", "EnableTriggerGroup('%1', false)", 1+8+128+1024+16384, "")
SetAliasOption("trigger1","send_to", 12)
AddAlias("trigger2", "^t\\+ (.*)$", "EnableTriggerGroup('%1', true)", 1+8+128+1024+16384, "")
SetAliasOption("trigger2","send_to", 12)
---t+ t-打开/关闭触发组

c.)做几个动作 alias：
“.blxxx”---xxx 为遍历路径名
“gg1”---快速走完遍历路径
“gg2”---快速回退遍历路径
如下：

```
AddAlias("ptbl1", "^\.bl(.*)$", "ptbl_start('%1')", 1+8+128+1024+16384, "")
SetAliasOption("ptbl1","send_to", 12)
---做了一个快捷键，想要遍历那里就敲入".blxx"吧，比如我要遍历 yz1 这个路径就敲入".blyz1"
AddAlias("ptbl2", "gg1", "bl_fast_goto()", 1+8+1024+16384, "")
SetAliasOption("ptbl2","send_to", 12)
---gg1:快速走完剩余遍历路程
AddAlias("ptbl3", "gg2", "bl_fast_back()", 1+8+1024+16384, "")
SetAliasOption("ptbl3","send_to", 12)
---gg2:快速回退遍历路程
```

d.)做几个底层函数：
revlj()---转换路径用的
Split()---将长字符按照分隔符转换为字符串列表
如下：
---函数名:revlj()
---作用:逆转路径
---输入参数:nowlj---待逆转路径
---输出:逆转好的路径
---调用函数:revfx

```
function revlj(nowlj)
    local lj=nowlj
    local i=table.getn(lj)
    local newlj={}
    for newi=1,i do
        newlj[newi]=revfx(lj[i+1-newi])
```

```
    end
    return newlj

---函数名:revfx()
---作用:逆转方向
---输入参数:fx---待逆转方向
---输出:逆转好的方向
function revfx(fx)
---有些路径是不可逆转的，怎么办自己想想办法吧。
        if fx==nil then return    "look"     end
        if fx=="south" then return "north"       end
        if fx=="east" then return "west"     end
        if fx=="west" then return "east"     end
        if fx=="north" then return "south"     end
        if fx=="southup" then return "northdown"     end
        if fx=="southdown" then return "northup"     end
        if fx=="westup" then return "eastdown"     end
        if fx=="westdown" then return "eastup"     end
        if fx=="eastup" then return "westdown"     end
        if fx=="eastdown" then return "westup"     end
        if fx=="northup" then return "southdown"     end
        if fx=="northdown" then return "southup"     end
        if fx=="northwest" then return "southeast"     end
        if fx=="northeast" then return "southwest"     end
        if fx=="southwest" then return "northeast"     end
        if fx=="southeast" then return "northwest"     end
        if fx=="enter" then return "out"     end
        if fx=="out" then return "enter"     end
        if fx=="up" then return "down"     end
        if fx=="down" then return "up"     end


        if fx=="u" then return "d"     end
        if fx=="d" then return "u"     end
        if fx=="s" then return "n"      end
        if fx=="e" then return "w"     end
        if fx=="w" then return "e"     end
        if fx=="n" then return "s"     end
        if fx=="su" then return "nd"     end
        if fx=="sd" then return "nu"     end
        if fx=="wu" then return "ed"     end
        if fx=="wd" then return "eu"     end
        if fx=="eu" then return "wd"     end
        if fx=="ed" then return "wu"     end
        if fx=="nu" then return "sd"     end
        if fx=="nd" then return "su"     end
        if fx=="nw" then return "se"     end
        if fx=="ne" then return "sw"     end
        if fx=="sw" then return "ne"     end
```

```
        if fx=="se" then return "nw"      end
        return fx
  end

---函数名:Split()
---作用:将字符串按照分隔符替换成数组
---输入参数:fullstring---要分割的源字符
---                    tar_string---分割判定符
---输出:分隔好的字符数组
---调用函数:
function Split(fullstring, tar_string)
local start_index = 1
local index = 1
local backstring = {}
local full_len=string.len(fullstring)
local del_len=string.len(tar_string)
while true do
    local last_index = string.find(fullstring, tar_string, start_index)
    if not last_index then
      backstring[index] = string.sub(fullstring, start_index, full_len)
      break
    end
    backstring[index] = string.sub(fullstring, start_index, last_index - 1)
    start_index = last_index + del_len
    index = index + 1
end

return backstring
end --end Split

---函数名:bldz_tran()
---作用:转换路径字符到变量
---输入参数:bldz---待遍历区域字符
---输出:转换好的变量名

function bldz_tran(bldz)
     if bldz==nil then return end
     if string.find(bldz,"yz1") then return yz1 end
end---end bldz_tran
```

e.)做一个触发：

AddTriggerEx ("ptbl2", "^[>]*你双手抱拳，作了个揖道：不好意思，在下自己先请了！", "ptblx()", 0 + 8 + 32 + 1024 + 16384, -1, 0, "","", 12, 100)

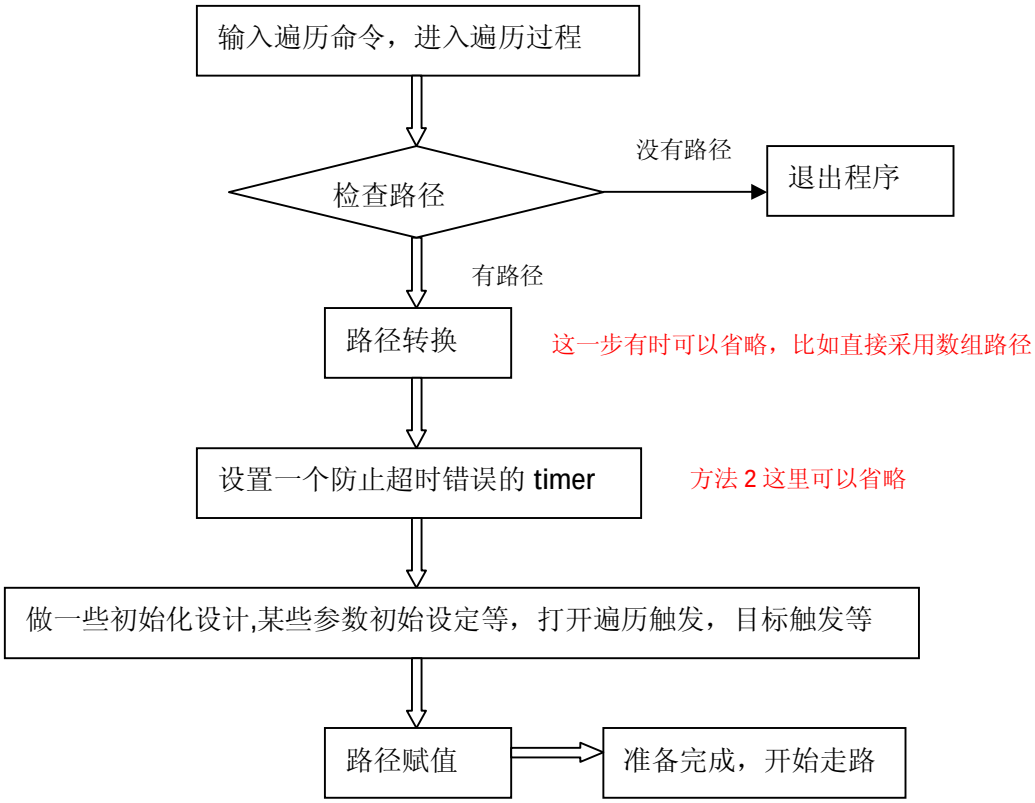
SetTriggerOption ("ptbl2", "group", "ptbl1")

方法 1 的遍历，需要这个触发，方法 1 基于触发

方法 2 的遍历，这个触发可以无视，方法 2 基于 timer

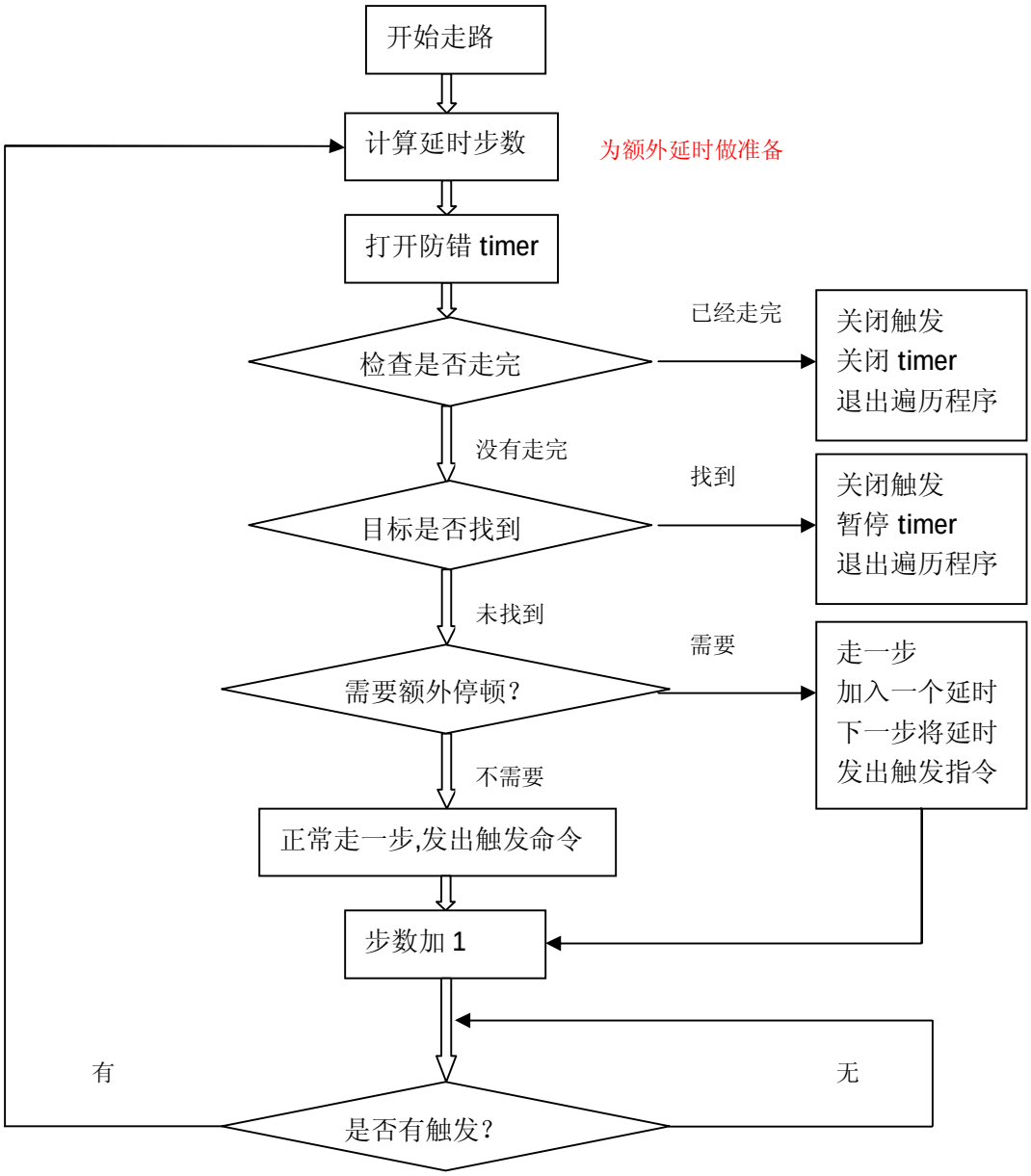
3．工作流程：

准备工作的流程：

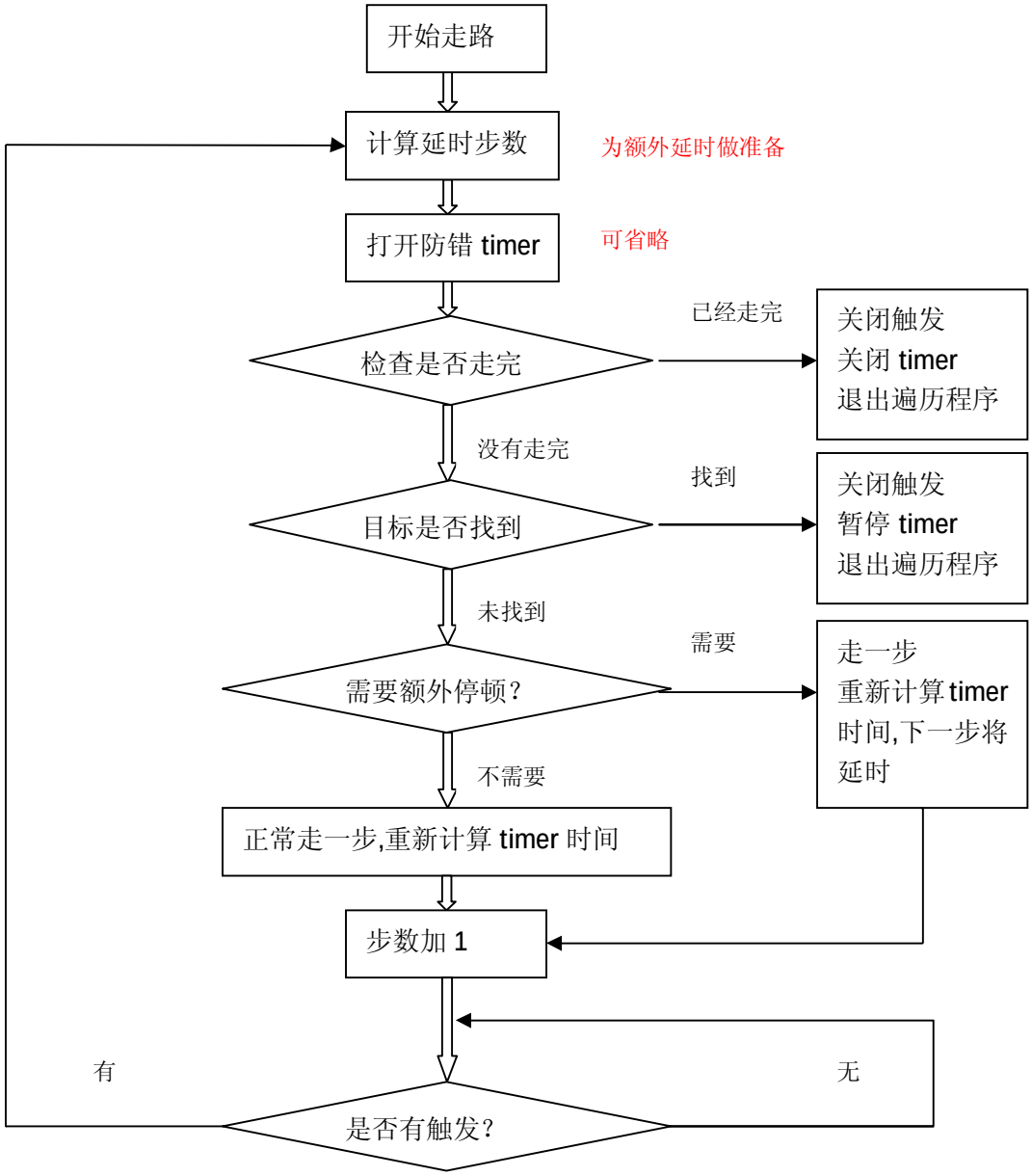

输入遍历命令，进入遍历过程
检查路径
没有路径
退出程序
有路径
路径转换
这一步有时可以省略，比如直接采用数组路径
设置一个防止超时错误的 timer
方法 2 这里可以省略
做一些初始化设计,某些参数初始设定等，打开遍历触发，目标触发等
路径赋值
准备完成，开始走路

遍历过程的流程：

基于触发（方法 1）的流程：



基于 timer（方法 2）的流程：



4. 正式程序：

a.)几个基本函数：

---函数名:ptblact1()
---作用:输出上一次遍历动作
---输入参数:
---输出:
---调用函数:

```
function ptblact1()
    local i=tonumber(GetVariable("bianli_i"))
    i=i-1
    if blpath==nil then return end
```

```
    DoAfterSpecial(1,"ptbly()",12)
end

---函数名:ptbl_start()
---作用:遍历前的一些准备工作
---输入参数:bl_zone---待遍历区域的字符串名字
---输出:遍历开始
---调用函数:bldz_tran,ptblx
function ptbl_start(bl_zone)
 dstep=tonumber(GetVariable("bl_dstep"))   ---取得延时步数
 dtime=tonumber(GetVariable("bl_dtime")) ---取得延时时间
 if dtime < 0.01 then dtime=0.01 end
 if dstep==0 then dstep=4 end   ---如果没设置，则自动配置一个
 Note("开始遍历"..bl_zone) ---输出调试信息
 local lj2=bldz_tran(bl_zone) ---将路径名转换为路径参数
 if lj2==nil then Note("遍历路径缺失!") return end
 print(lj2) ---输出调试信息
 local lj=Split(lj2,";")   ---如果路径是按照 mush 习惯做的可以不做这一步
 AddTimer("bl_anti_err",0,0,20,"ptblact1()",1+1024+16384,"")
 SetTimerOption("bl_anti_err","send_to",12)
 EnableTimer("bl_anti_err",true)
 ---加入一个防止超时错误的 timer
 findtarget=0  ---目标参数清 0
 SetVariable("bianli_i",1) ---设置遍历步数为初始步数 1
      Execute("t+ ptbl1;t+ target")   ---打开遍历触发，这种遍历方式可以不需要这一步
 blpathover={}   ---遍历走完路径清空
 blpath=lj   ---设置遍历路径
 Execute("set brief 2")   ---设置环境变量
 print (os.date ("%c"))  ----输出起始时间，起始就是看看自己能跑多快
 Note("开始遍历")   ---提醒信息
 ptblx() ---开始走路,如果使用方法 2，请用 ptbly()
 --ptbly()---开始走路
end

---函数名:bl_fast_back()
---作用:逆转遍历路径走完
---输入参数:
---输出:
---调用函数:ptblx
function bl_fast_back()
 local i=1
 Note("快速后退")
 local lj=revlj(blpathover)---注意有些路径是不可逆转的，怎么办?
      Execute("t+ ptbl1;t- target")
 blpath=lj
 findtarget=0
 SetVariable("bianli_i",1)
 ptblx()   ---如果是使用方法 2，这里请用 ptbly()
--blpathover={}
```

```
end --end bl_fast_back


---函数名:bl_fast_goto()
---作用:将遍历路径走完
---输入参数:
---输出:
---调用函数:ptblx

function bl_fast_goto()
 Note("快速前进")
 print (os.date ("%c"))
 local i=tonumber(GetVariable("bianli_i"))
 local lj=blpath
 findtarget=0
       Execute("t+ ptbl1;t- target")
 ptblx()---如果是使用方法 2，这里请用 ptbly()
--blpath={}
end -- end bl_fast_goto
```

b.)基于触发的方法 1 主要走路函数：

```
---函数名:ptblx()
---作用:输出遍历动作
---输入参数:
---输出:

function ptblx()
    local i=tonumber(GetVariable("bianli_i"))
    local j=math.fmod(i,dstep)---取余数，为多余延时做准备
    ResetTimer("bl_anti_err")
    EnableTimer("bl_anti_err",true)
    if blpath[i]==nil then
      Note("遍历走完")
      print (os.date ("%c"))
      Execute("t- ptbl1")
           DeleteTimer("bl_anti_err",0)
      return
    end
    if findtarget==0 then
              Note("现在为第"..GetVariable("bianli_i").."步") ---输出调试信息
      if j==0 then
           Execute(blpath[i])
                DoAfterSpecial(0.1+dtime,"hi "..idself,10)
                ---这个一般是为了高速遍历过程而做的延迟，比如 bl_dtime 设定为 0
      else
           Execute(blpath[i])
           DoAfterSpecial(dtime,"hi "..idself,10)
           ---正常延迟走路
      end
```

```
        table.insert(blpathover,i,blpath[i])
        i=i+1
        SetVariable("bianli_i",i)
        ---步数加 1
       else
            Execute("t- ptbl1;t- target")
            EnableTimer("bl_anti_err",false)
            ---暂停住，暂停 timer
        return
       end

 end---endptblx
```

b.)基于 timer 的方法 2 主要走路函数：

```
---函数名:ptbly()
---作用:输出遍历动作
---输入参数:
---输出:


function ptbly()
    local i=tonumber(GetVariable("bianli_i"))
    if dstep==nil then dstep=1000 end---设置 dstep
    local j=math.fmod(i,dstep)---取余数，为多余延时做准备
    ResetTimer("bl_anti_err")
    EnableTimer("bl_anti_err",true)
    if blpath[i]==nil then
      Note("遍历走完")
      print (os.date ("%c"))
      Execute("t- ptbl1")
        DeleteTimer("bl_anti_err")---删除防错 timer
      AddTimer("ptbl_step",0,5,0,"ptbly()",1+1024+16384,"")
 SetTimerOption("ptbl_step","send_to",12)
 EnableTimer("ptbl_step",false)
 ----一个奇怪的 mush bug???少于 1s 的 timer 删除有问题?
        DeleteTimer("ptbl_step")---删除走路触发
      return
    end
    if findtarget==0 then
            Note("现在为第"..GetVariable("bianli_i")..."步") ---输出调试信息
      if j==0 then
             Execute(blpath[i])
      AddTimer("ptbl_step",0,0,0.1+dtime,"ptbly()",1+1024+16384,"")
      SetTimerOption("ptbl_step","send_to",12)
              ---这个一般是为了高速遍历过程而做的延迟，比如 bl_dtime 设定为 0.01
      ---这里下一个 timer 时间延长了 0.1 秒
      EnableTimer("ptbl_step",true)
      else
       Execute(blpath[i])
```

```
            AddTimer("ptbl_step",0,0,dtime,"ptbly()",1+1024+16384,"")
            SetTimerOption("ptbl_step","send_to",12)
            EnableTimer("ptbl_step",true)
            ---正常走路，重设 timer
            end
            table.insert(blpathover,i,blpath[i])
            i=i+1
            SetVariable("bianli_i",i)
        ----步数加 1
            else
                Execute("t- target")
                EnableTimer("bl_anti_err",false)
            EnableTimer("ptbl_step",false)
            ---停止住，暂停 2 个 timer
            return
            end

end---endptbly
```

一些后面的话，越写就觉得自己水平不够，很多想说说的不会，如果有补充我会慢慢改进，但是本人水平估计不会有太大提高了，就这样吧。