#本文来自 http://blog.chinaunix.net/u1/34190/showart_263888.html

Fvwm 中文手册

Fvwm 中文手册-Copyright

Fvwm 中文手册-FVWM（一）

Fvwm

名称（NAME）：

Fvwm -X11 虚拟窗口管理器（F? Virtual Window Manager for X11 ）

概要（SYNOPSIS）：

fvwm [-c config-command] [-d displayname] [-f config-file] [-r] [-s [screen_num]] [-V] [-C visual-class | -I visual-id] [-l colors [-L] [-A] [-S] [-P]] [-D] [-h] [-i client-id] [-F state-file] [--debug-stack-ring] [-blackout]

描述（DESCRIPTION）：

Fvwm 是 X11 的窗口管理器，它致力于最小的内存消耗，并提供窗口边框的 3D 效果和虚拟桌面。

Fvwm 提供一个大的虚拟桌面（virtual desktop）和能被独立或一起使用的多个独立桌面（multiple disjoint desktops）。虚拟桌面让你的显示屏幕看起来好像非常大，多个独立桌面让你好像有多个屏幕同时在工作，但它们之间互相独立。

Fvwm 提供键盘加速（keyboard accelerators），使你仅使用键盘便可完成多数窗口管理器功能，包括移动和缩放窗口，菜单操作和快捷键。

在配置命令（configuration commands）和动作命令（action commands）之间，大多数窗口管理器都有区分，配置命令典型地实现字体、颜色、菜单内容、按键和鼠标的功能绑定等设置，而动作命令主要完成窗口提升（raise）和降低（lower）等功能。Fvwm 没有区分这两者的差别，它允许在任何时候改变任何事情。

在 Fvwm 和其它窗口管理器之间显著的不同还有 SloppyFocus 和 NeverFocus 焦点策略的引入。焦点策略可以为不同组的窗口分别指定。使用 SloppyFocus 时，窗口当鼠标指针移进它们时获得焦点，并保留焦点直到它被其它窗口获得。这样，当鼠标指针移进根窗口（root window）时原来拥有焦点的窗口不会丢失焦点。NeverFocus 可以使窗口从不获得焦点（例如 xclock, oclock, xbiff, xeyes, tuxeyes），比如，如果 SloppyFocus 类型的终端窗口拥有焦点，当鼠标指针移动进一个 NeverFocus 窗口时，它的焦点不会被剥夺。

选项（OPTIONS）：

下面是能够被 fvwm 识别的命令行选项：

-i | --clientid

id　当 fvwm 被一个会话管理器（session manager）启动时，id 选项被使用，用户不应该使用它。

-c | --cmd

config-command　使 fvwm 采用 config-command 取代‘Read config’作为它的初始化指令。

-d | --display

displayname　操纵名称为 displayname 的显示器（display），而不是环境变量$DISPLAY 所表示的。

-D | --debug

使 X 会话（X transactions）运行在同步模式，这将显著的降低效率，但是确保了 fvwm 的内部错误能够被纠正。这也使 fvwm 能够输出运行时的调试信息。

-f config-file

指定 fvwm 的配置文件，取代~/.fvwm/config 作为初始化配置文件。这相当于-c 'Read config-file'。

-h | --help

显示 fvwm 的用法帮助。

-r | --replace

尝试接管正在运行的另外一个窗口管理器，这要求那个窗口管理器必须 ICCCM 2.0 兼容，否则它将不起作用。

-F | --restore

state-file　当 fvwm 被一个会话管理器启动时使用这个选项。用户不应该使用它。

-s | --single-screen

【screen_num】 多屏显示时，使 fvwm 仅仅运行在 $DISPLAY 指定的显示器上或者使用-d 选项所指定的显示器上。正常情况下，多屏显示时，fvwm 尝试在所有显示器上启动。

-V | --version

在 stderr 上显示 fvwm 的版本号。也会显示一些有关 readline、rplay、stroke、xpm、png、gnome hints、EWMH hints、session management，bidirectional text 等支持信息。

-I | --visualid

id　使 fvwm 使用 id 作为窗口边框和菜单的虚拟 id (visual id)。id 可以使用十进制或十六进制。参看 xdpyinfo 的手册页。

-l | --color-limit

limit　这个选项仅当屏幕显示为 256 色或更少并且带有动态视觉效果时才有用。

-L | --strict-color-limit

如果屏幕显示为 256 色或更少并且带有动态视觉效果时，使 fvwm 使用它自己的调色板。

-P | --visual-palette

-A | --allocate-palette

-S | --static-palette

-blackout

这个选项仅为向后兼容而提供。

--debug-stack-ring

这个选项仅被开发者使用。

窗口剖析（ANATOMY OF A WINDOW）：

Fvwm 为多数窗口加上修饰边框。边框（border）包括每一边的栏（bar）和每个角落的 L 形（L-shaped）部分，还有显示窗口名称的标题栏（title-bar），另外，还有多达 10 个标题栏按钮（title-bar buttons）。顶部、四周和底部的 bar 被称作 side-bars，角落部分被称作 frame。

默认的，在 frame 或 side-bars 拖拉鼠标左键将进行窗口缩放。拖拉鼠标右键将进行移动操作。在 border 上单击会执行 raise/lower 操作。

标题栏按钮的作用完全由用户自定义。非常流行的配置是左边的一个按钮用来弹出窗口操作菜单，右边的两个按钮用来图示化（iconify）和最大化（maximize）窗口。另外一个比较流行的配置是在右边增加一个关闭按钮。使用的标题栏按钮的数目依赖于有没有鼠标动作（mouse action）和它绑定。参看下面的 Mouse 指令部分。

虚拟桌面（THE VIRTUAL DESKTOP）：

Fvwm 提供多个虚拟桌面供用户使用。显示屏幕（screen）只是一个桌面（desktop）的视口，即可见部分，桌面可能比显示屏幕更大。可以有几个不同的桌面供访问。因为每个桌面可能比物理显示屏幕大的多，将它们划分成 mxn 页有利于查看。

虚拟桌面的大小可以在任何时候通过 DeskTopSize 命令改变。所有的虚拟桌面必须有同样的大小。桌面的总数可以不必指定，但它有 40 亿的上限。所有的窗口都能够在 FvwmPager（桌面的微型视图）里面观察到。pager 模块只是一个辅助工具，对于窗口管理器来说并不是必须的。窗口也可以依照它们的位置在一个称作 FvwmWinList 的窗口列表里列出。

Fvwm 以层（layer）的方式将窗口叠放在桌面上，位于更低层的窗口从不和位于更高层的窗口相混淆。窗口位于的层可以使用 Layer 指令改变。层的概念是 fvwm 旧版本里 StaysOnTop 标记的扩展。StaysOnTop 和 StaysPut 两个 style 选项现在通过把窗口放在合适的层里来实现，同时增加了之前所没有的 StaysOnBottom 选项。

Sticky 窗口总是在屏幕上显示，好像粘在上面一样。这对于显示像 clocks 和 xbiffs 那样的应用非常方便，运行类似的应用时它将总是显示在屏幕上。如果需要图标也可以设置成 stick。

窗口的几何位置（geometry）是相对于当前视口（viewport）来说的，即：

```
xterm -geometry +0+0
```

表示在显示屏幕的左上角创建一个窗口。相对于虚拟桌面来指定窗口的几何位置是允许的，不过它可能会超出显示屏幕。例如：如果物理屏幕尺寸为 1000x1000，虚拟桌面大小为 3x3，当前视口是在桌面的左上角，调用：

```
xterm -geometry +1000+1000
```

将会在恰好超出屏幕的右下角的地方放置窗口。将鼠标移动到屏幕右下角，并等候它滚动到下一视图时，将看到这个窗口。下面的指令：

```
xterm -geometry -5-5
```

在距显示屏幕右下角 5 个像素的位置放置窗口。并不是所有的窗口都支持这样设置，你可能会发现一些不符合自己预想的情况。

有几种方法可以将窗口放在当前桌面或当前页之外的地方。上面提到的几何位置方法（指定比屏幕物理尺寸更大的 x、y 坐标）受到是相对于当前视口进行位置解析的限制，可能不会达到预想的效果，除非你总是从同一页调用应用程序。

把窗口放在不同页的更好的方法是，在你的配置文件里使用 StartsOnPage 或 StartsOnScreen style 标记。这样的话，放置的效果是比较理想的，并不依赖于你在虚拟桌面上的当前位置。

一些能够解析标准 X 命令行参数和 X 资源的应用，比如 xterm 和 xfongsel，允许用户在命令行指定开始桌面和页：

```
xterm -xrm "*Desk:1"
```

将在桌面 1 启动 xterm；

```
xterm -xrm "*Page:3 2 1"
```

将在桌面 3 的（2，1）页启动 xterm。并不是所有的应用都能够使用这样的选项，但是你能够通过在.Xdefaults 文件里添加下面的语句达到同样的效果：

```
XTerm*Desk: 1
```

或

```
XTerm*Page: 3 2 1
```

多屏显示（USE ON MULTI-SCREEN DISPLAYS）：

如果没有指定-s 命令行参数，fvwm 将自动在每个显示器上启动。每个启动的 fvwm 都被独立对待。在每个显示器上重启 fvwm 将互不影响。指令

```
EdgeScroll 0 0
```

在多屏显示时被强烈推荐。你可能需要在每个显示器上分别退出，才能完全的退出 X 会话。不要把它和 Xinerama 支持相混淆。

XINERAMA 支持（XINERAMA SUPPORT）

Fvwm 支持 Xinerama 扩展，类似于多屏显示但允许在屏幕之间移动窗口。如果 fvwm 编译时添加了 Xinerama 支持，则在通过 Xinerama 方式支持和使用多屏显示的 X server 上运行时，Xinerama 将被使用。没有使用

这个选项时，整个桌面被当作一个大的屏幕。EdgeResistance 命令明确的指定了，在两个 Xinerama 屏幕之间移动窗口通过屏幕边界时的阻力值（explicit resistance value）。Xinerama 支持可以在配置文件里使用 Xinerama 命令打开和关闭。很多模块和命令能够很好的与 Xinerama 一起工作。

在支持 X 格式 geometry 的地方，fvwm 的 Xinerama 扩展允许通过扩展 geometry 来指定一个屏幕。为了这个目的，需要在 geometry 的末尾添加‘@’，并于‘@’之后紧跟一个屏幕号（screen number）或一个字母。字母为‘g’时表示全局屏幕（由所有 Xinerama 屏幕组成的矩形），字母‘p’表示第一个屏幕，字母‘c’表示当前屏幕（鼠标指针当前所在的那个屏幕）。如果 X server 不支持 Xinerama 或者仅有一个屏幕被使用，这个字母（screen bit）将被忽略。

```
Style * IconBox 64x300-0-0@p
```

能够配置 Xinerama 支持使用一个 primary 屏幕。Fvwm 能够在这个屏幕上放置新的窗口和图标。primary 屏幕认为屏幕 0，但可以使用 XineramaPrimaryScreen 命令改变。

初始化（INITIALIZATION）

初始化期间，fvwm 需要搜索配置文件，文件的格式下面将描述。fvwm 首先使用下面的命令搜索配置文件：

```
Read config
```

它将在$FVWM_USERDIR 和$FVWM_DATADIR 目录下面搜索 config 文件，如果搜索失败，则将继续搜索更多的文件。下面是初始化时默认的查找路径，只有第一个发现的文件被使用：

```
$HOME/.fvwm/config

/usr/local/share/fvwm/config



$HOME/.fvwm/.fvwm2rc

$HOME/.fvwm2rc

/usr/local/share/fvwm/.fvwm2rc

/usr/local/share/fvwm/system.fvwm2rc

/etc/system.fvwm2rc
```

注意，并不确保后面的五个以后仍然支持。

如果没有查找到配置文件，则在根窗口使用鼠标左键，Help 或按键 F1 就会调出创建配置文件的菜单和表格。

fvwm 设置了两个环境变量，$DISPLAY 描述 fvwm 正在运行的显示器，可以是 unix:0.0 或:0.0，当通过 ssh 传递到其它机子的时候工作很不稳定。$HOSTDISPLAY 设置为显示器的网络描述（network-ready description），它总是使用 TCP/IP 传输协议（即使在本地），因此本地连接时使用$DISPLAY 速度将更快。

如果你希望启动一些应用或模块，可以添加下面的命令

```
Exec app
```

或

```
Module FvwmXxx
```

到你的配置文件里。但并不推荐这样做，注意仅仅当你知道你将做什么的时候这样做。在全部配置被读取之后启动应用或模块通常是比较危险的，因为它包含了风格（style）或模块配置，这些配置能够影响窗口的外观和功能。

在 fvwm 启动的时候启动应用或模块的标准方式是在初始化函数里添加它们（StartFunction 或 InitFunction）。这样它们将仅在 fvwm 完全读取并执行配置文件之后被启动。

Fvwm 针对初始化有三个专门的函数：StartFunction 在 fvwm 启动和重启时执行；InitFunction 和 RestartFunction 分别于 fvwm 初始化期间和重启期间在 StartFunction 之后执行。可以在配置文件里使用 AddToFunc 命令定义这三个函数，以便启动模块、xterms 或者你希望被 fvwm 启动的什么。

Fvwm 也有一个专门的退出函数：ExitFunction，它在退出的时候执行，可以使用它明确的杀死模块和应用。

如果 fvwm 在一个会话管理器上运行，函数 SessionInitFunction 和 SessionRestartFunction 取代 InitFunction 和 RestartFunction 函数被执行。这使用户的配置文件不管在不在会话管理器上运行都会很好的工作。一般来说，不要在"Session*"函数里启动 xterms 或其它的应用。同样，SessionExitFunction 将取代 ExitFunction 被执行。

```
DestroyFunc StartFunction

AddToFunc StartFunction

+ I Module FvwmPager * *

+ I Module FvwmButtons



DestroyFunc InitFunction

AddToFunc InitFunction

+ I Module FvwmBanner

+ I Module FvwmTaskBar

+ I xsetroot -solid cyan

+ I Exec xterm

+ I Exec netscape



DestroyFunc RestartFunction

AddToFunc RestartFunction

 + I Module FvwmTaskBar
```

```
DestroyFunc SessionInitFunction

AddToFunc SessionInitFunction

 + I Module FvwmBanner



DestroyFunc SessionRestartFunction

AddToFunc SessionRestartFunction

 + I Nop
```

你并不需要定义所有这些函数。注意，所有这些函数可以使用 StartFunction 和 ExitFunction 仿效，例如：

```
DestroyFunc StartFunction

AddToFunc StartFunction

 + I Test (Init) Module FvwmBanner

 + I Module FvwmPager * *

 + I Test (Restart) Beep



DestroyFunc ExitFunction

AddToFunc ExitFunction

 + I Test (Quit) Echo Bye-bye

 + I KillModule MyBuggyModule

 + I Test (ToRestart) Beep
```

Fvwm 中文手册-FVWM（二）

编译选项（COMPILATION OPTIONS）

Fvwm 有大量的编译选项。如果使用某个命令或特点时有麻烦，请确认在编译时支持已经支持它。可选的特点在编译时生成的 config.h 文件里描述。

图标和图像（ICONS AND IMAGES）

Fvwm 可以加载.xbm、.xpm 和.png 图像。XBM 图像是单色的。Fvwm 总是能够显示 XBM 文件。XPM 和 PNG 格式是彩色图像（color images）。是否能显示 XPM 或 PNG 图标和图像取决于编译时得选项。参看 INSTALL.fvwm 文件获取更多信息。

编译时添加 SHAPE 选项能使 fvwm 显示 spiffy shaped 图标。

模块（MODULES）

模块是通过管道跟 FVWM 通信的独立程序，用户能够写自己的模块来实现一些另类的操作。

为了能够设置两个和 fvwm 通信的管道，模块必须由 FVWM spawned。这两个管道在模块启动的时候就已经打开，管道的文件描述符作为命令行参数提供。

在 X 会话（X session）期间，fvwm 可以在任何时候通过 Moudle 命令启动模块。模块能够在整个 X 会话期间存在，也可以在完成一个单个的任务之后退出。如果 fvwm 退出的时候模块仍然是活动的，fvwm 需要关闭通信管道并等待一个来自模块的 SIGCHLD 信号，表明模块已经检测到管道关闭并已经退出了。如果模块未能检测到管道关闭，则 fvwm 在等候大概 30 妙后退出。同时可以运行的模块的数量由操作系统同时可以打开的文件数量决定，通常在 60 到 256 之间。

模块发送命令到 fvwm 命令引擎（fvwm command engine）。某些辅助信息也会被发送。细节请参考 MOUDLE COMMANDS 部分。

ICCCM 兼容（ICCCM COMPLIANCE）

Fvwm 力图去兼容 ICCCM 2.0。更多信息请查看 http://tronche.com/gui/x/icccm。另外，ICCCM 规定应用程序接受任何按键都是可能的，这与 fvwm 以及其它大多数窗口管理器中使用的键盘快捷键方式不一致。特别地，在 fvwm 和另外一个运行在 Xnest（运行在一个窗口里的嵌套 X server）内的 fvwm 间不能使用同样的快捷键方式。同样的问题在鼠标绑定上也是存在的。

ICCCM 规定拥有下面属性的窗口

```
WM_HINTS(WM_HINTS):

      Client accepts input or input focus: False
```

不应该由窗口管理器分配键盘输入焦点。这些窗口能够自己获得输入焦点。但是很多设置了这个属性的应用仍然期望窗口管理器为它们分配键盘焦点，因此 fvwm 提供了一种窗口风格，Lenience，允许 fvwm 忽略这个 ICCCM 规则。不过即使使用这个窗口风格，也不能保证应用能够接受到焦点。

ICCCM 1.1 和 ICCCM 2.0 之间的不同包括，从一个正在运行的 ICCCM 2.0 兼容的窗口管理器接管的能力，因而

```
fvwm; vi ~/.fvwm/config; fvwm -replace
```

模拟了 Restart 命令，但并不完全一样，因为在窗口管理器作为最后一个客户端（client）启动的情况下（.Xclients 或 .Xsession 文件里设置），杀死先前运行的窗口管理器可能会终止你的 X 会话。

更深层次的区别是对 client-side colormap installation 和 the urgency hint 的支持。客户端程序能够在窗口的 WM_HINTS 属性里设置这个提示（hint），并期待窗口管理器去引起用户对这个窗口的注意。Fvwm 重新定义了两个函数来达到这个目的："UrgencyFunc"和"UrgencyDoneFunc"，它们在这个标志被设置或清除的时候运行。它们默认的定义是：

```
AddToFunc UrgencyFunc

+ I Iconify off

+ I FlipFocus

+ I Raise
```

```
        + I WarpToWindow 5p 5p

      AddToFunc UrgencyDoneFunc

        + I Nop
```

GNOME 兼容（GNOME COMPLIANCE）

Fvwm 力图去兼容 GNOME（版本 1）。GNOMEIgnoreHints 可以用来禁止部分或所有窗口的 GNOME hint。

EWMH（EXTENDED WINDOW MANAGER HINTS）

Fvwm 力图去兼容 EWMH：http://www.freedesktop.org/wiki/Standards_2fwm_2dspec 以及这个规范的一些扩展。这允许 fvwm 与 KDE version>=2、GNOME v2 以及其它遵守这个规范的应用（一些基于 GTK+2 的应用）一起工作。遵守这个规范的应用被称作 ewmh 兼容应用（ewmh compliant applications）。

可以使用以 EWMH 为前缀的 style 和 command 配置这个支持。

对于 Key，PointerKey，Mouse 和 Stroke 命令有一个新的上下文（Context）'D'，它适用于桌面应用（比如 kdesktop 和 Nautilus 桌面）。

当一个遵守规范的任务栏要求 fvwm 激活一个窗口时（典型的是你点击任务栏上代表一个窗口的按钮时），fvwm 调用函数 EWMHActivateWindowFunc 来默认执行 Iconify Off, Focus 和 Raise。你可以重定义这个函数，例如：

```
      DestroyFunc EWMHActivateWindowFunc

      AddToFunc EWMHActivateWindowFunc I Iconify Off

        + I Focus

        + I Raise

        + I WarpToWindow 50 50
```

将鼠标指针 wrap 到窗口中央。

EWMH 引入了工作域（Working Area）的概念。不加 ewmh 支持，工作域是整个可见屏幕（或者你的所有屏幕，如果你使用多屏显示或使用 Xinerama）。遵守这个规范的应用（比如一个面板（panel））能够请求在屏幕的边缘预留空间。这种情况下，工作域是整个可见屏幕减去这些预留的空间。如果面板能够通过点击按钮隐藏，工作域不会改变（因为你能够在任何时候弹出这个面板），但是动态工作域（Dynamic Working Area）被更新：被面板预留的空间被删除（如果你弹出面板它将再次增加）。动态工作域可以当 fvwm 放置或最大化窗口时使用。为了知道一个应用是否预留有空间，你可以在终端里输入命令“xprop | grep _NET_WM_STRUT”然后使用鼠标选择这个应用，如果出现有四个数字，它们就表示这个应用的预留空间，正如 EwmhBaseStruts 命令里解释的。

MWM 兼容性（MWM COMPATIBILITY）

Fvwm 提供了模拟 Motif 窗口管理器（Mwm）的一些选项。细节请参考 Emulate 命令以及 Style 和 MenuStyle 命令的 Mwm 相关选项。

OPEN LOOK 和 XVIEW 兼容性（OPEN LOOK and XVIEW COMPATIBILITY）

Fvwm 支持所有的 Open Look 修饰 hints（除了 pushpints）。使用任何这样的应用，都需要添加下面的语句到你的配置文件里：

```
Style * OLDecor
```

大多数（或许所有）Open Look 应用都有一种奇怪的处理键盘焦点的方式，尽管 fvwm 和它们一起工作很好，你仍然可能遇到一些问题。推荐对所有应用使用 NeverFocus 焦点策略和 NoLenience 风格（窗口仍将得到焦点）：

```
Style <application name> NeverFocus, NoLenience
```

但是为了避免你不能使用那个焦点策略，你能够尝试使用下面的方式之一：

```
Style <application name> MouseFocus, Lenience

Style <application name> SloppyFocus, Lenience

Style <application name> ClickToFocus, Lenience
```

M4 预处理 (M4 PREPROCESSING)

M4 预处理由 fvwm 的一个模块处理。更多的细节请参看 FvwmM4 手册。简而言之，如果你希望 fvwm 解析你的 m4 文件，请在你的~/.fvwm/config 文件里使用 FvwmM4 取代 Read 命令，并启动 fvwm。

```
fvwm -cmd "FvwmM4 config"
```

CPP 预处理 (CPP PREPROCESSING)

Cpp 是 C 语言预处理器，借鉴 m4 预处理（m4 pre-processing），fvwm 提供了 cpp 处理（cpp processing）。重读上面的 M4 部分，并用"cpp"取代"m4"。

自动提升 (AUTO-RAISE)

通过使用 auto-raise 模块（FvwmAuto），收到焦点或收到焦点一些毫秒后，窗口能够自动提升。

配置文件 (CONFIGURATION FILES)

配置文件用来描述鼠标和按钮绑定（mouse and button bindings），颜色，虚拟显示尺寸（virtual display size），和相关的项。典型的初始配置文件是 config（或.fvwm2rc）。通过使用 Read 指令，读取你自定义的配置文件是很容易的。

以"#"开始的行被忽略，以"*"开始的行应该是模块配置指令（而不是 fvwm 自己的配置指令）。类似于在 shell 脚本里的换行，在配置文件里也可以通过反斜线符号（backslash）来换行，所有加上这个特点的行被当作一行对待。

Fvwm 不区分配置命令（configuration command）和操作命令（action command）。

已经提供的配置 (SUPPLIED CONFIGURATION)

一个示例配置文件 system.fvwm2rc 随着 fvwm 发布版提供。它有很详细的注释，可以基于它进行修改。可以将它复制为/usr/local/share/fvwm/config 文件。

可选地，fvwm 内置的菜单（没有发现配置文件时附加的）有创建初始配置文件的选项。

如果你刚开始使用 fvwm，可以尝试 fvwm-themes 包，它很好地演示了 fvwm 的功能。

Fvwm 中文手册-FVWM（三）

字体名称和字体加载（FONT NAMES AND FONT LOADING）

可以使用 Font 和 IconFont Style，Font MenuSytle 和 DefaultFont 命令来指定窗口标题（window title）、图标标题（icon title）、菜单（menu）和 geometry 窗口（geometry window）使用的字体。同样，所有使用文本的模块都有指定字体的配置命令。所有这些 styles 和 commands 使用字体名作为参数。这个部分解释了什么是字体名以及 fvwm 加载哪些字体。

首先，你可以使用所谓的常用字体名称（usual font name），例如：

```
-adobe-courier-bold-r-normal--10-100-75-75-m-60-ISO8859-1

-adobe-courier-bold-r-normal--10-*

-*-fixed-medium-o-normal--14-*-ISO8859-15
```

就是说，你能够使用一个 X Logical Font Description（缩写为 XLFD）。第一个匹配这个描述（description）的字体将被加载和使用。这依赖于你的字体路径和 locale，匹配 loclae 字符集（charset）的字体优先级要更高。例如：

```
-adobe-courier-bold-r-normal--10-*
```

如果 locale 字符集为 ISO8859-1，fvwm 会加载下面的字体：

```
-adobe-courier-bold-r-normal--10-*-ISO8859-1
```

locale 字符集为 ISO8859-15 时，会加载：

```
-adobe-courier-bold-r-normal--10-*-ISO8859-15
```

字体名可以作为扩展的 XLFD 给出，XLFD 字体名称列表使用逗号隔开，例如：

```
-adobe-courier-bold-r-normal--14-*,-*-courier-medium-r-normal--14-*
```

fvwm 会尝试去匹配每个字体名，直到发现匹配 locale 字符集的字体，如果匹配不成功，则将不考虑字符集的约束去尝试每个字体名。

关于 XLFD 更多的细节可以参看 X 手册，X Logical Font Description Conventions 文档、XLoadFont 以及 XcreateFontSet 的手册。有一些有用的工具：xlsfonts，fontsel，xfd 和 xset。

如果支持 xft，你可以指定带有 true type（或 Type1）字体前缀“xft:”的 Xft 字体名，例如：

```
"xft:Luxi Mono"

"xft:Luxi Mono:Medium:Roman:size=14:encoding=iso8859-1"
```

匹配上述名称的第一个字体被加载。这依赖于 Xft1 的 XftConfit 配置文件和 Xft2 的/etc/fonts/fonts.conf 文件。可以参考 Xft 手册页和 fontconfig 手册页。“xft:”后面紧跟的是 family 字段，在上面的第二个例子中，Luxi Mono 是 Family（其它的 XFree TTF families："Luxi Serif", "Luxi Sans"），Medium 是权重（Weight）（其它的 weight 可以是：Light，DemiBold，Bold，Black）。

为了决定哪个 Xft 字体被真正的加载，你可以在 fvwm 启动前设置环境变量 XFT_DEBUG=1，并观察错误日志。你也可以使用 fc-list 命令列出可用的 Xft2 字体。不管如何，Xft 支持只是实验性的。显示的质量依赖于 XFree 和 freetype 的版本以及你的显卡。

你可以在 Xft 字体后面添加 XLFD 字体，比如：

```
xft:Verdana:pixelsize=14;-adobe-courier-bold-r-normal--14-*
```

如果 Xft 字体加载不成功，或者 fvwm 不支持 Xft，fvwm 将加载字体"-adobe-courier-bold-r-normal--14-*"。这使得 fvwm 配置文件可移植。

字体和字符串编码 (FONT AND STRING ENCODING)

字体阴影效果 (FONT SHADOW EFFECTS)

字体可以有 3D 效果。在字体名称的开始添加：

```
Shadow=size [offset] [directions]:
```

size 表示阴影像素的数目，offset 是阴影偏移字母边缘的像素数目，默认 offset 为 0。directions 表示阴影偏离字母的方向。directions 是用空格隔开的方向列表。

```
N, North, Top, t, Up, u, -

E, East, Right, r, Right, r, ]

S, South, Bottom, b, Down, d, _

W, West, Left, l, Left, l, [

NE, NorthEast, TopRight, tr, UpRight, ur, ^

SE, SouthEast, BottomRight, br, DownRight, dr, >

SW, SouthWest, BottomLeft, bl, DownLeft, dl, v

NW, NorthWest, TopLeft, tl, UpLeft, ul, <

C, Center, Centre, .
```

阴影将在指定的方向显示。All 相当于所有方向。默认的方向是 BottomRight。使用 Center 方向时，阴影将环绕整个字符串。

阴影效果仅仅和 colorset 一起工作时有效。阴影的颜色使用 Colorset 命令的 fgsh 选项来定义。参考 COLORSETS 部分。

BI-DIRECTIONAL TEXT

快捷键 (KEYBOARD SHORTCUTS)

几乎所有的窗口管理器操作都能使用键盘来完成，因此不使用鼠标完全是有可能的。除了沿虚拟桌面滚动需要绑定 Scroll 命令到适当的按键外，Popup，Move，Resize 和其它任何命令能和任何按键绑定。一旦一个命令已经开始，可以使用 up，down，left，right 箭头移动光标，按下 return 键终止。按下 shift 键使光标以更大的步骤（larger step）移动，按下 ctrl 键使光标以更小的步骤（smaller step）移动。标准 emacs 和 vi 光标移动控制（n,p,f,b 和 j,k,h,l）能够取代箭头键。

会话管理（SESSION MANAGEMENT）

Fvwm 依照 X 会话管理协议（X Session Managerment Protocol）支持会话管理（session management）。它保存和恢复 window position, size, stacking order, desk, stickiness, shadiness, maximizedness, iconifiedness for all windows。此外，还将保存一些全局声明。

Fvwm 不保存有关 styles, decors, functions or menus 的信息。如果你在会话期间对改变了它们（例如，使用 Style 命令或使用各种模块），在重启会话后，这些改变将不再有效。你可以通过将改变添加到配置文件里来避免。

注意当使用多屏显示的时候可能会有下面的异常：第一次启动 fvwm 时，fvwm 通过在每个屏幕上 fork 自己的一个 copy 来管理所有的屏幕，每个 copy 知道它的 parent，发送 Quit 指令到 fvwm 的任何一个实例可以杀死 master 并进而杀死 fvwm 的所有 copy，当你保存和重启这个会话的时候，会话管理器在每个屏幕上产生（bring up）fvwm 的一个 copy，但这时，它们是作为单一的实例存在并仅仅管理一个屏幕，因而 Quit 仅仅杀死命令被发送的那个 copy。这可能不是一个严重的问题，因为你应该总是通过会话管理器来退出一个会话。必要的话，

```
Exec exec killall fvwm
```

将杀死 fvwm 的所有 copy。

布尔参数（BOOLEAN ARGUMENTS）

很多命令使用一个或多个布尔参数。输入“yes”，“on”，“true”，“t”和“y”相当于 true，同时“no”，“off”，“false”，“f”和“n”相当于 false。一些命令可以使用“toggle”参数，“toggle”为触发状态时表示这个特点被禁止。

条件命令和返回代码（CONDITIONAL COMMANDS AND RETURN CODES）

参考 CONDITIONAL COMMANDS 部分。

Fvwm 中文手册-FVWM（四）

内置的按键和鼠标绑定（BUILT-IN KEY AND MOUSE BINDINGS）

下面的命令是 fvwm 内置的：

```
Key Help R A Popup MenuFvwmRoot

Key F1 R A Popup MenuFvwmRoot

Key Tab A M WindowList Root c c NoDeskSort

Key Escape A MC EscapeFunc

Mouse 1 R A Menu MenuFvwmRoot
```

```
Mouse 1 T   A FuncFvwmRaiseLowerX Move

Mouse 1 FS  A FuncFvwmRaiseLowerX Resize

Mouse 2 FST A FuncFvwmRaiseLowerX Move

AddToFunc FuncFvwmRaiseLowerX

 + I Raise

 + M $0

 + D Lower
```

Help 和 F1 按键调用 fvwm 创建的内置菜单。这可能对于新用户很有用处。

Tab+Alt 调用窗口列表菜单。

在标题栏、边框上使用鼠标左键能够 move、raise 或 lower 窗口。

在窗口四个边角使用鼠标左键能够 resize、raise 或 lower 窗口。

你能够去掉或修改这些默认的绑定，比如，为了去掉 Tab+Alt 和窗口列表的绑定，可以使用：

```
Key Tab A M -
```

模块和函数命令 (MODULE AND FUNCTION COMMANDS)

如果 fvwm 遇到一个它无法识别的命令，它会检查这个命令是否应该是：

```
Function (rest of command)
```

或

```
Module (rest of command)
```

这允许调用模块和复杂的函数。

例如：在配置文件里有下面的语句：

```
HelpMe
```

Fvwm 查找名为"HelpMe"的 fvwm 命令，如果不成功，它将查找名为"HelpMe"的用户自定义函数，如果没有这样的函数存在，则 fvwm 将尽力执行一个名为"HelpMe"的模块。

命令的延迟执行 (DELAYED EXECUTION OF COMMANDS)

有很多能够产生特效的命令，比如 Style，Mouse，Colorset，TitleStyle 和很多其它命令。由于性能的原因，这些特效仅当 fvwm 空闲的时候才执行。特别的，在函数里设置的新的 Style 选项将在函数完成之后才执行。有时这可能会导致一些并不希望的效果。

UpdateStyles, Refresh 或 RefreshWindow 命令可以强制所有的改变立即生效。

引号 (QUOTING)

如果希望 fvwm 将两个或更多个单词当成一个单一的参数，引号是必须的。不过你可以使用“-”符号避开它，比如，有一个名为“Window-Ops”的弹出菜单，你不需要加引号：

```
Popup Window-Ops
```

但如果用空格取代“-”，则引号是必须的：

```
Popup "Window Ops"
```

fvwm 可以支持双引号，单引号和 reverse single quotes。所有三种引号被同样对待。

命令扩展 (COMMAND EXPANSION)

任何时候执行一个 fvwm 命令行，fvwm 都需要完成参数的展开。参数可以为$[...]或单个字符。如果 fvwm 遇到一个没有加引号的参数，则将它扩展为字符串。“$$”可以得到字符“$”。

注意，如果在一个窗口上下文外面调用命令，则“$[w.class]”将取代 class 名。

fvwm 能够识别的参数有：

$$

字母'$'

$

当前 Read 文件的绝对路径。

$0 ～$9

传递到函数的参数。“$0”代表第一个参数，“$1”代表第二个参数，依此类推。如果相应的参数没有被定义，“$...”从命令行里删除。

$*

传递到复杂函数的所有参数。包括“$9”之后的参数。

$[n]

传递到函数的第 n 个参数，从 0 开始计数。如果相应的参数没有定义，“$[n]”从命令行里删除。这个参数被不加引号的展开。

$[n-m]

传递到函数的参数，以参数 n 开始到参数 m 结束。如果所有相应的参数没有定义，，“$...”从命令行里删除。如果仅有部分参数定义，所有已定义的参数展开，剩余的忽略。所有参数被不加引号的展开。

$[n-]

所有从 n 开始的参数。如果所有相应的参数没有定义，“$...”从命令行里删除。

$[*]

所有参数，相当于$[0-]。

$[version.num]

版本号，比如“2.6.0”。

$[version.line]

--version 命令行选型输出的第一行。

$[vp.x] $[vp.y] $[vp.width] $[vp.height]

当前视口的坐标和宽高。

$[desk.n]

当前的桌面号。

$[desk.name<n>]

这个参数被桌面 n 的名字替换，在 DesktopName 命令里定义。如果没有定义名称，返回默认名称。

$[desk.width] $[desk.height]

整个桌面的宽高，例如，width 或 height 乘以 x 或 y 方向的页面数。

$[desk.pagesx] $[desk.pagesy]

桌面 x 或 y 方向的页面总数。和 DesktopSize 里设置的一样。

$[page.nx] $[page.ny]

X 和 Y 轴的当前页面号，从 0 开始。page 相当于 GNOME 语法里的 area。

$[w.id]

窗口 id (16 进制)。

$[w.name] $[w.iconname] $[w.class] $[w.resource] $[w.iconfile] $[w.miniiconfile]

窗口名称，图标名称，resource class 和 resource name，使用 Icon 或 MiniIcon style 定义的图标或 mini 图标文件名。

$[w.x] $[w.y] $[w.width] $[w.height]

当前窗口的坐标和宽高（没有图标化的时候）。

$[w.desk]

窗口所在的桌面号，如果窗口是 sticky 风格的，则使用当前桌面号。

$[cw.x] $[cw.y] $[cw.width] $[cw.height]

窗口客户端部分的 geometry。换句话说，窗口的边框和标题不被考虑。

$[i.x], $[it.x], $[ip.x] $[i.y], $[it.y], $[ip.y] $[i.width], $[it.width], $[ip.width] $[i.height], $[it.height], $[ip.height]

返回图标，图标标题，或图标 picture 的 geometry。

$[pointer.x] $[pointer.y]

返回屏幕上光标的位置。如果光标不在这个屏幕上，它们不被扩展。

$[pointer.wx] $[pointer.wy]

返回指针在所选择的窗口里的位置。如果指针不在屏幕上，这个窗口已经图标化或没有窗口被选择时，它们不被扩展。

$[pointer.cx] $[pointer.cy]

返回指针在所选择的窗口的客户区的位置。

$[screen]

fvwm 正在运行的屏幕号。仅对多屏显示时有效。

$[fg.cs<n>]

$[bg.cs<n>]

$[hilight.cs<n>]

$[shadow.cs<n>]

被 colorset <n>定义的前景色、背景色、高亮色、阴影色替换。例如，"$[fg.cs3]"被 colorset 3 的前景色替换。

$[schedule.last]

被 Schedule 命令调度的最后一个命令的 id 替换，即使这个命令已经执行。

$[schedule.next]

被 Schedule 命令调度的下一个命令的 id 替换，即使这个命令已经执行。

$[cond.rc]

最后一个条件命令的返回代码。这个变量只能使用在函数内部，不能被条件命令使用。

$[func.context]

$[gt.str]

返回 str 在当前 locale 下的翻译字符串。

$[...]
如果[]里面的字符串不属于上面中的任何一个，fvwm 将会尽力查找一个同名的环境变量来替换它。

脚本和复杂函数（SCRIPTING AND COMPLEX FUNCTIONS）
为了实现更加复杂的效果，fvwm 有很多提高它的脚本能力的命令。脚本（Script）可以通过三种方式获取，使用 Read 读取文件，使用 PipeRead 获得命令行输出，或者使用 AddToFunc 命令创建函数。fvwm FAQ 的第七部分显示了一些脚本的文件应用。参考 COMMANDS FOR USER FUNCTIONS AND SHELL COMMANDS 和 CONDITIONAL COMMANDS 部分。在一个复杂函数执行期间，fvwm 需要接收所有来自鼠标指针的输入。函数运行期间，其它程序不能够接收任何鼠标指针输入。

Fvwm 中文手册-FVWM（五）

FVWM 命令列表（THE LIST OF FVWM COMMANDS）

菜单（MENUS）

打开菜单前，首先应该使用 AddToMenu 命令对菜单进行定义，并且与 Key，PointerKey，或 Mouse 命令进行绑定（当然还有很多其它调用菜单的方式）。这些通常都是在配置文件里完成。

Fvwm 菜单是可以自己随意定义的，包括菜单字体，背景色，弹出子菜单的延迟时间，动态生成菜单和很多其它特点，即使最细微的差别都可以被改变。更多细节请参考 MenuStyle 命令。

菜单类型（Types of Menus）

共有四种类型的菜单，它们之间并没有太多的区别：

弹出菜单（Popup menus），它能独立地或作为窗口地一部分出现在屏幕的任何地方。使用 Popup 命令打开弹出菜单。如果弹出菜单在鼠标按键按下时被调用，按钮释放时它将消失。鼠标按键按下打开弹出菜单时，可以移动指针选择菜单项，位于指针下面的菜单项被激活，关联动作在按钮释放菜单消失后被执行。

Menu 命令与 Popup 命令有点相似，不过它打开的菜单能够保留的更加长久，通过单击鼠标按键操作打开菜单时，它将一直出现在屏幕上，而且不用像 Popup 命令那样按着鼠标按键来导航菜单项。可以移动鼠标指针选择一个菜单项，然后单击鼠标按键，该菜单项的关联动作将被执行。

Tear off menus 或 Pin up menus 是已经脱离了它们原始的上下文（context），并且像普通窗口那样（具有标题栏、边框等窗口的属性）停留在屏幕上的菜单。可以通过某些键盘或鼠标操作直接将已经打开的菜单转化为这种类型的菜单，也可以使用 TearMenuOff 命令从一个菜单内部创建这种菜单。

子菜单（Sub menus）是包含在菜单内部的菜单。关联 Popup 命令的菜单项被选择时，它的下级菜单会打开，任何菜单都可以有子菜单。

菜单解析（Menu Anatomy）

菜单包括通常出现在菜单顶端的标题（title），关联一定操作的菜单项，菜单项之间的分隔行（separator line，用于对菜单分组），tear off bars（一条水平虚线），和通过小三角形指出的子菜单项。所有上面的项都是可选的。

另外，如果菜单太长以至于不能在在屏幕上完整的显示，则多余的菜单项会放在附加菜单（continuation menu）和子菜单里。最后，菜单的任意一边（side bar）可以显示一副图片。

菜单导航（Menu Navigation）

键盘或鼠标都可以实现菜单导航。在菜单显示的时候，fvwm 将不能进行其它任何操作。比如，在菜单消失前新窗口不会出现。然而，对于 tear off menus 情况会有所不同。细节请参看 TearOffMenus 部分。

鼠标导航（Mouse Navigation）

在菜单上面移动鼠标指针可以选择指针下面的菜单项，通常菜单项周围会出现 3d 效果的边框来显示这一过程，并不是菜单的所有部分都能够被激活。在菜单打开的时候，选择一个菜单项，然后按下鼠标的任何一个按键都可以激活它。弹出菜单的菜单项在松开鼠标按键的时候也可以被激活。如果一个菜单项含有子菜单，则指针停留在该菜单项上面足够久，或者接近指示子菜单的三角形，子菜单都将显示。

在菜单上滚动鼠标滑轮时的操作依赖于 MouseWheel 菜单 style。

点击激活一个所选择的菜单项产生的操作也依赖于菜单项的类型。

在 title，separator，side bar，或者菜单外面单击鼠标都将关闭这个菜单（tear off menus 不能通过这种方式关闭）。在菜单标题或 tear off bar 上点击鼠标按键 2 会创建一个 tear off menu（和当前菜单内容一样，如果是子菜单的 tear off bar，则创建的 tear off menu 和子菜单内容一样）。点击一个普通的菜单项将执行与它关联的操作。点击一个子菜单项会关闭所有打开的菜单并执行该子菜单项的关联操作。

键盘导航（Keyboard Navigation）

像鼠标导航一样，指针下面的菜单项被选择。菜单打开时，所有键盘按键操作都会被菜单截获，其它应用不再能够获得键盘输入（对于 tear off menus 情况会有所不同）。

可以通过每个菜单项的热键直接选择菜单项，菜单项标签里的下划线指出了它关联的热键。使用 AutomaticHotkeys 菜单风格，fvwm 可以自动的为所有菜单项分配热键。

导航菜单的最基本按键是光标键（up 和 down 移动指针到一个菜单项，进入或离开一个子菜单），Space（激活菜单项）和 Escape（关闭菜单）。也有很多其它键可以用来导航菜单。

Enter, Return, Space activate the current item.

Escape, Delete, Ctrl-G exit the current sequence of menus or destroy a tear off menu.

J, N, Cursor-Down, Tab, Meta-Tab, Ctrl-F, move to the next item.

K, P, Cursor-Up, Shift-Tab, Shift-Meta-Tab, Ctrl-B, move to the prior item.

L, Cursor-Right, F enter a sub menu.

H, Cursor-Left, B return to the prior menu.

Ctrl-Cursor-Up, Ctrl-K Ctrl-P, Shift-Ctrl-Meta-Tab, Page-Up move up five items.

Ctrl-Cursor-Down, Ctrl-J Ctrl-N, Ctrl-Meta-Tab,P, Page-Down move down five items.

Home, Shift-Cursor-Up, Ctrl-A move to the first item.

End, Shift-Cursor-Down, Ctrl-E move to the last item.

Meta-Cursor-Up, Ctrl-Cursor-Left, Shift-Ctrl-Tab move up just below the next separator.

Meta-Cursor-Down, Ctrl-Cursor-Right, Ctrl-Tab move down just below the next separator.

Insert opens the "More..." sub menu if any.

Backspace tears off the menu.

Menu Bindings

Tear Off Menus

一个 tear-off 菜单是已经脱离了它们原始的上下文（context），并象普通窗口那样（具有标题栏、边框等窗口的属性）停留在屏幕上的菜单。有三种方式创建 tear-off 菜单：在菜单标题上点击鼠标按钮 2，在菜单上按 Backspace 键，或激活菜单的 tear off bar。通过 TearMenuOff 命令为菜单分配 tear off bar，使用方式和 Nop 命令类似。

Backspace 按键的操作不能被覆盖，但是在菜单标题上点击鼠标按键 2 的操作可以。为了删除内置的鼠标按键 2 操作，可以使用：

Mouse 2 M N -

为了分配其它按键实现tearoff，可以使用：

```
Mouse 1 M N TearOff
```

注意，修饰符（Modifier）必须是"N"（none）。

包含了菜单的窗口被当作普通窗口在屏幕上放置，如果你觉得这比较混乱，在你的配置文件里添加下面的语句：

```
Style fvwm_menu UsePPosition
```

之后，tear-off 菜单将会出现在它 tearoff 前的位置。

为了删除 tear-off 菜单的边框和按钮，但是保留菜单标题，可以使用：

```
Style fvwm_menu !Button 0, !Button 1

Style fvwm_menu !Button 2, !Button 3

Style fvwm_menu !Button 4, !Button 5

Style fvwm_menu !Button 6, !Button 7

Style fvwm_menu !Button 8, !Button 9

Style fvwm_menu Title, HandleWidth 0
```

Tear-off 菜单是介于窗口和菜单之间的产物。菜单被包含在窗口内部，菜单标题显示在窗口标题栏里。这么做主要的好处是菜单可以因此长久保留，即使在激活一个菜单项之后也不会关闭菜单，菜单可以因此多次使用，不用去反复的打开它。为了关闭 tear-off 菜单，关闭它的窗口或 Escape 键即可。

Tear-off 菜单和普通的菜单和窗口行为还是稍有些不同的。它们并不截获键盘焦点，但是当指针在菜单项上面的时候，所有的键盘按键操作还是会发送给它。只要指针位于 tear-off 菜单里面，或者位于它的子菜单里面，其它的 fvwm 按键绑定都会被禁止。指针离开这个区域的时候，所有的子菜单会被立即关闭。注意，包含 tear-off 菜单的窗口拥有焦点时从不高亮。

Tear-off 菜单是原始菜单的复制，并且独立于原始菜单，同样，向原始菜单增加菜单项，或改变原始菜单的风格不能影响 tear-off 菜单。

如果希望在不打开普通菜单的情况下创建 tear-off 菜单，可以使用 Menu 和 Popup 命令的 TearOffImmediately 选项。

Fvwm 中文手册-FVWM（六）

AddToMenu menu-name [menu-label action]

定义菜单。例如：

```
AddToMenu Utilities Utilities Title

+ Xterm           Exec  exec xterm -e tcsh

+ Rxvt            Exec  exec rxvt
```

```
+ "Remote Logins" Popup Remote-Logins
+ Top             Exec  exec rxvt -T Top -n \
                         Top -e top
+ Calculator      Exec  exec xcalc
+ Xman            Exec  exec xman
 + Xmag            Exec  exec xmag
 + emacs           Exec  exec xemacs
 + Mail            MailFunction \
                   xmh "-font fixed"
 + ""              Nop
 + Modules         Popup Module-Popup
 + ""              Nop
 + Exit Fvwm       Popup Quit-Verify
```

可以使用下面的命令调用这个菜单：

```
Mouse 1 R A Menu Utilities Nop
```

或

```
Mouse 1 R A Popup Utilities
```

AddToMenu 命令里没有结束符（end-of-menu），在配置文件里，多个菜单也不需要在相邻的区域里定义。上面例子里加引号的部分（或一个单词）是菜单的标签，其余部分是菜单项关联的命令。空标签""和 Nop 函数用来添加一个分隔符（separator）。

关键词 DynamicPopUpAction 和 DynamicPopDownAction 作为菜单项名称时具有其特殊的含义。它们关联的操作在菜单弹出或消失时执行，可以通过这种方式实现动态菜单。使用 DestroyMenu 自我销毁并从头重建是可能的，菜单销毁后，不要忘记再次添加动态操作（除非当销毁菜单的时候使用了 recreate 选项）。

注意：不要触发需要用户交互的操作，它们可能失败并弄糟你的菜单。参看 Silent 命令。

警告：在动态菜单操作时不要使用 MenuStyle 命令，它将使 fvwm 崩溃。

有几个随着 fvwm 一起发布的有关动态菜单的脚本。它们都有自己的手册页，其中一些，特别是 fvwm-menu-directory 和 fvwm-menu-desktop，可以和 DynamicPopupAction 一起使用来创建目录列表或 GNOME/KDE 应用列表。

例如（文件浏览器 File browser）：

```
# You can find the shell script fvwm_make_browse_menu.sh
```

```
# in the utils/ directory of the distribution.

AddToMenu BrowseMenu

 + DynamicPopupAction Piperead \

'fvwm_make_browse_menu.sh BrowseMenu'
```

例如（图像菜单 Picture menu）：

```
# Build a menu of all .jpg files in

# $HOME/Pictures

AddToMenu JpgMenu foo title

 + DynamicPopupAction Function MakeJpgMenu


AddToFunc MakeJpgMenu

 + I DestroyMenu recreate JpgMenu

 + I AddToMenu JpgMenu Pictures Title

 + I PipeRead 'for i in $HOME/Pictures/*.jpg; \

do echo AddToMenu JpgMenu "`basename $i`" Exec xv $i; done'
```

关键词 MissingSubmenuFunction 作为菜单项名称也有其特殊含义，它会在你试图弹出一个并不存在的子菜单时被执行。使用这个函数，你能够定义和销毁一个不工作的菜单。它后面可以使用任何命令，但是当跟随一个由 AddToFunc 命令定义的函数的时候，fvwm 执行下面的命令：

```
Function <function-name> <submenu-name>
```

submenu-name 作为第一个参数传递给这个函数。

上面提到的 fvwm-menu-directory 脚本可以和 MissingSubmenuFunction 一起使用来创建递归的目录列表。

例如：

```
# There is another shell script fvwm_make_directory_menu.sh

# in the utils/ directory of the distribution. To use it,

# define this function in your configuration file:


DestroyFunc MakeMissingDirectoryMenu

AddToFunc MakeMissingDirectoryMenu
```

```
+ I PipeRead fvwm_make_directory_menu.sh $0


DestroyMenu SomeMenu

AddToMenu SomeMenu

+ MissingSubmenuFunction MakeMissingDirectoryMenu

+ "Root directory" Popup /
```

这是文件浏览器的另外一种实现方式。

在关键词 Title 后面添加 top 选项，可以在菜单顶部添加标题，这将覆盖先前的标题。

```
AddToMenu Utilities Tools Title top
```

菜单标签里，第一个 Tab 键之前的文本与菜单左边对齐，右边的文本与第二列左对齐，剩余的文本右对齐第三列。所有其它的 Tab 被空格替换。可以使用 MenuStyle 命令的 ItermFormat 选项改变这个格式。

如果菜单标签里含有标记`&'，接下来的字母表示该菜单项的热键。标签里的热键会添加下划线。字母`&'使用"&&"的方式获得。

如果菜单标签里含有由标记`*'隔开的字符串，两个`*'之间的文本表示这个菜单项的图标名称。字母`*'使用"**"的方式获得。例如：

```
+ Calculator*xcalc.xpm* Exec exec xcalc
```

将在菜单项"Calculator"中插入图标"xcalc.xpm"，下面

```
+ *xcalc.xpm*           Exec exec xcalc
```

将删除标签，但留下图标。

如果菜单标签里含有由标记`%'隔开的字符串，两个`%'之间的文本表示这个菜单项的图标名称（所谓的显示在菜单标签左边的 mini icon）。同样方式，第二个 mini icon 显示这个菜单项的右边。字母`%'使用"%%"的方式获得。例如：

```
+ Calculator%xcalc.xpm% Exec exec xcalc
```

将在菜单项"Calculator"左边插入图标"xcalc.xpm"，下面

```
+ %xcalc.xpm%           Exec exec xcalc
```

将删除标签，但留下图标。通常这里使用的图片比较小（大概 16x16）。

注意：`%'之间的图标加在菜单项的左边或右边，但`*'不是。

如果菜单名称里（不是标签）包含由`@'隔开的字符串，两个`@'之间的文本表示菜单左边（与菜单底部对齐）显示的 xpm 或 bitmap 图像名称。你可以使用 MenuStyle 命令的 SidePic 选项达到同样的效果。字母`@'使用"@@"的方式获得。例如：

```
AddToMenu StartMenu@linux-menu.xpm@
```

创建一个左边带有 side picture 的菜单。

如果菜单名称里包含由'^'隔开的字符串，两个'^'之间的文本表示 side picture 使用的 X11 颜色的名字。当 side picture 的高度比创建的菜单要低的时候，这个颜色会用来填充剩余的部分。你可以使用 MenuStyle 命令的 SideColor 选项达到同样的效果。字母'^'使用"^^"的方式获得。例如：

```
AddToMenu StartMenu@linux-menu.xpm@^blue^
```

将在菜单的左下角添加一个图片。

所有上面的例子，菜单的名称都是必须的，其它位于不同分隔符之间的字符串都是可选的。

```
ChangeMenuStyle menustyle menu ...
```

改变菜单 menu 的风格为 menustyle。参数中可以指定多个菜单。例如：

```
ChangeMenuStyle pixmap1 \

        ScreenSaverMenu ScreenLockMenu
```

```
CopyMenuStyle orig-menustyle dest-menustyle
```

复制 orig-menustyle 为 dest-menustyle，orig-menustyle 是已经存在的菜单风格。如果 dest-menustyle 不存在，则创建它。

```
DestroyMenu [recreate] menu
```

销毁菜单，后面对它的使用不再有效。你能够通过这个命令在 fvwm 会话期间改变一个菜单的内容。这个菜单能够使用 AddToMenu 命令重建。参数 recreate 通知 fvwm 丢弃所有的菜单项，包括标题。

```
DestroyMenu Utilities
```

```
DestroyMenuStyle menustyle
```

销毁名为 menustyle 的菜单样式，所有使用这个样式的菜单将使用默认样式。默认的菜单样式不能被销毁。

```
DestroyMenuStyle pixmap1
```



```
Menu menu-name [position] [double-click-action]
```

用 sticky 的方式调用先前定义的菜单。也就是说，如果用户使用单击（click）而不是拖拉（drag）的方式调用菜单，这个菜单将一直保留（stays up）。打开（bring up）菜单时鼠标双击（或者菜单和键盘按键绑定的时候快速的敲两次键），命令 double-click-action 会被调用。如果没有指定双击操作，在菜单上双击将什么都不做。然而，如果菜单以一个菜单项（不是标题或分隔符）开始，并且没有指定双击操作，鼠标双击时将调用这个菜单的第一项（仅当指针确实在它上面时）。

如果菜单通过键盘来调用和关闭，当菜单被调用的时候，指针将出现在之前在的位置。

position 参数允许指定菜单放置在屏幕的任何地方，例如屏幕的中心或标题栏的上方。它基本的工作方式是：指定一个 context-rectangle 以及菜单左上角到这个矩形左上角的偏移。position 参数包括几个部

分：

```
[context-rectangle] x y [special-options]
```

context-rectangle 可能是：

Root

当前屏幕的根窗口。

XineramaRoot

整个 Xinerama 屏幕的根窗口。Xinerama 不支持时相当于“Root”。

Mouse

鼠标位置上 1x1 的矩形。

Window

上下文窗口的框架 (the frame of the context window) 。

Interior

上下文窗口内部 (the inside of the context window) 。

Title

上下文窗口或图标的标题 (the title of the context window or icon) 。

Button<n>

上下文窗口的按钮 n (button #n of the context window) 。

Icon

上下文窗口的图标 (the icon of the context window) 。

Menu

当前菜单。

Item

当前菜单项。

Context

当前窗口、菜单或图标。

This

指针所在的控件 (widget) 。 (比如窗口的角落或根窗口)

Rectangle <geometry>

用 X geometry 格式定义的矩形。宽和高默认为 1。

如果 context-rectangle 字段为空或非法，则默认为“Mouse”。注意，上面所列并非在任何环境下都有意义。

偏移 x 和 y 表示菜单左上角相比 context-rectangle 左上角的偏移。这个数字默认以 context-rectangle 宽/高的百分比表示，加后缀‘m’后以该菜单宽/高的百分比表示，加后缀‘p’表示像素值。

你可以使用列表形式的值来取代单个值，它们以自己的符号（sign，正负符号）为分割，不要使用任何其它的分隔符。

如果 x 或 y 以“o<number>”为前缀，菜单和 context-rectangle 的指定位置将重叠，context-rectangle 宽/高的<number>百分比位置刚好在菜单宽/高的<number>百分比位置上。因此“o0”表示菜单和 context-rectangle 的 top/left 边重叠（x 为 o0 时 top 边重叠，y 为 o0 时 left 边重叠），“o100”表示 bottom/right 边重叠（x 为 o100 时 right 边重叠，y 为 o100 时 bottom 边重叠），“o50”表示它们的中心相重叠。默认为“o0”。前缀“o<number>”是“+<number>-<number>m”的缩写。

‘c’被预定义为“o50”，例如：

```
# window list in the middle of the screen

WindowList Root c c



# menu to the left of a window

Menu name window -100m c+0



# popup menu 8 pixels above the mouse pointer

Popup name mouse c -100m-8p



# somewhere on the screen

Menu name rectangle 512x384+1+1 +0 +0



# centered vertically around a menu item

AddToMenu foobar-menu

 + "first item" Nop

 + "special item" Popup "another menu" item \

                  +100 c
```

```
 + "last item" Nop


# above the first menu item

AddToMenu foobar-menu

 + "first item" Popup "another menu" item \

                    +0 -100m
```

注意，你可以通过上面的方式将子菜单放在远离当前菜单的位置，鼠标不离开当前菜单你将不能够到达它，如果指针沿着子菜单的大体方向离开当前菜单，菜单仍将 stays up。

special-options：

可以使用 TearOffImmediately 选项在不打开普通菜单的情况下创建 tear-off 菜单，菜单将首先以普通菜单的形式打开，然后瞬间转换为 tear-off 菜单。tear-off 菜单的位置定义和其它窗口一样。

```
# Forbid fvwm to place the menu window

Style <name of menu> UsePPosition

# Menu at top left corner of screen

Menu Root 0p 0p TearOffImmediately
```

animated 和 Mwm 或 Win 菜单风格或许会将菜单移动到屏幕的其它地方，如果不希望如此，可以增加 Fixed 选项。比如你希望菜单总是出现在屏幕右上角位置的时候，可以这么做。

你希望菜单在你单击它的菜单项的时候出现什么地方那？默认是把标题放在鼠标光标下面，但如果你希望它处于参数指定的位置，可以使用 SelectInPlace 选项。如果你希望指针出现在菜单的标题上，使用 SelectWarp 选项。注意，这些选项仅应用在 PopupAsRootMenu MenuStyle 选项使用的时候。

注意，对于不带位置参数的普通菜单，special-option 不将有效。

Fvwm 中文手册-FVWM（八）

MenuStyle stylename options

设置新的菜单风格或者改变先前定义的菜单风格。stylename 是风格名称，如果它包含空格或 tab，则应该加上引号。名称“*”预留给默认的菜单风格。默认菜单风格用于那些还没有使用 ChangeMenuStyle 命令改变风格的菜单类（menu-like）的对象（例如，WindowList 命令创建的窗口）。参看 DestroyStyle。

options 是以逗号分隔的关键词列表，可以包含如下关键词：Fvwm / Mwm / Win, BorderWidth, Foreground, Background, Greyed, HilightBack / !HilightBack, HilightTitleBack, ActiveFore / !ActiveFore, MenuColorset, ActiveColorset, GreyedColorset, TitleColorset, Hilight3DThick / Hilight3DThin / Hilight3DOff, Hilight3DThickness, Animation / !Animation, Font, TitleFont, MenuFace, PopupDelay, PopupOffset, TitleWarp / !TitleWarp, TitleUnderlines0 / TitleUnderlines1 / TitleUnderlines2, SeparatorsLong / SeparatorsShort, TrianglesSolid / TrianglesRelief, PopupImmediately / PopupDelayed, PopdownImmediately / PopdownDelayed,

PopupActiveArea, DoubleClickTime, SidePic, SideColor, PopupAsRootMenu / PopupAsSubmenu / PopupIgnore / PopupClose, RemoveSubmenus / HoldSubmenus, SubmenusRight / SubmenusLeft, SelectOnRelease, ItemFormat, VerticalItemSpacing, VerticalTitleSpacing, AutomaticHotkeys / !AutomaticHotkeys, MouseWheel, ScrollOffPage / !ScrollOffPage, TrianglesUseFore / !TrianglesUseFore。

上面的列表里面，以'/'分隔成组列出的选项之间彼此排斥，加上!前缀的选项表示具有相反的效果。

Fvwm，Mwm，Win 重置所有的选项为同名风格。默认为 Fvwm 风格。除了 Foreground, Background, Greyed, HilightBack, ActiveFore 和 PopupDelay，它们能够覆盖所有其它的选项。因此应该将它们作为第一个选项来指定，或者使用它们将风格重置为已定义的形式。通过逐个的设置所有其它的菜单项可以达到同样的效果。

Mwm 和 Win 风格的菜单将自动弹出子菜单。Win 风格的菜单使背景色变暗来指出当前菜单项。Fvwm 风格的子菜单与父菜单重叠，Mwm 和 Win 风格的菜单从不与父菜单重叠。

Fvwm 风格相当于设置!HilightBack, Hilight3DThin, !ActiveFore, !Animation, Font, MenuFace, PopupOffset 0 67, TitleWarp, TitleUnderlines1, SeparatorsShort, TrianglesRelief, PopupDelayed, PopdownDelayed, PopupDelay 150, PopdownDelay 150, PopupAsSubmenu, HoldSubmenus, SubmenusRight, BorderWidth 2, !AutomaticHotkeys, PopupActiveArea 75。

Mwm 样式相当于设置!HilightBack, Hilight3DThick, !ActiveFore, !Animation, Font, MenuFace, PopupOffset -3 100, !TitleWarp, TitleUnderlines2, SeparatorsLong, TrianglesRelief, PopupImmediately, PopdownDelayed, PopdownDelay 150, PopupAsSubmenu, HoldSubmenus, SubmenusRight, BorderWidth 2, !AutomaticHotkeys, PopupActiveArea 75。

Win 样式相当于设置 HilightBack, Hilight3DOff, ActiveFore, !Animation, Font, MenuFace, PopupOffset -5 100, !TitleWarp, TitleUnderlines1, SeparatorsShort, TrianglesSolid, PopupImmediately, PopdownDelayed, PopdownDelay 150, PopupAsSubmenu, RemoveSubmenus, SubmenusRight, BorderWidth 2, !AutomaticHotkeys, PopupActiveArea 75。

BorderWidth 表示菜单周围边框的宽度 (thickness)。可以是 0 到 50 之间的像素值，默认为 2。使用非法值时将恢复默认设置。

Foreground 和 BackGround 使用颜色名作为参数，用来显示菜单的文本或背景，可以省略颜色名来恢复默认设置。

Greyed 使用颜色名作为参数，指定菜单项被选中时的颜色。应用指定 Mwm hints 时被禁止。参数省略时表示使用菜单的背景色。

HilightBack 和!HilightBack 决定菜单项被选中时背景是否高亮。HilightBack 使用颜色名作为参数，如果参数省略，则使用菜单的背景色。ActiveColorset 选项覆盖这里指定的颜色。

HilightTitleBack 决定菜单标题的背景是否高亮。如果 TitltColorset 被设置，可以作为这里的背景色，否则使用菜单的背景色。

ActiveFore 和!ActiveFore 决定菜单项被选中时前景是否高亮。ActiveFore 使用颜色名作为参数，参数为空并且 ActiveColorset 指定时，打开高亮。ActiveFore 完全关闭高亮。ActiveColorset 覆盖这里指定的颜色。

MenuColorset 决定一个 colorset 是否被用来代替 Foreground，Background 和 MenuFace。MenuColorset 后面跟随一个大于等于 0 的数字，表示使用之前定义的哪个 colorset。如果数

字省略，则colorset被关闭，将再次使用普通的菜单风格。菜单项的前景色和背景色被colorset中定义的颜色所取代，如果colorset以pixmap定义，这个pixmap将被用作菜单的背景。注意，MenuFace已经为内存消耗进行过优化，消耗的内存比从colorset获得背景少的多。细节请参COLORSETS部分。

ActiveColorset类似MenuColorset。

GreyedColorset类似MenuColorset。

TitleColorset类似MenuColorset，只是仅用于菜单标题。

Hilight3DThick, Hilight3DThin 和 Hilight3DOff决定菜单项被选中时是否使用3D效果高亮。

Hilight3DThickness以-50到+50之间的像素值为参数。前面的三个命令分别相当于thickness为2，1，和0。

Animation 和 !Animation打开和关闭菜单动画。

Font 和 TitleFont以字体名为参数。如果这里指定的字体存在，则它将应用于所有菜单项的文本。如果不存在则使用默认字体。如果指定了TitleFont，则所有菜单的标题都将使用它。

MenuFace选项将在菜单上产生fancy的背景。

比如：

```
MenuFace DGradient 128 2 lightgrey 50 blue 50 \
  white
MenuFace TiledPixmap texture10.xpm
MenuFace HGradient 128 2 Red 40 Maroon 60 \
  White
MenuFace Solid Maroon
```

PopupDelay 需要一个数字参数。这个数字表示当指针移动到包含子菜单的菜单项上面时，子菜单弹出前的延迟时间。如果值为0，则不自动弹出。如果值为空，将使用默认值。如果PopupImmediately选项被使用，则这个延迟时间将无效。

PopupImmediately

PopdownDelay类似PopuuDelay，不过指定的是PopupDelayed风格的超时时间。

PopdownImmediately

PopupOffset 使用两个正整数作为参数。指定子菜单相对于父菜单的位置。如果都为 0，子菜单的左边和父菜单的左边重叠。如果第一个值非 0，表示子菜单的左边相对于父菜单的左边向右偏离了多少像素（如果为负则向左偏离）。如果第二个值非 0，表示子菜单向右或向左偏离父菜单宽度的百分比。

PopupActiveArea 带一个 51 到 100 之间的正整数作为参数。

TitleWarp 和 !TitleWarp

TitleUnderlines0, TitleUnderlines1 和 TitleUnderlines2 指定菜单标题下有多少行。

SeparatorsLong 和 SeparatorsShort 设置菜单分隔符的长度。

TrianglesSolid 和 TrianglesRelief 将影响指示子菜单的小三角形如何显示。

DoubleClickTime 使用数字作为参数，表示在菜单项上单击两次之间间隔多久将被当作双击。默认 450 毫秒，空参数恢复默认设置。

SidePic 后跟图像文件名。

SideColor

PopupAsRootMenu, PopupAsSubmenu, PopupIgnore 和 PopupClose

RemoveSubmenus

SelectOnRelease

ItemFormat

VerticalItemSpacing 和 VerticalTitleSpacing

SubmenusLeft

AutomaticHotkeys 和 !AutomaticHotkeys

MouseWheel

ScrollOffPage

TrianglesUseFore

Fvwm 中文手册-FVWM（九）

MenuStyle forecolor backcolor shadecolor font style [anim]

Popup PopupName [position] [default-action]

这个命令有两个目的：绑定一个菜单到鼠标或键盘操作；绑定一个子菜单菜单到另一个菜单。position 参数和 Menu 命令里的一样。按下鼠标按键调用菜单并且立即释放按键时（或者与键盘绑定时快速的敲两次键），default-action 命令被调用。如果没有指定 default-action，在菜单上双击将什么都不做。然而，如果菜单以一个菜单项（不是标题或分隔符）开始，并且没有指定双击操作，鼠标双击时将调用这个菜单的第一项（仅当指针确实在它上面时）。

可以为弹出菜单绑定鼠标或键盘操作。下面的例子绑定鼠标按键 2 和 3 到"Window Ops"弹出菜单。鼠标按键 2 或 3 在窗口 frame 上按下时将调用这个弹出菜单。

```
Mouse 2 FST N Popup "Window Ops"

Mouse 3 FST N Popup "Window Ops"
```

弹出菜单可以和键盘按键绑定，此时，不使用鼠标，仅使用 up，down 和 enter 键便可操作菜单。

可以把弹出菜单作为子菜单绑定到另外一个菜单。下面的例子定义子菜单"Quit-Verify"，然后绑定到"RootMenu"主菜单。

```
AddToMenu Quit-Verify
 + "Really Quit Fvwm?" Title
 + "Yes, Really Quit"  Quit
 + "Restart Fvwm"      Restart
```

```
 + "Restart Fvwm 1.xx" Restart fvwm1 -s
 + ""                  Nop
 + "No, Don't Quit"    Nop


AddToMenu RootMenu "Root Menu" Title
 + "Open XTerm Window" Popup NewWindowMenu
 + "Login as Root"     Exec exec xterm \
                         -fg green -T Root \
                         -n Root -e su -
 + "Login as Anyone"   Popup AnyoneMenu
 + "Remote Hosts"      Popup HostMenu
 + ""                  Nop
 + "X utilities"       Popup Xutils
 + ""                  Nop
 + "Fvwm Modules"      Popup Module-Popup
 + "Fvwm Window Ops"   Popup Window-Ops
 + ""                  Nop
 + "Previous Focus"    Prev (AcceptsFocus) Focus
 + "Next Focus"        Next (AcceptsFocus) Focus
 + ""                  Nop
 + "Refresh screen"    Refresh
 + ""                  Nop
 + "Reset X defaults"  Exec xrdb -load \
                       $HOME/.Xdefaults
 + ""                  Nop
 + ""                  Nop
 + Quit                Popup Quit-Verify
```

TearMenuOff

在菜单里添加 tear off bar（水平虚线）。如果设置有标签，则标签会取代水平虚线。

```
AddToMenu WindowMenu
+ I "" TearMenuOff

AddToMenu RootMenu
+ I "click here to tear me off" TearMenuOff
```

Title

添加标题。

Fvwm 中文手册-FVWM（十）

MISCELLANEOUS COMMANDS

BugOpts [option [bool]], ...

FlickeringMoveWorkaround， 禁止在应用移动时发送 ConfigureNotify 事件。如果一些窗口移动时附带有令人厌恶的闪烁，这个选项将会非常有帮助。注意，出现这个现象并不是一个 fvwm bug，而是应用本身的问题。

MixedVisualWorkaround，会在执行使用根窗口 visual 的操作前安装 root colormap。仅当 fvwm 启动时加上-visual 选项才有效。

ModalityIsEvil，选项控制 Motif 应用是否可以有模态对话框（modal dialog，只有关闭它才能执行其它操作的对话框）。默认应用不能有模态对话框。要谨慎使用这个选项，一旦打开这个选项，要关闭它的话，就必须重启 fvwm。

RaiseOverNativeWindows，运行在 Windows 或 Windows NT 下的一些 X servers 会需要这个选项，因此，在这些 X server 上运行时，fvwm 会设法检测并初始化这个选项。

RaiseOverUnmanaged，

FlickeringQtDialogsWorkaround，使用具有模态对话框的 KDE 或 Qt 应用时，这个选项禁止焦点窗口的闪烁。这个选项默认情况下是打开的。对于其它应用，如果使用的窗口不能够被 fvwm 管理，它可能会看起来令人厌烦，但是这样的应用非常少，置这个选项为默认设置多数情况下是比较安全的。

EWMHIconicStateWorkaround，

DisplayNewWindowNames 选项打开时，fvwm 输出新窗口的名称，图标名称，resource 和 class 到控制台上，这或许能够帮助用户正确使用 Style 命令。

ExplainWindowPlacement 选项打开时，无论什么时候放置新窗口，或使用 PlaceAgain, Recapture，RecaptureWindow 中的一个命令，fvwm 都会输出一条消息到控制台上。这条消息解释了这个窗口放在哪个 desk，page，Xinerama screen 和 position 上，以及为什么这么放。这个选项可以推断为什么窗口没有出现在你预期的位置上。

DebugCRMotionMethod 选项对用户没有什么帮助。

BusyCursor [Option bool], ...

控制某些命令执行期间的光标。Option 可以是 DynamicMenu, ModuleSynchronous, Read, Wait, *。option 必须跟随一个布尔参数 bool。多个 option 之间使用逗号分割。如果设置一个 option 为“True”，相应命令执行时，fvwm 显示 CursorStyle 命令中 WAIT 上下文的光标。设置为“False”则不显示这个光标。

默认是：

```
BusyCursor DynamicMenu False, \
  ModuleSynchronous False, Read False, \
  Recapture True, Wait False
```

*指所有可用的 option。

Read 选项也可以控制 PipeRead 命令。

DynamicMenu 影响 AddToMenu 命令的 DynamicPopupAction 和 MissingSubmenuFunction 选项。如果它设置为“False”，在动态菜单命令期间，busy cursor 不显示，即使此时是一个 Read 或 PipeRead 命令并且 Read 选项为“True”。

Wait 影响 root cursor。一个 wait 暂停期间，root cursor 被 busy cursor 取代，并且 fvwm 仍然可以进行其它操作（你能够 escape 这个 wait，参考 EscapeFunc 命令）。如果你希望使用这个选项，并且不想使用默认的 root cursor，你应该使用 CursorStyle 命令设置 root cursor。

ClickTime [delay]

指定在按下与释放鼠标按键之间的时间。默认是 150 毫秒。省略参数恢复默认值。

ColorLimit limit

已经废除，看 fvwm 的--color-limit 选项。

ColormapFocus FollowsMouse|FollowsFocus

指定 fvwm 控制颜色的方式。followsmouse 表示鼠标光标所在窗口的 colormap 就是当前的 colormap。followsfocus 表示当前拥有键盘输入焦点的窗口的 colormap 就是当前所采用的 colormap。默认为 FollowsMouse。

CursorStyle context [number| name | xpm | None | Tiny [fore back]]

定义特定上下文的光标风格。注意，这个命令并不控制应用使用的光标形状。context 可以是：

POSITION (top_left_corner)

used when initially placing windows

TITLE (top_left_arrow)

used in a window title-bar

DEFAULT (top_left_arrow)

used in windows that do not set their cursor

SYS (hand2)

used in one of the title-bar buttons

MOVE (fleur)

used when moving or resizing windows

RESIZE (sizing)

used when moving or resizing windows

WAIT (watch)

used during certain fvwm commands (see BusyCursor for details).

MENU (top_left_arrow)

used in menus

SELECT (crosshair)

used when the user is required to select a window

DESTROY (pirate)

used for DESTROY, CLOSE, and DELETE commands

TOP (top_side)

used in the top side-bar of a window

RIGHT (right_side)

used in the right side-bar of a window

BOTTOM (bottom_side)

used in the bottom side-bar of a window

LEFT (left_side)

used in the left side-bar of a window

TOP_LEFT (top_left_corner)

used in the top left corner of a window

TOP_RIGHT (top_right_corner)

used in the top right corner of a window

BOTTOM_LEFT (bottom_left_corner)

used in the bottom left corner of a window

BOTTOM_RIGHT (bottom_right_corner)

used in the bottom right corner of a window

TOP_EDGE (top_side)

used at the top edge of the screen.

RIGHT_EDGE (right_side)

used at the right edge of the screen.

BOTTOM_EDGE (bottom_side)

used at the bottom edge of the screen.

LEFT_EDGE (left_side)

used at the left edge of the screen.

ROOT (left_ptr)

used as the root cursor.

STROKE (plus)

used during a StrokeFunc command.

各种 context 默认的光标风格显示在相应括号里。

置第二个参数为空可以恢复默认设置。第二个参数可以是 X11/cursorfont.h 文件里定义的光标值，或者它的名称（不加 XC_ prefix），或者 xpm 文件名。xpm 文件应该包含一个 3 色（none，black，white）的 pixmap，和一个可选的 hot-spot，如果没有定义 hot-spot，hot-spot 被放在图像中间。此外，第二个参数还可以是 None（没有光标）或 Tiny（单个像素）。例如：

```
# make the kill cursor be XC_gumby (both forms work):

CursorStyle DESTROY 56

CursorStyle DESTROY gumby
```

```
CursorStyle TOP_LEFT topl.xpm

CursorStyle ROOT nice_arrow.xpm yellow black
```

可选的 `fg` 和 `bg` 参数指定了光标的前景色和背景色，默认为 `black` 和 `white`。

下面是 `xpm` 文件的例子：

```
/* XPM */

static char *nice_arrow_xpm[] = {

/* width height num_colors chars_per_pixel hot-spot */

"    14    14        3            1          1 1",

/* colors */

"  c None",

". c black",

"# c white",

/* pixels */

"...           ",

".##..         ",

".####..       ",

" .#####..     ",

" .#######..   ",

"  .########.. ",

"  .##########.",

"   .#######.. ",

"   .######.   ",

"    .######.  ",

"    .###.###. ",

"     .#. .###.",

"     .#.  .#. ",

"      .    .  ",
```

```
        };
```

Fvwm 中文手册-FVWM（十一）

```
DefaultColors [foreground background]
```

指定由 fvwm 创建的各种窗口使用的默认背景色和前景色，比如，移动或缩放操作时出现的 geometry 反馈窗口。foreground 或 background 为‘-’表示不改变原来的设置，两个都为空恢复默认设置。注意，菜单，窗口标题或图标标题不使用默认颜色。

```
DefaultColorset [num]
```

指定 DefaultColors 控制的窗口使用的 colorset。使用

```
        DefaultColorset -1
```

可以恢复 DefaultColors 设置的颜色，

```
DefaultFont [fontname]
```

设置默认字体为 fontname。在没有专门指定字体的时候，使用这里设置的默认字体。参数为空时恢复默认设置。菜单、窗口标题、图标标题，移动或缩放操作时出现的 geometry 反馈窗口都可以使用这里的默认字体。为了在特定的上下文里替换这里的默认字体，可以使用 Style * Font, Style * IconFont, 和 MenuStyle 命令。

```
DefaultIcon filename
```

指定默认图标。

```
DefaultLayers bottom put top
```

指定 Style 命令 StaysOnBottom, StaysPut, StaysOnTop 选项使用的层。可以使用 2，4，6。

```
Deschedule [command_id]
```

删除所有被 Schedule 调度的命令，除非这个命令已经执行。如果 command_id 为空，使用$ [schedule.last]替代。

```
Emulate Fvwm|Mwm|Win
```

```
EscapeFunc
```

默认的组合键 Ctrl-Alt-Escape 可以解除一个 Wait 暂停和锁定的 ModuleSynchronous 命令。和 Key 命令一起使用可以配置组合键，例如：

```
        Key Escape A MC -

        Key Escape A  S EscapeFunc
```

将使用 Shift-Escape 替换 Ctrl-Alt-Escape。如果被 Key 命令之外的其它命令使用，EscapeFunc 将什么都不做。

```
FakeClick [command value] ...


FakeKeypress [command value] ...


GlobalOpts [options]
```

已经废除，请依据下面的列表在配置文件里替换 `GlobalOpts` 设置的全局选项。

```
        GlobalOpts WindowShadeShrinks
          -->
        Style * WindowShadeShrinks


        GlobalOpts WindowShadeScrolls
          -->
        Style * WindowShadeScrolls


        GlobalOpts SmartPlacementIsReallySmart
          -->
        Style * MinOverlapPlacement


        GlobalOpts SmartPlacementIsNormal
          -->
        Style * TileCascadePlacement


        GlobalOpts ClickToFocusDoesntPassClick
          -->
        Style * ClickToFocusPassesClickOff


        GlobalOpts ClickToFocusPassesClick
```

```
  -->

Style * ClickToFocusPassesClick


GlobalOpts ClickToFocusDoesntRaise

  -->

Style * ClickToFocusRaisesOff


GlobalOpts ClickToFocusRaises

  -->

Style * ClickToFocusRaises


GlobalOpts MouseFocusClickDoesntRaise

  -->

Style * MouseFocusClickRaisesOff


GlobalOpts MouseFocusClickRaises

  -->

Style * MouseFocusClickRaises


GlobalOpts NoStipledTitles

  -->

Style * !StippledTitle


GlobalOpts StipledTitles

  -->

Style * StippledTitle
```

```
GlobalOpts CaptureHonorsStartsOnPage

  -->

Style * CaptureHonorsStartsOnPage


GlobalOpts CaptureIgnoresStartsOnPage

  -->

Style * CaptureIgnoresStartsOnPage


GlobalOpts RecaptureHonorsStartsOnPage

  -->

Style * RecaptureHonorsStartsOnPage


GlobalOpts RecaptureIgnoresStartsOnPage

  -->

Style * RecaptureIgnoresStartsOnPage


GlobalOpts ActivePlacementHonorsStartsOnPage

  -->

Style * ManualPlacementHonorsStartsOnPage


GlobalOpts ActivePlacementIgnoresStartsOnPage

  -->

Style * ManualPlacementIgnoresStartsOnPage


GlobalOpts RaiseOverNativeWindows

  -->

BugOpts RaiseOverNativeWindows on
```

```
          GlobalOpts IgnoreNativeWindows

            -->

          BugOpts RaiseOverNativeWindows off
```

```
HilightColor textcolor backgroundcolor
```

已经废除，使用

```
          Style * HilightFore textcolor, \

                  HilightBack backgroundcolor
```

代替。

```
HilightColorset [num]
```

已经废除，使用

```
          Style * HilightColorset num
```

代替。

```
IconFont [fontname]
```

已经废除，使用

```
          Style * IconFont fontname
```

代替。

```
IconPath path
```

已经废除，使用 `ImagePath` 代替。

```
ImagePath path
```

指定图像文件的搜索路径（以“:”分隔的目录列表）。`fvwm` 会搜索这里列出的每个目录查找使用的图像，并使用匹配的第一个文件。

如果目录以“`/some/dir;.ext`”的形式给出，表示这个目录里的所有图像文件强制性的使用扩展名“`.ext`”，原始的图像名称（可能没有扩展名或有其它形式的扩展名）不被查询，其扩展名直接用"`.ext`"替换。例如，如果用户有包含“`.xpm`”图像的目录，同时有另外名称相同但是包含“`.png`”图像的目录。

`path` 可以使用环境变量`$HOME` (或`${HOME}`)，‘+’表示之前定义过的那个 `path`。例如：

```
          ImagePath $HOME/icons:+:/usr/include/X11/bitmaps
```

```
LocalePath path
```

指定"locale path"目录列表。一个 locale path 由一个目录路径和一个文本域组成。默认的 locale path 是：

```
/install_prefix/share/locale;fvwm
```

install_prefix 是 fvwm 的默认安装目录。使用这个 locale path 时，

类似于 ImagePath 命令，path 可以包含环境变量，'+'表示之前定义过的那个 path。

例如，fvwm-themes 包使用

```
LocalePath ";fvwm-themes:+"
```

添加 locale 目录。

默认的 fvwm 目录包含一些 fvwm 自己使用的字符串以及一些默认配置文件和 FvwmForm 配置使用的字符串。可以在 fvwm 源文件的 po/目录里获得翻译成不同语言的字符串列表。目前只有少数的语言被支持。

Fvwm 中文手册-FVWM（十二）

```
PixmapPath path
```

已经废除，使用 ImagePath 代替。

```
PrintInfo subject [verbose]
```

在 stderr 上输出 subject 有关的信息。可选的整数参数 verbose 定义输出信息的层。当前有效的 subjects 是：

Color 输出 fvwm 使用的颜色信息。如果 verbose 为 1 或更大，输出 fvwm 使用的调色板。如果你的调色板色彩有限，这个命令可能有所帮助。

Locale 输出 fvwm 使用的 locale 和字体信息。verbose 可以是 1 或 2。

nls 输出 fvwm 使用的 locale 目录信息。

style 输出 fvwm styles 有关的信息，verbose 可以是 1。

```
Repeat
```

重复执行上一次执行的命令。Function 命令，Read 或 PipeRead 命令，菜单不被重复。

```
Schedule [Periodic] delay_ms [command_id] command
```

```
State state [bool]
```

```
WindowFont [fontname]
```

已经废除，使用

```
Style * Font fontname
```

代替。

```
WindowList [(conditions)] [position] [options] [double-click-action]
```

```
XSync
```

XSync 被调用时，同名的 X 函数发送所有待处理的 X 请求到服务器。仅供调试使用。

```
XSynchronize [bool]
```

```
+
```

用来增加 decor，function 或 menu。参考 AddToDecor, AddToFunc, 和 AddToMenu 命令。

Fvwm 中文手册-FVWM（十三）

```
COMMANDS AFFECTING WINDOW MOVEMENT AND PLACEMENT
```

```
AnimatedMove x y [Warp]
```

在移动窗口时添加动画效果。和下面的 Move 命令有点类似。它们有相同的选项，但是这个命令并不是必须的，因为在移动窗口时添加交互和动画效果并没什么意义。参数 Warp 指定时，指针被 wrap 到窗口里。

```
HideGeometryWindow [Never | Move | Resize]
```

移动或缩放窗口时，隐藏显示的位置和尺寸窗口。Never 参数表示不隐藏。

```
Layer [arg1 arg2] | [default]
```

把当前窗口放到一个新的层里。如果 arg1 非 0，新的层是当前层加上 arg1。如果 arg1 为 0，新的层是 arg2。

default 把窗口放在它的默认层里，例如，它初始化时所在的层。这也适用于没有参数或参数无效时。

```
Lower
```

仅在窗口所在的层里降低窗口。为了使窗口处于最底层的最下面，可以使用：

```
AddToFunc lower-to-bottom
 + I Layer 0 0
 + I Lower
```

```
Move [[screen screen]
```

[w|m]x[p] [w|m]y[p] [Warp]] | [pointer]

移动窗口。如果在根窗口上调用这个命令，用户需要指定目标窗口。

screen 表示屏幕号，它的坐标是相对于指定屏幕的坐标。screen 和 MoveToScreen 命令里描述的一样。参数 Warp 指定时，指针被 wrap 到窗口里。指定 pointer 时，窗口的左上角在开始操作前被移动到指针的位置。

这个操作可以被 Escape 键或其它鼠标按键终止。默认鼠标按键 2 取消移动操作。你可以使用 Mouse 命令和上下文'P'一起改变这个设置。

窗口条件 PlacedByButton 可以用来检查特定按键是否按下来放置窗口。（参看 Current 命令）

如果指定了 x 和 y，则不再需要和用户交互，窗口将立即移动。每个参数可以指定一个绝对或相对位置。默认地，x 和 y 表示屏幕宽/高的百分比，后缀'p'时表示像素值，前缀'w'时表示相对于它的当前位置移动。前缀'm'表示相对于当前指针的位置移动。

下面是几个简单的例子：

```
# Interactive move

Mouse 1 T A Move

# Move window to top left is at (10%,10%)

Mouse 2 T A Move 10 10

# Move top left to (10pixels,10pixels)

Mouse 3 T A Move 10p 10p
```

更复杂一些的例子：

```
# Move window so bottom right is at bottom

# right of screen

Move -0 -0



# Move window so top left corner is 10 pixels

# off the top left screen edge

Move +-10 +-10



# Move window 5% to the right, and to the

# middle vertically

Move w+5 50
```

```
# Move window up 10 pixels, and so left edge

# is at x=40 pixels

Move 40p w-10p



# Move window to the mouse pointer location

Move m+0 m+0
```

MoveToDesk [prev | arg1 [arg2] [min max]]

移动窗口到另一个桌面（desktop）。参数和命令 GotoDesk 的参数相同。不使用任何参数时，窗口移动到当前桌面。MoveToDesk 命令取代了旧的 WindowDesk 命令。

MoveThreshold [pixels]

当用户在一个对象上按下鼠标按键时，fvwm 会等待并判断是一个单击操作还是一个拖拉操作。如果鼠标移动距离大于 pixels 像素，将被当作拖拉操作。

pixels 默认为 3，如果为负或为空，采用默认值。

MoveToPage [options] [x[p|w] y[p|w]] | [prev]

将窗口移动到page(x,y)。x 和 y 表示 X 轴和 Y 轴上 page 的位置坐标，左上角的 page 坐标为（0，0），右上角的 page 坐标为（M，0），M 值为 DeskTopSize 命令指定的水平 page 数目减去 1。类似，左下角 page 坐标为（0，N），右下角 page 坐标为（M，N）。负的 x 值/y 值表示相对于最有边/最下边的值。x 和 y 没有指定时，窗口移动到当前 page（一个拥有焦点，但是在屏幕外（off-screen）的窗口可以通过这种方式找回）。后缀‘p’时将窗口移动到相对于当前页的 page，后缀‘w’时表示相对当前位置移动窗口，单独的参数 prev 移动窗口到前一页。

通常不能移动窗口超出桌面边界限制。

选项 options 可能是 wrapx 和 wrapy，适用于移动窗口时超出桌面边界的情况。比如，使用 wrapx 时，如果窗口超出了桌面边界，则会从桌面左边界重新算起。选项 nodesklimitx 和 nodesklimity 允许移动窗口到桌面之外。

例子：

```
# Move window to page (2,3)

MoveToPage 2 3



# Move window to lowest and rightmost page

MoveToPage -1 -1
```

```
         # Move window to last page visited

         MoveToPage prev


         # Move window two pages to the right and one

         # page up, wrap at desk boundaries

         MoveToPage wrapx wrapy +2p -1p
```

`MoveToScreen [screen]`

移动窗口到另一个 Xinerama 屏幕上。screen 参数为‘p’时表示第一个屏幕（primary screen），‘c’表示当前屏幕（包含鼠标指针），‘g’表示全局屏幕，或屏幕号（从 0 开始计数）。

`OpaqueMoveSize [percentage]`

`PlaceAgain [Anim] [Icon]`

`Raise`

仅在窗口所在的层提升窗口。为了使窗口处于最顶层的最上边，可以使用

```
         AddToFunc raise-to-top

          + I Layer 0 ontop

          + I Raise
```

上面的 ontop 是使用的最高层。

`RaiseLower`

提升或降低窗口。如果窗口被其它窗口覆盖，将会提升它，否则会降低它。

`Resize [[frame] [direction dir [warptoborder]] [fixeddirection] [w]width[p|c] [w]height[p|c]] | [bottomright | br x y]`

`ResizeMaximize [resize-arguments]`

合并了 Resize 和 Maximize 两个命令的作用。resize-arguments 和 Reise 命令的参数一样。

ResizeMove resize-arguments move-arguments

ResizeMoveMaximize resize-arguments move-arguments

RestackTransients

SetAnimation milliseconds-delay [fractions-to-move-list]

SnapAttraction [proximity [behavior] [Screen]]

SnapGrid [x-grid-size y-grid-size]

WindowsDesk arg1 [arg2]

已经废除，必须使用 MoveToDesk 代替，注意，它们的语法已经改变，不能仅仅简单的替换命令名称。

XorPixmap [pixmap]

XorValue [number]

Fvwm 中文手册-FVWM（十四）

COMMANDS FOR FOCUS AND MOUSE MOVEMENT

CursorMove horizontal[p] vertical[p]

移动鼠标指针。全部参数或两个参数中的一个都可以为负值。参数值表示的是 page 的百分比。

```
CursorMove 100 100
```

表示移动到 page 的右下角。

```
CursorMove 50 25
```

表示沿 x 方向移动 50%page，沿垂直方向移动 25%page。后缀‘p’时表示移动的像素值。例如

```
CursorMove -10p -10p
```

表示沿 x 轴和 y 轴的相反方向各移动 10 个像素。弹出菜单不能调用这个命令。

FlipFocus [NoWarp]

执行 Focus 命令，好像用户使用指针选择了那个窗口。这个命令和单击一个窗口获得焦点一样都会改变 WindowList 的顺序，也就是说，目标窗口从 WindowList 里删除并放在开始的位置。推荐和 Direction 命令一起使用。

```
Focus [NoWarp]
```

设置键盘焦点到所选择的窗口。如果指定 NoWarp 参数，则只做这么多，否则，它还要移动窗口使它可见。它并不会自动提升窗口。不要将指针 wrap 进这个窗口。不要反图标化。这个命令不改变 WindowList 的顺序，它循环 WindowList 使这个窗口出现在列表的开始。

指定 NoWarp 参数时，不能将焦点传送到其它桌面上的窗口。

为了使用 Focus 或 FlipFocus 时，提升（raise）和/或 wrap 指针到一个窗口，使用类似下面的函数：

```
AddToFunc SelectWindow

+ I Focus

+ I Iconify false

+ I Raise

+ I WarpToWindow 50 8p

WarpToWindow x[p] y[p]
```

```
COMMANDS CONTROLLING WINDOW STATE

Close
```

这个命令发送关闭消息给窗口。如果窗口不能理解，则被销毁，像 Destroy 命令做的一样。注意，如果窗口接受了这个消息但没有响应，则窗口不被删除。

```
Delete
```

发送删除消息给窗口，常引发应用退出。

```
Destroy
```

销毁一个应用窗口，常引发应用崩溃。

```
Iconify [bool]
```

图标化窗口。bool 是布尔参数。"True"表示仅仅允许图标化，"False"表示仅允许反图示化。"toggle"在两种状态间切换。

有很多 Style 选项影响图标的外观和行为（例如，StickyIcon，NoIcon）。

```
Maximize [flags] [bool] [horizontal[p]] [vertical [p]]
```

不使用参数时，Maximize 使窗口在全屏大小和正常大小间切换。bool 为"True"或"False"时可以强迫窗口

最大化或正常尺寸。

horizontal 和 vertical 表示屏幕的百分比，用户可以控制窗口的新尺寸。后缀'p'表示采用像素值。horizontal 大于 0，则窗口的水平尺寸为 horizontal*screen_width/100。如果小于 0，则用屏幕宽度去减，例如-25 和 75 同样效果。

Recapture

已经废除，不应再使用。

RecaptureWindow

已经废除，不应再使用。

Refresh

刷新屏幕上的所有窗口。所有窗口待处理的 styles 和 looks 立即生效。

RefreshWindow

刷新目标窗口。这个窗口待处理的 styles 和 looks 立即生效。

Stick [bool]

如果 bool 为空或"toggle"，Stick 命令使窗口 sticky 或 non-sticky。"True"使窗口 sticky，不管它当前状态是什么。"False"相反。

StickAcrossPages [bool]

StickAcrossDesks [bool]

WindowShade [bool] | [[ShadeAgain] direction]

WindowShadeAnimate [steps[p]]

已经废除，请使用 Style 命令的 WindowShadeSteps 选项替代。

Fvwm 中文手册-FVWM（十五）

COMMANDS FOR MOUSE, KEY AND STROKE BINDINGS

IgnoreModifiers [Modifiers]

指定进行鼠标键盘绑定时，哪个修饰符将被忽略。IgnoreModifiers 也影响 ClickToFocus 风格。最好在配置文件里使用它，如果你的 fvwm 会话已经正在运行，执行这个命令所造成的结果是不可预知的。它应该在一些

应用或模块启动前使用。

Modifiers 和 Mouse 或 Key 绑定的语法相同，另外还有‘L’表示大写锁定键，默认是‘L’。Modifiers 为空，表示不忽略任何修饰符。在 num-lock 和 scroll-lock 键妨碍了你的快捷方式的时候，这个命令会派得上用场。在 XFree86 下，‘2’通常表示 num-lock 修饰符，‘5’表示 scroll-lock 修饰符。为了关闭所有这些烦人得修饰符，你可以使用下面得命令：

```
IgnoreModifiers L25
```

如果 Modifiers 参数是字符串“default”，则使用默认值“L”。

注意：这个命令创造了大量额外的网络流量，依赖于你得 CPU，网络连接，配置文件里 Key 或 Mouse 命令得数量，和你希望忽略的修饰符的数量。如果你没有一个速度快的机子或有比较多的绑定，你不应该忽略两个以上得修饰符。比如，如果 scroll-lock 没有什么问题，就不要忽略它。在 FAQ 里，你可以发现一个更好得解决方法。

```
EdgeCommand [direction [Function]]
```

将指定的 fvwm 命令 Function 与屏幕的边缘（edge）绑定。direction 可以是“North”，“Top”，“West”，“Left”，“South”，“Bottom”，“Right”和“East”。如果 Function 为空，绑定被删除。如果不带参数执行 EdgeCommand，则所有边缘的绑定被删除。

Function 当指针进入围绕可见屏幕的 the invisible pan frames 时执行。仅当 EdgeThickness 的值大于 0 时，这个绑定有效。如果一个函数绑定到一个边上，EdgeScroll 针对这个边指定的滚动被禁止。将函数绑定到一些边上，则另外在另外一些边上执行滚动是可以的。当指针进入一个边时，这个命令有意提升或降低某些窗口。FvwmAuto 可以用来获得提升或降低窗口的延迟。下面的例子里，指针进入屏幕 top 边时，提升 FvwmButtons：

```
# Disable EdgeScrolling but make it possible

# to move windows over the screen edge

EdgeResistance 10000 20



# Set thickness of the edge of the screen to 1

EdgeThickness 1



# Give focus to FvwmButtons if the mouse

# hits top edge

EdgeCommand Top Next (FvwmButtons) Focus

# Make sure the Next command matches the window

Style FvwmButtons CirculateHit
```

```
Module FvwmButtons

Module FvwmAuto 100 \

   "Silent AutoRaiseFunction" \

   "Silent AutoLowerFunction"


# If any window except FvwmButtons has

# focus when calling this function

# FvwmButtons are lowered

DestroyFunc AutoLowerFunction

AddToFunc AutoLowerFunction

+ I Current (!FvwmButtons) \

   All (FvwmButtons) Lower


# If FvwmButtons has focus when calling \

# this function raise it

DestroyFunc AutoRaiseFunction

AddToFunc AutoRaiseFunction

+ I Current (FvwmButtons) Raise
```

Normally, the invisible pan frames are only on the screen edges that border virtual pages。如果屏幕边缘绑定有命令，the pan frame is always created on that edge。

EdgeLeaveCommand [direction [Function]]

将指定的 fvwm 命令 Function 与屏幕的边缘（edge）绑定。direction 可以是“North”，“Top”，“West”，“Left”，“South”，“Bottom”，“Right”和“East”。如果 Function 为空，绑定被删除。如果不带参数执行 EdgeLeaveCommand，则所有边缘的绑定被删除。

Function 当指针进入围绕可见屏幕的 the invisible pan frames 时执行。仅当 EdgeThickness 的值大于 0 时，这个绑定有效。如果一个函数绑定到一个边上，EdgeScroll 针对这个边指定的滚动被禁止。将函数绑定到一些边上，则另外在另外一些边上执行滚动是可以的。当指针进入一个边时，这个命令有意提升或降低某些窗口。FvwmAuto 可以用来获得提升或降低窗口的延迟。参看 EdgeCommand 的例子。

Normally, the invisible pan frames are only on the screen edges that border virtual pages。如果屏幕边绑定有命令，the pan frame is always created on that edge。

GnomeButton

与 Mouse 命令一起将根窗口上的鼠标按键按下（presses）操作传递到一个 GNOME 程序（例如 GMC）。下面的例子传递鼠标按键 1 和 3 的 presses 到这样的程序：

```
Mouse 1 R A GnomeButton

Mouse 3 R A GnomeButton
```

```
Key [(window)] Keyname Context Modifiers Function
```

将键盘按键操作和 fvwm 命令绑定，如果 Function 为‘-’，则绑定删除。语法和 Mouse 绑定相同，除了鼠标按键数被 Keyname 取代。通常，键盘绑定当按键按下时被激活。Keyname 是一个标准的 X11 按键名，在/usr/include/X11/keysymdef.h 里定义，（不带 XK_前缀），或者在 keysym 数据库/usr/X11R6/lib/X11/XKeysymDB 里定义。Only key names that are generated with no modifier keys or with just the Shift key held are guaranteed to work。Context 和 Modifiers 域和 Mouse 绑定里定义的相同。然而当你按下按键时，context 窗口是有键盘焦点的窗口。不需要与指针下面的窗口相同（带有 SloppyFocus 或 ClickToFocus）。注意，在‘R’上下文上的键盘绑定和 SloppyFocus 和 ClickToFocus 一起并不会很好的工作。如果你遇到某些问题，使用 PointerKey 命令替换。

下面的例子绑定内置的（built-in）窗口列表到 Alt-Ctrl-Shift-F11 序列，无论指针在什么地方：

```
Key F11 A SCM WindowList
```

细节参考 Mouse 命令。

```
Mouse [(window)] Button Context Modifiers Function
```

```
PointerKey [(window)] Keyname Context Modifiers Function
```

工作方式和 Key 命令类同，唯一的区别是在指针下面的窗口上执行绑定操作。通常是在焦点所在的窗口执行绑定操作。如果你正在使用 SloppyFocus 或 ClickToFocus，PointerKey 命令可以用来绑定按键到根窗口。然而一些应用不能够处理这个按键，即使指针在 xterm 窗口上面。推荐仅为不必在任何一个应用窗口内的按键组合使用 PointerKey 命令。

例如：

```
Style * SloppyFocus

PointerKey f1 a m Menu MainMenu
```

```
Stroke [(window)] Sequence Button Context Modifiers Function
```

```
StrokeFunc [Options]
```

Fvwm 中文手册-FVWM（十六）

```
THE STYLE COMMAND (CONTROLLING WINDOW STYLES)
```

```
FocusStyle stylename options
```

工作方式类似 Style 命令，但仅接受与以“FP”为前缀的风格相关的焦点策略。“FP”前缀可以省略，但要付出少量时间的代价。FocusStyle 可以增加配置文件的可读性。例如：

```
FocusStyle * EnterToFocus, !LeaveToUnfocus
```

相当于

```
Style * FPEnterToFocus, !FPLeaveToUnfocus
```

DestroyStyle style

删除风格 style，改变立即生效。注意，style 不是一个 wild-carded 搜索字符串，而是一个 exactly match the original Style command 的 case-sensitive 字符串。

销毁风格“*”是可以的，但并不推荐这么做。例如：

```
DestroyStyle Application*
```

将删除名为“Application*”的风格的所有设置，而不是以“Application”开头的所有风格。

DestroyWindowStyle

删除 WindowStyle 命令在所选择的窗口上指定的风格，改变立即生效。

UpdateStyles

所有窗口待处理的 styles 和 looks 请求立即生效。

Style stylename options ...

Style 命令取代了 fvwm 旧版本里的全局命令 NoBorder, NoTitle, StartsOnDesk, Sticky, StaysOnTop, Icon, WindowListSkip, CirculateSkip, SuppressIcons, BoundaryWidth, NoBoundaryWidth, StdForeColor, and StdBackColor。它用来设置窗口的属性，和设置窗口管理器的默认风格。

stylename 可以是窗口的名称，class，或 resource string，可以包含‘*’和‘?’通配符。可以在单个 Style 命令里包含多个 options，它们从左到右依次读取，好像在使用多个单一的 Style 命令进行定义。如果新定义的风格与之前定义的有所冲突，则覆盖先前的定义。

options 是一个逗号分隔的包含下面关键词中的一个或更多个的列表。每一组使用'/'间隔，其中的最后一个为默认。BorderWidth, HandleWidth, !Icon / Icon, MiniIcon, IconBox, IconGrid, IconFill, IconSize, !Title / Title, TitleAtBottom / TitleAtLeft / TitleAtRight / TitleAtTop, LeftTitleRotatedCW / LeftTitleRotatedCCW, RightTitleRotatedCCW / RightTitleRotatedCW, TopTitleRotated / TopTitleNotRotated, BottomTitleRotated / BottomTitleNotRotated, !UseTitleDecorRotation / UseTitleDecorRotation, StippledTitle / !StippledTitle, StippledIconTitle / !StippledIconTitle, IndexedWindowName / ExactWindowName, IndexedIconName / ExactIconName, !Borders / Borders, !Handles / Handles, WindowListSkip / WindowListHit, CirculateSkip / CirculateHit, CirculateSkipShaded / CirculateHitShaded, CirculateSkipIcon / CirculateHitIcon, Layer, StaysOnTop / StaysOnBottom / StaysPut, Sticky / Slippery, StickyAcrossPages / !StickyAcrossPages, StickyAcrossDesks / !StickyAcrossDesks, ! StickyStippledTitle / StickyStippledTitle, !StickyStippledIconTitle / StickyStippledIconTitle, StartIconic / StartNormal, Color, ForeColor, BackColor,

Colorset, HilightFore, HilightBack, HilightColorset, BorderColorset,
HilightBorderColorset, IconTitleColorset, HilightIconTitleColorset,
IconBackgroundColorset, IconTitleRelief, IconBackgroundRelief,
IconBackgroundPadding, Font, IconFont, StartsOnDesk / StartsOnPage /
StartsAnyWhere, StartsOnScreen, ManualPlacementHonorsStartsOnPage /
ManualPlacementIgnoresStartsOnPage, CaptureHonorsStartsOnPage /
CaptureIgnoresStartsOnPage, RecaptureHonorsStartsOnPage /
RecaptureIgnoresStartsOnPage, StartsOnPageIncludesTransients /
StartsOnPageIgnoresTransients, IconTitle / !IconTitle, MwmButtons / FvwmButtons,
MwmBorder / FvwmBorder, MwmDecor / !DecorHint, MwmFunctions / !FuncHint,
HintOverride / !Override, !Button / Button, ResizeHintOverride / !
ResizeHintOverride, OLDecor / !OLDecor, GNOMEUseHints / GNOMEIgnoreHints,
StickyIcon / SlipperyIcon, StickyAcrossPagesIcon / !StickyAcrossPagesIcon,
StickyAcrossDesksIcon / !StickyAcrossDesksIcon, ManualPlacement /
CascadePlacement / MinOverlapPlacement / MinOverlapPercentPlacement /
TileManualPlacement / TileCascadePlacement / CenterPlacement / UnderMousePlacement,
MinOverlapPlacementPenalties, MinOverlapPercentPlacementPenalties,
DecorateTransient / NakedTransient, DontRaiseTransient / RaiseTransient,
DontLowerTransient / LowerTransient, DontStackTransientParent /
StackTransientParent, SkipMapping / ShowMapping, ScatterWindowGroups /
KeepWindowGroupsOnDesk, UseDecor, UseStyle, !UsePPosition / NoPPosition /
UsePPosition, !UseUSPosition / NoUSPosition / UseUSPosition, !UseTransientPPosition
/ NoTransientPPosition / UseTransientPPosition, !UseTransientUSPosition /
NoTransientUSPosition / UseTransientUSPosition, !UseIconPosition / NoIconPosition /
UseIconPosition, Lenience / !Lenience, ClickToFocus / SloppyFocus / MouseFocus|
FocusFollowsMouse / NeverFocus, ClickToFocusPassesClickOff /
ClickToFocusPassesClick, ClickToFocusRaisesOff / ClickToFocusRaises,
MouseFocusClickRaises / MouseFocusClickRaisesOff, GrabFocus / GrabFocusOff,
GrabFocusTransientOff / GrabFocusTransient, FPFocusClickButtons,
FPFocusClickModifiers, !FPSortWindowlistByFocus / FPSortWindowlistByFocus,
FPClickRaisesFocused / !FPClickRaisesFocused, FPClickDecorRaisesFocused / !
FPClickDecorRaisesFocused, FPClickIconRaisesFocused / !FPClickIconRaisesFocused, !
FPClickRaisesUnfocused / FPClickRaisesUnfocused, FPClickDecorRaisesUnfocused / !
FPClickDecorRaisesUnfocused, FPClickIconRaisesUnfocused / !
FPClickIconRaisesUnfocused, FPClickToFocus / !FPClickToFocus, FPClickDecorToFocus /
!FPClickDecorToFocus, FPClickIconToFocus / !FPClickIconToFocus, !FPEnterToFocus /
FPEnterToFocus, !FPLeaveToUnfocus / FPLeaveToUnfocus, !FPFocusByProgram /
FPFocusByProgram, !FPFocusByFunction / FPFocusByFunction,
FPFocusByFunctionWarpPointer / !FPFocusByFunctionWarpPointer, FPLenient / !
FPLenient, !FPPassFocusClick / FPPassFocusClick, !FPPassRaiseClick /
FPPassRaiseClick, FPIgnoreFocusClickMotion / !FPIgnoreFocusClickMotion,
FPIgnoreRaiseClickMotion / !FPIgnoreRaiseClickMotion, !FPAllowFocusClickFunction /
FPAllowFocusClickFunction, !FPAllowRaiseClickFunction / FPAllowRaiseClickFunction,
FPGrabFocus / !FPGrabFocus, !FPGrabFocusTransient / FPGrabFocusTransient,
FPOverrideGrabFocus / !FPOverrideGrabFocus, FPReleaseFocus / !FPReleaseFocus, !
FPReleaseFocusTransient / FPReleaseFocusTransient, FPOverrideReleaseFocus / !
FPOverrideReleaseFocus, StartsLowered / StartsRaised, IgnoreRestack / AllowRestack,
FixedPosition / VariablePosition, FixedUSPosition / VariableUSPosition,
FixedPPosition / VariablePPosition, FixedSize / VariableSize, FixedUSSize /
VariableUSSize, FixedPSize / VariablePSize, !Closable / Closable, !Iconifiable /
Iconifiable, !Maximizable / Maximizable, !AllowMaximizeFixedSize /
AllowMaximizeFixedSize, IconOverride / NoIconOverride / NoActiveIconOverride,
DepressableBorder / FirmBorder, MaxWindowSize, IconifyWindowGroups /
IconifyWindowGroupsOff, ResizeOpaque / ResizeOutline, BackingStore /

BackingStoreOff / BackingStoreWindowDefault, Opacity / ParentalRelativity, SaveUnder / SaveUnderOff, WindowShadeShrinks / WindowShadeScrolls, WindowShadeSteps, WindowShadeAlwaysLazy / WindowShadeBusy / WindowShadeLazy, EWMHDonateIcon / EWMHDontDonateIcon, EWMHDonateMiniIcon / EWMHDontDonateMiniIcon, EWMHMiniIconOverride / EWMHNoMiniIconOverride, EWMHUseStackingOrderHints / EWMHIgnoreStackingOrderHints, EWMHIgnoreStateHints / EWMHUseStateHints, EWMHIgnoreStrutHints / EWMHUseStrutHints, EWMHIgnoreWindowType / ! EWMHIgnoreWindowType, EWMHMaximizeIgnoreWorkingArea / EWMHMaximizeUseWorkingArea / EWMHMaximizeUseDynamicWorkingArea, EWMHPlacementIgnoreWorkingArea / EWMHPlacementUseWorkingArea / EWMHPlacementUseDynamicWorkingArea, MoveByProgramMethod, Unmanaged, State。

Focus policy

ClickToFocus 表示单击窗口时，窗口获得焦点。MouseFocus （别名为 FocusFollowsMouse）表示指针进入窗口时，窗口获得焦点，当指针离开窗口时，窗口失去焦点。SloppyFocus 类似，但当指针离开窗口进入根窗口，或者进入一个 ClickToFocus 窗口时（除非单击它），原来得窗口并不会失去焦点。NeverFocus 风格得窗口从不获得焦点，常用于 FvwmButton 这样得模块。

上面得焦点模型可以被几个额外得选项扩展，在 2.5.3 之后得版本里，有很多以“FP”或“!FP”开始得选项，这些选项将在某一天取代旧得选项。但是使用它们将会限制向后得兼容性。通常来说，以“FP”开始得选项打开一个特点，以“!FP”开始得选项表示关闭一个特点。

聚焦窗口 (Focusing the window)

FPEnterToFocus，指针进入窗口时，该窗口收到焦点。

FPLeaveToUnfocus，指针离开窗口时，该窗口失去焦点。

FPClickToFocus, FPClickDecorToFocus 或 FPClickIconToFocus，单击窗口内部、修饰、图标时，该窗口收到焦点。

FPFocusByProgram，允许窗口自己获取焦点。

!FPFocusByFunction，禁止窗口通过 Focus 和 FlipFocus 命令获得焦点。

FPFocusByFunctionWarpPointer，使用 Focus 命令时，指针是否 wrap 到被选择得窗口。

FPLenient，允许聚焦一个不希望获得焦点得窗口。比如 FvwmPager 或 xclock。

FPFocusClickButtons，

FPFocusClickModifiers，

FPPassFocusClick，

FPAllowFocusClickFunction，

FPIgnoreFocusClickMotion，

FPSortWindowlistByFocus 和 !FPSortWindowlistByFocus，

单击提升窗口 (Clicking the window to raise)

FPClickRaisesFocused, FPClickDecorRaisesFocused 和 FPClickIconRaisesFocused ,

FPClickRaisesUnfocused, FPClickDecorRaisesUnfocused 和 FPClickIconRaisesUnfocused ,

FPPassRaiseClick ,

FPAllowRaiseClickFunction ,

FPIgnoreRaiseClickMotion ,

新窗口创建时获取焦点 (Grabbing the focus when a new window is created)

FPGrabFocus 或 FPGrabFocusTransient ,

OverrideGrabFocus ,

FPReleaseFocus, FPReleaseFocusTransient 和 FPOverrideReleaseFocus ,

ClickToFocusPassesClickOff 和 ClickToFocusPassesClick ,

ClickToFocusRaisesOff/MouseFocusClickRaisesOff 和 ClickToFocusRaises/MouseFocusClickRaises ,

Fvwm 中文手册-FvwmAnimate

FvwmAnimate

名称 (NAME) :

FvwmAnimate -Fvwm 动画制作模块

概要 (SYNOPSIS) :

Module FvwmAnimate [ModuleAlias]

FvwmAnimate 只能被 fvwm 调用(fork),不能从命令行启动。

有两种启动 FvwmAnimate 的方式，在.fvwm2rc 文件中增加下面的语句：

Module FvwmAnimate

或者可以使用弹出菜单 (pop-up menu) :

DestroyMenu Module-Popup

AddToMenu Module-Popup "Modules" Title

+ "Fvwm Animate Icons" Module FvwmAnimate ModuleAlias

描述 (DESCRIPTION) :

FvwmAnimate 模块能够产生图示化 (iconfy) 或反图示化 (de-iconfy) 时的动画效果,目前有 6 种效果可

供选择。

调用（INVOCATION）：

FvwmAnimate 只能被 fvwm 窗口管理器启动。当启动语句中含有 OptionalName 参数时， 使用 ModuleAlias 而不是 FvwmAnimate 来查找配置命令，配置文件，以及默认生成的菜单和窗体（form）。启动的过程中 FvwmAnimate 定义配置和控制 FvwmAnimate 的菜单和窗体。默认的菜单名是 "MenuFvwmAnimate" ，窗体名是"FormFvwmAnimate"。如果 OptionalName 参数被指定,默认的菜单名将变为 "Menu<ModuleAlias>" ，而窗体将变为 "form<ModuleAlias>" 。

如果你已经创建了一个菜单 "Module-Popup" ,你可以通过如下的方式使用 FvwmAnimate：

```
AddToFunc "StartFunction" "I" Module FvwmAnimate

AddToMenu "Module-Popup" "Control Animation" Popup MenuFvwmAnimate
```

配置选项（CONFIGURATION OPTIONS）：

因为弹出菜单 "MenuFvwmAnimate" 已经可以完全控制 FvwmAnimate 模块，你并不需要详细了解下面这些配置选项。

FvwmAnimate 从 fvwm 的模块配置数据库（参看 fvwm（1）的 MODULE COMMANDS 部分）获得配置信息，此外，还读取$HOME/.FvwmAnimate 文件，并且运行时可以接受来自 fvwm 以及其它模块的指令。

```
*FvwmAnimate: Color color
```

指定 FvwmAnimate 所使用的颜色。依赖于你所使用的显示模式，所达到的效果将会不同。尤其在 8-bit 显示时，如果是纯色的背景，它将有所帮助。你必须通过试验来了解它如何工作。

```
*FvwmAnimate: Pixmap pixmap
```

指定 FvwmAnimate 所显示的 pixmap，如果 *FvwmAnimate: Color 达不到一个比较好的效果，它将非常有用。

```
*FvwmAnimate: Delay msecs
```

告诉 FvwmAnimate 两个动画帧之间的时间间隔，单位是毫秒。

```
*FvwmAnimate: Iterations iterations
```

告诉 FvwmAnimate 动画有多少 step。

```
*FvwmAnimate: Twist twist
```

告诉 FvwmAnimate 窗口图示化（iconfy）时的旋转次数。

```
*FvwmAnimate: Width width
```

告诉 FvwmAnimate 画行时的宽度，默认值 0，表示宽度为 1 的 fast line

```
*FvwmAnimate: Effect mode
```

指定 FvwmAnimate 使用的动画效果，目前可供选择的效果有：Frame, Lines, Flip, Turn, Zoom3D, Twist Random, and None。在配置文件里，通常设置为 None 来防止 FvwmAnimate 自动启动。

*FvwmAnimate: Stop

关闭动画效果。

*FvwmAnimate: Save

保存当前的配置到 ".FvwmAnimate" 文件，该文件在 FvwmAnimate 启动期间自动被读取。

命令 (COMMANDS) ：

可以使用 "SendToModule" 命令要求 FvwmAnimate 产生动画,命令的格式是：

SendToModule FvwmAnimate animate sx sy sw sh dx dy dw dh

第二个字段一定要和 FvwmAnimate 启动时的名称相匹配。animate 后的 8 个字段必须是数字。前 4 个为动画的起始位置，后四个为动画的目的位置，组成两个长方形。每组数字的前两个表示长方形右上角的位置，后两个是宽度和高度。

此外可以使用的命令有：pause, play, push, pop，reset。

pause 使模块暂时无效，不产生任何动画效果。play 使模块重新运行。push 保存当前的状态，pop 来恢复它。reset 删除所有保存的状态并开始播放。

如果你不希望等候所有 40 个 xterm 逐个产生动画，你可以使用如下命令：

SendToModule FvwmAnimate pause

All (XTerm) Iconify on

如果你不想因此破坏当前的状态，可以如下操作：

SendToModule FvwmAnimate push pause

All (XTerm) Iconify on

SendToModule FvwmAnimate pop

Fvwm 中文手册-FvwmAuto

FvwmAuto

名称 (NAME) ：

FvwmAuto -Fvwm 自动提升 (auto-raise) 模块。

概要 (SYNOPSIS) ：

FvwmAuto 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) ：

FvwmAuto 常常用来自动提升焦点窗口。

调用 (INVOCATION) ：

```
Module FvwmAuto Timeout [-passid] [-menter|-menterleave|-mfocus] [EnterCommand
[LeaveCommand]]

AddToMenu Modules

+ "Auto Raise (300 ms)"  Module FvwmAuto 300

+ "Auto Raise/Lower"     Module FvwmAuto 300 "Silent Raise" "Silent Lower"
```

Timeout 参数是必须的，单位为毫秒，任何一个大于 0 的正整数都是有效的。它指定在这个命令执行前，窗口必须保留输入焦点多长时间。

如果指定 -passid 选项，刚刚进入或离开的窗口的 id 被附加到这个命令上发送给 fvwm，它能被 fvwm 的 WindowId 命令使用。

选项-menter，-menterleave，和-mfocus 只能使用一个，如果是-mfocus 模式，FvwmAuto 提升焦点窗口。如果是-menter 模式，当鼠标指针进入一个窗口时，FvwmAuto 提升指针下面的窗口。在鼠标指针进入新窗口前，LeaveCommand 在指针下面的窗口上执行。当指针离开一个窗口并进入根窗口时，EnterCommand 也被执行，但没有窗口供操作。-menterleave 模式的工作方式类似于-menter 模式，但是在鼠标指针离开一个窗口而且还没有进入一个新的窗口时，LeaveCommand 也被执行。后面的两种模式当窗口不接受焦点时非常有用。

注意：-menterleave 模式可以和一些应用的弹出窗口交互。一个例子是 Ghostview 的 zoom 菜单，不用担心，这只是 Ghostview 的一个 bug。

EnterCommand 和 LeaveCommand 是可选的。EnterCommand 在窗口得到输入焦点 Timeout 毫秒后执行，LeaveCommand 在窗口失去焦点 Timeout 毫秒后执行。注意，你应该在命令前一直使用'Silent'关键字。如果你忘记它，fvwm 会自动把"Silent"加在这个命令上。如果没有添加这个前缀，在命令被 fvwm 处理前窗口已经死掉时，fvwm 将会向你请求一个窗口来进行操作。

EnterCommand 默认为"Silent Raise"，但其它一些 fvwm 函数也是可以的。可以进行下面的试验：

```
    Module FvwmAuto 0 Nop "Silent Lower"

    Module FvwmAuto 0 Nop "Silent Iconify"
```

下面是自动提升 ClickToFocus 窗口的例子：

```
    Style * ClickToFocus

    FvwmAuto 0 -menter "Silent Raise"
```

下面是自动提升和降低部分窗口的例子：

启动 FvwmAuto：

```
    FvwmAuto 0 -passid -menter \

    "Silent selective_raiselower raise" \

    "Silent selective_raiselower lower"
```

加入.fvwm2rc：

```
AddToFunc selective_raiselower
+ I WindowId $1 (FvwmIconMan) $0
+ I WindowId $1 (FvwmButtons) $0
+ I WindowId $1 (xclock) $0
```

下面是一个更复杂的例子：（有三个 FvwmAuto 正在运行）

```
DestroyFunc RestoreIconified
AddToFunc   RestoreIconified
+ I Current (Iconic) Iconify false


DestroyFunc RegisterFocus
AddToFunc   RegisterFocus
+ I Exec date +"%T $n focused" >>/tmp/focus-stats.txt
DestroyFunc RegisterUnfocus
AddToFunc   RegisterUnfocus
+ I Exec date +"%T $n unfocused" >>/tmp/focus-stats.txt
KillModule FvwmAuto
Module FvwmAuto 250 Raise Nop
Module FvwmAuto 800 RestoreIconified Nop
Module FvwmAuto   0 RegisterFocus RegisterUnfocus
```

Fvwm 中文手册-FvwmBacker

FvwmBacker

名称（NAME）：

FvwmBacker -FVWM 背景变换模块

概要（SYNOPSIS）：

FvwmBacker 只能被 fvwm 调用(fork),不能从命令行启动。

描述（DESCRIPTION）：

FvwmBacker 模块主要实现在切换桌面的同时改变背景。任何一个命令都可以用来改变背景。实际上，任意的命令都能够利用这个模块发送给 fvwm 执行，我们可以利用这个特点，在切换桌面的同时改变窗口边框颜色等。

初始化（INITIALIZATION）：

初始化期间，FvwmBacker 从 fvwm 的模块配置数据库里获取配置信息。

调用（INVOCATION）：

可以通过在~/.fvwm2rc 文件加入下面的语句

```
AddToFunc StartFunction I Module FvwmBacker
```

进而在 fvwm 初始化期间启动 FvwmBacker.

如果 fvwm 正在运行，可以使用‘Module FvwmBacker’命令启动 FvwmBacker，和使用‘KillModule FvwmBacker’命令停止 FvwmBacker.

配置选项（CONFIGURATION OPTIONS）：

下面的选项可以在.fvwm2rc 文件里使用。

```
*FvwmBacker: Command (Desk d, Page x y) command
```

当视口和位置参数(Desk d, Page x y) 匹配时执行命令 command。这三个数字都可以使用‘*’替代表示任意值。

如果 Desk 或 Page 之一为空，即使另外一个匹配，command 命令也不执行。如果都为空，command 命令仅当模块启动时执行。与使用‘*’的情况不同：如果使用‘*’，命令将总是被执行。

如果 command 是-solid，FvwmBacker 使用接下来的一个参数作为颜色值，并且设置它为背景色（不使用 xsetroot 调用，只能使用单色）。

如果 comand 是 colorset，FvwmBacker 使用 coloerset n 指定的颜色集作为背景。

否则，command 将被发送到 fvwm 执行。

```
*FvwmBacker: RetainPixmap
```

```
*FvwmBacker: DoNotRetainPixmap
```

取消上一个选项实现的效果，这是默认的。

运行时配置（RUN-TIME CONFIGURATION）：

运行时改变 FvwmBacker 的配置是可以的（但是删除目前的配置行是不可能的）。有很多方法可以实现，比如创建一个 fvwm 函数，或者使用模块 FvwmCommand 或 FvwmConsole 中的一个。

例如：

```
DestroyModuleConfig FvwmBacker*
```

```
    *FvwmBacker: Command (Desk 0) -solid black

    *FvwmBacker: Command (Desk 1) -solid blue
```

配置示例 (SAMPLE CONFIGURATION) :

下面的例子引用自从一个.fvwm2rc 文件

```
    ####

    # Set Up Backgrounds for different desktop pages (2 desks, 3x2 pages).

    ####

    *FvwmBacker: Command (Page 2 *) -solid steelblue

    *FvwmBacker: Command (Desk 0, Page 0 0) Exec fvwm-root $[HOME]/bg2.xpm

    *FvwmBacker: Command (Desk 0, Page 0 1) -solid midnightblue

    *FvwmBacker: Command (Desk 0, Page 1 *) -solid yellow

    *FvwmBacker: Command (Desk 1, Page * 0) -solid navy

    *FvwmBacker: Command (Desk 1, Page * 1) Colorset 5
```

Fvwm 中文手册-FvwmBanner

FvwmBanner

名称 (NAME) :

FvwmBanner -FVWM 徽标模块

概要 (SYNOPSIS) :

FvwmBanner 只能被 fvwm 启动(fork),不能从命令行启动。

描述 (DESCRIPTION) :

FvwmBanner 在屏幕中心显示一个 Fvwm Logo 3 秒钟。

调用 (INVOCATION) :

FvwmBanner 可以使用命令"Module FvwmBanner"启动。可以在 .fvwm2rc 文件的 menu 或 key-stroke 部分绑定这个命令。但是更多情况下是在 StartFunction 或 InitFunction 函数里添加这个命令，比如：

```
    AddToFunc InitFunction "I" Module FvwmBanner
```

我们可以为这个命令指定一个文件参数，比如"FvwmBanner doomface.xpm"。或者通过配置选项（见下面的"*FvwmBanner: Pixmap")指定一个默认的图片。 Fvwm 将在 ImagePath 里寻找这副图片。当然，我们也可以使用图片的绝对路径。

配置选项（CONFIGURATION OPTIONS）：

*FvwmBanner: NoDecor

告诉 FvwmBanner 创建一个不受 Fvwm 所管理和装饰的窗口。

*FvwmBanner: Pixmap file

指定 FvwmBanner 显示 file 指定的图像文件。

*FvwmBanner: Timeout sec

指定 FvwmBanner 显示徽标的时间长度。

Fvwm 中文手册-FvwmCommand

FvwmCommand

名称（NAME）：

FvwmCommand - FVWM 外部命令接口

概要（SYNOPSIS）：

FvwmCommand [-cmrvw] [-S name] [-i level] [-f name] [-F level] [command...]

描述（DESCRIPTION）：

FvwmCommand 使你能够监控 fvwm 的执行（fvwm transaction），并从 shell 命令行或者脚本文件发送 fvwm 命令。FvwmCommand 把每个参数都当作一个 fvwm 命令。如果命令包含空格，则必须加上引号，如

```
FvwmCommand 'FvwmPager 0 1'
```

调用（INVOCATION）：

在调用 FvwmCommand 之前，FvwmCommandS 需要被 fvwm 调用一次，通过.fvwm2rc 文件，菜单，或者 FvwmConsole 都可以。之后，就可以通过 shell 或脚本调用 FvwmCommand。

通过.fvwm2rc 文件调用 FvwmCommandS：

```
Module FvwmCommandS
```

或

```
AddToFunc StartFunction "I" Module FvwmCommandS
```

然后，通过脚本文件或 shell 调用 FvwmCommand：

```
FvwmCommand  'popup Utilities'
```

选项（OPTIONS）：

-c

通知 FvwmCommand 从标准输入读取多个命令，而不是命令行参数中指定的一个命令。它禁止了-m 或-i 选项。

```
(echo "Exec xload"; echo "Beep") | FvwmCommand -c
```

-F <level>

指定了 FvwmCommand 输出窗口标记（fvwm window flags）的层次。

0 没有窗口标记输出

1 如果信息层（information level），即-i 选项，为 2 或 3，则输出全部窗口标记，

-f <name>

指定一个和 server 通信的可选 FIFO。默认 FIFO 设置是/var/tmp/FvwmCommand-${DISPLAY}C。FvwmCommand..C 用来发送命令和 FvwmCommand。M 用来接受消息。如果这个路径不可用，将使用 ${FVWM_USERDIR}/FvwmCommand-${DISPLAY} 取代。必须在 FvwmCommand 调用前使用同样的<name>作为第一个参数调用 FvwmCommandS。可选的，选项-S 可以被使用。参考选项-S。

-i <level>

指定 FvwmCommand 输出的信息的层次。

0 仅显示错误信息。

```
FvwmCommand -i 0 FvwmBanner
```

将显示一个不带任何输出的徽标（banner）。

另一方面，

```
FvwmCommand -i 0 foobar
```

将返回

```
[FVWM][executeModule]: <<ERROR>> No such module

'foobar' in ModulePath '/usr/lib/X11/fvwm'
```

注意，Fvwm 本身将不返回任何一个错误信息，以避免象下面这样的情况，因为'windowid'自己是一个有效的命令。

```
FvwmCommand -i 0 `windowid foo bar'
```

1 显示错误和窗口配置信息。为默认设置。

```
FvwmCommand send_windowlist
```

将显示类似下面的信息。

```
0x02000014 window               FvwmConsole
```

```
0x02000014 icon                 FvwmConsole
0x02000014 class                XTerm
0x02000014 resource             FvwmConsole
0x01c00014 window               console
0x01c00014 icon                 console
0x01c00014 class                XTerm
0x01c00014 resource             console
0x01000003 window               Fvwm Pager
0x01000003 icon
0x01000003 class                FvwmModule
0x01000003 resource             FvwmPager
0x00c0002c window               emacs: FvwmCommand.man
0x00c0002c icon                 FvwmCommand.man
0x00c0002c icon file            xemacs.xpm
0x00c0002c class                Emacs
0x00c0002c resource             emacs
end windowlist
```

第一列表示窗口的 ID 号，它们可以用作'windowid'命令的参数。第二列表示信息类型。最后一列表示信息内容。如果没有信息返回，需要增加-w <time>或-r 选项，在负载很重的系统里这将很有帮助。

2 显示上面的信息和静态的窗口信息

```
FvwmCommand -i2 'FvwmPager 0 1'
```

将显示类似下面的信息。

```
0x03c00003 frame                x 962, y 743, width 187, height 114
0x03c00003 desktop              0
0x03c00003 StartIconic          no
0x03c00003 OnTop                yes
0x03c00003 Sticky               yes
0x03c00003 WindowListSkip       yes
```

```
0x03c00003 SuppressIcon         no
0x03c00003 NoiconTitle          no
0x03c00003 Lenience             no
0x03c00003 StickyIcon           no
0x03c00003 CirculateSkipIcon    no
0x03c00003 CirculateSkip        no
0x03c00003 ClickToFocus         no
0x03c00003 SloppyFocus          no
0x03c00003 SkipMapping          no
0x03c00003 Handles              no
0x03c00003 Title                no
0x03c00003 Mapped               no
0x03c00003 Iconified            no
0x03c00003 Transient            no
0x03c00003 Raised               no
0x03c00003 Visible              no
0x03c00003 IconOurs             no
0x03c00003 PixmapOurs           no
0x03c00003 ShapedIcon           no
0x03c00003 Maximized            no
0x03c00003 WmTakeFocus          no
0x03c00003 WmDeleteWindow       yes
0x03c00003 IconMoved            no
0x03c00003 IconUnmapped         no
0x03c00003 MapPending           no
0x03c00003 HintOverride         yes
0x03c00003 MWMButtons           no
```

```
0x03c00003 MWMBorders          no

0x03c00003 title height        0

0x03c00003 border width        4

0x03c00003 base size           width 8, height 7

0x03c00003 size increment      width 9, height 9

0x03c00003 min size            width 8, height 7

0x03c00003 max size            width 32767, height 32767

0x03c00003 gravity             SouthEast

0x03c00003 pixel               text 0xffffff, back 0x7f7f7f

0x03c00003 window              Fvwm Pager

0x03c00003 icon                Fvwm Pager

0x03c00003 class               FvwmModule

0x03c00003 resource            FvwmPager
```

3 显示所有可用的信息。

```
FvwmCommand -i3 'Killmodule Fvwm*'
```

将报告哪个窗口被关闭。

```
0x03400003 destroy

0x02400002 destroy
```

-m

监控 fvwm 的窗口信息处理。FvwmCommand 不断地输出它接受到的信息。这个选项能够用在后台任务里，常和-i3 选项一起动态控制窗口。

```
FvwmCommand -mi3 | grep 'iconify'
```

将报告窗口什么时候图标化（iconfy）或反图标化（de-iconfy）。

注意：FvwmCommand 不缓冲它的输出，但很多 grep 或 sed 这样的使用块缓冲的机制。下一个例子的输出将直到 FvwmCommand 被终止或 stdout 缓冲被填满时才显示。

```
FvwmCommand -mi3 | grep ' map' |

sed 's/\(0x[0-9a-f]*\).*/windowid \1 move 0 0/'
```

作为替代，可以使用缓冲控制工具，比如 pty 或 perl。下面是一个例子。

```
    Fvwm -mi3 | perl -ne '

    $|=1;

    print "windowid $1 iconify\n" if /^(0x\S+) add/;

    ' > ~/.FvwmCommandC
```

-r

退出前等待一个回复。在一个指定的时间之内，如果没有返回任何信息或错误，FvwmCommand 将退出。指定选项-r 时，将等候至少一条返回消息。收到第一条消息之后，它将在指定的时间限制内，等候另外一条返回消息。

-S <name>

-v

显示 FvwmCommand 的版本号。

-w <time>

等候一个消息<time>微妙。在一个指定的时间之内，如果没有返回任何信息或错误，FvwmCommand 将退出，除非-m 被使用。默认值为 500ms。

错误：

如果显示下面的错误消息，很有可能是没有运行 FvwmCommandS。

```
    FvwmCommand error in opening message fifo

    --No such file or directory--
```

fvwm 的模块不返回错误消息给 fvwm，而是显示在 stderr 上。这些错误消息将被当作 FvwmCommand 消息显示。

Fvwm 中文手册-FvwmConsole

FvwmConsole

名称（NAME）：

FvwmConsole - FVWM 命令输入接口

概要（SYNOPSIS）：

FvwmConsole 只能被 fvwm 调用(fork)，不能从命令行启动。

描述（DESCRIPTION）：

FvwmConsole 通过一个控制台终端交互式的接收用户输入的 fvwm 配置命令，并立即执行。可以利用它来测

试新的配置，或者暂时地改变 fvwm 的外观。

调用（INVOCATION）：

FvwmConsole 能够解析所有的 xterm 选项。

可以在.fvwm2rc 文件的初始化函数里使用命令‘Moudle FvwmConsole’启动 FvwmConsole，也可以将这个命令与菜单操作、鼠标操作或 key-stroke 绑定。

配置选项（CONFIGURATION OPTIONS）：

FvwmConsole 使用 xterm 作为自己的控制台。所有有关 xterm 的设置都被继承，除非使用类似下面的命令专门指定 FvwmConsole 使用的 xterm 风格。

```
Module FvwmConsole -g 40x10 -fg black -bg green3
```

可以通过-termainal 选项指定使用其它的终端作为自己的控制台，不过，该终端必须能够解析-name，-title 和-e 选项，例如

```
Module FvwmConsole -terminal rxvt
```

FvwmConsole 的早期版本支持使用-e 选项选择一个前端（font-end）。尽管仍然向后兼容这个选项，但是不建议使用它，除非你知道自己在做什么：

```
Module FvwmConsole -e FvwmConsoleC.pl
```

FvwmConsole 使用的 X 资源（X resources）可以在~/.Xdefaults 文件里设置：

```
FvwmConsole*VT100*geometry: 40x4

FvwmConsole*font: 7x14
```

命令编辑（COMMAND EDITING）：

如果 GNU readline 库是可用的，则可以使用它。

如果已经安装了 Perl5，则可以使用 FvwmConsole.pl 作为命令编辑器。有两种方式：可以复制 FvwmConsole.pl 到 fvwmlib 目录下面并重命名为 FvwmConsoleC；或者在调用 FvwmConsole 时使用-e 选项。例如：

```
Module FvwmConsole -e FvwmConsoleC.pl
```

如果都没有安装，则只有简单的读取功能，没有编辑能力。GNU readline 库和 FvwmConsoleC.pl 有一些通用的操作命令。这些命令类似于 emacs。更多细节请参考 GNU readline 手册和 FvwmConsoleC.pl 手册。

```
Ctrl-A

    - beginning of line

Ctrl-B

    - previous char

Ctrl-D
```

- delete char

Ctrl-E

- end of line

Ctrl-F

- next char

Ctrl-H

- backspace

Ctrl-K

- erase to the end of line

Ctrl-N

- next line

Ctrl-P

- previous line

Ctrl-R

- search reverse

Ctrl-U

- delete line

Meta-B

- previous word

Meta-F

- next word

Esc <

- beginning of history

Esc >

- end of history

退出 (EXITING) :

FvwmConsole 可以通过输入 EOF 字符（通常情况下是 CTRL-D）终止。

注意，不要在 FvwmConsole 控制台的命令行使用“quit”命令，它将导致 fvwm 完全退出。

Fvwm 中文手册-FvwmConsoleC.pl

FvwmConsoleC.pl

名称（NAME）：

FvwmConsoleC.pl - FvwmConsole 的命令编辑器

概要（SYNOPSIS）：

FvwmConsoleC.pl -e /usr/X11/lib/fvwm/FvwmConsole.pl

描述（DESCRIPTION）：

FvwmConsoleC.pl 提供命令编辑功能，编辑命令的方式类似 emacs。它也提供代换功能（substitution），在发送命令前，将命令里包含的模式（pattern）转换为一个字符串。

功能：

下面是可用的功能和相应的默认功能键。

bind

    Meta-k, Ctrl-x Ctrl-b

    列出功能键。

boh

    移动到第一个历史记录（history）。

boh_ign_mode Esc-<

    移动到第一个历史记录，如果处于搜索模式，继续。

bol

    Home, Ctrl-a

    移动光标到行首。

bs[(n)] BackSpace，Ctrl-h

    后退 n 次，n 默认为 1。

cancel

    Ctrl-x Ctrl-k

    取消当前输入。

del_back_line

删除开始到光标之间的行

del_back_word Ctrl-w

删除开始到光标之间的单词。

del_char [(n)] Delete, Ctrl-d

删除光标到结束之间的行

del_forw_word Meta-d

删除光标到结束之间的单词。

del_line Ctrl-u

删除完整的行。

enter

Enter, Ctrl-j, Ctrl-m

完成合适的替换（substitution），并发送当前行到Fvwm。

enter_wo_subst Meta-Enter

发送当前行到Fvwm，不进行替换（substitution）。

eoh

移动到历史记录的结尾。

eoh_ign_mode Esc->

移动到历史的结尾。如果是搜索模式，继续。

eol

End, Ctrl-e

移动光标到行尾。

ins_char (str)

在光标位置插入字符串。

ins_last_word Esc-.

在光标位置插入先前命令的最后一个参数。

ins_nth_word Meta-[1..9]

在光标位置插入先前命令的第 n 个参数。

list_func Meta-l

列出可用的编辑功能。

next_char Right, Ctrl-f

移动光标到下一个字符。

next_line Down, Ctrl-n

移动到下一个历史纪录。

next_word Meta-f

移动光标到下一个单词。

prefix

等候下一个字符输入。

prev_char Left, Ctrl-b

移动光标到先前的字符。

prev_line Up, Ctrl-p

移动到前一个历史纪录。

prev_word Meta-b

移动光标到前一个单词。

quote

Ctrl-q

逐字的插入下一个字符到缓冲区。

search

Ctrl-s

在历史记录里搜索模式 (pattern) 。

search_rev Ctrl-r

在历史纪录里使用倒序搜索模式 (pattern) 。

subst

Meta-s

替换所有的模式（pattern）为字符串，并重新输出行。

配置（CONFIGURATION）：

功能键可以在 fvwm 的模块配置里定义：

```
*FvwmConsole: Key \ck prev_line
```

没有空格的字母组合不必加引号。删除最后一个参数可以取消之前的功能键定义：

```
*FvwmConsole: Key \ck
```

调用（INVOCATION）：

FvwmConsoleC.pl 应该被 FvwmConsole 调用。

Fvwm 中文手册-FvwmButtons

FvwmButtons

名称（NAME）：

FvwmButtons -FVWM buttonbox 模块

概要（SYNOPSIS）：

Module FvwmButtons [-g geometry] [-transient | -transientpanel] [name[configfile]]

FvwmButtons 只能被 fvwm 调用(fork)，不能从命令行启动。

描述（DESCRIPTION）：

FvwmButtons 模块在桌面上提供了一个窗口，这个窗口可以作为按钮的容器，在它里面放置多个按钮，而我们可以使用 FvwmButtons 模块去管理和布局这些按钮。我们可以在任何时候通过去点击这些按钮来触发与之关联的操作，比如启动一个应用程序。FvwmButtons 只能在 fvwm 窗口管理器里使用。

FvwmButtons 提供的按钮容器可以被配置成任意的形状，并可以处于任意位置。它可以容纳具有关联操作的单色或彩色图标，甚至还可以在它里面集成其它的应用或模块。

我们可以通过点击一个按钮去打开面板，参看 CREATING PANELS 部分。

选项（OPTIONS）：

-g，指定主窗口的 geometry。这里的命令行选项优先于配置文件里的 geometry 选项。

-transient，使用 transient 选项调用 FvwmButtons 模块时，一旦其中按钮关联的操作的被执行，FvwmButtons 就会自动关闭。

-transientpanel 选项类似于-transient 选项，但是并不会关闭整个 FvwmButtons，仅仅是将它隐藏起来。这通常使用在一个 FvwmButtons 作为一个另一个 FvwmButtons 的子面板（subpanel）被启动时。

调用（INVOCATION）：

FvwmButtons 只能被 fvwm 调用(fork)，不能从命令行启动。

可以通过在配置文件里添加‘Module FvwmButtons OptionalName’语句调用 FvwmButtons 模块。如果希望 FvwmButtons 在 fvwm 初始化期间启动，它需要添加在 StartFunction 函数里，否则，可以将它与菜单、鼠标、键盘等操作绑定在一起。

如果指定了 OptionalName 参数，则使用 OptionalName 查找配置命令，而不是 FvwmButtons，例如：

```
AddToFunc StartFunction Module FvwmButtons MyButtonBox
```

之后，FvwmButtons 将使用以“*MyButtonBox”字符串开始的配置，而不是默认的"*FvwmButtons"。

配置选项（CONFIGURATION OPTIONS）：

下面是目前可用的一些配置选项：

```
*FvwmButtons: Back color
```

指定按钮的背景色，relief 和阴影的颜色可以通过它计算出来。

```
*FvwmButtons: BoxSize algorithm
```

指定遵守 Rows 和 Columns 选项的程度。可以是 dumb，fixed，smart。

如果使用 fixed，且 Rows 和 Columns 选项被指定为非 0 值，FvwmButtons 使用固定的行数和列数，如按钮容器（box）太小以至于不能容纳所有的按钮，这个模块将不能成功启动。

如果使用 smart，FvwmButtons 将可以扩大按钮容器来容纳所有的按钮。列的数目将至少增加到最宽的按钮的宽度，并且新的行将被增加直到已经容纳所有的按钮。使用 smart 选项可以减少配置错误的发生。

dumb 即不表示 fixed，也不表示 smart，这是默认值。

```
*FvwmButtons: Colorset colorset
```

指定模块窗口背景使用的颜色集（colorset）。关于 colorset 的细节可以参考 FvwmTheme。

```
*FvwmButtons: ActiveColorset colorset
```

指定当鼠标光标移动到按钮上面时，它的背景和标题（title）使用的 colorset。

```
*FvwmButtons: PressColorset colorset
```

指定当按钮被按下时，它的背景和标题（title）使用的 colorset。

```
*FvwmButtons: Columns columns
```

指定按钮容器窗口里包含按钮的列数。如果没有指定，则列数被设置成按钮的总数除以行数。如果行数和列数都被指定，但是按钮的总数大于它们所允许的范围，则列数被忽略，除非 BoxSize 选项的值是 fixed。

```
*FvwmButtons: File filename
```

指定从文件 filename 里查找配置选项。filename 可以是绝对路径，或者假定位于 fvwm 的启动目录下。这个文件和 fvwm 的配置文件有同样的格式，但是其中的每一行都以字符串“*FvwmButtons”为前缀。

```
*FvwmButtons: Font font
```

指定按钮标签使用的字体，默认为 None。

*FvwmButtons: Fore color

指定按钮标签的文本和单色图标的使用的颜色。

*FvwmButtons: Frame width

指定按钮周围 relief 的宽度。如果为负数，则正常情况下按钮的轮廓是凹下去的。

*FvwmButtons: Geometry geometry

指定按钮容器窗口的位置和尺寸。

*FvwmButtons: ButtonGeometry geometry

指定按钮容器串口里单个按钮的位置和尺寸。

*FvwmButtons: Padding width height

指定 FvwmButtons 窗口里每个按钮周围的空白空间，width 表示水平方向的空白为，height 表示垂直方向的空白，单位为像素。除了 FvwmButtons 窗口里集成的其它窗口和容器，正常情况下，按钮的 relief 和它的内容之间，左右两边的间隙是 2 个像素，上面和下面的间隙是 4 个像素。

*FvwmButtons: Pixmap pixmapfile

指定 FvwmButtons 窗口的背景使用的 pixmap。none 表示透明背景。

*FvwmButtons: Rows rows

指定按钮行数，默认值为 2。

*FvwmButtons: (options) [title icon command]

指定 FvwmButtons 窗口里单个按钮的内容，多个选项之间使用逗号分隔：

geometry

指定按钮的大小和在 FvwmButtons 窗口里的大小和位置，以普通按钮的宽和高为单位。明确指定位置参数的按钮将首先被放置，如果两个以上的按钮发生重叠，则 FvwmButtons 将产生一条错误消息，然后退出。

Action [(options)] command

指定按钮被激活时（按下回车键或使用鼠标点击时）执行的 fvwm 命令。如果 command 包含逗号或

者括号，则需要加引号。

目前 Action 的当前选项可以是：Mouse n – 表示 command 仅与鼠标按键 n 关联。

在 command 字段里，可以使用预定义的变量：$left, $right, $top 和 $bottom 表示按钮被按下时按钮的 left，right，top，和 bottom 坐标。$-left, $-right, $-top 和 $-bottom 具有同样的含义，不过是从屏幕的底部或右边界开始算起（比如距屏幕右边界 5 个像素，则 $-right 值为 5）。$width 和 $height 表示按钮的宽和高。变量 $fg 和 $bg 表示 Back 或 Fore 选项指定的背景色和前景色。字母‘$’可以通过‘$$’获

得。

例如：

```
*FvwmButtons: (Title xload, Action (Mouse 1) \

`     'Exec exec xload -fg $fg -bg $bg -geometry -3000-3000')
```

注意，在 2.5.0 之前的 fvwm 版本，Action 选项不能被分配给集成（swallowed）了一个应用窗口的按钮。这样的操作只能通过点击按钮四周的 border 执行。目前的版本修正了这个问题，为了兼容之前的版本，可以使用 ActionIgnoresClientWindow：

```
*FvwmButtons: (Action beep, ActionIgnoresClientWindow, \

     Swallow xeyes "Exec exec xeyes")
```

上面的例子里，command 仅当你点击这个按钮的 border 或 xeyes 窗口的透明部分时执行。

ActionIgnoresClientWindow

参看上面 Action 部分。

ActionOnPress

通常情况下，除了 Popup 操作，其它关联操作都是当鼠标按钮释放时执行。ActionOnPress 可以改变这种行为，使关联操作在按钮按下时执行。

Back color

指定 box 的背景色。relief 和阴影的颜色可以通过它计算出来。

Center

按钮的内容位于按钮中央。默认是这样的，但可以通过 Left 或 Right 改变。

Colorset colorset

colorset 可以应用于一个容器，一个集成的（swallowed）应用和简单的一个按钮。为了应用于按钮和容器，把这个选项添加到描述按钮和容器的行里即可。使用 colorset 为单个按钮和容器绘制背景需要和 X server 进行大量的通信。因此，在显示多个按钮时，如果你不能满意使用 colorse 的速度，就不要去使用它。应用于集成的应用时没有这个约束，但完全依赖于这个应用本身。

如果集成的窗口是一个 fvwm 模块，则 colorset 不能适用于它。我们应该在该模块的配置里使用 colorset。如果这个模块的背景使用了透明的 colorset，则 FvwmButtons 窗口的背景被使用作这个模块的背景。参考 FvwmTheme 模块。

ActiveColorset colorset

指定当鼠标光标移动到这个按钮上面时，它的背景和标题（title）使用的 colorset。

PressColorset colorset

指定当这个按钮被按下时，它的背景和标题（title）使用的 colorset。

Container [(options)]

指定这个按钮将包含一个小型的按钮容器（buttonbox），相当于集成了另外一个 FvwmButtons 模块。Options 将影响这个容器里的所有按钮。适用于这个用法的选项有 Back, Font, Fore, Frame 和 Padding。Title 和 Swallow 选项的标记可以使用 Title(flags) 和 Swallow(flags)设置。同样也可以指定"Columns width"或"Rows height"，"Rows 2"是默认值。

这个集成的容器按钮可以使用 Frame 和 Padding 选项，也可以关联特定的命令。

典型地，你至少希望给容器设置一个 widthxheight 的大小。

End

指定不会再向在当前的按钮容器添加更多的按钮，之后的按钮将被添加在这个容器的父容器（parent）里。这个选项应该独占一行，例如：

```
    *FvwmButtons: (End)
```

Font fontname

指定这个按钮的标签使用的字体。

Fore color

指定按钮标签的文本和单色图标的使用的颜色。

Frame width

指定按钮周围 relief 的宽度。如果为负数，则正常情况下按钮的轮廓是凹下去的。

Icon filename

指定按钮的图标。FvwmButtons 将从 ImagePath 配置的路径里查找图标文件。

ActiveIcon filename

指定当鼠标光标移动到这个按钮上面时，显示的图标。如果没有明确指定，使用和 Icon 同样的配置。

PressIcon filename

指定当这个按钮被按下时，显示的图标。如果没有明确指定，使用和 Icon 同样的配置。

Id id

按钮的 id，第一个字符必须为字母。

Fvwm 中文手册-FvwmCpp

FvwmCpp

名称（NAME）：

FvwmCpp -FVWM Cpp 预处理器

概要 (SYNOPSIS) :

```
Module FvwmCpp [options] filename
```

FvwmCpp 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) :

FvwmCpp 执行时，将对参数里指定的文件进行 Cpp 的预处理，之后，fvwm 将会去执行预处理输出文件里包含的命令。

调用：

FvwmCpp 可以通过.fvwm2rc 文件、菜单操作、鼠标操作、或者任何能够执行 fvwm 命令的任何方式调用。

如果用户希望使用 FvwmCpp 预处理自己的.fvwm2rc 文件，则应该使用下面的命令启动 fvwm：

```
    fvwm -cmd "Module FvwmCpp .fvwm2rc"
```

注意，选项"-cmd"的参数需要加引号。

FvwmCpp 作为一个模块运行时，可以与 fvmw 异步运行。如果在.fvwm2rc 里调用 FvwmCpp，则它生成的命令有可能不能够在执行.fvwm2rc 文件中的下一条命令时运行。按照下面的命令调用 FvwmCpp 达到异步的目的：

```
    ModuleSynchronous FvwmCpp -lock filename
```

选项 (OPTIONS) :

```
-cppopt option
```

传递一个选项给 cpp 程序。

```
-cppprog name
```

使用 name 指定的程序替代调用"/usr/lib/cpp"

```
-outfile filename
```

指定输出文件的名称。注意，写这个文件前，FvwmCpp 会尝试首先删除它，因此不要在它里面保存任何重要的信息，即使已经加上写保护。

```
-debug
```

保留 Cpp 输出的临时文件。它通常是"/tmp/fvwmrcXXXXXX"

```
-lock
```

如果希望使用这个选项，你需要使用 ModuleSynchronous 来启动 FvwmCpp。-lock 使 fvwm 等待预处理过程结束，并在继续下一步之前，FvwmCpp 会请求 fvwm 读取预处理过的文件。

```
-noread
```

使 fvwm 不去读取预处理后的文件。

配置选项 (CONFIGURATION OPTIONS) ：

FvwmCpp 定义了一些可以在预处理文件里使用的常量：

TWM_TYPE：

    总是“fvwm”

SERVERHOST

    运行 X server 的机器名

CLIENTHOST

    运行 fvwm 的机器名

HOSTNAME

    运行 fvwm 的主机名，通常与 CLIENTHOST 相同

OSTYPE

    CLIENTHOST 上的操作系统

USER

    运行 fvwm 的用户的名字

HOME

    用户主目录

VERSION

    X11 版本

REVISION

    X11 版本的修订号

VENDOR

    X server 厂商

RELEASE

    X server 版本号

SCREEN

    显示屏幕号

WIDTH

屏幕宽度

HEIGHT

屏幕高度

X_RESOLUTION

水平方向上的距离

Y_RESOLUTION

垂直方向上的距离

PLANES

BITS_PER_RGB

CLASS

COLOR

FVWM_CLASS

FVWM_COLOR

FVWM_VERSION

fvwm 版本号

OPTIONS

FVWM_MODULEDIR

fvwm 配置文件存放的目录。

FVWM_USERDIR

环境变量 $FVWM_USERDIR 的值

SESSION_MANAGER

环境变量 $SESSION_MANAGER **的值**

示例 (EXAMPLE PROLOG)

```
#define TWM_TYPE fvwm

#define SERVERHOST spx20

#define CLIENTHOST grumpy

#define HOSTNAME grumpy

#define OSTYPE SunOS

#define USER nation

#define HOME /local/homes/dsp/nation

#define VERSION 11

#define REVISION 0

#define VENDOR HDS human designed systems, inc. (2.1.2-D)

#define RELEASE 4

#define SCREEN 0

#define WIDTH 1280

#define HEIGHT 1024

#define X_RESOLUTION 3938

#define Y_RESOLUTION 3938

#define PLANES 8

#define BITS_PER_RGB 8

#define CLASS PseudoColor

#define COLOR Yes

#define FVWM_VERSION 2.0 pl 1

#define OPTIONS SHAPE XPM Cpp

#define FVWM_MODULEDIR /local/homes/dsp/nation/modules

#define FVWM_USERDIR /local/homes/dsp/nation/.fvwm
```

```
    #define SESSION_MANAGER local/grumpy:/tmp/.ICE-unix/440,tcp/spx20:1025
```

Fvwm 中文手册-FVWMDEBUG

FVWMDEBUG

名称（NAME）：

FVWMDEBUG - FVWM 模块调试器

概要（SYNOPSIS）：

可以在配置文件里使用下面的命令启动这个模块：

```
    Module FvwmDebug [optional-params]
```

下面的命令将停止这个模块。

```
    KillModule FvwmDebug
```

描述（DESCRIPTION）：

FVWMDEBUG 能够将所有的 fvwm 事件细节，和其它的一些信息写入标准错误输出或者文件。可以将输出重定向到 xconsole 或类似的窗口。

调用（INVOCATION）：

```
FvwmDebug [ --args|--noargs ] [ --events|--noevents ] [ --log file ] [ --xconsole ]
[ --mask mask ] [ --xmask mask ] [ --debug level ] [ --track tracker-name ] [ --
send-configinfo ] [ --send-windowlist ]
```

--noargs

仅输出事件的名称。默认为--args。

--noevents

连事件名称都不输出，已经暗含了-noargs 选项。类似于将--mask 和-xmask 设置为 0，但是实际上事件已经被模块接受，仅仅不输出而已。

默认为-events

-l|--log file

使用特定的日志文件取代标准错误输出。如果这个文件不能写，则仍使用默认的标准错误输出。

-xc|--xconsole

这个选项是下面命令的快捷方式：

```
    FvwmDebug --log '|xconsole -file /dev/stdin -geometry 600x400 -notify'
```

在 xconsole 上显示输出，而不是标准错误输出。

-m|--mask mask

设置模块掩码（mask），31 位的整数。默认为监控几乎所有的事件。

-x|--xmask mask

设置扩展的模块掩码。

-d|--debug level

使用 Perl 库的调试机制。可用的 levels 是 2-4。

-t|--track tracker-name

-sc|--send-configinfo

启动时发送 Send_configinfo 命令给 fvwm。

-sw|--send-windowlist

启动时发送 Send_WindowList 命令给 fvwm。

Fvwm 中文手册-FvwmDragWell

FvwmDragWell

名称（NAME）：

FvwmDragWell - XDND drag well

概要（SYNOPSIS）：

FvwmDragWell 是一个 drag well。用户通过命令 SendToModule 发送数据和数据类型到 drag well，这个 drag well 将通过一个简单的动画显示它收到了数据，然后，用户可以拖拉（drag）这些信息到其它支持 XDND 标准的应用。

初始化（INITIALIZATION）：

初始化期间，FvwmDragWell 从 fvwm 的模块配置数据库里获得配置信息。如果 FvwmDragWell 被链接到其它应用，比如 ln -s FvwmDragWell OtherDragWell ，则模块 OtherDragWell 将被启动。

用法（USE）：

用户通过命令 SendToModule 发送数据和数据类型到 drag well，格式为：

```
SendToModule modulename dragtype type-string, dragdata data
```

type-string 需要加引号，声明了数据的类型和如何导出数据。type-string 将被转化为 XAtom 类型。用户应该使用标准的包含小写字母的 Mime 类型字符串。注意，这个模块并不进行数据转换。

上面的格式里，dragtype 可以被省略，此时，FvwmDragWell 将假定用户正在传递一个文件或目录的路径。这种情况下，字符串"file://$hostname/"被附加到数据的前面，且导出数据的类型为"text/uri-list"。例如：

```
SendToModule FvwmDragWell dragdata /usr/local/libexec/fvwm/2.4.0
```

FvwmDragWell 将转换数据为“file://saturn/usr/local/libexec/fvwm/2.4.0”，并且将数据以“text/uri-list”类型导出。这个例子的完整格式是：

```
SendToModule FvwmDragWell dragtype text/uri-list, \
                          dragdata file://saturn/usr/local/libexec/fvwm/2.3.8
```

调用（INVOCATION）：

FvwmDragWell 只能被 fvwm 调用(fork),不能从命令行启动。

配置选项（CONFIGURATION OPTIONS）：

FvwmDragWell 从 fvwm 的配置文件.fvwm2rc 里查找自己的配置信息。

*FvwmDragWell: Geometry geometry

完全或部分的指定窗口的位置和尺寸。

*FvwmDragWell: DragWellGeometry geometry

指定 FvwmDragWell 窗口里拖拉盒（drag box）的位置和尺寸。

*FvwmDragWell: Colorset colorset

指定 drag well 的背景和阴影使用的 colorset。

*FvwmDragWell: Fore Color

前景色，默认为 grey60。

*FvwmDragWell: Back Color

背景色，默认为黑色。

*FvwmDragWell: Shadow Color

按钮 relief 的阴影色。

*FvwmDragWell: Hilite Color

按钮 relief 的高亮色。

Fvwm 中文手册-FvwmForm（一）

FvwmForm

名称（NAME）：

FvwmForm - Fvwm 输入表格（form）模块。

概要（SYNOPSIS）：

```
Module FvwmForm [ Alias ]
```

FvwmForm 只能被 fvwm 调用(fork)。如果以命令行的形式调用 FvwmForm，则它将在显示版本号之后退出。

描述：

FvwmForm 提供了一种获取用户输入并执行相应操作的机制，它通过使用户填充一定的表格，和选择希望 fvwm 执行的操作来实现。一个表格包含了五种类型的项目：文本标签（text labels），单行文本输入框（single-line text inputs），单选框（mutually-exclusive selections），多选框（multiple-choice selections），操作按钮（action buttons）。

l 文本标签（text labels）只起解释的作用，并不接受任何输入。

l 超时项（timeout entry）指出超时发生时默认执行的操作。倒计时时钟的显示类似于文本标签，除了它会自动更新剩余的时间。

l 文本输入域（text input field）可以用来编辑单行字符串。FvwmForm 可以接受 Emacs 风格的光标移动快捷键。细节请参看 FvwmFormInput。不支持使用鼠标进行复制，但是可以粘贴。

l 一个选择框控件（selection）包括几个选项。每个选项包括一个按钮（push-button），和一个解释性的文本标签。单选框的按钮显示为圆形，多选框的按钮显示为圆形。

l 操作按钮（action buttons）被激活时将发送一个或多个命令给 Fvwm，或者执行 shell 命令。执行的 shell 命令可以包含文本输入域的内容并反映选择框控件的设置。操作按钮可以使用键盘或鼠标激活。

初始化（INITIALIZATION）：

Alias 参数为空时调用 FvwmForm，将使用以“*FvwmForm”开始的配置。

通常来说，你应该使用一个别名调用 FvwmForm。例如，命令“Module FvwmForm Rlogin”使用以“*Rlogin”开始的配置命令，并读取可选的配置文件“Rlogin”。

但是，所有的表格首先都会查找以“*FvwmFormDefault”开始的配置命令。这些命令通常来自于内置的“FvwmForm-Form”表格，它将命令在“.FvwmForm”文件。

“.FvwmForm”文件仅在 FvwmForm 第一次被调用时读取，或者在“FvwmForm-Form”更新了整个“.FvwmForm”文件之后。

可以通过发送命令“Read .FvwmForm Quiet”给 fvwm 来读取“.FvwmForm”文件。考虑到“read”命令的工作方式，这个文件能够保存在你的用户目录里，或者在 fvwm 的数据目录里。

然后，FvwmForm 读取剩余的配置。初始的配置来自.fvwm2rc 文件。

让 FvwmForm 和 fvwm 读取文件时，要记得这些文件包含了可能执行 shell 命令的命令，因此你应该谨慎设置这些文件的权限。

FvwmForm 在一个窗口上下文被调用时，例如窗口菜单，所有发送给 Fvwm 的命令将有那个窗口上下文。这将允许 FvwmForm 控制那个窗口。

所有的配置命令都已经被读取之后，FvwmForm 将显示这些命令定义的表格。

默认（DEFAULTS）：

FvwmForm 创建一个内置的“FvwmForm-Form”表格，它创建了“.FvwmForm”文件。这个文件包含默认的表格颜色和字体。其它表格将使用这些默认值，除非在它们的定义里重新定义这些值。

这个默认创建的表格通常通过“module menu”调用。例如，如果你的模块菜单“Module-Popup”，将增加下面的行：

```
AddToMenu "Module-Popup" "FvwmForm Defaults" FvwmForm FvwmForm-Form
```

当从模块菜单里选择“FvwmForm Defaults”时，将显示一个表格，它显示了当前表格设置的默认值，并允许用户去改变。如果激活“Save Restart Me”按钮，“.FvwmForm”文件将被重写，“FvwmForm-Form”退出并重启，并显示新的默认值。

下面是执行保存操作后这个文件的示例：

```
# This file last created by FvwmForm-Form on Sun Nov 28 11:18:26 EST 1999.

*FvwmFormDefault: Font 10x20

*FvwmFormDefault: InputFont 8x13bold

*FvwmFormDefault: ButtonFont 10x20

*FvwmFormDefault: TimeoutFont 10x20

*FvwmFormDefault: Fore white

*FvwmFormDefault: Back cornflowerblue

*FvwmFormDefault: Colorset -1

*FvwmFormDefault: ItemFore green

*FvwmFormDefault: ItemBack gray40

*FvwmFormDefault: ItemColorset -1

*FvwmFormDefault: ButtonPointer hand2

*FvwmFormDefault: ButtonInPointer star

*FvwmFormDefault: InputPointer gumby

*FvwmFormDefault: ButtonPointerFore blue

*FvwmFormDefault: ButtonPointerBack gray

*FvwmFormDefault: ButtonInPointerFore gray

*FvwmFormDefault: ButtonInPointerBack blue

*FvwmFormDefault: InputPointerFore
```

```
*FvwmFormDefault: InputPointerBack
```

这个文件里的命令与其它的 FvwmForm 命令相同，除了它们以“*FvwmFormDefault”开始。

FvwmForm 仅在启动时读取一次这个文件，或者在这个文件被“FvwmForm-Form”更改后读取。通过发送命令“*FvwmFormDefault: Read x”来完成读取操作。“x”可以设置为“y”或“n”，设置为“n”时，FvwmForm 发送一个“read .FvwmForm quiet”命令给 fvwm。

变量替换（VARIABLE SUBSTITUTION）：

如果使用类似下面的命令调用 FvwmForm：

```
Module FvwmForm MyForm ACTION=Browse "TITLE=Browse Form"
```

则所有的 FvwmForm 输入命令将经历变量替换。上面命令的变量被导出，然后，每个命令使用这些变量进行扩展。例如，有关上面“MyForm”的调用命令将会发生变化：

替换前： *MyForm: Text "$TITLE, Home $HOME, Going to $ACTION"

替换后： *MyForm: TEXT "Browse Form, Home /home/me, Going to Browse"

利用这个特点，一个表格能够用于设置不同的输入数据。

Fvwm 中文手册-FvwmForm（二）

配置（CONFIGURATION）：

下面的命令可以在.fvwm2rc 文件里被设置，或者可以通过其它 fvwm 能接受的任何方式。最简单的方式是在 [PREFIX/share/fvwm] 或你的个人 fvwm 目录[$HOME/.fvwm]里创建一个文件，并使用表格的别名作为配置命令的前缀。

下面的段落里，字符串“FvwmForm”通常应该是表格别名。

FvwmForm 在表格显示前，或者表格正在显示的时候读取命令。

下面的命令在表格显示前被接受：

```
Back

Button

ButtonFont

ButtonInPointer

ButtonInPointerFore

ButtonInPointerBack

ButtonPointer

ButtonPointerFore
```

```
ButtonPointerBack

Choice

Command

Colorset

Font

Fore

GrabServer

Input

InputFont

InputPointer

ItemBack

ItemColorset

ItemFore

InputPointerFore

InputPointerBack

Line

Message

PadVText

Position

Selection

Text

Timeout

TimeoutFont

Title

UseData

WarpPointer
```

下面的命令在表格显示时被接受：

```
    Map

    Stop

    UnMap
```

“Map”, “UnMap”和 “Stop”工具仍然正在开发之中，这里将不做解释，因为它们有可能改变。

这些选项处理的顺序，第一个是背景文本色，“*FvwmFormBack”，设置整个表格的默认背景色。

其它的，比如颜色，字体，文本，选择，和按钮可以以任意顺序处理。在表格尺寸，表格项的数目，字体或颜色的数目上没有限制。

```
*FvwmForm: GrabServer
```

使 FvwmForm 在启动时获得鼠标指针的焦点。

```
*FvwmForm: WarpPointer
```

使 FvwmForm 在启动时 warp 指针到它的窗口。

```
*FvwmForm: Geometry geometry
```

指定 FvwmForm 的窗口位置。

```
*FvwmForm: Position x y
```

把 FvwmForm 窗口放在屏幕的 (x,y) 位置。如果这个选项为空，FvwmForm 窗口将位于的屏幕中心。

```
*FvwmForm: Colorset n
```

指定这个模块使用 colorset n。

```
*FvwmForm: Back color
```

指定 FvwmForm 窗口的背景色和文本色。

```
*FvwmForm: Fore color
```

指定文本标签的前景色。覆盖了 Colorset 选项。

```
*FvwmForm: ItemColorset n
```

指定 FvwmForm 中各项使用 colorset n。

```
*FvwmForm: ItemBack color
```

指定文本输入窗口和按钮的背景色。按钮被显示为 3D depressable 按钮。输入域被显示为 3D indented。

```
*FvwmForm: ItemFore color
```

指定文本输入字符串和按钮文本的前景色。

```
*FvwmForm: Font font
```

指定文本的字体。

*FvwmForm: ButtonFont font

指定操作按钮的文本的字体。

*FvwmForm: InputFont font

指定输入文本的字体。

*FvwmForm: TimeoutFont font

指定超时计数器和相关文本的字体。

*FvwmForm: Line justification

开始一个新行。一行可以包含任意数目的文本，输入，按钮，和选择项。一个 FvwmForm 窗口可以有任意数目的行。窗口的宽度取决于最长的行。

justification 指定了行内各项的对齐状态，可以是下面中的一个：

left 左对齐

right 右对齐

center 中心对齐

expand

*FvwmForm: Message

定义一个文本域，包含了来自 fvwm 的最后一条错误消息。这个消息域的宽度默认为 80 字节。它的实际长度和接受的消息一样。如果这个消息大于 80 字节，你能够通过缩放表格窗口查看剩余的部分。

*FvwmForm: PadVText Pixels

指定文本项之间，行与行之间垂直填充的象素数。默认为 6。

*FvwmForm: Text string

显示字符串为普通文本。必须通过多个*FvwmForm: Line 和 *FvwmForm: Text 选项实现断行。空白可以用来在各个项之间提供额外的填充。

*FvwmForm: Title string

显示 string 作为窗口标题。这个字符串必须加双引号。不加引号时会显示空白标题。如果没有使用这个命令，窗口标题是表格的别名。

*FvwmForm: Input name size init_string

指定一个名为 name 的文本输入项。size 字符宽的子窗口用来编辑文本。如果 init_string 指定，它是 FvwmForm 启动和重启时的初始字符串。默认的初始字符串是“”。

可以使用鼠标按键 2 粘贴文本到输入域，按键 1 和 3 移动光标到输入域。

输入域总是处于插入模式，不支持 overtyping。

Emacs 风格的 keystrokes 被支持。

ctrl-a，Home 和 Begin 移动到输入域的开始。ctrl-e 和 End 移动到输入域的最后。ctrl-b 和 Left 向左移动，ctrl-f 和 Right 向右移动。ctrl-p，Up，Shift-Tab 向上移动。

*FvwmForm: Selection name type

这个选项创建名为 name 的选择框控件。它的各个选项在之后的配置命令里指定。type 是下面中的一个：

single 单选

multiple 多选

*FvwmForm: Choice name value on | off string

创建选择框控件的一个选项。它的名字为 name，值为 value。字符串 string 显示在选项按钮的右边。

*FvwmForm: Button type string [key]

创建一个操作按钮。string 为它的标签，激活时执行一组 Fvwm 命令。

*FvwmForm: Command command

指定与当前按钮关联的 Fvwm 命令。一个按钮可以关联多个命令。如果在任意一个*FvwmForm: Button 选项之前指定 command，则该命令在启动时被执行，这种情况下，通常是引起用户注意的蜂鸣声。

以"！"开始的命令被 FvwmForm 执行，所有其它的命令被发送到 Fvwm 执行。在发送命令给 Fvwm 之前，FvwmForm 识别下面形式的变量，并替换它们。

$(name)

如果 name 对应文本输入域，则使用用户输入的字符串替换。

如果 name 对应于一个选择项，则使用该选择项的值替换。

如果 name 对应一个选择框控件（selection），则使用被选择的所有选择项的值的列表替换。

$(name?string)

如果 name 对应输入文本域，且它的值不是空字符串，则使用 string 替换。

如果 name 是一个选择项且被选中，则使用 string 替换。

$(name!string)

同上，除了使用相反的条件。

*FvwmForm: UseData datafile leading

指定 FvwmForm 读取一个数据文件，并从匹配"leading"参数的模块命令、输入域、选择项、选择框变量提取数据。

*FvwmForm: ButtonPointer pointername

改变鼠标指针移动到按钮上面时的形状。

*FvwmForm: ButtonInPointer pointername

改变按钮按下时的鼠标指针的形状。

*FvwmForm: InputPointer pointername

改变文本域上的鼠标指针形状。

*FvwmForm: ButtonPointerFore|Back color

改变鼠标指针移动到按钮上面时的默认前景色和背景色。

*FvwmForm: ButtonInPointerFore|Back color

改变按钮按下时鼠标指针的默认前景色和背景色。

*FvwmForm: InputPointerFore|Back color

改变鼠标指针移动到文本域上方时，默认的前景色和背景色。

*FvwmForm: Timeout seconds command text

设置 seconds 时间后 FvwmForm 超时。当定时器为 0 时，执行 command。

例子（EXAMPLES）：

略，参看 http://www.fvwm.org/documentation/manpages/unstable/FvwmForm.php

Fvwm 中文手册-FvwmGtk

FvwmGtk

名称（NAME）：

FvwmGtk -Fvwm GTK 模块。

概要（SYNOPSIS）：

FvwmGtk 只能被 fvwm 调用(fork)。可以在配置文件里使用下面的语句调用 FvwmGtk：

```
Module FvwmGtk [ name ]
```

也可以从一个 fvwm 弹出菜单里调用 FvwmGtk：

```
DestroyMenu Module-Popup
```

```
    AddToMenu   Module-Popup "Modules" Title

    AddToMenu   Module-Popup "Gtk" Module FvwmGtk [ name ]
```

描述（DESCRIPTION）：

FvwmGtk 模块在 fvwm 中实现了基于 GTK 的 GUI 元素，即内置菜单和 FvwmForm 对话框。

调用（INVOCATION）：

FvwmGtk 只能被 fvwm 调用(fork),不能从命令行启动。指定参数 name 时，使用 name 来查找配置命令和配置文件，而不是使用“FvwmGtk”。

配置选项（CONFIGURATION OPTIONS）：

FvwmGtk 仅有定义菜单和对话框内容的选项，包括标签和显示的 pixmaps；菜单和对话框外观的配置必须通过 GTK 的 rc 文件实现。

FvwmGtk 从 fvwm 的模块配置数据库获得配置信息。另外，FvwmGtk 运行时能够接受来自 fvwm 和其它模块的命令。

如果使用参数 name 启动 FvwmGtk，name 在所有的命令，消息，和 FvwmGtk 生成的菜单和表格中使用。不像其它的模块，这里很少会使用到 name。

所有的对话框和菜单必须在它们可用前通过配置命令进行定义。对话框和菜单可以通过将它们的名字发送给 FvwmGtk 进行调用。对于菜单，与之绑定的按钮也需要一起发送。

```
SendToModule FvwmGtk menu-example 1

SendToModule FvwmGtk dialog-example
```

菜单（MENUS）：

下面的命令用来定义菜单。

```
*FvwmGtk: Menu name
```

通知 FvwmGtk 将之后的菜单项添加到 name 指定的菜单上。注意，你可以“reopen”一个菜单，并继续添加项给它。

```
*FvwmGtk: Title label [ icon [ r_label ] ]
```

添加一个 title 到当前的菜单上。如果指定可选的参数，它应该是一个 xpm 文件的名称。这个图标将出现在菜单项文本的左边。如果 FvwmGtk 已经编有 imlib 支持，任何 imlib 能够读取的图像格式都可以被使用。这个标签可以包含一个“&”，指示跟随的字符将添加下划线来表示快捷键。

```
*FvwmGtk: Item label action [ icon [ r_label ] ]
```

添加一个菜单项到当前打开的菜单上。第一个参数是显示的文本，第二个参数是激活时发送给 fvwm 的命令。如果指定可选的参数，它应该是一个 xpm 文件的名称。图标将显示在文本左边。这个标签可以包含一个"&"，指示跟随的字符将添加下划线来表示快捷键。

```
*FvwmGtk: Submenu label name [ icon ]
```

添加一个菜单项到当前打开的菜单上。第一个参数是显示的文本，第二个参数是子菜单的名称。如果子菜单不存在，它将被创建。如果指定可选的参数，它应该是一个 xpm 文件的名称。图标将显示在本本左边。这个标签可以包含一个"&"，指示跟随的字符将添加下划线来表示快捷键。

```
*FvwmGtk: Tearoff
```

添加一个专门的 tear-off 项。它被激活时，菜单将变成一个窗口，再次激活它菜单将消失。

窗口列表 (WINDOW LISTS) ：

窗口列表是是动态创建的菜单。它提供了 fvwm 管理的所有窗口的列表。

```
*FvwmGtk: WindowList name [ option... ]
```

创建一个名为 name 的窗口列表。这个列表的格式依据所所给定的选项而不同。下面的选项是当前所支持的，它们与 fvwm 内置的窗口列表选项有同样的意义： "NoGeometry", "NoMiniIcon", "UseIconName", "Desk <desk>", "CurrentDesk", "Icons/NoIcons/OnlyIcons", "Sticky/NoSticky/OnlySticky", "Normal/NoNormal/OnlyNormal", "NoDeskSort", "Alphabetic", "Function <func>"。下面的选项是新支持的："Title <label> <icon> <r_label>"。

对话框 (DIALOGS) ：

对话框由不同的控件 (widgets) 所组成。可以是容器控件，比如包含一个或多个其它控件，也可以是原子控件。容器控件通过指示子控件列表的开始和结束的一对命令定义。原子控件使用单个命令定义。一些控件从用户请求数据。

定义控件的命令通常有下面的形式：

```
*FvwmGtk: Widget widget-specific args [ -- general-args ]



*FvwmGtk: Dialog name title [ center ]
```

启动或重新打开名为 name 的对话框。title 是窗口标题。如果指定参数 center，对话框将被放到屏幕中心。否则它将放到鼠标的位置。

```
*FvwmGtk: Box [ vertical ] [ homogeneous ] [ spacing [ border ] ]

*FvwmGtk: EndBox
```

开始和结束一个容器控件的定义。参数将影响子控件的位置。

```
*FvwmGtk: Frame label [ border ]

*FvwmGtk: EndFrame



*FvwmGtk: Label label
```

添加一个标签控件。

*FvwmGtk: Entry name [initial-value]

添加一个能够接受用户输入的控件。initial-value 表示显示的初始字符串。

*FvwmGtk: Button label cmd ...

添加一个按钮控件，label 表示按钮的标签，cmd 表示按钮按下时触发的命令。

*FvwmGtk: CheckButton name label on-value off-value [on]

*FvwmGtk: RadioGroup name

*FvwmGtk: EndRadioGroup

*FvwmGtk: RadioButton label on-value [on]

*FvwmGtk: Notebook label

打开一个标签为 label 的 notebook 页。如果已经有一个打开的 notebook，则这里打开的页将附加到它上面。否则，创建一个新的 notebook。

*FvwmGtk: EndNotebook

关闭 notebook 控件。

*FvwmGtk: Color name [initial-value]

*FvwmGtk: Scale name [vertical] value lower

*FvwmGtk: SpinButton name value lower

*FvwmGtk: OptionMenu name

*FvwmGtk: EndOptionMenu

*FvwmGtk: Item label value [on]

用来创建 Option 菜单。

公用配置（COMMON CONFIGURATION）：

下面的命令菜单和对话框都可以使用：

*FvwmGtk: Separator

添加一个分隔行到当前打开的菜单上。

*FvwmGtk: Destroy name

销毁指定的菜单和对话框。

*FvwmGtk: RCFile file

注意，这个命令应该在定义任意一个菜单或对话框之前发送。

*FvwmGtk: IconSize [width height]

如果 FvwmGtk 已经编有 imlib 支持，图标将会被缩放到参数指定的尺寸。

命令（COMMANDS）：

可以使用命令 SendToModule 调用之前定义的菜单或对话框。

SendToModule FvwmGtk name button

将弹出 name 指定的菜单或对话框。第二个参数表示菜单所关联的按钮。例如：

```
Mouse 3 R A SendToModule FvwmGtk Window-Ops 3

Key F10 R A SendToModule FvwmGtk Applications-Menu

Mouse 1 R A SendToModule FvwmGtk Quit-Verify-Dialog
```

Fvwm 中文手册-FvwmGtkDebug

FvwmGtkDebug

名称（NAME）：

FvwmGtkDebug - 图形接口的 FVWM 模块调试器。

概要（SYNOPSIS）：

FvwmGtkDebug 应该被 fvwm 启动。

可以通过在配置文件里添加下面的命令来调用这个模块：

```
Module FvwmGtkDebug
```

关闭它的 GUI 窗口即可停止这个模块，KillModule 也可以。

也可以在命令行用 dummy 模式启动这个模块，但是这时的 GUI 仅显示 dummy 事件数据。

描述（DESCRIPTION）：

这个模块能够监视所有的 fvwm 事件信息，并用交互式的 gtk+应用显示它们。

调用（INVOCATION）：

FvwmGtkDebug [--mask mask] [--xmask mask] [--debug level]

-m|--mask mask 设置初始掩码，31 位整型数。这个掩码可以在任何时候改变。默认监视几乎所有的事件。

-x|--xmask mask 设置扩展的初始掩码，31 位整型数。这个掩码可以在任何时候改变。默认监视几乎所有的事件。

-d|--debug level 使用 Perl 库调试机制。可用的 levels 是 2-4。

Fvwm 中文手册-FvwmIconBox

FvwmIconBox

名称（NAME）：

FvwmIconBox -FVWM 图标盒（icon box）模块。

概要（SYNOPSIS）：

FvwmIconBox [name]

FvwmIconMan 只能被 fvwm 调用(fork)，不能从命令行启动。

描述（DESCRIPTION）：

FvwmIconBox 模块提供一个图标管理器。用户可以使用鼠标和键盘对显示在这个模块里的每个图标操作，比如图标化（iconify）和反图标化（de-iconify）。

初始化（INITIALIZATION）：

初始化期间，FvwmIconBox 从 fvwm 的模块配置数据库中获得配置信息。

调用（INVOCATION）：

可以将命令 'Module FvwmIconBox'绑定到一个菜单项或 key-stroke 来调用 FvwmIconBox。Fvwm 将在 ModulePath 配置选项指定的目录里查找 FvwmIconBox。

配置选项（CONFIGURATION OPTIONS）：

仅仅 NoIcon 应用时，FvwmIconBox 显示图标。注意，NoIcon 应该在 Icon 属性指定之后设置。否则，icon-box 模块可能变成一个空的模块。FvwmIconBox 启动时读取.fvwm2rc 文件，并寻找下面的语句：

*FvwmIconBox: Fore color

指定窗口前景色为 color，而不是白色。这个选项仅仅影响由*FvwmIconBox: Pixmap 指定的 background_bitmap 的前景色。

*FvwmIconBox: Colorset colorset

指定使用 colorset 取代 Fore 和 Back。

*FvwmIconBox: IconColorset colorset

指定使用 colorset 取代 IconFore 和 IconBack。

*FvwmIconBox: IconHiColorset colorset

指定使用 colorset 取代 IconHiFore 和 IconHiBack。

*FvwmIconBox: Back color

指定模块窗口背景色为 color，而不是黑色。

*FvwmIconBox: IconFore color

指定没有选中时图标的文本色为 color，而不是黑色。

*FvwmIconBox: IconBack color

指定没有选中时图标的文本背景色为 color，而不是黑色。

*FvwmIconBox: IconHiFore color

指定选中时图标的文本色为 color，而不是黑色。

*FvwmIconBox: IconHiBack color

指定选中时图标的背景色为 color，而不是白色。

*FvwmIconBox: Pixmap pixmap

指定使用 pixmap 作为模块窗口的 background_pixmap。

*FvwmIconBox: Font fontname

指定模块的文本使用 fontname。

*FvwmIconBox: SortIcons option

指定按字母顺序排列图标。option 可以是 WindowName, IconName, ResClass, 和 ResName。

*FvwmIconBox: Padding number

指定图标之间的空白像素数目。默认为 5。

*FvwmIconBox: SBWidth number

指定水平或垂直滚动栏（scrollbars）的宽度，默认为 9。

*FvwmIconBox: Placement primary secondary

指定图标的布局策略。primary 和 secondary 可以是 Top，Bottom，Left 和 Right。可以使用下面的 8 个组合：

primary	secondary
Left	Top
Left	Bottom
Right	Top
Right	Bottom
Top	Left
Top	Right
Bottom	Left
Bottom	Right

Top：图标从上到下布局。

Bottom：图标从下到上布局。

Left：图标从左到右布局。

Right：图标从右到左布局。

例如，为“Left Top”时，图标从左到右放置，且新的行从上到下增加。默认为"Left Bottom"。

*FvwmIconBox: UseSkipList

指定 FvwmIconBox 不显示配置文件里 WindowListSkip 行列出的窗口。

*FvwmIconBox: Lines

指定一行里图标的数目。如果 primary 是 Left 或 Right，这个值指定了列的数目。如果 pirmary 是 Top 或 Bottom，这个值指定了行的数目。例如，如果*FvwmIconBox: Lines 是 7 且*FvwmIconBox: Placement 是"Left Top"，则一行包含了 7 个图标。默认为 6。

*FvwmIconBox: HideSC direction

指定一个并不显示的滚动栏 (un-displayed scroll bar) 。direction 可以是 Horizontal 或 Vertical。

*FvwmIconBox: Geometry <width>x<height>{+-}<X>{+-}<Y>

指定 FvwmIconBox 窗口的位置。width 和 height 以图标为单位。默认为 6x1+0+0。

*FvwmIconBox: MaxIconSize <width>x<height>

指定图标位图的最大尺寸。如果一个位图比这个尺寸大，则剪切到适合这个尺寸。默认为 48x48。如果高为 0，图标位图不显示，仅仅显示图标标签。

*FvwmIconBox: Mouse Button Action Response[, Response]

指定鼠标按键 Button 执行 Action 操作时，FvwmIconBox 的响应。Response 是 Fvwm 的内置命令（例如 Iconify，Delete，Focus），Actions 可以是 Click 和 DoubleClick。

*FvwmIconBox: Key Key Response[, Response]

指定键盘按键 Key 按下时，FvwmIconBox 的响应 Response。除了 Fvwm 的内置命令，Response 还可以是下面 6 个 FvwmIconBox 内置命令：Next, Prev, Left, Right, Up, 和 Down。

Next：改变高亮图标为下一个。

Prev：改变高亮图标为上一个。

Left：向左移动水平滚动栏的滑块。图标相应右移。

Right：向右移动水平滚动栏的滑块。图标相应左移。

Up：向上移动垂直滚动栏的滑块。图标相应下移。

Down：向下移动垂直滚动栏的滑块。图标相应上移。

*FvwmIconBox: windowname bitmap-file

指定窗口 windowname 在 FvwmIconBox 里显示的位图。windowname 可以是窗口名字，窗口类名（class name），或资源名称。Windowname 可以包含"*"和"?"。bitmap-file 是一个位图文件的绝对路径，或者 ImagePath 目录里的文件，如果 bitmap-file 为"-"，则窗口 windowname 的图标将不被显示。

*FvwmIconBox: SetWMIconSize

指定模块设置根窗口的 XA_WM_ICON_SIZE 属性。

*FvwmIconBox: HilightFocusWin

将拥有键盘焦点的窗口的图标高亮显示。高亮图标的前景和背景色通过*FvwmIconBox: IconHiFore 和 *FvwmIconBox: IconHiBack 命令指定。

*FvwmIconBox: Resolution resolution

如果 resolution 是 Desk，FvwmIconBox 仅显示当前桌面（desk）的图标。目前，Desk 是唯一可以使用的值。

*FvwmIconBox: FrameWidth width1 width2

指定 FvwmIconBox 窗口的边框宽度（frame-width）。width1 表示窗口边框到滚动栏的宽度，width2 表示滚动栏到窗口内部图标的宽度。

*FvwmIconBox: NoIconAction action

指定当 NoIcon 风格的窗口图标化或非图标化时 FvwmIconBox 执行的操作 action。

*FvwmIconBox: NoIconifiedParentheses

指定 FvwmIconBox 不装入一个图标化窗口的标题。

*FvwmIconBox: NormalTitleRelief num

设置窗口没有图标化时，图标标题周围 relief 的宽度，默认为 2 像素。

*FvwmIconBox: IconifiedTitleRelief num

设置窗口图标化时，图标标题周围 relief 的宽度，默认为 2 像素。

*FvwmIconBox: NormalTitleInvertedRelief

*FvwmIconBox: IconifiedTitleInvertedRelief

配置示例 (SAMPLE CONFIGURATION) :
略，参看 http://www.fvwm.org/documentation/manpages/unstable/FvwmIconBox.php

Fvwm 中文手册-FvwmIdent

FvwmIdent

名称 (NAME) :

FvwmIdent -FVWM identify-window 模块。

概要 (SYNOPSIS) :

FvwmIdent 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) :

如果 FvwmIdent 没有从一个窗口上下文启动，它将提示用户选择一个目标窗口。之后，它将弹出一个窗口，显示被选中的窗口的信息。

初始化 (INITIALIZATION) :

初始化期间，FvwmIdent 从 fvwm 的模块配置数据库里获得配置信息，并决定使用什么颜色和字体。

调用 (INVOCATION) :

可以将命令'Module FvwmIdent'绑定到菜单或 .fvwm2rc 文件的 key-stroke 部分来调用 FvwmIdent。Fvwm 将在 ModulePath 指定的目录里查找 FvwmIdent。

配置选项 (CONFIGURATION OPTIONS) :

启动时，FvwmIdent 读取.fvwm2rc 文件并查找下面的语句：

*FvwmIdent: Colorset n

指定使用 colorset n。

*FvwmIdent: Fore color

指定文本色为 color，而不是黑色。

*FvwmIdent: Back color

指定窗口背景色为 color，而不是白色。

*FvwmIdent: Font fontname

指定文本的字体为 fontname。

*FvwmIdent: MinimalLayer layer

FvwmIdent 在目标窗口的层里放置它自己的窗口，但不能在这里设置的最小层下面，默认为 4。如果 layer 为 0，目标窗口的层总是可用的。如果 layer 为"default"或没有指定，恢复默认设置。如果 layer 为"none"，FvwmIdent 被作为普通窗口对待，即使目标窗口在它上面。

Fvwm 中文手册-FvwmM4

FvwmM4

名称 (NAME) ：

FvwmM4 - FVWM M4 预处理器。 (M4 是一个宏处理器。将输入拷贝到输出，同时将宏展开。宏可以是内嵌的也可以是用户定义的。除了可以展开宏，m4 还有一些内建的函数，用来引用文件，执行 Unix 命令，整数运算，文本操作，循环等。m4 既可以作为编译器的前端也可以单独作为一个宏处理器。)

概要 (SYNOPSIS) ：

Module FvwmM4 [options] filename

FvwmM4 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) ：

FvwmM4 执行时，将对参数里指定的文件进行 M4 的预处理，之后，fvwm 将会去执行预处理输出文件里包含的命令。

调用 (INVOCATION) ：

FvwmM4 可以通过.fvwm2rc 文件、菜单操作、鼠标操作、或者任何能够执行 fvwm 命令的任何方式调用。

如果用户希望使用 FvwmM4 预处理自己的.fvwm2rc 文件，则应该使用下面的命令启动 fvwm：

```
fvwm -cmd "Module FvwmM4 .fvwm2rc"
```

注意，选项"-cmd"的参数需要加引号。

FvwmM4 作为一个模块运行时，可以与 fvmw 异步运行。如果在.fvwm2rc 里调用 FvwmM4，则它生成的命令有可能不能够在执行.fvwm2rc 文件中的下一条命令时运行。按照下面的命令调用 FvwmM4 达到异步的目的：

```
ModuleSynchronous FvwmM4 -lock filename
```

选项 (OPTIONS) ：

-m4-prefix

使所有的 m4 指令要求前缀"m4_"。

-m4-prefix-defines

-m4opt option

传递 option 到m4 程序。

-m4-squote character

-m4-equote character

-m4prog name

使用 name 指定的程序替代调用“M4”

-outfile filename

指定输出文件的名称。注意，写这个文件前，FvwmM4 会尝试首先删除它，因此不要在它里面保存任何重要的信息，即使已经加上写保护。

-debug

保留 M4 输出的临时文件。它通常是"/tmp/fvwmrcXXXXXX"

-lock

如果希望使用这个选项，你需要使用 ModuleSynchronous 来启动 FvwmM4。-lock 使 fvwm 等待预处理过程结束，并在继续下一步之前，FvwmCpp 会请求 fvwm 读取预处理过的文件。

-noread

使 fvwm 不去读取预处理后的文件。

配置选项 (CONFIGURATION OPTIONS) ：

FvwmM4 定义了一些可以在预处理文件里使用的常量：

TWM_TYPE：

总是“fvwm”

SERVERHOST

运行 X server 的机器名

CLIENTHOST

运行 fvwm 的机器名

HOSTNAME

运行 fvwm 的主机名，通常与 CLIENTHOST 相同

OSTYPE

CLIENTHOST 上的操作系统

USER

运行 fvwm 的用户的名字

HOME

用户主目录

VERSION

X11 版本

REVISION

X11 版本的修订号

VENDOR

X server 厂商

RELEASE

X server 版本号

SCREEN

显示屏幕号

WIDTH

屏幕宽度

HEIGHT

屏幕高度

X_RESOLUTION

水平方向上的距离

Y_RESOLUTION

垂直方向上的距离

PLANES

BITS_PER_RGB

CLASS

COLOR

FVWM_CLASS

FVWM_COLOR

FVWM_VERSION

fvwm 版本号

OPTIONS

FVWM_MODULEDIR

fvwm 配置文件存放的目录。

FVWM_USERDIR

环境变量 $FVWM_USERDIR 的值

SESSION_MANAGER

环境变量 $SESSION_MANAGER 的值

示例 (EXAMPLE PROLOG)

```
define(TWM_TYPE,``fvwm'')dnl
define(SERVERHOST,``spx20'')dnl
define(CLIENTHOST,``grumpy'')dnl
```

```
    define(HOSTNAME,``grumpy'')dnl
    define(OSTYPE,``SunOS'')dnl
    define(USER,``nation'')dnl
    define(HOME,``/local/homes/dsp/nation'')dnl
    define(VERSION,``11'')dnl
    define(REVISION,``0'')dnl
    define(VENDOR,``HDS human designed systems, inc. (2.1.2-D)'')dnl
    define(RELEASE,``4'')dnl
    define(SCREEN,``0'')dnl
    define(WIDTH,``1280'')dnl
    define(HEIGHT,``1024'')dnl
    define(X_RESOLUTION,``3938'')dnl
    define(Y_RESOLUTION,``3938'')dnl
    define(PLANES,``8'')dnl
    define(BITS_PER_RGB,``8'')dnl
    define(CLASS,``PseudoColor'')dnl
    define(COLOR,``Yes'')dnl
    define(FVWM_VERSION,``1.24l'')dnl
    define(OPTIONS,``SHAPE XPM M4 '')dnl
    define(FVWM_MODULEDIR,``/local/homes/dsp/nation/modules'')dnl
    define(FVWM_USERDIR,``/local/homes/dsp/nation/.fvwm'')dnl
    define(SESSION_MANAGER,``local/grumpy:/tmp/.ICE-unix/440,tcp/spx20:1025'')dnl
```

Fvwm 中文手册-FvwmIconMan (一)

FvwmIconMan

名称 (NAME) :

FvwmIconMan -Fvwm 图标管理器 (icon manager)

概要 (SYNOPSIS) :

FvwmIconMan 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) :

FvwmIconMan 模仿 TWM 的图标管理器。用户可以有多个图标管理器，每个负责管理一类窗口。例如，用户可以有一个管理器只负责管理 emacs 窗口，同时可以有另外一个管理器负责管理所有其它窗口。你也可以分别指定每个图标管理器使用的策略，例如，可以指定一个管理所有桌面上的窗口，而另一个仅仅管理当前桌面（desk）、页（page）、屏幕（screen）上的窗口。FvwmIconMan 能够显示 fvwm 提供的窗口微型图标。图标管理器可以有多个列（垂直增长），也可以有多个行（水平增长），或者固定尺寸大小并调整窗口图标的尺寸来适应它。

你能指定鼠标、键盘事件关联的操作，例如，你可以首先绑定鼠标按键反图标化所选择的窗口，然后抛开鼠标，直接绑定箭头键来浏览这个管理器。

FvwmIconMan 能够显示当前哪个窗口拥有键盘焦点，通过绑定 select 事件到 fvwm 的 Focus 函数，你能模拟 TWM 的图标管理器。

初始化 (INITIALIZATION) :

在初始化期间，FvwmIconMan 搜索 fvwm 配置文件来发现用户配置的选项。建议将 FvwmIconMan 设置为一个 sticky 窗口。如果你想利用 followfocus 选项，或者绑定一个操作到 Focus，你应该使用 FvwmIconMan clicktofocus。使用 Shape 选项时，建议不要修饰 FvwmIconMan 窗口。

调用 (INVOCATION) :

可以在 .fvwmrc 文件里添加‘Module FvwmIconMan’命令来启动 FvwmIconMan。如果希望在初始化期间启动 FvwmIconMan，应该将这个命令添加到 StartFunction 函数里，否则可以将它绑定到菜单，鼠标，或键盘操作以便后来启动 FvwmIconMan。

如果你希望以 transient 模式使用 FvwmIconMan，比如内置在一个窗口列表里，则应该传递 Transient 参数，‘Module FvwmIconMan Transient’。使用这个模式时，FvwmIconMan 将在鼠标光标的位置弹出一个管理器窗口，鼠标按键释放后，它执行一定的操作然后退出。

FvwmIconMan 也可以接受一个别名作为参数，例如“Module FvwmIconMan FvwmIconMan-Variant2”。

配置选项索引 (CONFIGURATION OPTIONS REFERENCE CHART) :

FvwmIconMan 有非常多的选项，这里给出它们的索引，之后的部分将有更详尽的描述。

Name	Description	Default
NumManagers	number of managers	1
Action	binds command to event	Mouse 0 N sendcommand Iconify
Background	default background	gray
ButtonGeometry	size of button in pixels	
Colorset	default colorset	
DontShow	list of windows to ignore	

```
DrawIcons            use mini icons                false

FocusAndSelectButton                            flat grey black

FocusAndSelectColorset

FocusButton          style for focused buttons  up grey black

FocusColorset

FollowFocus         show which win has focus   false

Font                                              8x13

Foreground           default text color           white

Format               describes button label       "%c: %i"

IconName             manger icon name              FvwmIconMan

IconButton           style for icon buttons       up black grey

IconColorset

ManagerGeometry         size of manager in buttons 0x1

MaxButtonWidth      max width of a button

MaxButtonWidthByColumns

NoIconAction         animate iconification        NOP

PlainButton          style for normal buttons   up black grey

PlainColorset

ReliefThickness         size of button relief        2

Resolution           global/desk/page/screen      page

Reverse              normal, icon or none         none

SelectButton         style for selected buttons flat black grey

SelectColorset

Shape                 use shape extension          false

Show                  list of windows to show

ShowOnlyIcons       only icons visible           false

ShowNoIcons          icons are not displayer      false
```

```
ShowTransient       transient windows visible  false

Sort                  keep managers sorted         name

SortWeight           weight for sorting

Tips                  Tool Tips mode                none

TipsDelays           Tool Tips mapping delays   1000 300

TipsFont             Font for Tool Tips           default fvwm font

TipsColorset         Tool Tips Colorset           0

TipsFormat           describes Tips label         the Format value

TipsBorderWidth         Tool Tips border size        1

TipsPlacement       Tips placement vs button   updown

TipsJustification   Tips Just vs button              leftup

TipsOffsets          Tips placement Offsets       3 2

Title                 manager title                FvwmIconMan

TitleButton          style for title button       raisededge black grey

TitleColorset

UseWinList           honor WinListSkip?           true
```

配置选项（CONFIGURATION OPTIONS）：

除 nummanagers 选项外，其它的选项都可以针对某一个管理器定义。例如，你可以有一个红色背景的 emacs 管理器，同时也可以有一个蓝色背景的 xterm 管理器。因此，一个配置语句可能有两种形式：

```
*FvwmIconMan: OptionName OptionValue
```

选项 OptionName 的值 OptionValue 适用于每个管理器。

```
*FvwmIconMan: ManagerId OptionName OptionValue
```

选项 OptionName 的值 OptionValue 只适用于 ID 为 ManagerId 的管理器。MangerId 可以是一个正整数，也可以是字符串"transient"。正整数 ID 是指正常运行时 FvwmIconMan 创建的管理器，"transient"是指 FvwmIconMan 临时运行时创建的管理器。

下面是可供使用的各个选项：

```
*FvwmIconMan: NumManagers num
```

num 表示图标管理器的总数。FvwmIconMan 希望在处理一个特定选项的之前知道一共有多少管理器，它应该首先被设置，默认为 1。

*FvwmIconMan: [id] Action type binding

绑定一个 FvwmIconMan 命令到一个事件。Type 的值可以是：Key, Mouse, 或 Select。Action 在下面的 ACTIONS 部分详细描述。

*FvwmIconMan: [id] Background background

指定默认的背景色。

*FvwmIconMan: [id] ButtonGeometry geometry

以像素为单位指定单个按钮的初始尺寸。如果 height 为 0，则按钮的高取决于字体的大小。X 和 Y 坐标被忽略。

*FvwmIconMan: [id] Colorset colorset

默认的 colorset。覆盖 background 和 foreground 的设置。参看 FvwmTheme。

*FvwmIconMan: [id] DrawIcons value

如果你的 fvwm 版本能够使用 mini icons，这个选项可以决定是否显示 mini icons。否则，它将产生一个错误消息。"true"表示 mini icons 当窗口图标化时显示，"false"表示 mini icons 从不显示，"always"表示 mini icons 总是显示。

*FvwmIconMan: [id] PlainButton style [forecolor backcolor]

指定普通按钮如何显示。style 可以是：flat，up，down，raisededge，或 sunkedge。两个颜色选项是可选的，如果没有被设置，则使用默认的颜色。在单色屏幕 (monochrome screen) 上，style 选项将被忽略，但仍然必须被设置。

*FvwmIconMan: [id] PlainColorset colorset

类似 PlainButton，但指定的是 colorset。style 设置对 PlainButton 仍然有效。参看 FvwmTheme。

*FvwmIconMan: [id] FocusAndSelectButton style [forecolor backcolor]

类似 plainbutton 选项，指定被选中并有键盘焦点时按钮如何显示。

*FvwmIconMan: [id] FocusAndSelectColorset colorset

作用类似 focusandselectbutton，但指定的是 colorsets。style 设置对 focusandselectbutton 仍然有效。参看 FvwmTheme。

*FvwmIconMan: [id] FocusButton style [forecolor backcolor]

类似 plainbutton，指定拥有键盘焦点时按钮如何显示。参看 FvwmTheme。

*FvwmIconMan: [id] FocusColorset colorset

类似 FocusButton，但指定的是 colorsets。style 设置对 FocusButton 仍然有效。参看 FvwmTheme。

*FvwmIconMan: [id] IconButton style [forecolor backcolor]

类似 plainbutton，指定图标化时按钮如何显示。

*FvwmIconMan: [id] IconColorset colorset

类似 IconButton，但指定的是 colorsets。style 设置对 IconButton 仍然有效。参看 FvwmTheme。

*FvwmIconMan: [id] SelectButton style [forecolor backcolor]

类似 plainbutton，指定鼠标光标位于按钮上方时按钮如何显示。

*FvwmIconMan: [id] SelectColorset colorset

类似 SelectButton，但指定的是 colorset。style 设置对 slectbutton 仍然有效。参看 FvwmTheme。

上面五组选项，如果 colorset 被设置，则 forecolor backcolor 将不起作用，但是 style 仍然有效。因此，如果 colorset 设置的时候，forecolor backcolor 就可以不再设置。

*FvwmIconMan: [id] FollowFocus boolean

如果 boolean 为 true，按钮的外观反映出当前哪个窗口拥有焦点。默认 false。

*FvwmIconMan: [id] Font font

指定按钮标签的字体。默认是 8x13

*FvwmIconMan: [id] Foreground foreground

指定默认的前景色。

*FvwmIconMan: [id] Format formatstring

一个 printf 格式的字符串，描述管理器内按钮标签内容的显示格式。可能的 flag 有：%t,%i,%c,和%r，分别表示窗口的 title、icon、class 和 resource name。默认是"%c:%i"。

*FvwmIconMan: [id] IconName iconstring

指定管理器窗口的图标名称。Iconstring 可以是单个单词或被加上引号的字符串。默认是“FvwmIconMan”。

*FvwmIconMan: [id] ManagerGeometry geometry

指定管理器的初始位置和尺寸，以按钮为单位。如果 height 为 0，则这个管理器将有 width 列，一旦它拥有多于 width 个窗口，它将垂直伸长。同样，如果 width 为 0，则它有 height 行，并水平伸长。如果两个都为非 0 值，则该管理器窗口将有确定的大小。如果指定负的 y 坐标，这个管理器将向上伸长。否则，向下伸长。

*FvwmIconMan: [id] MaxButtonWidth width

以像素为单位定义按钮的最大宽度。默认没有最大值。为 0 表示恢复默认设置（没有最大值）。最大值仅适用于固定大小的管理器（width 和 height 都被指定为非 0 值）。

*FvwmIconMan: [id] MaxButtonWidthByColumns col

设置按钮宽度的另外一种方式。col 是图标的列数。按钮宽度取决于 FvwmIconMan 的宽度除以列数。例如，如果 FvwmIconMan 管理器窗口的宽度为 1024，MaxButtonWidthByColumns 值为 4，则MaxButtonWidth 为 256。当你不知道管理器窗口宽度的时候，它将非常有用。

*FvwmIconMan: [id] NoIconAction action

指定一个 NoIcon 风格的窗口图标化和反图标化时 FvwmIconMan 执行的 action。一个 action 的例子："*FvwwmIconMan: NoIconAction SendToModule FvwmAnimate animate"。

*FvwmIconMan: [id] ReliefThickness num

指定 non-flat 按钮（up，down，raisededge，或 sunkedge）边缘的形状（凹凸效果，正数时凹下，负数时凸起）。如果 num 为 0，所有 FocusAndSelectButton, FocusButton, IconButton, PlainButton, SelectButton, 和 TitleButton 将重置为 flat 按钮。如果 num 为负，按钮将凸起，好像针对所有按钮类型都设置了 Reverse 一样。

*FvwmIconMan: [id] Resolution resolution

指定管理器在什么时候显示哪些窗口，resolution 可以采用下面的值：global，desk，page，screen，!global，!desk，!page，或!screen。global 表示所有窗口都被显示。desk 表示仅仅那些当前 desk 上的窗口被显示。page 表示仅仅那些当前 page 上的窗口被显示。screen 表示仅仅当前 Xinerama 屏上的窗口被显示。!desk 与 desk 相反，表示仅仅显示不在当前 desk 上的窗口。同样，!page 仅仅显示那些不在当前 page 上的窗口，!screen 仅仅显示那些不在当前 Xinerama 屏上的窗口。默认为 page。如果 Xinerama 没有激活，或者仅有一个屏幕被使用，则 page 和 screen 产生同样的效果。

FvwmIconMan 运行时 resolution 可以被动态的改变。

*FvwmIconMan: [id] Reverse class

使某类按钮的 relef 配置，up 和 down 风格颠倒。对 flat 按钮无效。class 可以是 icon，normal，none。默认为 none。

*FvwmIconMan: [id] Shape boolean

如果为 True，窗口被 shaped。或许仅当你有多行或多列时有用。如果 FvwmIconMan 编译时不支持 Shape 扩展，这将引发一个错误。当使用 shaped 窗口时，建议去掉 FvwmIconMan 窗口的边框。否则，fvwm 将混淆。

*FvwmIconMan: [id] Sort value

name 表示管理器内图标按名字排列。namewithcase 表示按名字排列，但大小写敏感。id 表示按窗口 id 排列，窗口 id 在窗口创建后从不改变。weighted 表示按 weight 排列（参看 sortweight 的描述）。为 none，不按一定的顺序排列。默认为 name。

*FvwmIconMan: [id] SortWeight weight pattern-list

分配指定的 weight 给匹配 pattern-list 的窗口。这个列表由 type=pattern 形式的字段组成，type 可以是 class，resource，title，或 icon；pattern 和 style 命令的格式相同。可以指定多个 sort weights，每个窗口按照顺序去匹配，并被设置成第一个匹配到的 weight。低 weight 值的窗口被放置在管理器列表的开始。例如：

*FvwmIconMan: Sort weighted

FvwmIconMan: SortWeight 1 class=XTerm title=special

*FvwmIconMan: SortWeight 10 class=XTerm

*FvwmIconMan: SortWeight 5

上面的例子里，标题以"special"开始的 xterm 窗口（weight 1）被放在列表的开始，后面是其它的非 xterm 窗口（weight 5），而标题不以"special"开始的 xterm 窗口位于列表的最后。默认的 weight 为 0（仅当 sort 类型为 weight 时有效）。

*FvwmIconMan: [id] Title title-string

指定管理器窗口的标题字符串。titlestring 可以是单个单词，或者加上引号的字符串。默认是"FvwmIconMan"。它将显示在管理器窗口的标题栏（title bar）上。

*FvwmIconMan: [id] TitleButton style [forecolor backcolor]

类似 plainbutton，指定标题按钮（title button）（管理器为空时显示这个按钮）如何显示。管理器的标题显示在标题按钮里面。

*FvwmIconMan: [id] UseWinList boolean

true 表示如果某类应用使用了 WindowListSkip 属性，那么管理器不收集它们。否则，所有的窗口依据 show 和 dontshow 列表接受管理。

Fvwm 中文手册-FvwmIconMan（二）

下面的两个选项控制哪个窗口将被哪个管理器处理。一个管理器可以有两个列表：Show 和 DontShow。如果仅指定 Show 列表，管理器将仅显示该列表里的窗口。如果仅指定 DontShow 列表，管理器将显示所有该列表之外的窗口。如果同时指定两个列表，不在 DontShow 列表而在 Show 列表里的窗口将被显示。最后，如果两个列表都没有指定，管理器将处理所有的窗口。每个列表由 type=pattern 形式的字段组成，type 可以是 class,resource，title,或 icon；pattern 和 style 命令里的格式相同。如果一个窗口能够被多个管理器操作，id 更小的管理器将获得它。

*FvwmIconMan: [id] Show pattern list

窗口匹配 pattern 列表中的一个时，它将能被该管理器处理。

*FvwmIconMan: [id] DontShow pattern list

窗口匹配 pattern 列表中的一个时，它将不能被该管理器处理。

*FvwmIconMan: [id] ShowTransient boolean

在管理器的列表里显示临时窗口，默认为 false。临时窗口是指应用操作过程中产生的弹出窗口等，比如点击 gedit 菜单中的"打开"选项，会弹出一个窗口让你选择要打开的文件，这个窗口就是临时窗口。默认这样的窗口是不在管理器中显示的。

*FvwmIconMan: [id] ShowOnlyIcons boolean

true 表示仅显示图标化的窗口。

*FvwmIconMan: [id] ShowNoIcons boolean

true 表示仅显示没有图标化的窗口。

*FvwmIconMan: [id] ShowOnlyFocused boolean

true 表示仅显示焦点窗口。

下面的两个选项控制 tips：

*FvwmIconMan: [id] Tips value

value 可以是 always，needed 或 false。默认为 false，不显示 tips。always，tips 总是被显示。needed 表示仅当按钮字符串被截短或 tip 字符串不等于按钮字符串时显示 tip。

*FvwmIconMan: [id] TipsDelays delay [mappeddelay]

delay 和 mappeddelay 表示毫秒为单位的超时时间。如果 mappeddelay 值没有明确的指定，将假设它等于 delay。默认是 1000 300。当鼠标光标移动到一个按钮上时，FvwmIconMan 在显示 tip 前需要等待 delay 毫秒。如果 tip 已经被绘制但此时光标移动到了另一个按钮，FvwmIconMan 在显示一个新的 tip 前需要等待 mappeddelay 毫秒。

*FvwmIconMan: [id] TipsFont fontname

指定 tip 的字体。默认是 fvwm 字体。

*FvwmIconMan: [id] TipsColorset colorset

指定 tips 窗口的颜色。默认是 colorset 0。参看 FvwmTheme。

*FvwmIconMan: [id] TipsFormat formatstring

类似 Format 选项，表示 tips 显示的格式。默认是 Format 选项的格式字符串。

*FvwmIconMan: [id] TipsBorderWidth pixels

以像素为单位指定 tips 窗口的边框宽度。默认为 1。

*FvwmIconMan: [id] TipsPlacement value

value 可以是：up，down，left，updown，或 leftright。指定了 tips 窗口相对于按钮的位置。默认为 updown，当按钮位于屏幕的上半部时 tip 位于按钮下面，否则相反。

*FvwmIconMan: [id] TipsJustification value

value 可以是 leftup，rightdown，或 center。指定了 tips 窗口放置时相对于按钮的方向。默认是 leftup，表示 tip 位于按钮的上面或下面时，tip 和按钮的左边框对齐，tip 位于按钮的左面或右面时，它们的顶部边框对齐。rightdown 和 center 类似 leftup，只是方向有所区别。

*FvwmIconMan: [id] TipsOffsets placementoffset justoffset

placementoffset 和 justoffset 分别是 TipsPlacement 和 TipsJustification 选项的偏移值。默认为 3 2。

操作（ACTIONS）：

Actions 是绑定到某一事件的命令。事件的类型可以是：键盘按键， 鼠标点击， 或者鼠标光标移动到管理器，绑定的命令类型分别为：Key，Mouse，和 Select。

通常，与鼠标单击绑定的命令当鼠标按键按下时执行。但是在使用 transient 模式时，它在鼠标按键释放时执行。提示：FvwmIconMan 可以在识别鼠标按键的同时识别键盘修饰键，比如说，如果将 FvwmIconMan 绑定到meta-button3，则当 meta-button3 事件发生时，actions 将被执行。

actions 的语法如下：

Key actions: Key Keysym Modifiers FunctionList

Keysym 和 Modifiers 与 fvwm 的 Key 命令一样含义。

Mouse actions: Mouse Button Modifiers FunctionList

Button 和 Modifiers 与 fvwm 的 Mouse 命令一样含义。

Select actions: Select FunctionList

FunctionList 是以逗号为分隔的命令序列。它们以从左到右的顺序执行，并且共享同一上下文—目前仅包括指向当前按钮（“current” button）的指针。如果在 actions 执行时，选中了一个按钮，则当前的按钮初始化为那个选中的按钮。

大部分命令通过下面的方式修改当前按钮：移动它，使它变成选中的按钮，发送命令到它代表的窗口上。注意，当这个当前按钮被初始化为所选中的按钮时，那个按钮并不随当前按钮移动，通过这种方式，在不改变所选中按钮的情况下，用户能够发送命令到不同的窗口。

命令可以接受五种类型的参数：Integer, Manager, Window, Button, 和 String。String 如果是单个单词，可以不加引号。通过在命令之间添加逗号，你可以绑定一个命令序列到一个事件上。

Window 和 Button 的类型在.fvwm2rc 文件里看起来是一样的，但是它们分别描述一个管理器窗口，和代表一个窗口的 FvwmIconMan 按钮。它们可以是一个整数，或下列字符串之一：Select, Focus, Up, Down, Right, Left, Next, Prev。Select 和 Focus 指向当前选中的或拥有焦点的按钮或窗口。Up, Down, Right, Left 指向位于管理器中当前按钮上下左右方向的按钮或窗口。Next 和 Prev 指向当前按钮后面和前面的按钮和窗口，如果管理器中的按钮按照一定顺序排列，则 Next 和 Prev 依赖排序的顺序。

Manager 的类型可以是 integer, Next, 或 Prev。含义类似于 button 类型。

下面是当前支持的命令：

bif Button Integer/String

bifn Button Integer/String

gotobutton Button

gotomanager Manager

jmp Integer/String

跳转命令。不允许向后跳转。

label String

提供 jmp 命令使用的标签。

print String

输出 String 到控制台，供调试使用。

printdebug

输出已定义的 action 到控制台，仅供开发者使用。

quit

退出 FvwmIconMan。

refresh

刷新所有的管理器窗口。

ret

停止执行完整的 action。

searchback String

searchforward String

select

选择当前的按钮。

sendcommand Command

发送 fvwm 命令 Command 到当前按钮代表的窗口。

warp

warp 光标到当前按钮。

例子：

```
gotobutton select, gotobutton Down, select


gotobutton Up, select


gotobutton 0, select


gotobutton -1, select


gotobutton focus, select


gotobutton focus, Iconify


bif Next 3, gotobutton 0, select, ret, gotobutton Next, select


bif select 7, bif focus 3, gotomanager 0, select, ret, gotobutton focus, \
       select, ret, gotobutton down, select


bif select Select, bif focus Focus, gotomanager 0, select, ret, label Focus, \
       gotobutton focus, select, ret, label Select, gotobutton down, select
```

除了能够绑定到键盘和鼠标，actions 也能够通过 SendToModule 命令从 FvwmIconMan 发送到 fvwm。此时，commond 不需要加引号。

配置示例（SAMPLE CONFIGURATIONS）：

下面的例子有一个管理器，处理所有的窗口。

```
############################################################

# Load any modules which should be started during

# fvwm initialization

ModulePath /usr/lib/X11/fvwm:/usr/bin/X11

Module FvwmIconMan


# Make FvwmIconMan title-bar-less, sticky, and give it an icon

Style "Fvwm*"      Icon toolbox.xpm,NoTitle,NoHandles,Sticky

Style "FvwmIconMan" HandleWidth 5, Handles, BorderWidth 5


############################################################

############################################################

#Definitions used by the modules


*FvwmIconMan: NumManagers        1

*FvwmIconMan: Resolution         global

*FvwmIconMan: Background         slategrey

*FvwmIconMan: Foreground         white

*FvwmIconMan: Font               7x13

*FvwmIconMan: ButtonGeometry     100x0

*FvwmIconMan: ManagerGeometry    1x0-0+0
```

Fvwm 中文手册-FvwmPager

FvwmPager

名称（NAME）：

FvwmPager –FVWM Pager 模块。

概要（SYNOPSIS）：

FvwmPager [–transient] [name] [first desk [last desk]]

FvwmPager 只能被 fvwm 调用(fork),不能从命令行启动。

desk 号在 first dest 和 last dest 之间的所有桌面被显示。如果 last desk 为空，则仅 first desk 显示。如果两者都为空，则显示当前桌面。如果 first desk 为‘*’，则pager 将总是显示当前桌面，即使你切换到其它桌面。

可以在.fvwm2rc 文件里添加下面的语句启动 FvwmPager：

```
Module FvwmPager 0 3
```

或

```
Module FvwmPager *
```

也可以绑定到一个弹出菜单里面：

```
AddToMenu Module-Popup Modules Title

+ Audio        Module FvwmAudio

+ Auto         Module FvwmAuto 200

+ Buttons      Module FvwmButtons

+ Console      Module FvwmConsole

+ Ident        Module FvwmIdent

+ Banner       Module FvwmBanner

+ Pager        Module FvwmPager 0 3
```

或

```
+ Pager        Module FvwmPager *
```

如果启动时使用-transient 选项，则按钮释放时，pager 将关闭。注意，仅当 pager 窗口的风格为‘sticky’时，这个选项才有效。

例子：

```
Style FvwmPager Sticky, StaysOnTop

*FvwmPager: Rows       1

*FvwmPager: Columns    1
```

```
Mouse 3 R C Module FvwmPager -transient
```

如果在根窗口上同时按下 ctrl 和鼠标按键 3，pager 在鼠标光标的位置弹出，同时视口随鼠标移动。

描述（DESCRIPTION）：

FvwmPager 模块显示参数里指定的 Fvwm 桌面的微型视图。这可以提醒你活动窗口的位置。pager 里的窗口使用它们的窗口修饰同样的颜色显示。

pager 可以用来改变你的视口（viewport）到当前桌面。

在 pager 里面，按鼠标按键 1 将使你的视口改变到所选中的桌面的选中 page。如果在 desk-lable 区域上单击鼠标按键 1，你将切换桌面，而不是切换桌面内的 page。

在窗口的微型视图上拖拉鼠标按键 2，将使窗口移动到鼠标按键释放的地方，但你的视口并不改变。如果你拖拉这个窗口到 pager 外面并且拖拉到你的桌面上，这个窗口将完整的显示出来。没有将一个完整的窗口移动到一个 pager 里面的方法。因为一些鼠标没有按键 2，所以也可以在按下 modifier-1（通常是 Alt）的同时拖拉鼠标按键 3 在 pager 内移动窗口。

在一个位置单击鼠标按键 3，将使视口切换到所选择的位置，并在必要的时候切换桌面，但是并不对齐视口到页面（page）的边界。拖拉鼠标按键 3 将移动视口，就好像你拖拉视口一样，但是并不切换桌面，即使鼠标指针移动到了另一个桌面。

初始化（INITIALIZATION）：

初始化期间，FvwmPager 从 fvwm 的模块配置数据库获得配置信息。

可以使用 name 参数调用 FvwmPager，这可以让你同时使用几个不同配置的 FvwmPager，例如"Module FvwmPager OtherPager"，OtherPager 将仅仅读取以"*OtherPager"开始的配置选项。

键盘焦点控制（KEYBOARD FOCUS CONTROL）：

你可以通过单击鼠标按键 2，使键盘焦点移动到当前桌面的任一窗口。这个窗口不必是可见的，但它必须在当前 page 里。

调用（INVOCATION）：

参看概要部分。

配置选项（CONFIGURATION OPTIONS）：

```
*FvwmPager: Geometry geometry
```

完全或部分地指定 pager 窗口的尺寸和位置。为了使窗口比例协调，你可能希望忽略 geometry 的 width 或 height。

```
*FvwmPager: Rows rows
```

pager 窗口内包含桌面的行数。

```
*FvwmPager: Columns columns
```

pager 窗口内包含桌面的列数。

*FvwmPager: IconGeometry geometry

指定 pager 图标窗口的尺寸和位置。

*FvwmPager: StartIconic

使 pager 启动时图标化。

*FvwmPager: NoStartIconic

使 pager 正常方式启动。

*FvwmPager: LabelsBelow

在 desk 下面显示相应的 desk 标签。

*FvwmPager: LabelsAbove

在 desk 上面显示相应的 desk 标签。

*FvwmPager: ShapeLabels

隐藏所有桌面的标签，除了当前 desk。

*FvwmPager: NoShapeLabels

显示所有可见桌面的标签。

*FvwmPager: Font font-name

指定桌面标签使用的字体。如果 font-name 为"none"，桌面标签不显示。

*FvwmPager: SmallFont font-name

指定 pager 里窗口标签使用的字体。如果没有指定，则窗口标签被忽略。如果 font-name 为"none"，则不显示任何窗口的名称。

*FvwmPager: Fore color

指定桌面标签的文本和 page 之间网格（page-grid）使用的颜色。

*FvwmPager: Back color

指定 FvwmPager 窗口的背景色。

*FvwmPager: Hilight color

活动桌面和 page 被高亮显示时的颜色。

*FvwmPager: HilightPixmap pixmap

活动 page 被高亮显示时的 pixmap。

*FvwmPager: DeskHilight

使用当前的高亮 color/pixmap 高亮显示活动 page。常用于取消 NoDeskHilight 选项的设置。

*FvwmPager: NoDeskHilight

不高亮显示活动 page。

*FvwmPager: WindowColors fore back hiFore hiBack

改变窗口正常和高亮时（normal/highlight）的颜色。fore 和 hiFore 指定窗口内部字体使用的颜色。back 和 hiBack 指定填充窗口使用的颜色。

*FvwmPager: WindowLabelFormat format

指定窗口（mini window）标签使用一个 printf 风格的格式。可能的 flag 是：%t,%i,%c 和%r 分别对应窗口的 title，icon，class，resource name。默认为%i。

*FvwmPager: Label desk label

为 pager 窗口里的 desk 分配文本 label。

```
*FvwmPager: Label 1 Mail

*FvwmPager: Label 2 Maker

*FvwmPager: Label * Matlab
```

注意，目前有更好的方法，即使用 DesktopName 命令，设置全局的桌面名称（不仅在 FvwmPager 里），因此可以不再使用这个选项。

*FvwmPager: DeskColor desk color

为 page 窗口里的 desk 分配颜色 color。这覆盖了 desk 的背景色。仅当 pager 处于实际大小时有效。图标化时，pager 使用 *FvwmPager: Back 选项指定的颜色。

*FvwmPager: Pixmap pixmap

指定 pager 的背景为 pixmap。

*FvwmPager: DeskPixmap desk pixmap

为 pager 窗口里的 desk 分配 pixmap。这覆盖了 desk 的背景 pixmap。

*FvwmPager: DeskTopScale number

如果没有指定 geometry，则一个桌面换算系数（desktop reduction factor）用来计算 pager 的尺寸。pager 窗口里的部件按实际尺寸的 1/number 显示。

*FvwmPager: MiniIcons

允许 pager 显示一个窗口的 mini 图标（如果有的话），而不是显示窗口的名称。

*FvwmPager: MoveThreshold pixels

定义一个窗口在使用鼠标按键 2 拖拉发生移动前，鼠标指针需要移动的距离。默认为 3 个像素。如果鼠标指针移动的距离小于 pixels，则该窗口回到它原来所在的位置。如果 pixels 小于 0，则使用默认值。

*FvwmPager: SloppyFocus

如果指定了 SloppyFocus 选项，你就不再需要单击 pager 里的 mini 窗口来使窗口获得焦点。简单的将鼠标指针放在 pager 内这个窗口的上面就可以了。

*FvwmPager: SolidSeparators

pager 窗口里虚拟桌面的 page 默认使用虚线分隔，这个选项更改为使用实线。

*FvwmPager: NoSeparators

取消 pager 窗口里虚拟桌面的 page 之间的分隔线。

*FvwmPager: Balloons [type]

鼠标指针移动到 pager 内的窗口上面时，显示描述这个窗口提示信息。默认格式可以使用 BalloonStringFormat 来改变。如果 type 为 Pager，则只对非图标化的窗口起作用。如果 type 为 Icon，则只对图标化的窗口起作用。除此之外，提示信息总是显示。

*FvwmPager: BalloonFore color

指定提示信息的文本色。

*FvwmPager: BalloonBack color

指定提示窗口的背景色。

*FvwmPager: BalloonFont font-name

指定提示信息文本使用的字体。

*FvwmPager: BalloonBorderWidth number

设置提示窗口边框的宽度。默认为 1。

*FvwmPager: BalloonBorderColor color

设置提示窗口边框的颜色。默认为黑色。

*FvwmPager: BalloonYOffset number

*FvwmPager: BalloonStringFormat format

类似*FvwmPager: WindowLabelFormat，仅指定提示窗口里显示的字符串的格式，默认为"%i"。

*FvwmPager: Colorset desk colorset

指定 desk 使用的 colorset。如果设置 desk 为"*"，则 colorset 对所有 desk 都有效。

*FvwmPager: BalloonColorset desk colorset

指定 desk 上的提示窗口使用的 colorset。如果设置 desk 为"*"，则 colorset 对所有 desk 都有效。

*FvwmPager: HilightColorset desk colorset

以*FvwmPager: WindowColors 的方式使用 colorset。

*FvwmPager: WindowBorderWidth n

指定 mini 窗口周围的边框宽度。同时设置 mini 窗口的最小尺寸为(2 * n + 1)，默认为 1。

*FvwmPager: Window3DBorders

指定 mini 窗口应该有一个 3d 效果的边框。

*FvwmPager: UseSkipList

指定 FvwmPager 不显示 WindowListSkip 风格的窗口。

Fvwm 中文手册-FvwmProxy

FvwmProxy

名称（NAME）：

FvwmProxy -FVWM Proxy 模块。

概要（SYNOPSIS）：

FvwmProxy 只能被 fvwm 调用(fork)，不能从命令行启动。

描述（DESCRIPTION）：

通过使用不相重叠的小的代理窗口，FvwmProxy 允许用户定位和控制被其它窗口覆盖的窗口。默认的功能包括提升（raise）和降低（lower）所代理的窗口。

使用简单的配置，按下 Alt-Tab 组合键将循环这些窗口，并允许在这些 proxy 上单击操作。释放 Alt 键解除 proxy 窗口。默认，一个 proxy 窗口上鼠标左键和右键提升或降低关联的窗口。

Proxy 窗口总是顶层窗口，并设法收集它们代理的窗口。

调用（INVOCATION）：

可以在配置文件里添加'Module FvwmProxy'命令来调用 FvwmProxy。如果在 fvwm 初始化期间启动，则这个命令可以作为单独的行添加，否则，它可以和菜单、鼠标和键盘操作绑定在一起。Fvwm 在 ModulePath 指定的目录里查找 FvwmProxy。

配置选项：

*FvwmProxy: Colorset n

指定没有选中时 proxy 窗口的 colorset。

*FvwmProxy: SelectColorset n

指定选中时 proxy 窗口的 colorset。

*FvwmProxy: IconifiedColorset n

指定图标化时 proxy 窗口的 colorset。只在 ProxyIconified 选项打开时有意义。

*FvwmProxy: Font font

指定所有 proxy 窗口文本的字体。

*FvwmProxy: SmallFont font

*FvwmProxy: Width w

指定每个 proxy 窗口的宽度。默认为 180。

*FvwmProxy: Height h

指定每个 proxy 窗口的高度。默认为 60。

*FvwmProxy: Separation d

指定 proxy 窗口之间的最小距离。默认为 10。

*FvwmProxy: ShowMiniIcons bool

如果为 true，proxy 窗口显示它们所代理的窗口的 mini 图标（如果有的话）。默认为 true。

*FvwmProxy: EnterSelect bool

如果为 true，鼠标移动到 proxy 上面时 proxy 自动被选中，即使没有按下鼠标键。默认为 false。

*FvwmProxy: ProxyMove bool

如果为 true，移动一个 proxy 窗口将移动它代理的窗口。

*FvwmProxy: ProxyIconified bool

如果为 true，当窗口图标化时，继续被代理。

*FvwmProxy: Action mouseaction response

当给定的 action 执行时，FvwmProxy 做出指定的反应 response。当前支持的鼠标操作是：Click1, Click2, Click3 等等。默认，这个模块共支持 3 个鼠标按键，但可以支持更多。Click1, Click2, Click3 默认对应的 response 分别是 Raise，Nop，和 Lower。

*FvwmProxy: Action Select command

*FvwmProxy: Action Show command

指定一个 FvwmProxy Show 命令执行期间调用的 fvwm 函数。默认为 Nop。

*FvwmProxy: Action Hide command

指定一个 FvwmProxy Hide 命令执行期间调用的 fvwm 函数。默认为 Nop。

*FvwmProxy: Action Abort command

指定一个 FvwmProxy Abort 命令执行期间调用的 fvwm 函数。默认为 Nop。

*FvwmProxy: Action Mark command

*FvwmProxy: Action Unmark command

*FvwmProxy: Action ModifierRelease modifiers command

指定一个 proxies 已经被显示且指定的修饰符（modifiers）被释放的时候调用的 fvwm 函数。modifiers 使用 Mouse 命令里同样的语法。command 默认为 Nop。

*FvwmProxy: Group groupname command pattern

*FvwmProxy: Group groupname flag

*FvwmProxy: SlotWidth w

*FvwmProxy: SlotHeight h

*FvwmProxy: SlotSpace d

*FvwmProxy: GroupSlot n

*FvwmProxy: GroupCount n

*FvwmProxy: SlotStyle n style

*FvwmProxy: SlotAction n mouseaction response

*FvwmProxy: UndoLimit n

指定了 undo 缓冲里储存的操作的数目。决定了你可以取消之前的操作步数。默认为 8。

命令 (COMMANDS) :

SendToModule FvwmProxy Show

激活当前 desk 上所有窗口的 proxy，除了 WindowListSkip 风格的窗口。如果 desk 被切换，新的 proxies 被自动生成。

SendToModule FvwmProxy Hide

解除所有 proxy 窗口。如果一个 proxy 被选择，Select 操作在这个 proxy 代表的窗口上调用。默认操作包括提升窗口并 warp 鼠标到窗口上的某个位置。

SendToModule FvwmProxy ShowToggle

如果被显示，则隐藏。如果隐藏则显示。

SendToModule FvwmProxy Abort

解除所有 proxy 窗口。与 Hide 命令不同的是没有操作被执行。

SendToModule FvwmProxy Circulate command

通知 FvwmProxy 运行一个条件命令并标记结果。内置命令 SendToModule FvwmProxy Mark 在可选的条件后自动附加。

SendToModule FvwmProxy Next (temporary)

SendToModule FvwmProxy Prev (temporary)

SendToModule FvwmProxy SoftToggle

SendToModule FvwmProxy IsolateToggle

SendToModule FvwmProxy PrevIsolated

SendToModule FvwmProxy NextIsolated

SendToModule FvwmProxy Undo

SendToModule FvwmProxy Redo

配置示例 (SAMPLE CONFIGURATION) :

下面是在.fvwm2rc 文件里调用 FvwmProxy 的例子:

```
    Key Tab A M SendToModule FvwmProxy Circulate \
            ScanForWindow East South (CurrentPage)
    Key Tab A SM SendToModule FvwmProxy Circulate \
            ScanForWindow West North (CurrentPage)
    *FvwmProxy: Action ModifierRelease M SendToModule FvwmProxy Hide
```

但 Meta-Shift-Tab 组合键很难使用，因此 Meta-Q 可能是更好的选择。

```
    Key Q A M SendToModule FvwmProxy Circulate \
         ScanForWindow West North (CurrentPage)
```

为了让 proxies 当你按下 Alt 键时立即弹出，添加

```
    Key Meta_L A N SendToModule FvwmProxy Show
```

Fvwm 中文手册-FvwmRearrange

FvwmRearrange

名称 (NAME) :

FvwmRearrange - 重新排列 FVWM 窗口。

概要 (SYNOPSIS) :

FvwmRearrange 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) :

这个模块可以用来平铺（tile）或者层叠（cascade）窗口。

平铺时，遵守一定的限制，FvwmRearrange 尝试平铺当前屏幕上的窗口。水平或垂直平铺被执行，以便每个窗口之间不会互相重叠，默认每个窗口被缩放到它最接近的缩放增量（resize increment）（这就是有时平铺之后窗口之间会出现一些间隙的原因）。

层叠时，遵守一定的限制，FvwmRearrange 尝试层叠当前屏幕上的窗口。层叠时，将执行分层（layering）操作，每个窗口都要让位于它下面的前一个窗口标题可见。

调用（INVOCATION）：

最好通过菜单、按钮等方式调用 FvwmRearrange。有很多选项用来限制分层（layering）。作为一个例子，你可以使用下面的方式调用 FvwmRearrange：

```
FvwmRearrange -tile -h 10 10 90 90
```

或

```
FvwmRearrange -cascade -resize 10 2 80 70
```

第一个例子将在参数指定的范围内平铺窗口，第二个例子将在参数指定的范围内层叠窗口，因为指定了-resize，窗口将被缩放到受限的宽和高。参数中的数字表示的是屏幕尺寸的百分比。

FvwmRearrange 模块可以使用 FvwmTile 或 FvwmCascade 命令调用，它们相当于分别加上了-tile 或-cascade 选项。这种调用形式已经被废除，但是为了提供向后兼容，仍然可以在新版本里使用。

下面描述的是调用 FvwmRearrange 时可以使用的参数：

-a

影响所有类型的窗口，即使 WindowListSkip 风格的窗口也不例外。

-animate

尝试在窗口移动时添加动画效果，使用-resize 或-maximize 选项时被忽略。

-cascade

层叠窗口。是默认操作。这个选项必须位于所有其它选项的前面。

-desk

平铺和层叠当前桌面上的所有窗口，而并不仅仅是当前屏幕上的窗口。

-flatx

禁止增加边框的宽度，仅当层叠时使用。

-flaty

禁止增加边框的高度，仅当层叠时使用。

-h

水平平铺（默认为垂直平铺），仅当平铺操作时使用。

-incx arg

添加到层叠窗口的水平增量，arg 表示屏幕宽度的百分比，或者在后缀 p 时表示像素值。默认为 0。只用于层叠操作时。

-incy arg

添加到层叠窗口的垂直增量，arg 表示屏幕高度的百分比，或者在后缀 p 时表示像素值。默认为 0。只用于层叠操作时。

-m

指定影响最大化的窗口。（暗含于-a 选项）

-maximize

移动或缩放窗口时，将它置于最大化的状态。

-mn arg

-noanimate

移动时不添加动画效果。

-nomaximize

不将窗口置于最大化的状态。

-noraise

-noresize

-nostretch

-r

颠倒窗口的顺序。

-resize

强制所有窗口缩放到受限的宽和高。平铺时这是默认的。

-s

影响 sticky 风格的窗口。（暗含于-a 选项）

-sp

-sd

-t

影响临时窗口

-tile

平铺窗口。这个选项必须位于所有其它选项的前面。

-u

影响没有标题的（untitled）窗口。（暗含于-a 选项）

最后，还有四个数字参数，第一对数字表示第一个窗口的 x 和 y 偏移（默认为 0，0）。第二对数字依赖于一定的操作模式。平铺时，它表示右下方向的绝对坐标。层叠时，它表示各层窗口最大的宽度和高度，如果某个窗口超出了这里指定的宽或高，则将它缩放为最大的宽或高。这几个数字以 p 为后缀时表示像素值，否则，它们表示屏幕尺寸的百分比。

Fvwm 中文手册-FvwmSave

FvwmSave

名称（NAME）：

FvwmSave - 保存 FVWM 的桌面布局（desktop-layout）。

概要（SYNOPSIS）：

FvwmSave 只能被 fvwm 调用(fork),不能从命令行启动。

描述（DESCRIPTION）：

FvwmSave 尝试将你当前的桌面布局保存到一个新的.xinitrc 文件里。理想地，这个文件看起来很像.xinitrc，但实际上，获得一个有用的配置你将不得不重新编辑它，因此备份旧的.xinitrc 文件是必要地。

你的应用必须提供某些到 X 窗口系统的 hints。例如，Emacs 没有提供相应地 hints，因此 FvwmSave 不可能获得任何关于它的信息。

同时，FvwmSave 假定某些命令行选项被应用全部接受，这或许并不是事实。

Fvwm 中文手册-FvwmSaveDesk

FvwmSaveDesk

名称（NAME）：

FvwmSaveDesk - 另外一个保存 FVWM 桌面布局（desktop-layout）的模块。

概要（SYNOPSIS）：

FvwmSaveDesk 只能被 fvwm 调用(fork),不能从命令行启动。

描述（DESCRIPTION）：

FvwmSaveDesk 将你当前的桌面布局作为 InitFunction 函数的额外行保存在你主目录里的.fvwm2desk 文件。这个函数在 fvwm 启动期间被调用。你需要在.fvwm2rc 文件里， Initfunction 函数定义之后包括（include）它，这可以通过 FvwmM4 或 FvwmCpp 模块实现。

你的应用必须提供某些到 X 窗口系统的 hints。例如，Emacs 和 Netscape 没有提供这些 hints，因此 FvwmSaveDesk 不可能获得任何有关它们的信息。

同时，FvwmSave 假定某些命令行选项被应用全部接受，这或许并不是事实。

使用 FvwmM4 时的设置（SETUP USING FVWMM4 MODULE）：

```
fvwm -cmd "FvwmM4 -m4-prefix -m4opt -I$HOME $HOME/.fvwm2rc"
```

然后添加下面的行到.fvwm2rc 文件的末尾

```
m4_include(`.fvwm2desk')
```

使用 FvwmCpp 时的设置（SETUP USING FVWMCPP MODULE）：

```
fvwm -cmd "FvwmCpp -C-I$HOME $HOME/.fvwm2rc"
```

然后添加下面的行到.fvwm2rc 文件的末尾

```
#include ".fvwm2desk"
```

调用（INVOCATION）：

可以通过菜单，鼠标，或键盘操作调用 FvwmSaveDesk。Fvwm 将在 ModulePath 指定的目录里查找 FvwmSaveDesk。

可以将下面的行添加到菜单定义里，

```
    + "Save Desktop" Module FvwmSaveDesk
```

绑定到一个鼠标按键操作并不是很有用，但仍然可以这样做，例如添加

```
    Mouse 3  R  CS  Module FvwmSaveDesk
```

然后，如果你在根窗口上按下鼠标右键，同时按下 shift 和 ctrl，FvwmSaveDesk 将被调用。

可以使用下面的命令将 FvwmSaveDesk 绑定到功能键 F10

```
    Key  F10  A  Module FvwmSaveDesk
```

Fvwm 中文手册-FvwmScript (一)

FvwmScript

名称 (NAME) :

FvwmScript - 创建 GUI

概要 (SYNOPSIS) :

FvwmScript 只能被 fvwm 调用(fork),不能从命令行启动。

描述 (DESCRIPTION) :

FvwmScript 可以使你创建很多图形应用，比如桌面附件，带弹出菜单的按钮面板，模态对话框……。启动时，FvwmScript 读取命令行指定的文件。这个文件包括了要执行的脚本，这个脚本并不被包含在 Fvwm 的配置文件里。

通过键盘可以完全控制一个 FvwmScript 脚本。(Shift)-Tab 循环各个控件 (widget)，Return 模拟鼠标单击操作，箭头可以移动光标或改变控件的值，Escape 可以"cancels"菜单和弹出菜单。

调用 (INVOCATION) :

可以将命令'Module FvwmScript name_of_script'添加在.fvwm2rc 文件里来调用 FvwmScript。name_of_script 文件可以斜线 (slash) 开始，这时，它应该是一个绝对路径。否则，FvwmScript 将在几个不同的位置寻找这个文件。如果.fvwm2rc 包括了 '*FvwmScript: Path path_of_the_script_directory'语句，FvwmScript 将尝试在指定的目录里查找脚本文件，如果没有发现脚本文件，则继续尝试在系统配置目录 (system configuration directory) 和用户配置目录 (user configuration directory) 里查找。

配置选项 (CONFIGURATION OPTIONS) :

下面的命令可以用在配置文件里。仅当调用的脚本里没有使用相应的脚本命令时，它们才有效。

*FvwmScript: DefaultFont font

指定默认字体。如果没有指定，同时脚本文件里也没有使用 Font 命令指定，则使用 fixed 字体。

*FvwmScript: DefaultFore color

指定默认前景色。如果没有指定，同时脚本文件里也没有使用 ForeColor 命令指定，则使用黑色。

*FvwmScript: DefaultBack color

指定默认背景色。如果没有指定，同时脚本文件里也没有使用 BackColor 命令指定，则使用 grey85。

*FvwmScript: DefaultHilight color

指定默认的高亮色。如果没有指定，同时脚本文件里也没有使用 HilightColor 命令指定，则使用 grey100。

*FvwmScript: DefaultShadow color

指定默认的阴影色。如果没有指定，同时脚本文件里也没有使用 ShadowColor 命令指定，则使用 grey55。

*FvwmScript: DefaultColorset colorset

指定默认的 colorset。

脚本剖析 (ANATOMY OF A SCRIPT) ：

FvwmScript 使用一种独特的编程语言。一个 FvwmScript 脚本由五个部分组成。Heading 包括窗口的一般特征和所有控件的默认属性。第二部分包括脚本启动时执行的命令。第三部分包括每秒周期执行的任务。第四部分包括退出时执行的命令。最后一部分包括控件的描述。一个控件可以包含 11 种不同类型的项 (item) ：文本标签 (text labels) ，单行文本输入域 (single-line text inputs) ，单选按钮 (radio buttons) ，多选框 (checkbox) ，按钮 (push buttons) ，水平和垂直滚动栏 (horizontal and vertical scrollbars) ，矩形 (rectangles) ，弹出菜单 (pop up menus) ，swallowexecs 和 mini scrollbars。

Heading：

WindowTitle string

设置窗口标题。

WindowSize width height

设置窗口尺寸。

WindowPosition x y

设置窗口位置。

ForeColor {color}

设置所有控件默认的前景色。

BackColor {color}

设置所有控件默认的背景色。

HilightColor {color}

设置所有控件默认的高亮色。

ShadowColor {color}

设置所有控件默认的阴影色。

Colorset {n}

设置所有控件默认的 colorset。

Font {font}

设置所有控件的默认字体。

UseGettext [locale_path]

打开用于 WindowLocaleTitle, LocaleTitle, ChangeLocaleTitle 命令和 Gettext 函数的 gettext 机制。如果参数为空，则使用默认的 FvwmScript locale 目录。这个目录在 fvwm 的安装目录下面，text domain 是 FvwmScript (install_prefix/share/locale/*/LC_MESSAGES/FvwmScript.mo) 。你能够使用 LocalePath 命令修改或增加 locale 目录。这个命令必须放在 WindowLocaleTitle 的前面。

WindowLocaleTitle string

设置窗口标题，但使用的是 UseGettext 定义的 locale。

初始化 (INITIALISATION) ：

这个部分包括启动时执行的命令，例如：

```
    Init

        Begin

            Do "Exec cat tada.voc > /dev/dsp"

            WarpPointer 1

            Set $ToDo=Restart

         End
```

这些命令用来播放声音，移动鼠标指针到控件 1，并初始化$ToDo 为“Restart”。

定时任务 (PERIODIC TASKS) ：

这个部分包括每秒都周期执行的命令，例如：

```
    PeriodicTasks

        Begin

            If (RemainderOfDiv (GetTime) 10)==0 Then

            Do {Exec xcalc}

        End
```

这个例子显示了如何每隔 10 秒执行 xcalc。

退出函数 (THE QUIT FUNCTION) ：

这个部分包括脚本退出时执行的命令 (Quit 命令之后，或者如果你使用 Close，Delete 或 Destroy 命令关闭窗口时) 。例如：

```
    QuitFunc
```

```
        Begin

            Do {Echo bye, bye}

        End
```

如果使用 KillModule 命令关闭脚本，一些依赖于脚本和 fvwm 之间联系的命令或函数不将被执行。

脚本主体 (MAIN OF A SCRIPT) ：

这个部分包括每个控件的描述。每个控件描述有两部分。第一个部分描述初始化属性，第二个部分包括控件收到消息时执行的命令。所有控件能够发送和接受消息。所有的消息通过数字来识别。当用户操作一个控件时，消息“UserAction”被发送到这个控件。

第一个部分的语法是：

```
    Widget      id # A number between 1 and 999 inclusive

    Property

        Type    string

        Size width height

        Position   x y

        Title      { string }

        Value      int

        MaxValue   int

        MinValue   int

        Font    string

        ForeColor  { color }

        BackColor  { color }

        HilightColor  { color }

        ShadowColor   { color }

        Colorset   int

    Flags      flagsOpt
```

上面最后一行 Flags 命令的 flagsOpt 选项是一个空格隔开的关键字列表，可以是 Hidden, NoReliefString, NoFocus, Left / Center / Right。Hidden 指定控件是否启动时隐藏。NoReliefString 指定字符串是否带 relief 显示。NoFocus 指定控件能否获得键盘焦点。默认所有的控件都可以获得焦点，除了 Rectangle，HDipstick 和 VDipstick。当使用 (Shift-)Tab 切换控件时，NoFocus 控件被忽略。Left / Center / Right 指定文本位置，它们仅应用于 ItemDraw，List，Menu，PopupMenu and PushButton，对于 ItemDraw 和 PushButton 默认为 Center，其它默认为

Left。

如果使用 UseGettext 定义的 locale 目录，LocaleTitle 可以用来取代 Title。

每个控件的位置必须指定。

第二个部分的语法是：

```
    Main

        Case message of

            SingleClic:

            Begin

            # list of instructions which will be

            # executed when widget receives

            # message "SingleClic". This message is

            # generated by the user.

            End

            1 :

            Begin

            # list of instructions which will be

            # executed when widget receives

            # message 1

            End

        End
```

Fvwm 中文手册-FvwmScript（二）

控件列表（LIST OF WIDGETS）：

共有 15 种控件。

CheckBox：多选框。

Title：多选框的标题。

Value：如果等于 1，这个复选框被选中，否则没有。

Size 属性被忽略。

HDipstick：水平量度计。

可以用来显示硬盘使用情况。

Value：当前值。

MinValue：最小值。

MaxValue：最大值。

最小尺寸强制为 30x11。

HScrollBar：水平滚动栏。

Value：thumb 的位置。

MaxValue：Value 的上限。

MinValue：Value 的下限。

hegiht 属性被忽略，强制一个最小的 width。

ItemDraw：显示一个图标和/或一个字符串。

Title：显示的字符串。

Icon：显示的图标。

MaxValue：光标的 x 坐标。

MinValue：光标的 y 坐标。

Size 设置为足够大来包括 title 和/或 icon。

List：列表。

让用户在不同的选项中选择。

Value：指定选中了哪个选项。

MinValue：第一个可见的选项。

Title：包括列表里显示的选项。语法是：{Option 1|Option 2|...|Option N}。菜单在窗口顶部显示。

最小 height 强制为 3 个 item，width 至少为 108。

Menu：菜单。

菜单项沿窗口顶部从左到右布局。size 和 position 属性被忽略。

Value：指定选中了哪个选项。

Title：包括菜单里包含的选项。语法是：{Option 1|Option 2|...|Option N}。

MiniScroll：显示一个非常小的垂直滚动栏。

Value：thumb 的位置。

MaxValue：Value 的上限。

MinValue：Value 的下限。

size 设置为 19x34。

PopupMenu：弹出菜单。

Value：指定选中了哪个选项。

Title：语法是：{Option 1|Option 2|...|Option N}。"Option 1|Option 2|...|Option N"是按下鼠标时显示的弹出菜单。

size 属性被忽略。

PushButton：按钮。

Title：按钮的标题。语法是：{Title of the button|Option 1|Option 2|Option3|...|Option N}。"Option 1|Option 2|...|Option N"是按下鼠标右键时显示的弹出菜单。

Icon：按钮的图标。

按钮足够大来适合图标和标签。

RadioButton：单选按钮。

Title：单选按钮的标题。

Value：如果等于 1，被选中。否则没有。

size 属性被忽略。

Rectangle：显示一个矩形。

可以用来装饰窗口。

SwallowExec

这种控件使 FvwmScript spawn 一个进程，捕获第一个名称或 resource 等于 Title 的窗口，并在脚本窗口里显示。

Title：指定被捕获和显示的窗口名称。

SwallowExec：指定执行的命令行。模块也可以被 swallowed。

Value：指定边框的样子。可能的值是：-1,0,1。

size 至少为 30x30。

TextField：文本输入域。

能够用来编辑单行字符串。

Title：文本域的内容。

Value：插入点的位置。

MinValue：结束的位置。

MaxValue：标题的第一个可见字符。

height 属性被忽略，width 至少比初始内容宽 40 个像素。

VDipstick：垂直量度计。

Value：指定当前值。

MinValue：指定最小值。

MaxValue：指定最大值。

size 至少是 11x30。

VScrollBar：垂直滚动栏。

Value：thumb 的位置。

MaxValue：Value 的上限。

MinValue：Value 的下限。

width 属性被忽略，并强制一个最小 height。height 至少是所有可选值的范围加上 37，例如，最小为 0 最大为 10 范围为 11，因此最小 height 为 48。

指令（INSTRUCTIONS）：

HideWidget id

隐藏编号为 id 的控件。

ShowWidget id

显示编号为 id 的控件。

ChangeValue id1 id2

设置 id1 控件的 Value 等于 id2。

ChangeMaxValue id1 id2

设置 id1 控件的 MaxValue 等于 id2。

ChangeMinValue id1 id2

设置 id1 控件的 MinValue 等于 id2。

ChangeTitle id1 id2

设置 id1 控件的 Title 等于 id2。

ChangeLocaleTitle id1 id2

类似 ChangeTitle，但使用 UseGettext 定义的 locale。

ChangeIcon id1 id2

设置 id1 控件的 Icon 等于 id2。

ChangeForeColor id1 {color}

设置 id1 控件的前景色为 color。

ChangeBackColor id1 {color}

设置 id1 控件的背景色为 color。

ChangeColorSet id1 id2

设置 id1 控件的 colorset 等于 id2。

ChangePosition id1 x y

移动 id1 控件到位置 (x，y) 。

ChangeSize id1 width height

设置 id1 控件的尺寸为 (width，height) 。

ChangeFont id1 newfont

设置 id1 控件的字体为 newfont。

WarpPointer id

warp 鼠标指针到 id 控件。

WriteToFile filename {str1} {str2} etc

写字符串到文件 filename 里。

Do {command args}

在 Do 代码块里执行 fvwm 命令。在 fvwm2 手册里描述的任何命令都可以被使用。命令从这个模块发送到 fvwm 主程序。命令和参数的长度不能超过 988 个字符。

Set $var={str1} {str2} etc

连接所有参数到一个字符串，并设置变量$var 为这个字符串。

Quit

退出程序。

SendSignal id1 id2

从 id1 控件发送消息到 id2 控件。

SendToScript id_script {str1} {str2} etc

发送消息到脚本 id_script，消息是所有 str1，str2 等的联接。

Key Keyname Modifier id sig str1 str2 etc

绑定键盘操作到指令

SendSignal id sig

参数 (ARGUMENTS) ：

多数命令都使用参数。有两种类型的参数：数字和字符串。数字参数是位于-32000 和+32000 之间的值。变量总是以"$"开始，并可以包括数字和字符串。

Fvwm 中文手册-FvwmScript（三）

函数（FUNCTIONS）：

所有函数都使用参数。函数可以返回字符串和数字。语法是：

```
    (function argument1 argument2 etc)
```

下面是完整的参数列表：

(GetTitle id)

返回 id 控件的标题。

(GetValue id)

返回 id 控件的当前 Value。

(GetMinValue id)

返回 id 控件的当前 Minvalue。

(GetMaxValue id)

返回 id 控件的当前 Maxvalue。

(GetFore id)

返回 id 控件的当前 RGB 前景色，用 16 进制格式 RRGGBB 表示。

(GetBack id)

返回 id 控件的当前 RGB 背景色，用 16 进制格式 RRGGBB 表示。

(GetHilight id)

返回 id 控件的当前 RGB 高亮色，用 16 进制格式 RRGGBB 表示。

(GetShadow id)

返回 id 控件的当前 RGB 阴影色，用 16 进制格式 RRGGBB 表示。

(GetOutput {str} int1 int2)

执行命令 str，得到标准输出，并返回位于行 int1 和位置 int2 的单词。如果 int2 等于-1，返回完整的行。

(NumToHex int)

返回关于 int 的十六进制值。

(HexToNum {str})

返回 str 的十进制值，str 必须是十六进制值。

(Add int1 int2)

返回(int1+int2)的结果。

(Mult int1 int2)

返回(int1*int2)的结果。

(Div int1 int2)

返回(int1/int2)的结果。

(StrCopy {str} int1 int2)

返回位于 int1 和 int2 之间的字符串。例如，(StrCopy {Hello} 1 2)返回{He}。

(LaunchScript {str})

启动名为 str 的脚本，并返回一个数字标识符。

(GetScriptArgument {int})

返回用于 LaunchScript 参数的脚本。如果 int 等于 0，则返回脚本的名称。

(GetScriptFather)

返回父脚本的 id 号。

(ReceivFromScript {int})

返回 int 脚本发送的消息。

(RemainderOfDiv {int1 int2}): t

返回(int1/int2)的余数。

(GetTime)

返回时间。

(GetPid)

返回脚本的进程 id。

(Gettext {str})

返回翻译字符串。

(SendMsgAndGet {comId} {cmd} bool)

```
(Parse {str} int)
```

str 必须是下面格式的字符串：

```
    X1S1X2S2X3S3...SnXn
```

```
(LastString)
```

返回 Key 和 SendString 的“current working string”。启动时这个字符串是空的，但是当一个 Key 绑定被探测到时，这个字符串被设置为关联这个 Key 命令的字符串。

条件循环 (CONDITIONAL LOOPS) ：

有三种条件循环。“If-Then-Else”语法是：

```
    If $ToDo=={Open xcalc} Then

        Do {Exec xcalc &}  # List of instructions

    Else

    Begin

        Do {Exec killall xcalc &} # List of instructions

        Do {Exec echo xcalc killed > /dev/console}

    End
```

其中的“Else-Begin-End”是可选的。如果这个循环仅有一个指令，Begin 和 End 可以省略。

“While-Do”语法是：

```
    While $i<5 Do

    Begin

        Set $i=(Add i 1)  # List of instructions

    End
```

字符串可以使用“==”比较，数字可以用“<”,“<=”,“==”,“>=”,“>”比较。

“For-Do-Begin-End”语法是：

```
    For $i=1 To 20 Do

    Begin

        Do {Exec xcalc &}# List of instructions

    End
```

命令（COMMANDS）：

下面的 FVWM 命令可以在任何时候执行。

```
SendToModule ScriptName SendString id sig str
```

它发送字符串给任何一个别名或名称匹配 ScriptName 的模块。

看一个例子：

```
Widget 50
Property
    Type PushButton
    Title {Quit}
    ...
Main
Case message of

    SingleClic:
    Begin
        Quit
    End

    1 :
    Begin
        Set $str = (LastString)
        If $str == {Quit} Then
            Quit
        Else
            ChangeTitle 33 $str
    End
```

```
End
```

然后，命令

```
SendToModule MyScript SendString 50 1 str
```

强制 MyScript 退出，如果 str 等于"Quit"且不改变 控件 33 标题为 str 的话。

示例（EXAMPLES）：

你能够在 fvwm 的配置目录里发现脚本的例子。

通信协议（A COMMUNICATION PROTOCOL）：

FvwmScript 是一个弱的（但是简单）编程语言。如果你需要处理很多数据，并且/或者你需要使用复杂的算法，你应该使用外部程序（例如 perl），并"send"期望的信息到你的 FvwmScript 脚本。第一种方式是使用 GetOutput 函数。这比较简单，但是每次你需要信息的时候，都要再次运行外部程序（这或许引发性能问题）。第二种方式是使用 SendMsgAndGet 函数，通过使用一些能够处理命名管道的编程语言扩展 FvwmScript。这个部分将描述这种方式。（第三种方式是使用 fvmw-themes-com，但事实上 SendMsgAndGet 是 fvmw-themes-com 在 FvwmScript 内部的一个实现，并能够得到更好的性能）。

基本上，你从你的 FvwmScript 脚本启动一个外部程序。这个程序在后台运行，你在你的脚本里使用 SendMsgAndGet 函数请求问题或下指令给这个程序。这个程序必须严格遵守某个通信协议。首先有一个标识符 comId，它应该包括脚本的进程 id。这个协议使用两个 fifos，在 FVWM 用户目录里，名为：.tmp-com-in-comId 和 .tmp-com-out-comId。这个程序应该在.tmp-com-in-comId fifo 上创建和监听。然后，当 FvwmScript 以下面的形式执行一个函数时

```
Set $answer = (SendMsgAndGet {comId} {cmd} bool)
```

FvwmScript 在这个 fifo 上写 cmd。这种方式，这个程序能够读 cmd 并执行相应的操作。

Fvwm 中文手册-FvwmScroll

FvwmScroll

名称（NAME）

FvwmScroll - FVWM 滚动栏。

概要（SYNOPSIS）

FvwmScroll 只能被 fvwm 调用(fork),不能从命令行启动。

描述（DESCRIPTION）

如果不是从一个窗口上下文启动的 FvwmScroll，它将提示用户选择一个目标窗口。之后，它将在目标窗口上增加滚动栏，以便减少被窗口所占用的桌面空间。

FvwmScroll 不应该用在移动或缩放自己的窗口上，也不应该用在设置了 WM_COLORMAP_WINDOWS 属性的窗口上。

初始化（INITIALIZATION）：

初始化期间，FvwmScroll 从 fvwm 的模块配置库里获得配置信息，决定使用哪个颜色。

调用 (INVOCATION) :

可以通过将命令'Module FvwmScroll x y'与菜单操作或.fvwm2rc 文件里的 key-stroke 绑定来调用 FvwmScroll。参数 x 和 y 可以是整数，或以 p 为后缀的整数，描述窗口的水平和垂直尺寸。

配置选项 (CONFIGURATION OPTIONS) :

*FvwmScroll: Colorset n

指定 FvwmScroll 使用 colorset n。

*FvwmScroll: Back color

指定 FvwmScroll 窗口的背景色使用 color，而不是黑色。覆盖了 Colorset 选项。

Fvwm 中文手册-FvwmTaskBar

FvwmTaskBar

名称 (NAME) :

FvwmTaskBar - FVWM 任务栏。

概要 (SYNOPSIS) :

FvwmTaskBar [name]

FvwmTaskBar 只能被 fvwm 调用(spawned),不能从命令行启动。

描述 (DESCRIPTION) :

FvwmTaskBar 模块提供了一个任务栏，该任务栏由按行排列的按钮构成，每个按钮对应 FVWM 正在管理的一个窗口。单击鼠标第一个按键能够使对应的顶层窗口 (top level window) 获得焦点，单击中间的按键将隐藏一个顶层窗口，第三个按键为将来的扩展所保留。类似其它模块，FvwmTaskBar 仅当窗口管理器是 fvwm 的时候有效。

启动时，任务栏显示为填充全部屏幕宽度的单行按钮，但在运行期间，可以被缩放到适应 8 行。另外，如果使用了 AutoStick 选项，任务栏将自动放置在屏幕的顶端和底部，且能够从一个位置拖拉到另外一个位置。

任务栏的第一个按钮标签为"Start"，被按下时能够发送一个"Popup StartMenu"命令到 FVWM。

在 FvwmTaskBar 的右边可以显示当前时间和内置的邮件指示器。

初始化 (INITIALIZATION) :

初始化期间，FvwmTaskBar 从 FVWM 获得配置信息。

调用 (INVOCATION) :

可以在.fvwm2rc 文件里添加'Module FvwmTaskBar'命令来调用 FvwmTaskBar。

配置选项 (CONFIGURATION OPTIONS) :

*FvwmTaskBar: Geometry {+-}<X>{+-}<Y>

指定 FvwmTaskBar 的位置。除了 x 和 y 偏移，还能包括宽度和高度，但是仅 x 和 y 被使用。

FvwmTaskBar 的实际宽度总是屏幕的宽度，高度由 *FvwmTaskBar: Rows 选项控制。

如果指定了 AutoStick 选项，任务栏自动的固定（sticks）到屏幕的顶端或底部，具体是两者中的哪一个，依赖于它们哪一个更接近于 Geometry 选项的设置。

设置为"+0-0"可以把任务栏放在屏幕底部。

*FvwmTaskBar: Rows r

指定 FvwmTaskBar 的初始的行数。默认为 1，最大为 8。

*FvwmTaskBar: Font font

指定没有按下时（depressed）按钮标签使用的字体。默认为 fixed 字体。

*FvwmTaskBar: SelFont font

指定压下时按钮使用的字体。注意，Start 按钮总是使用这个字体，即使它没有被按下。如果没有指定这个选项，恢复默认设置。

*FvwmTaskBar: StatusFont font

指定时钟（clock）和提示（tip）窗口使用的字体。默认为 fixed 字体。

*FvwmTaskBar: Fore color

指定按钮名称使用的颜色。

*FvwmTaskBar: Back color

指定任务栏和按钮使用的背景色。

*FvwmTaskBar: Colorset colorset

指定按钮窗口的背景色和前景色使用的 colorset。

*FvwmTaskBar: IconFore color

指定代表图标化窗口的按钮的名称使用的颜色。

*FvwmTaskBar: IconBack color

指定代表图标化窗口的按钮使用的颜色。

*FvwmTaskBar: IconColorset colorset

指定代表图标化窗口的按钮使用的 colorset。

*FvwmTaskBar: FocusFore color

指定代表焦点窗口的按钮使用的颜色。如果没有指定这个选项，*FvwmTaskBar: Fore 或 *FvwmTaskBar: Colorset 指定的颜色将被使用。

*FvwmTaskBar: FocusBack color

指定代表焦点窗口的按钮使用的颜色。如果没有指定这个选项，*FvwmTaskBar: Back 或 *FvwmTaskBar: Colorset 指定的颜色将被使用。注意，这个按钮也将被高亮显示。

*FvwmTaskBar: FocusColorset colorset

指定代表焦点窗口的按钮使用的 colorset。

*FvwmTaskBar: NoBrightFocus

默认情况下，代表焦点窗口的按钮将被高亮显示。这个选项禁止了这个特点。

*FvwmTaskBar: TipsFore color

指定提示窗口的文本使用的颜色。

*FvwmTaskBar: TipsBack color

指定提示窗口使用的背景色。

*FvwmTaskBar: TipsColorset colorset

指定提示窗口使用的 colorset。

*FvwmTaskBar: AutoStick

使任务栏固定（stick）在屏幕的顶部和底部。

*FvwmTaskBar: AutoFocus

当光标停留在任务栏里的一个按钮上时，提升该按钮代表的窗口并显示提示窗口。

*FvwmTaskBar: AutoHide [pixels]

使任务栏隐藏，仅在屏幕底部留下一个箭头。当鼠标移动到那个箭头时，任务栏重新显示。这个选项自动打开 AutoStick。

*FvwmTaskBar: UseSkipList

不显示配置文件中 WindowListSkip 语句里包含的窗口。

*FvwmTaskBar: DeskOnly

仅显示当前桌面上的窗口。切换桌面时，窗口列表相应改变。

*FvwmTaskBar: PageOnly

仅显示和任务栏同一 page 上的窗口。

*FvwmTaskBar: ScreenOnly

仅显示和任务栏同一 Xinerama 屏幕上的窗口。

*FvwmTaskBar: UseIconNames

指定 FvwmTaskBar 使用窗口的图标名称，而不是完整的窗口名称。常用于按钮的宽度很小时。

*FvwmTaskBar: ShowTransients

显示应用的临时窗口。默认不显示。

*FvwmTaskBar: Action action response

指定执行 action 时 FvwmTaskBar 的 response。当前支持的 action 是 Click1, Click2, Click3 等，默认支持 5 个鼠标按键，但可以编译时可以加入更多的支持。当前支持的 response 为任意的 fvwm 命令。

在 response 部分，能够使用一些预定义的变量：$left, $right, $top 和 $bottom，它们分别被按钮按下时的 left, right, top 和 bottom 坐标替换。$-left, $-right, $-top 和 $-bottom 类似。$width 和 $height 被按钮的宽和高替换。

*FvwmTaskBar: Button Title title, Icon icon, Action action

将一个快捷方式按钮（shortcut minibutton）放到任务栏里，单击时执行 action。这些按钮将依照它们在配置文件中定义的顺序紧靠 Start 右侧放置。

为了在操作不同的鼠标按键时执行不同的操作，使用下面的语法：

```
*FvwmTaskBar: Button Title title, Icon icon, \

    Action (Mouse 1) action1, Action (Mouse 2) action2
```

*FvwmTaskBar: ButtonWidth width

指定按钮的最大宽度。

*FvwmTaskBar: Pad width

指定按钮之间的空白。默认为 3。

*FvwmTaskBar: WindowButtonsLeftMargin margin

指定最左边的窗口按钮（window button）的左边界和 Start 按钮的右边界之间的空白。默认为 4。

*FvwmTaskBar: WindowButtonsRightMargin margin

指定最右边的窗口按钮的右边界和时钟窗口的左边界之间的空白。默认为 2。

*FvwmTaskBar: StartButtonRightMargin margin

指定最左边的快捷方式按钮（shortcut minibutton）的左边界和 Start 按钮的右边界之间的空白。默认为 0。

*FvwmTaskBar: 3DFvwm

默认情况下，按钮使用一个专门的 3D 效果（3D look）。这个选项指定使用一个更经典的 3D look。

*FvwmTaskBar: HighlightFocus

如果鼠标指针位于任务栏上方，当前按钮下面的窗口被激活。如果你将 FollowMouse 焦点风格和这个选项一起使用，任务栏将变成 ClickToFocus。

*FvwmTaskBar: ShowTips

打开提示窗口，默认禁止的。

*FvwmTaskBar: NoIconAction action

指定当 NoIcon 风格的窗口图标化或反图标化时执行的操作。比如：

```
*FvwmTaskBar: NoIconAction SendToModule FvwmAnimate animate
```

下面的选项处理在任务栏右边显示的状态指示器。

*FvwmTaskBar: ClockFormat format-string

指定数字时钟使用的时间格式。与 strftime 格式兼容。默认为"%R"。最多显示 24 个字符，显示的字符串依赖于系统的 locale。

*FvwmTaskBar: DateFormat format-string

指定时钟提示的日期和/或时间格式。与 strftime 格式兼容。默认为"%A, %B %d, %Y"。最多显示 40 个字符。显示的字符串依赖于系统的 locale。

*FvwmTaskBar: UpdateInterval seconds

指定时钟刷新的频率。默认 60 妙。

*FvwmTaskBar: BellVolume volume

设置检测到邮件时铃声的音量。为 0-100 之间的值。默认为 20。

*FvwmTaskBar: MailBox path

指定在指定的位置寻找邮件。是用户 mailbox 的绝对路径。默认为/var/spool/mail/$USER_LOGIN。‘None’表示没有 mail 指示器。

*FvwmTaskBar: MailDir

用户 mailbox 的格式默认为 mbox。如果指定这个选项，则使用 MailDir 格式取代。

*FvwmTaskBar: MailCommand command

指定双击 mail 图标时执行的 fvwm 命令。

*FvwmTaskBar: MailCheck seconds

指定检测新邮件的时间间隔。默认为 10 秒。0 或者负值关闭邮件检测。

*FvwmTaskBar: IgnoreOldMail

如果没有新邮件，则不显示位图。

下面的选项处理任务栏左边的开始按钮：

*FvwmTaskBar: StartCommand command

指定 start 按钮被按下时执行的命令。

```
*FvwmTaskBar: StartCommand Popup StartMenu rectangle \

        $widthx$height+$left+$top 0 -100m
```

为了单击不同的鼠标按键时调用不同的命令，使用下面的语法：

```
*FvwmTaskBar: StartCommand (Mouse 1) Popup Mouse1Menu

*FvwmTaskBar: StartCommand (Mouse 3) Popup Mouse3Menu
```

*FvwmTaskBar: StartName string

指定显示在 Start 按钮里的字符串。（默认为‘Start’）。如果参数为空，则不显示任何字符串。

*FvwmTaskBar: StartMenu string

指定 Start 按钮被按下时调用的弹出菜单。（默认为‘StartMenu’）。FvwmTaskBar 将发送‘Popup StartMenu’命令到 fvwm 窗口管理器。

为了单击不同的鼠标按键时调用不同的命令，使用下面的语法：

```
*FvwmTaskBar: StartMenu (Mouse 1) Mouse1Menu

*FvwmTaskBar: StartMenu (Mouse 3) Mouse3Menu
```

*FvwmTaskBar: StartIcon icon-name

指定 Start 按钮显示的图标的名称。

*FvwmTaskBar: NoDefaultStartButton

如果没有给定 Start 按钮的配置选项，则删除默认的 start 按钮。
配置示例（SAMPLE CONFIGURATION）：

略，参看 http://www.fvwm.org/documentation/manpages/unstable/FvwmTaskBar.php

Fvwm 中文手册-FvwmWinList

FvwmWinList

名称（NAME）：

FvwmWinList - FVWM 窗口列表模块。

概要（SYNOPSIS）：

FvwmWinList [name]

FvwmWinList 只能被 fvwm 调用(fork)，不能从命令行启动。

描述（DESCRIPTION）：

FvwmWinList 提供了一个由按钮组成的窗口列表，其中的每个按钮对应于 FVWM 正在管理的一个窗口。单击三个鼠标按键中的任何一个将执行默认的操作，用户也可以自己去配置这个操作。

初始化（INITIALIZATION）：

初始化期间，FvwmWinList 从 fvwm 的模块配置数据库里获得配置信息。

调用（INVOCATION）：

可以在 fvwm 配置文件里加入‘Module FvwmWinList’命令来调用 FvwmWinList。

FvwmWinList 也可以绑定到 keystroke，鼠标按键或菜单操作。使用‘Transient’参数时 FvwmWinList 将类似内置的窗口列表。

配置选项（CONFIGURATION OPTIONS）：

*FvwmWinList: Geometry {+-}<X>{+-}<Y>

指定 FvwmWinList 窗口的位置，目前不支持尺寸的设置，FvwmWinList 将随着里面按钮的增加自动缩放。

*FvwmWinList: Font font

指定按钮标签使用的字体。

*FvwmWinList: Colorset n

指定按钮使用的 colorset，默认为 0。

*FvwmWinList: Fore color

指定按钮名字使用的颜色。覆盖*FvwmWinList: Colorset 的设置。

*FvwmWinList: Back color

指定按钮的颜色。覆盖*FvwmWinList: Colorset 的设置。

*FvwmWinList: FocusColorset n

指定焦点窗口对应按钮的 colorset。默认为 1。

*FvwmWinList: FocusFore color

指定焦点窗口对应按钮的名称使用的颜色。如果参数为空，*FvwmWinList: Fore 被使用。覆盖 FocusColorset 的设置。

*FvwmWinList: FocusBack color

指定焦点窗口对应按钮使用的颜色。如果参数为空，*FvwmWinList: Back 被使用。覆盖 FocusColorset 的设置。

*FvwmWinList: IconColorset n

指定图标化窗口对应按钮使用的 colorset。默认为 0。

*FvwmWinList: IconFore color

指定图标化窗口对应按钮的名称使用的颜色。如果参数为空，则使用*FvwmWinList: Fore。

*FvwmWinList: IconBack color

指定图标化窗口对应按钮使用的颜色。如果参数为空，则使用*FvwmWinList: Back。

*FvwmWinList: DontDepressFocus

缺省的，FvwmWinList 将显示焦点窗口对应按钮为按下的状态。这个选项禁止这个特点。

*FvwmWinList: ButtonFrameWidth width

指定 FvwmWinList 内按钮周围 3D 边框的宽度。

*FvwmWinList: FollowWindowList

指定 FvwmWinList 以与 FVWM 内同样的顺序显示按钮列表。FVWM 的"WindowList NoDeskSort"命令显示了这个顺序。

*FvwmWinList: UseSkipList

通知 FvwmWinList 不显示在 WindowListSkip 行列出的窗口。

*FvwmWinList: ShowCurrentDesk

指定仅显示当前 desk 上的窗口。

*FvwmWinList: NoAnchor

*FvwmWinList: UseIconNames

指定 FvwmWinList 使用窗口的图标名，而不是窗口的完整名称。

*FvwmWinList: LeftJustify

缺省的，FvwmWinList 在图标的中央显示图标文本。这个选项使它对齐图标的左边界。使用 MiniIcons 时，这个选项自动被打开。

*FvwmWinList: MinWidth width

*FvwmWinList: MaxWidth width

指定按钮收缩和伸长的最小和最大宽度。按钮将自动缩放适应最长的按钮名称，但有些应用使用的图标名称可能会填充整个屏幕，这两个选项使它限制在两个值之间。

*FvwmWinList: TruncateLeft

如果设置了*FvwmWinList: MaxWidth，按钮的名称过长时通常从右边进行切除，仅名称的开始可见。设置这个选项将从左边开始切除，即名称的末尾可见。常用于窗口的标题包括目录或文件名的时候。

*FvwmWinList: Action action response

指定执行 action 的时候，FvwmWinList 的 response。目前支持的 action 为：Click1, Click2, Click3 等等。默认支持 3 个鼠标按键，但可以在编译时添加更多的支持。目前支持的 response 可以为任意的 fvwm 命令。

*FvwmWinList: NoIconAction action

指定当 NoIcon 风格的窗口图标化和反图标化时执行的 action。下面是一个 action 的例子：

*FvwmWinList: NoIconAction SendToModule FvwmAnimate animate

配置示例（SAMPLE CONFIGURATION）：

```
    ########

    # Pop up the window list in transient mode on button 3 press & hold


    Mouse 3   R   A   Module FvwmWinList Transient


    AddToFunc DeiconifyAndRaise

    + I Iconify off

    + I Raise


    ########################## Window-Lister ################################

    *FvwmWinList: Back DarkOliveGreen

    *FvwmWinList: Fore PaleGoldenRod

    *FvwmWinList: Font -*-new century schoolbook-bold-r-*-*-*-120-*-*-*-*-*-*-*

    *FvwmWinList: Action Click1 Function DeiconifyAndRaise
```

```
    *FvwmWinList: Action Click2 Iconify
    *FvwmWinList: Action Click3 Module FvwmIdent
    *FvwmWinList: UseSkipList
    *FvwmWinList: UseIconNames
    *FvwmWinList: Geometry -50-85
    *FvwmWinList: MinWidth 70
    *FvwmWinList: MaxWidth 120
    # I prefer the text centered
    #*FvwmWinList: LeftJustify
    # I like it anchored
    #*FvwmWinList: NoAnchor
    # A flat list in most recently focused order
    #*FvwmWinList: FollowWindowList
    #*FvwmWinList: BorderReliefWidth 0
    # pretend to be a taskbar
    *FvwmWinList: NoIconAction SendToModule FvwmAnimate animate
```

Fvwm 中文手册-FvwmWindowMenu

FvwmWindowMenu

名称 (NAME) :

FvwmWindowMenu - 具有窗口列表功能的 fvwm 菜单。

概要 (SYNOPSIS) :

从 StartFunction 函数启动 FvwmWindowMenu :

```
    AddToFunc StartFunction
    + I Module FvwmWindowMenu
```

描述 (DESCRIPTION) :

用于替换 fvwm 内置的 WindowList，但是使用 Perl 编写，容易定制。不像 FvwmIconMan 或 FvwmWinList 模块，FvwmWindowMenu 并不显示自己的窗口，它创建一个 fvwm 菜单并让 fvwm 弹出它。

用法（USAGE）：

从 StartFunction 函数启动 FvwmWindowMenu：

```
Module FvwmWindowMenu
```

并添加下面的命令调用 FvwmWindowMenu 创建的菜单：

```
    Key Menu A N SendToModule FvwmWindowMenu \

         Post Root c c SelectOnRelease Menu
```

或

```
    Mouse 2 A N SendToModule FvwmWindowMenu Popup
```

额外的参数是一些有效的 Menu 命令参数（不带菜单名称）。有效的 action 是 Post 和 Popup，它们创建 fvwm 菜单并使用相应的命令 Menu 和 Popup 调用它们。

设置 Show 和 DontShow 选项来显示和不显示窗口，语法是：

```
    *FvwmWindowMenu: ShowName pattern

    *FvwmWindowMenu: ShowClass pattern

    *FvwmWindowMenu: ShowResource pattern

     *FvwmWindowMenu: DontShowName pattern

    *FvwmWindowMenu: DontShowClass pattern

    *FvwmWindowMenu: DontShowResource pattern
```

Pattern 是一个 perl 表达式，处于 m//上下文。例如：

```
    *FvwmWindowMenu: ShowResource ^gvim

    *FvwmWindowMenu: ShowName Galeon|Navigator|mozilla-bin|Firefox
```

将定义两个分别包括浏览器和 GVim 的部分。剩下的第三部分将包括所有其它的窗口。

为了仅包括匹配的窗口，添加：

```
    *FvwmWindowMenu: DontShowName .*
```

类似于：

```
    *FvwmWindowMenu: DontShowName ^Fvwm

    *FvwmWindowMenu: DontShowClass Gkrellm
```

将使这个菜单忽略名字以 Fvwm 开始的窗口或 class gkrellm 的窗口。

其它选项（Other options）：

*FvwmWindowMenu: OnlyIconified {on|off}

仅显示图标化窗口。

*FvwmWindowMenu: AllDesks {on|off}

显示所有 desk 的窗口。

*FvwmWindowMenu: AllPages {on|off}

显示所有 pages 的窗口。

*FvwmWindowMenu: MaxLen 32

菜单项字符的最大长度。

*FvwmWindowMenu: MenuName MyMenu

弹出菜单的名称。

*FvwmWindowMenu: MenuStyle MyMenuStyle

使用的 MenuStyle 名称。

*FvwmWindowMenu: Debug {0,1,2,3}

输出调试信息的层，0 表示没有 debug。

*FvwmWindowMenu: Function MyWindowListFunc

在菜单项上调用的函数。默认为 WindowListFunc。

*FvwmWindowMenu: ItemFormat formatstring

如何格式化菜单项：

%n, %i, %c, %r

窗口名称，图标名称，class 或 resource。

%x, %y

窗口 x 或 y 坐标。

%X, %Y

窗口 x 或 y 坐标。

%d

窗口 desk 号。

%m

窗口的 mini-icon。

%M

图标化窗口的 mini-icon。

%t

tab。

%%

字母%。

格式字符串必须加引号。默认字符串是"%m%n%t%t(+%x+%y) - Desk %d"。

更多的例子 (MORE EXAMPLES) :

```
CopyMenuStyle * WindowMenu

MenuStyle WindowMenu SelectOnRelease Super_R

*FvwmWindowMenu: MenuStyle WindowMenu


AddToFunc StartFunction I Module FvwmWindowMenu


Key Super_R A A SendToModule FvwmWindowMenu Post Root c c WarpTitle
```

Fvwm 中文手册-FvwmPerl

FvwmPerl

名称 (FvwmPerl) :

FvwmPerl - FVWM perl 控制器和预处理器。

概要 (SYNOPSIS) :

FvwmPerl 应该被 fvwm spawned。

可以在配置文件里添加命令

```
Module FvwmPerl [params]
```

或

```
ModuleSynchronize FvwmPerl [params]
```

运行 FvwmPerl 模块。

描述（DESCRIPTION）：

FvwmPerl 使用 perl 脚本扩展 fvwm 命令。它通过在 perl 配置文件里嵌入 perl 表达式来构造 fvwm 命令。

调用（INVOCATION）：

建议在 StartFunction 函数里调用 FvwmPerl

```
AddToFunc StartFunction I Module FvwmPerl
```

有一些命令行开关（switchs）

```
FvwmPerl [ --eval line ] [ --load file ] [ --preprocess [ --quote char ] [ --winid
wid ] [ --cmd ] [ --nosend ] [ --noremove ] [ line | file ] ] [ --export [names] ]
[ --stay ] [ --nolock ] [ alias ]
```

长开关可以使用一个字母的短开关替换

-e|--eval line 捕获给定的 perl 代码

-l|--load file 捕获给定文件里的 perl 代码

-p|--preprocess [file] 预处理给定的 fvwm 配置文件

下面的五个选项仅在与--preprocess 一起使用时有效

-c|--cmd line 将被预处理的 fvwm 命令

-q|--quote char 改变默认的‘%’quote

-w|--winid wid 明确的设置窗口上下文

--nosend 不发送预处理后的文件给 fvwm，默认是发送的。常用于预处理非 fvwm 配置文件

--noremove 在将预处理后的文件发送给 fvwm 后，不将它删除，默认是删除的。常用于调试

-x|--export [names] 定义 fvwm 快捷方式功能

-s|--stay 在处理—eval，--load 或 -preprocess 后继续执行。

--nolock 当给定三个操作选项之一时，这个选项立即解锁 fvwm。默认异步执行请求的操作。除非使用下面的调用

```
ModuleSynchronous FvwmPerl --preprocess someconfig.ppp
```

如果上面添加了--nolock，则 ModuleSynchronous 立即返回。

使用别名（USING ALIAS）

别名允许同时运行几个不同的模块。

```
ModuleSynchronous FvwmPerl FvwmPerl-JustTest
```

```
    SendToModule FvwmPerl-JustTest eval $a = 2 + 2; $b = $a

    SendToModule FvwmPerl-JustTest eval cmd("Echo 2 + 2 = $b")

KillModule FvwmPerl FvwmPerl-JustTest
```

预处理示例 (PREPROCESSING EXAMPLE)

有效的预处理方法之一是，传递内嵌 perl 代码的 fvwm 配置文件给“FvwmPerl --preprocess”。另外一个方法是写一个脚本，产生 fvwm 命令并发送它们给 fvwm 执行，可以使用“FvwmPerl --load”装载这个脚本。最后，还可以预处理单独的配置行。