Emacs 作为 X Window 管理器

叶家炜

Apr 13, 2025

在软件工程领域，Emacs 长期扮演着「可编程操作系统」的角色。当开发者通过 `M-x shell` 启动终端时，或许不会想到这个行为暗示着更深层的可能性：既然 Emacs 能够托管终端模拟器，为何不能接管整个图形界面？这正是 EXWM（Emacs X Window Manager）项目的哲学起点 —— 将窗口管理这一系统级功能纳入 Emacs 的可编程宇宙。

传统窗口管理器如 i3 或 AwesomeWM 通过配置文件定义行为，而 EXWM 直接将窗口管理逻辑编码为 Elisp 函数。这种设计使得窗口切换快捷键可以与代码编辑操作共享同一套键绑定体系，实现了从文本编辑到系统控制的认知无缝衔接。当我们按下 `C-x b` 切换 Buffer 时，EXWM 将其重新映射为切换图形窗口的指令，这种思维模式的统一性正是 Emacs 生态的独特魅力。

1　技术背景与实现原理

1.1　X Window 系统的通信模型

X Window 系统的 Client-Server 架构为 EXWM 的实现奠定了基础。X Server 负责硬件交互和图形渲染，而 X Client（如 Firefox 或 Terminal）通过 X11 协议与 Server 通信。窗口管理器本身也是一个特殊权限的 X Client，通过 `XGrabKey` 等函数接管键盘事件和窗口布局控制。

EXWM 的突破性在于将 Emacs 进程转化为一个具备窗口管理权限的 X Client。当执行 (`require 'exwm`) 时，Emacs 会通过 `XSetInputFocus` 声明自身为焦点窗口，并通过 `XSelectInput` 订阅所有窗口状态变更事件。这一系列操作使得 X Server 将所有窗口管理决策权委托给 Emacs。

1.2　事件循环与键绑定

EXWM 的核心挑战在于如何将 X Event 整合到 Emacs 的主事件循环中。当用户在 Firefox 窗口中输入字符时，X Server 首先将事件传递给 EXWM，后者通过 `xcb-event->emacs-event` 函数将其转换为 Emacs 可处理的事件结构体。这个过程涉及位掩码操作和状态机转换，确保修饰键状态（如 Ctrl 或 Meta）能被正确传递。以下代码展示了 EXWM 如何将 X 的键盘事件绑定到 Elisp 函数：

```
(exwm-input-set-key (kbd "s-p")
 (lambda ()
   (interactive)
   (start-process "firefox" nil "firefox")))
```

此处 `s-p` 表示 Super 键（Windows 键）与 p 的组合。`exwm-input-set-key` 在底层调用 `XGrabKey`，将物

理按键映射到 Elisp 闭包。这种机制使得窗口管理快捷键与 Emacs 原生快捷键共享同一分发器，避免了传统桌面环境中全局快捷键冲突的问题。

1.3 窗口映射与缓冲区化

EXWM 将每个 X Window 映射为一个 Emacs Buffer，这一设计带来了革命性的交互方式。当启动 LibreOffice 时，EXWM 执行以下步骤：

- 通过 `XCreateWindow` 创建容器窗口
- 使用 `XReparentWindow` 将应用窗口嵌入容器
- 创建关联的 Buffer 对象，设置 `exwm-class-name` 等属性
- 将该 Buffer 挂载到选定的 Frame（虚拟桌面）

这种转换使得图形应用可以像文本 Buffer 一样被管理。例如，`C-x k` 不仅可以关闭文本 Buffer，也可以关闭图形窗口。`winner-mode` 能够回滚窗口布局变更，这得益于所有状态变化都被记录为标准 Emacs 撤销事件。

2 实践配置与使用指南

2.1 基础环境搭建

在 Arch Linux 上安装 EXWM 需要满足以下依赖：

```
pacman -S emacs xorg-server xorg-xinit
```

随后通过 MELPA 安装 EXWM 包，并在 `init.el` 中添加：

```
(require 'exwm)
(require 'exwm-config)
(exwm-config-default)
```

这段代码加载 EXWM 并应用默认配置，包括工作区切换、窗口移动等基本快捷键。`exwm-config-default` 实际上是一组预定义的 Elisp 函数，例如将 `s-&` 绑定到启动浏览器，其底层通过 `call-process` 执行 shell 命令。

2.2 多显示器支持

EXWM 将每个物理显示器映射为独立的 Emacs Frame。配置双显示器时，需要指定主显示器分辨率：

```
(setq exwm-randr-workspace-output-plist '(0 "HDMI-1" 1 "DP-1"))
(exwm-randr-enable)
```

`exwm-randr-workspace-output-plist` 定义工作区与显示器的映射关系，数字 0 和 1 表示工作区编号，字符串为 X11 的输出名称。`exwm-randr-enable` 会监听 RandR 扩展事件，在显示器热插拔时自动调整布局。

2.3 动态窗口规则

通过 Elisp 的模式匹配，可以实现智能窗口管理。以下代码使所有 GIMP 窗口浮动显示：

```
(add-hook 'exwm-manage-finish-hook
  (lambda ()
    (when (string= exwm-class-name "Gimp")
      (exwm-floating-toggle-floating))))
```

exwm-manage-finish-hook 在窗口创建完成后触发，检查窗口的 class-name 属性。exwm-floating-toggle-floating 调用 XConfigureWindow 将窗口设为浮动状态，绕过平铺布局系统。这种基于谓词逻辑的规则系统，远超传统窗口管理器的静态配置能力。

3 优劣分析与适用场景

在性能关键型场景中，EXWM 的架构存在固有瓶颈。由于所有 X Event 需经 Elisp 解释器路由，在 60Hz 刷新率的屏幕上，输入延迟可能达到 $t = \frac{1}{60} \times 2 \approx 33\text{ms}$（两个事件周期）。这对于打字编辑无感知，但在实时性要求高的场景（如游戏）会产生明显卡顿。

然而，对于开发者而言，EXWM 提供了无与伦比的整合能力。想象这样的工作流：在同一个 Emacs 实例中，C-x 5 2 新建 Frame 显示文档，C-x b 切换到终端 Buffer 执行编译，s-<right> 将浏览器窗口移至右侧显示器 —— 所有操作无需离开 Emacs 的键绑定体系。这种一致性将肌肉记忆的效率提升到新的维度。

EXWM 的存在证明了 Emacs 哲学的终极力量：系统不应限制用户，而应成为可塑的粘土。当我们在 Elisp 中定义窗口布局算法时，实际上是在参与一场持续了四十年的软件实验 —— 创造一个完全透明的计算环境。正如 Richard Stallman 在 GNU Manifesto 中所说：「自由意味着控制你自己的计算。」EXWM 将这一理念扩展到了像素的领域。