Broadview
www.broadview.com.cn

NiJili's Series
倪继利作品系列

凝聚研发经验
具备专业深度

Qt及Linux操作系统窗口设计

倪继利 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Qt及Linux操作系统窗口设计

倪继利 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

如何在 Linux 内核上建立窗口系统，这是嵌入式设备软件开发工程师必须了解的。Qt/Embedded 是高端嵌入式设备 PDA 及手机的主流开发工具。作为一项成熟的技术，KDE 桌面系统对于嵌入式窗口系统的设计有很大参考价值；Qtopia 给嵌入式窗口系统提供了很好的框架。本书不仅阐述了 KDE 及 X Window 的机制，分析了 Qt 及 Qt/Embedded 开发工具的核心技术，而且还详细介绍了如何在嵌入式设备上建立 Qtopia 窗口系统。掌握了本书中所介绍的技能，读者可以在 Linux 内核上建立窗口系统，开发应用程序并能够在 PC 上模拟运行为嵌入式设备开发的应用程序。

本书主要针对从事开发 Linux 应用程序的软件工程师，也很适合作为大学教材和参考书。

图书在版编目（CIP）数据

Qt 及 Linux 操作系统窗口设计 / 倪继利编著. —北京：电子工业出版社，2006.4
ISBN 7-121-02434-9

Ⅰ. Q… Ⅱ. 倪… Ⅲ. Linux 操作系统—程序设计 Ⅳ. TP316.89

中国版本图书馆 CIP 数据核字（2006）第 026232 号

责任编辑：顾慧芳
印　　刷：北京天宇星印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店
开　　本：850×1168　1/16　印张：29　字数：780 千字
印　　次：2006 年 4 月第 1 次印刷
印　　数：5000 册　定价：68.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前　言

背景知识

Linux 操作系统已被广泛地使用到 PC 和各种嵌入式设备。笔者在《Linux 内核分析及编程》（电子工业出版社 2005 年 9 月出版）中分析了内核的各种机制，本书作为它的配套书，主要讲述 Linux 操作系统的窗口系统设计。

目前主流的 Linux 窗口系统有：PC 上使用的基于开放源代码的 Gnome 和 KDE、嵌入式设备上使用的开放源代码 Opie 和 Gpe，以及一些商业化的版本（如 Motorola 的 PDA 手机的窗口系统）。

GPE（Gpe Palmtop Environment）窗口系统最类似于桌面领域中的 GNOME，它们都基于 GTK，即一种基于 C 语言的 GUI 界面开发工具。

Opie（Open Palmtop Integrated Environment）窗口系统类似于桌面领域中的 KDE，Opie 基于 Qtopia，Qtopia 是基于 Qt/Embedded 的简化窗口系统。Opie 在 Qtopia 的窗口系统机制上扩展了类库及应用程序，从而更加实用化，可以在多款 PDA 及 PDA 手机上完美运行。KDE 窗口系统基于 Qt/X11，Qt/X11 又基于 X Window 系统。Qt/X11 与 Qt/Embedded 具有相同接口的类库，对于应用程序编程者来说，可以使用相同的类进行开发。Qt 是基于面向对象的 C++的 GUI 界面开发工具，另外还提供了 KDevelop 和 QDesigner IDE 图形界面开发环境。

由于 Qt 具有面向对象、技术成熟等优点，并且目前高端嵌入式设备生产商几乎都选择 Qt 作为开发工具，因此本书选择 Qt 作为界面开发工具，选择 KDE 作为 PC 桌面环境，选择 Qtopia 嵌入式设备窗口系统进行分析介绍。

操作系统的窗口系统内容很广泛，对于 PC，KDE 桌面环境包括 KOffice 办公软件、网页/文件浏览器和系统工具等应用程序，它所依赖的 X Window 系统包括多种协议及 API 库；对于嵌入式设备，Opie 包括了许多应用程序，Qt/Embedded 包括了许多类库。这么多的内容在一本书中要想全面涉及是不可能的，本书主要针对专业开发所需要的深度，有重点地进行讲述。

本书特色与重点

本书主要讲述如何在 Linux 内核上建立窗口系统。

对于 PC 窗口系统，本书重点介绍 KDE 窗口系统如何在启动时从 Linux 内核上建立起来，以及输入设备及图形显示卡硬件如何与窗口界面中的控件进行交互；对于嵌入式设备窗口系统，介绍了 Qtopia 的应用程序管理器如何从 Linux 内核上启动运行，以及输入设备及图形显示卡硬件如何通过 Qt/Embedded 客户/服务器与窗口界面中的控件进行交互。由于 KDE 与 Qtopia 的窗口系统运行机制有许多相似之处，因此，本书重点讲述了 Qtopia 的应用程序管理器以及 KDE 的核心技术。笔者认为只要精通了 Qtopia 的编程，知道了 KDE 的核心技术以及与 Qtopia 编程的差别，并且仔细阅读 KDE 类库的说明文档，也能编写 KDE 应用程序。

在本书中，笔者几乎没有对 KDE 及 Qt 的类库进行说明。因为它们的 API 类库说明文档已说明得很详细，还可以参照源代码。通过对照阅读，读者就会明白各种类的用法。

关于作者

笔者从清华大学电子工程系读研究生起，就开始从事 Linux 内核编程工作，之后又一直在外国著名公司从事 Linux 内核编程工作；除了内核编程工作外，还从事过嵌入式设备的文件管理器及应用程序管理器等的开发工作。

读者对象及阅读方法

本书的读者对象为从事或准备从事 Linux 嵌入式设备开发的人员。为阅读本书，读者需具备 C++ 面向对象编程方面的知识。

建议读者在阅读本书时在 PC Linux 操作系统下编译运行 X 86 平台的 Qtopia，然后对照所运行的 Qtopia 窗口系统和 Qtopia 源代码来阅读，效果会更好。

写作目的

本书的写作目的是让读者对 Linux 窗口的建立过程，以及应用程序从窗口系统启动、文档的打开过程有个清晰的了解。本书中介绍的窗口的分层设计、接口标准化、菜单的通用化，以及应用程序查询硬件信息等设计方法，开发者有较高的参考价值。同时，本书还分析了 Qt 应用程序的编译过程，即 qmake 命令如何将项目文件转换成 Makefile、依赖关系如何传递等，这些内容对开发者查找编译错误可提供清晰的思路。

结构安排

笔者认为，对于应用程序编程者来说，应该重点掌握类的继承关系及对象管理链表、事件、进程间通信、应用程序与桌面环境的交互等。因此，本书重点介绍了这些内容，而舍去了对类库及利用类库编写应用程序方面的介绍。

本书介绍了 Qtopia，而没有介绍 Opie。这是因为 Opie 使用了与 Qtopia 一样的机制，Opie 只是扩展了类库与应用程序。更重要的是 Qtopia 由 Trolltech 公司维护，其提供了各种详细的文档。而且 Qtopia 代码量少，结构清晰，便于理解。如果读者理解了 Qtopia，那么再看 Opie 代码就会很容易理解。

Opie 中的代码更接近实际的产品，甚至 Opie 中的应用程序比许多产品化的应用程序写得还要完善，只要对它进行简单的修改便可用到产品中。

笔者长期从事 Linux 内核程序的开发，也经常要开发应用程序界面，比如使用 Qt 开发嵌入式设备的应用程序界面。有时，笔者还借助于应用程序界面与内核模块共同完成某些功能，如病毒防火墙。由于版权问题，笔者不可能将工作中开发的代码用做书中的源代码。但笔者在本书中尽量选择最贴近产品开发的开放源代码进行讲解，力求对读者的实际开发工作有帮助。

章节简介

第 1 章 “X Window”阐述了 X 的体系分层结构、显卡驱动程序、X 协议，说明了建立在 X 上的各种编程方法，还分析了 X 客户端应用程序 Xlib 函数如何发出请求以及 X 服务器的工作原理及源代码。

第 2 章 “Qt 编程核心技术”介绍了 Qt 对象模型、国际化方法、元对象及代码生成、进程间通信、窗口部件的基类、模板库和集合类、Qt 线程、鼠标拖放、键盘焦点、会话管理，以及调试等方面的技术。

第 3 章 “KDE 窗口系统”介绍了 KDE 的 MIME 类型处理、服务、KIO 框架、组件框架和国际化等，还分析了 KDE 的启动过程及 kicker 应用程序。

第 4 章 “Qt/Embedded 客户/服务器”讲述了将应用程序下载到嵌入式设备运行的方法，介绍了各种嵌入式 GUI 窗口系统，阐述了 Qt/Embedded 的客户/服务器模型工作过程。本章还分析了设备输入事件在客户与服务器之间的传递机制、应用程序窗口的显示原理，并说明了运行 Qt/Embedded 程序的方法。

第 5 章 “Qtopia 核心技术”介绍了 Qtopia 开放源代码的核心技术及类库，其中着重分析了插件的装载过程、应用程序快速启动器的机制、类 MimeType，以及编写输入法的方法等。

第 6 章 “Qtopia 服务器”主要说明组成 Qtopia 服务器的各个类的功能，介绍了 Qtopia 服务器的启动过程，分析了 Qtopia 服务器如何启动应用程序及插件，还分析了电源监控、外观设置、应用程序安装与卸载以及 AppServices 等与系统管理相关的小应用程序。

第 7 章 “Qtopia 的 sysinfo 应用程序分析”详细分析了 sysinfo 应用程序的源代码。

第 8 章 “Qtopia 编译及系统集成”说明了 Qtopia 的定制与集成，阐述了.pro 项目文件的语法。并且详细分析了由.pro 项目文件生成 Makefile 文件的过程以及描述了 Qtopia 的打包、安装和调试方法。

本书用到的软件版本

Qtopia 2.1.1
Qt/Embedded 3.4
Qt/X11 3.4
KDE 3.5
X Window 的 x11R682

致谢

由于有上一本书与电子工业出版社博文视点公司成功合作的愉快经历，所以笔者继续将此书信任地交给该公司的朋友们编辑出版。郭立女士及她的同事对本书的关注和用心使笔者非常感激，感谢他们为我的上一本书及本书所做的大量工作。

建议与问题

笔者的上一本书《Linux 内核分析及编程》出版后，许多读者通过网络发表了自己的评论和意见。为了方便读者发表自己的见解，笔者建立了自己的工作博客，网址如下：

http://nijili.blogcn.com

欢迎读者和专家访问指教，您可以提出自己的建议，指出书中的错误，还可以与网友一起分享和讨论阅读本书的收获以及有利于本书进一步改进的所有问题。

倪继利
2005 年 12 月 28 日于北京

目　录

第 1 章　X Window ····· 1
1.1　X Window 的体系结构 ····· 1
1.1.1　X Server ····· 2
1.1.2　X Client ····· 2
1.1.3　X Protocol ····· 3
1.2　窗口管理器 ····· 4
1.3　X Window 启动过程 ····· 5
1.4　XFree86 配置文件分析 ····· 7
1.5　X Window 程序设计简介 ····· 9
1.5.1　Xlib 编程 ····· 9
1.5.2　Motif 编程 ····· 11
1.5.3　GTK/GNOME ····· 12
1.5.4　KDE ····· 14
1.6　显示驱动程序 ····· 14
1.6.1　显卡驱动方式 ····· 14
1.6.2　帧缓冲 ····· 15
1.6.3　在台式机上使用通用帧缓冲 ····· 15
1.6.4　帧缓冲设备驱动程序 ····· 18
1.7　X Window 协议 ····· 21
1.7.1　X 协议 ····· 22
1.7.2　X 传输接口 ····· 26
1.7.3　XDMCP 协议 ····· 30
1.8　X Client 应用程序源代码分析 ····· 31
1.8.1　xinit 分析 ····· 31
1.8.2　XOpenDisplay ····· 35
1.8.3　函数 XCreateWindow 分析 ····· 39
1.9　X Server 应用程序分析 ····· 43
1.9.1　X Server 概述 ····· 43
1.9.2　DIX 层 ····· 44
1.9.3　OS 层 ····· 48
1.9.4　DDX 层 ····· 56
1.9.5　Screen ····· 62
第 2 章　Qt 编程核心技术 ····· 81
2.1　Qt 概述 ····· 81
2.2　Qt 对象模型 ····· 81
2.2.1　信号和槽 ····· 83
2.2.2　元对象系统 ····· 85
2.2.3　元对象编译器限制 ····· 87
2.2.4　属性 ····· 89
2.3　QObject 类 ····· 90
2.3.1　对象树 ····· 91
2.3.2　事件处理过程 ····· 92
2.3.3　事件运行机制 ····· 95
2.3.4　事件过滤器 ····· 95
2.3.5　定时器 ····· 96
2.3.6　连接函数 connect ····· 98
2.3.7　字符串翻译函数 ····· 98
2.4　Qt 国际化 ····· 98
2.4.1　软件中字符串国际化方法 ····· 98
2.4.2　创建译本 ····· 100
2.4.3　编码支持 ····· 101
2.5　QMetaObject 元对象类 ····· 102
2.5.1　相关的数据结构 ····· 102
2.5.2　QMetaObject 对象 ····· 103
2.5.3　Q_OBJECT 宏及 moc 生成代码分析 ····· 106
2.6　进程间通信——QCopChannel ····· 109
2.7　窗口部件类 ····· 111
2.7.1　窗口部件的基类 QWidget ····· 112
2.7.2　QFrame 类 ····· 113
2.7.3　QScrollView 类 ····· 114
2.8　Qt 风格机制 ····· 117
2.8.1　风格类 ····· 117
2.8.2　窗口系统风格更新机制 ····· 120
2.9　布局类 ····· 125
2.10　Qt 插件 ····· 130
2.11　Qt 模板库 ····· 131
2.11.1　迭代器 ····· 131
2.11.2　算法 ····· 132
2.11.3　数据流串行化 ····· 134
2.12　集合类 ····· 134

2.12.1 基于指针的容器的结构 …… 135
2.12.2 管理集合条目 …… 135
2.12.3 迭代器 …… 136
2.13 Qt 线程 …… 137
2.13.1 线程类 QThread …… 137
2.13.2 线程安全的事件传递 …… 138
2.14 鼠标拖放 …… 140
2.15 键盘焦点 …… 143
2.15.1 焦点移动的方式 …… 143
2.15.2 焦点策略及操作函数 …… 145
2.16 会话管理 …… 145
2.16.1 会话管理 …… 145
2.16.2 测试和调试会话管理 …… 146
2.17 调试技术 …… 147
2.17.1 命令行参数 …… 147
2.17.2 打印警告和调试消息 …… 147
2.17.3 调试宏 …… 149
第 3 章 KDE 窗口系统 …… 151
3.1 KDE 3.5 源代码说明 …… 151
3.2 KDE 类库 …… 153
3.3 系统资源访问 …… 154
3.3.1 标准资源目录 …… 154
3.3.2 KDE 系统配置缓冲 …… 155
3.4 图形支持 …… 155
3.5 用户界面 …… 156
3.5.1 Action 模式 …… 156
3.5.2 用 XML 定义菜单和工具栏 …… 156
3.5.3 在线帮助 …… 157
3.5.4 复杂窗口部件 …… 157
3.6 MIME 类型处理 …… 157
3.6.1 一个应用 MIME 类型的例子 …… 158
3.6.2 映射 MIME 类型到一个应用程序或服务 …… 159
3.7 KDE 服务 …… 160
3.7.1 定义服务类型 …… 161
3.7.2 定义共享库服务 …… 161
3.7.3 定义 DCOP 服务 …… 162
3.8 KIO 框架 …… 163
3.8.1 同步使用 KIO …… 164
3.8.2 同步使用 KIO …… 164
3.8.3 MetaData …… 164
3.8.4 调度 …… 165
3.8.5 定义一个 ioslave …… 166
3.9 DCOP …… 167
3.9.1 DCOP 工具 …… 167
3.9.2 DCOP 收发数据 …… 168
3.10 KDE 应用程序启动其他应用程序的方法 …… 173
3.11 国际化和本地化 …… 174
3.12 编译方法 …… 175
3.13 桌面框架 …… 179
3.13.1 Panel Applets …… 179
3.13.2 控制中心模块 …… 181
3.14 组件框架 …… 182
3.14.1 KParts 组件 …… 182
3.14.2 系统托盘 …… 183
3.14.3 集成 Java …… 183
3.15 KDE 协议说明 …… 184
3.16 KDE 启动 …… 185
3.16.1 startkde 脚本分析 …… 186
3.16.2 startkde 中的应用程序说明 …… 189
3.16.3 桌面组件 …… 191
3.17 kicker 应用程序 …… 193
3.17.1 kicker 应用程序的类继承关系 …… 194
3.17.2 桌面配置文件 …… 196
3.17.3 kicker 窗口的建立过程 …… 197
3.17.4 K 菜单 …… 203
第 4 章 Qt/Embedded 客户/服务器 …… 212
4.1 各种嵌入式 GUI …… 212
4.1.1 X Window …… 213
4.1.2 MICROWindows …… 213
4.1.3 MiniGUI …… 213
4.1.4 QT/Embedded …… 214
4.2 Qt/Embedded 的应用示例 …… 217
4.2.1 编译 Linux 内核 …… 217
4.2.2 root 文件系统 …… 217
4.2.3 裁剪 Qt/Embedded 库 …… 217
4.2.4 编译应用程序 …… 218
4.2.5 拷贝 Qt 库及应用程序 …… 218
4.2.6 打包 …… 218
4.2.7 下载及运行 …… 219
4.3 帧缓冲 …… 220
4.3.1 DirectFB 介绍 …… 220
4.3.2 虚拟帧缓冲 qvfb …… 220
4.4 创建应用程序 …… 221
4.4.1 QApplication 类 …… 221

4.4.2 QApplication 构造函数 …… 222
4.5 构建服务器 …… 224
4.5.1 QWSServer 类说明 …… 224
4.5.2 QWSServer 类构造函数分析 …… 225
4.6 Qt/Embedded 客户端与服务器通信 …… 228
4.6.1 服务器相关的类说明 …… 229
4.6.2 客户与服务器间的接收事件处理 …… 230
4.6.3 服务器对窗口的协调管理 …… 235
4.7 输入设备与应用程序通信 …… 237
4.7.1 鼠标键盘相关类 …… 238
4.7.2 打开鼠标设备的过程 …… 238
4.7.3 设备文件与 socket 连接 …… 241
4.7.4 服务器捕获设备输入信号 …… 243
4.7.5 服务器发送事件给客户 …… 244
4.8 应用程序显示 …… 246
4.8.1 与显示相关的类 …… 246
4.8.2 与字体相关的类 …… 247
4.8.3 函数 init_display …… 248
4.8.4 QWSDisplay 构造函数分析 …… 249
4.8.5 QScreen …… 252
4.8.6 与图形显示相关的类 …… 259
4.8.7 图形显示 …… 262
4.9 运行 Qt/Embedded 应用程序 …… 268
4.10 字符输入 …… 269
4.11 增加一个加速显示卡驱动程序到 Qt/Embedded …… 270
4.12 Qt/Embedded 作为 VNC 服务器 …… 271
第 5 章 Qtopia 的核心技术 …… 272
5.1 Qtopia 简介 …… 272
5.1.1 Qtopia PDA …… 272
5.1.2 Qtopia 手机版 …… 273
5.1.3 手机库 …… 274
5.1.4 Qtopia Desktop …… 275
5.2 信道和消息 …… 275
5.3 国际化 …… 278
5.3.1 翻译操作步骤 …… 278
5.3.2 .qm 文件位置 …… 278
5.3.3 Desktop 文件国际化 …… 279
5.3.4 文档国际化 …… 279
5.4 Qtopia 文件系统标准 …… 279
5.5 Qtopia 插件 …… 281
5.5.1 建立插件的步骤 …… 281
5.5.2 装载插件的方法 …… 283
5.6 输入法插件 …… 286
5.6.1 创建弹出输入方法 …… 287
5.6.2 创建复合输入法 …… 290
5.7 快速启动应用程序 …… 294
5.7.1 改进主窗口创建时间 …… 294
5.7.2 Quick Launcher …… 295
5.7.3 应用 Quick Launcher …… 296
5.7.4 Quick Launcher 的宏定义 …… 296
5.7.5 运行 Quick Launcher 分析 …… 299
5.8 Qtopia 核心类库说明 …… 302
5.8.1 QPEApplication 类 …… 302
5.8.2 应用程序配置与连接类 …… 304
5.8.3 PIM 管理类 …… 310
5.8.4 桌面相关类 …… 312
5.8.5 字体与输入法相关类 …… 314
5.8.6 StorageInfo 和 FileSystem 类 …… 315
5.8.7 DeviceButtonManager 和 DeviceButton 类 …… 318
5.8.8 图像处理相关类说明 …… 319
5.8.9 多媒体播放相关类说明 …… 319
5.8.10 网络相关类说明 …… 320
5.8.11 软件模块接口类 …… 322
5.8.12 Service 和 ServiceRequest 类 …… 324
5.8.13 MimeType 类 …… 325
5.8.14 其他类说明 …… 330
第 6 章 Qtopia 服务器 …… 332
6.1 Qtopia 服务器特定的类 …… 332
6.1.1 ServerApplication 类 …… 333
6.1.2 Server 类 …… 333
6.1.3 Launcher 类 …… 334
6.1.4 LauncherTabWidget、LauncherTabBar 和 LauncherTab 类 …… 335
6.1.5 LauncherView 和 LauncherItem 类 …… 337
6.1.6 AppLauncher 和 QuickLauncher 类 …… 337
6.1.7 LoadingWidget 和 DocumentList 类 …… 338
6.1.8 TaskBar 类 …… 338
6.1.9 StartMenu 和 StartPopupMenu 类 …… 339
6.1.10 InputMethodSelector 和 InputMethods 类 …… 339
6.1.11 RunningAppBar 和 SysTray 类 …… 341
6.1.12 服务器的其他类说明 …… 341
6.2 Qtopia 服务器启动过程 …… 343
6.2.1 Qtopia 服务器的 main 函数 …… 343

6.2.2 ServerApplication 类构造函数 …… 345
6.2.3 Server 类构造函数 …… 347
6.2.4 创建服务器主窗口界面 UI …… 348
6.2.5 创建应用程序图标视图 …… 350
6.3 应用程序启动过程 …… 354
6.4 文档的打开过程 …… 360
6.5 电池监控小应用程序 …… 362
6.5.1 电池状态窗口类 …… 363
6.5.2 电源状态管理器类 …… 365
6.5.3 插件装载 …… 365
6.6 外观设置 …… 366
6.6.1 main 函数 …… 367
6.6.2 SampleWindow 类 …… 367
6.6.3 AppearanceSettings 类 …… 368
6.6.4 AppearanceSettings 类的构造函数分析 …… 368
6.7 安装与卸载应用程序 …… 376
6.8 AppServices 应用程序 …… 377
6.8.1 AppServices 类 …… 378
6.8.2 ASCheckListItem 类 …… 381
6.9 插件管理器 …… 382
第 7 章 Qtopia 的 sysinfo 应用程序分析 …… 384
7.1 SystemInfo 类 …… 384
7.2 VersionInfo 类 …… 385
7.3 StorageInfoView 类 …… 387
7.4 负载信息 LoadInfo …… 396
7.5 内存信息 …… 399
7.6 DataView 类 …… 400
第 8 章 Qtopia 编译及系统集成 …… 401
8.1 qmake 介绍 …… 401
8.1.1 qmake 编译过程 …… 401
8.1.2 .pro 文件的语法分析 …… 402
8.2 Qtopia 编译方法 …… 404
8.2.1 Qtopia 编译步骤 …… 404
8.2.2 Qtopia Desktop 编译步骤 …… 406
8.2.3 编译一个应用程序 …… 407
8.2.4 使用 NFS 运行应用程序 …… 409
8.2.5 应用程序调试 …… 410
8.3 为设备定制 Qtopia …… 411
8.3.1 定制 Qtopia 启动器用户界面 …… 411
8.3.2 设备硬件因素 …… 412
8.3.3 不安装及不支持的组件 …… 414
8.3.4 安全模式需要的插件 …… 414
8.3.5 电源管理 …… 414
8.3.6 可移去的存储卡 …… 415
8.3.7 I18N …… 416
8.3.8 访问权限和仅读文件系统 …… 416
8.3.9 MMS 客户端 …… 416
8.3.10 GSM 模块集成 …… 417
8.3.11 定制键盘 …… 417
8.3.12 配置硬件按钮 …… 419
8.4 系统集成 …… 421
8.4.1 创建设备特定交叉编译工具配置文件 …… 421
8.4.2 定制电源、背景灯控制和键盘扫描码 …… 421
8.4.3 在 Qt/Embedded 中创建键盘处理函数 …… 422
8.4.4 配置指针设备 …… 423
8.4.5 配置适合设备的 Qt/Embedded 库 …… 424
8.4.6 配置适合设备的 Qtopia …… 428
8.4.7 为目标设备创建和安装 Qtopia image 文件 …… 428
8.4.8 在设备上配置适合于 Qtopia 的环境变量 …… 428
8.4.9 安装另外的字体 …… 428
8.4.10 集成 Java …… 429
8.5 Qtopia 编译系统 …… 429
8.5.1 内部的编译系统项目文件说明 …… 430
8.5.2 项目文件生成 Makefile …… 434
8.5.3 src-components.pro 和 src.pro 文件 …… 434
8.5.4 configure 脚本生成 Makefile …… 437
8.6 Qtopia 的编译过程 …… 442
8.6.1 编译库 …… 442
8.6.2 相互独立的应用程序及插件的编译 …… 444
8.6.3 文件的安装 …… 444
8.6.4 打包分发 …… 446
8.7 生成 img 文件并烧录 …… 449
主要参考文献 …… 450

第 1 章　X Window

1984 年，麻省理工学院（MIT）电脑科学研究室开始开发 X Window。1986 年，MIT 和数字设备公司（DEC）对 X Window 协议进行重新设计，推出了 X Window 第 11 版，即 X11。

X11 协议定义一个系统所必须具备的功能，任何系统若能满足此协议并符合 X 协会的其他规范，便可称为 X。

XFree86 是基于 Intel PC 平台的 X 系统，其含义是“提供基于 Intel 的 PC 平台，自由的 X 服务”。

X Window 是高度可配置的，它只定义了一系列最基本的显示功能调用，而没有规定图形窗口的样式。图形窗口的样式由程序员开发的窗口管理程序完成，从而使得 X Window 的风格多种多样，用户可根据自己的喜好定制图形界面。

本章阐述了 X 的体系分层结构、显卡驱动程序、X 协议，说明了建立在 X 上的各种编程方法，还分析了 X 客户端应用程序 Xlib 函数如何发出请求，以及 X 服务器的工作原理及源代码。

1.1　X Window 的体系结构

X Window 的运行基于一种客户/服务器（Client/Server）模式，它由 X Server、X Clients 和通信通道 3 个部分组成，X Server 和 X Clients 通过 X 协议在网络上通信完成应用任务。

X Server 是控制输入输出的程序，它和底层硬件直接通信控制实际的显示器、鼠标及键盘的软件，它只在接收 X Client 程序的请求后完成建立窗口和绘制图形等工作。

X Client 是基于 X Server 的客户程序，作为使用系统的视窗功能的程序，它请求 X Server 在指定的窗口执行各种操作。它不负责显示，只是给 X Server 发送一个请求，由服务器完成操作。

X 协议是 X Server 和 X Client 之间沟通的语言，Xlib 库封装了可完成这种通信功能的 API，可以使用这些 API 开发 X Client 程序。

X Window 运行分层如图 1-1 所示。

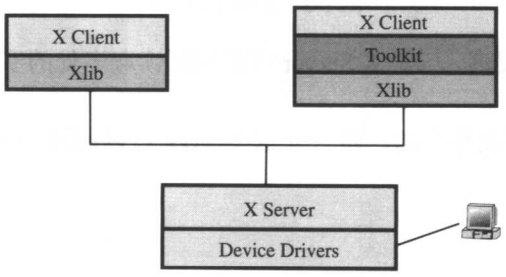

图 1-1　X Window 运行分层

其中最底层的是 X Server，其上层是 X 网络协议，该层使远程运行 X Window 成为可能。再上层是称为“Xlib”的底层函数接口，它介于网络和基础系统与较高层的程序之间，应用程序的实现通过调用这一层的函数实现。最顶层是管理窗口的窗口管理器，即 WM（Window Manager），如 AfterStep、Blackbox、Enlightenment、ctwm、ftwm、sawfish、twm 和 Window Maker 等，这些管理器中的每一个

都提供了一个不同的界面。另外，KDE 和 GNOME 桌面环境（Desktop Environments）都有自己的视窗管理器与桌面集成，每一个视窗管理器也有一个不同的配置机制。

X Window 的客户机/服务器模式有如下的优点。

（1）客户程序可以在远程计算机上执行计算任务，而使用的 X 服务器仅负责复杂的图形显示，从而充分发挥 X 服务器在显示上的优势。

（2）只有 X 服务器与硬件打交道，所有的客户程序都与硬件无关，从而很容易在不同的平台上移植。

（3）使用不同的视窗管理器会使得 X Window 的外观看起来截然不同。

1.1.1 X Server

X Server 是控制显示器和输入设备（键盘和鼠标）及一些 X 资源的软件，与硬件设备相关，处理 X Client 所传递的 X 事件。X 资源包括色彩和字形等，X 事件包括键盘的输入、鼠标移动及窗口大小改变，等。

每一套显示设备只对应一个惟一的 Server，X Server 通常根据其管理的显卡类型来命名，如 XF86_SVGA、XF86_VGA16 和 XF86_MONO 等。

X Server 的实现可以是单线程或多线程的，现有的 X Server 包括 Workstation、X Terminal、PC X Server 和 Web Browser。

X11R5 增加了一项“字库服务器”（font server）功能，其基本概念是将字库分布存储在网络中的机器上，每个 X Server 都可以通过网络取得所需要的字型。X Server 可以没有自己的字库，在一个网络中一套字库只须存储一份，如图 1-2 所示。

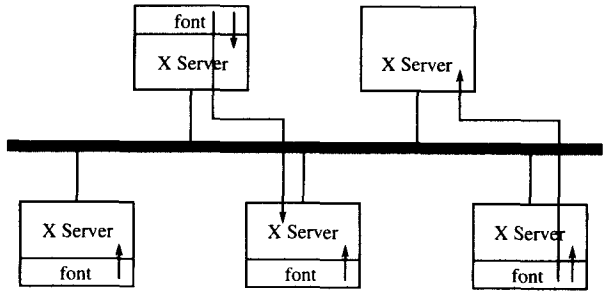

图 1-2 X font server

X font server 的结构非常灵活，一个 X Server 可以同时读取多处的字库，一个 X font server 也可以同时为多个 X Server 服务。

X font server 还可以是层次体系的，即一个 X Server 可以通过一个 font server 访问其他的 font server，如图 1-3 所示。

1.1.2 X Client

X Client 是使用系统视窗功能的一些应用程序，如窗口管理器（Windows Manager，WM）。X Client 与硬件设备无关，无法直接影响视窗，而只能送一个请求（request）给 Server，由 Server 来完成其请求。

典型的 X 客户程序有窗口管理器和桌面系统，窗口管理器决定窗口外观的客户进程，可改变窗口的大小或位置、将窗口缩成图标，并重新安排窗口在堆栈中的位置等。Linux 支持多种窗口管理器，如 fvwm 和 Kdm 等。

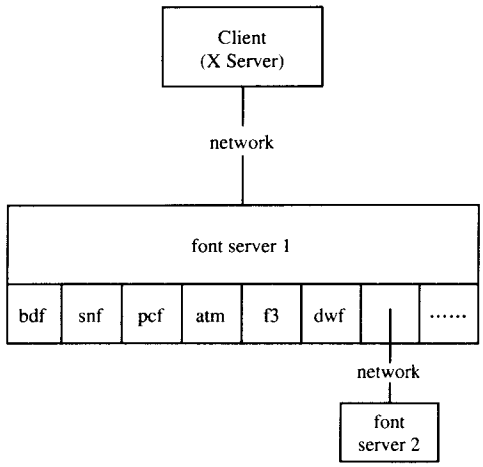

图 1-3　X font server 的层次体系

桌面系统是一个客户进程，它控制桌面上应用程序图标、“开始”菜单及其内容，以及鼠标在桌面上进行键击和拖动操作所产生的效果等。桌面系统集成了窗口管理器和其他工具，目前 Linux 系统上主要的桌面系统有 KDE 和 GNOME。KDE 采用 Kdm 作为窗口管理器。

用户可以通过系统提供的程序或第三方软件使用 Client 程序，或者为了某种特殊应用而编写自己的 Client 程序。

1.1.3　X Protocol

X Protocol 定义了应用及其显示之间的客户/服务器关系，通过这个协议，应用与其显示被分离开来。更进一步，X 协议被定义为两层，即设备相关和设备无关。X Client 与 X Server 之间的通信是异步的双向协议，任何提供字节流通信的方式都可以使用。可以是 IPC，也可以是 TCP/IP。X Protocol 隐藏了操作系统和硬件的特殊性，这样大大简化了应用的开发和 X Window 系统的可移植性。

X Server 和 X Client 通信有如下两种基本操作模式。

（1）X Server 和 X Client 在同一台机器上运行，它们可以使用机器上任何可用的通信方法。在这种情况下，X 可以同其他传统的视窗系统一样高效工作。

（2）X Client 和 X Server 在不同机器上运行，二者通信就必须遵守网络协议进行，最常用的协议为 TCP/IP。通过测试 TCP/IP，可以看到 X Server 在 TCP 6000 端口上侦听。

例如如果应用程序在远程计算机上运行，输入与显示在本地计算机上运行，那么本地计算机是 X Server，远程计算机是 Client，可使用类似如下命令：

```
rsh remote_pc run_program -display local_pc:0
```

这个命令的含义是在名为 remote_pc 的远程计算机上运行程序 run_program，结果显示在名为 local_pc 的本地计算机的第 0 号显示器上。

X Server 与 X Client 间的交互如图 1-4 所示，通信过程如下。

（1）用户通过鼠标键盘对 X Server 下达操作命令。

（2）X Server 利用 Event 给 X Client 传递用户操作信息。

（3）X Client 执行处理。

（4）X Client 利用 Request 传回所要显示的结果。

（5）X Server 将结果显示在屏幕上。

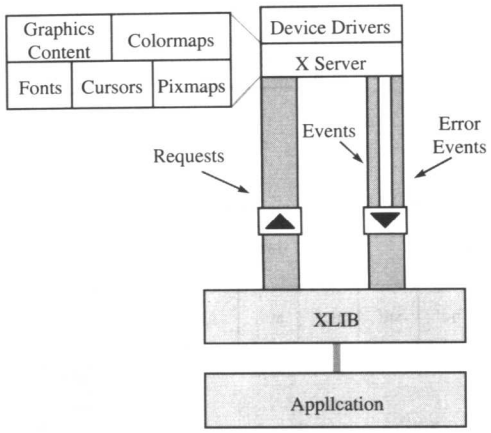

图 1-4　X Server 与 X Client 间的交互

X 协议具有以下客户机和服务器之间的主要通信信息。

（1）请求

X Client 向 X Server 发出服务请求，如创建窗口等。为了提高速度，X Client 通常不等待响应。请求保留给可靠的网络层进行传送。X 协议的请求是 4 个字节的倍数。

（2）答复

答复是 X Server 是对某个的 X Client 请求的响应，但并不是所有的请求都需要响应。X 协议的答复是 4 字节的整数倍，最小为 32 个字节。

（3）事件

X Server 转发事件给 X Client，这些事件包括键盘或鼠标输入等。X 事件为 32 个字节。

（4）差错

X Server 应 X Client 请求报告差错。X 差错大小与事件相同，它们被发送到 X Client 的差错处理程序。X 差错为 32 个字节。

1.2　窗口管理器

X Window 的设计目的是支持多种不同的用户界面，它提供了一般性的架构。即由不同的窗口管理器或桌面环境提供具体的窗口样式，如可以选择 Window Maker 窗口管理器或桌面环境 KDE（K Desktop Environment），它们都建立在 X Window 之上。

窗口管理器是控制窗口属性的一个 Client 程序，它管理窗口的外观形式、桌面菜单、图标、虚拟桌面，以及按钮等显示的样式。作为系统的一部分，它控制屏幕最上层的窗口，其功能包括改变窗口大小或位置、将窗口系统叠放或排列平放或变为一个图标等。用户可以使用不同的窗口管理器，屏幕显示的桌面将因窗口管理器的不同而不同。

桌面环境是包括窗口管理器及各类整合工具和应用程序的集合，Linux 最常用的桌面环境有 GNOME 和 KDE。GNOME 是以 GTK+作为 GUI 图形开发的工具包，包括许多小工具和大量的应用软件。

常见的窗口管理器说明如下。

（1）Twm

Twm 全称为“标签窗口管理器”（Tab Window Manager），作为一种极为简单的 X Window 窗口管理器，它不使用任何图标。用户需要点击背景，打开菜单项来完成工作。对于机器内存较小的用户，

Twm 窗口管理器是其首选。

（2）Window Maker

Window Maker 是一个优秀的窗口管理器，除了提供常用的窗口管理器功能之外，还提供支持 XPM、PNG、JPEG、TIFF、GIF 及 PPM 图标，以及多国语言等。其特色是在不重新启动 X 的状态下，可以修改 Window Maker 的菜单、界面颜色，以及字体。

（3）Fvwm

Fvwm 是一个仅拥有最基本框架的窗口管理器，是在最早的 Twm 窗口管理器基础上开发的。它没有提供华丽的界面和强大的功能，比较适合于性能比较差的计算机。

（4）mwm

mwm（Motif Window Manager）遵循 OSF（Open Source Foundation，开放源代码基金会）制定的 Motif 窗口管理标准。Motif 付费后才能使用。

（5）Sawfish

Sawfish 是使用 Lisp 脚本语言开发的可扩展窗口管理器，用户可以通过修改个人配置文件.sawfishrc 中的 Lisp 代码，或者通过整合的用户定制系统来修改配置信息、更改组件并控制用户界面。

Sawfish 提供了菜单、交互式的窗口移动和改变大小、虚拟桌面、框架主题定义，以及其他标准窗口管理器类似的功能，是 GNOME 默认的窗口管理器。

1.3　X Window 启动过程

启动 X Window 的方法，第一是在/etc/inittab 文件中，把系统的默认运行级别定为 5，重新开机启动系统即可进入 X Window；二是用 startx 脚本手动启动。

startx 主要设置 client 和 server 脚本文件所在的位置并处理相关参数，最后交给 xinit 处理。xinit 根据 startx 传递的参数启动 X Server，成功后根据 xinitrc 启动 X Client。在启动 X Client 时根据脚本 X Client 的配置，可以启动窗口管理器，如 Twm 等，也可以启动 KDE 桌面系统。

当需要启动窗口系统时，在终端窗口输入 startx 命令就可以启动窗口系统，下面是 startx 脚本源代码分析：

```
#!/bin/sh
userclientrc=$HOME/.xinitrc      #用户定义的 client 脚本文件
userserverrc=$HOME/.xserverrc    #用户定义的 server 脚本文件
sysclientrc=/usr/X11R6/lib/X11/xinit/xinitrc    #系统定义的 client 脚本文件
sysserverrc=/usr/X11R6/lib/X11/xinit/xserverrc  #系统定义的 server 脚本文件
defaultclient=/usr/X11R6/bin/xterm       #默认的 client 运行程序
defaultserver=/usr/X11R6/bin/X         #默认的 server 运行程序
defaultclientargs=""       #默认的 client 参数变量
defaultserverargs=""       #默认的 server 参数变量
clientargs=""           #client 参数变量
serverargs=""          #server 参数变量

#选择 client 脚本文件
if[-f $userclientrc ]; then
   defaultclientargs=$userclientrc
elif[-f $sysclientrc ]; then
   defaultclientargs=$sysclientrc
fi

#选择 sever 脚本文件
if[-f $userserverrc ]; then
   defaultserverargs=$userserverrc
elif[-f $sysserverrc ]; then
   defaultserverargs=$sysserverrc
```

```
fi

#下面循环处理 client 和 server 的参数
whoseargs="client"
while[x"$1" != x ]; do   #运行 startx 所带第 1 个参数为空，退出循环
      case "$1" in
     # 需要阻止 cpp 把"/*"当作一个 C 注释
     #如果$1 是/*或者./*形式(xinit 程序要求其参数里的 client 程序和 server 程序必须以/或./开头;否则会被视为 client 程序和
server 程序的参数，见 man xinit)。
/''*|\./''*)
    if["$whoseargs" = "client" ]; then  #如果当前是在处理 client 的参数
       if[x"$clientargs" = x ]; then  #如果 clientargs 为空，则赋值$1 给 client 变量
          client="$1"
       else
          #否则 clientargs 赋值为$clientargs $1，$1 是 startx 命令行 options 参数
          clientargs="$clientargs $1"
       fi
    else    #当前在处理 server 的参数，
       if[x"$serverargs" = x ]; then
         server="$1"
       else
         serverargs="$serverargs $1"
       fi
    fi
;;

#如果$1 为--，则表示开始处理 server 的参数，--为 client 和 server 参数的分界
--)
whoseargs="server"
;;
#为默认值时
*)

if["$whoseargs" = "client" ]; then #处理给 client 程序的参数
   clientargs="$clientargs $1"
else #处理给 server 程序的参数

#display 屏幕编号必须为第 1 个给 server 程序的参数，以:x 的形式(x 为数字)
   if[x"$serverargs" = x ] && expr "$1" : ':[0-9][0-9]*$' > /dev/null 2>&1; then
      display="$1"
   else #处理屏幕编号以外的参数
      serverargs="$serverargs $1"
   fi
fi
;;
esac
shift   #所有参数左移一次
done

# 处理 client 参数
if[x"$client" = x ]; then    #如果 client 程序为空
   #如果没有 client 参数，使用 rc 文件代替
   if[x"$clientargs" = x ]; then   #clientargs 为空
      client="$defaultclientargs"
   else
      client=$defaultclient     #使用默认的 client 程序
   fi
fi

# 处理 server 参数
if[x"$server" = x ]; then
# 如果没有 server 参数或 display，使用 rc 文件代替
   if[x"$serverargs" = x -a x"$display" = x ]; then
      server="$defaultserverargs"
   else
```

```
        server=$defaultserver
    fi
fi

#……省略授权代码若干
xinit $client $clientargs -- $server $display $serverargs  #把处理过的参数交由 xinit 程序处理
```

从上面代码可以看出，startx 为了查找 client 参数，先后查找$HOME/.xinitrc 和/etc/X11/xinit/xinitrc；为了查找 server 参数，先后查找$HOME/.xserverrc 和/etc/X11/xinit/xserverrc。

该 startx 命令启动 X 时，实际上等价于命令 xinit $HOME/.xinitrc -- $HOME/.xserverrc，或 xinit /usr/X11R6/lib/X11/xinit/xinitrc -- /usr/X11R6/lib/X11/xinit/xserver。

xinit 根据 startx 传递的参数启动 X Server，成功后根据 xinitrc 启动 X Client。

在单台计算机中，startx 启动过程如下。

（1）startx 查找 client 和 server 脚本的位置，得到相应参数并调用 xinit。

（2）xinit 根据传递的参数启动 X Server，成功后呼叫 X Client。xinit 先启动 X Server。X Server 建立自己的根窗口，并且一直监控着键盘及鼠标。如果没有 Client 程序需要被告知键盘或鼠标事件，键盘或鼠标的输入传入 X Server 后被放弃。

（3）根据 xinitrc 设置相关资源，启动窗口管理器，输入法和其他应用程序等 X Client 程序。如在/etc/X11/Xclients 脚本中，有如下语句运行 Twm 窗口管理器：

```
exec /usr/X11R6/bin/twm
```

对于内存较大的机器，用户可以使用 KDE 或者 GNOME。在 clients 脚本文件中使用下面的命令行运行 KDE 桌面系统：

```
exec /usr/X11R6/bin/startkde
```

1.4　XFree86 配置文件分析

X Window 系统的程序和字体大多放在/usr/X11R6 目录中，配置文件放在/etc/X11 目录中，初始化程序（如 xfs 字体服务器）放在系统初始化目录/etc/init.d 目录中。XFree86 系统目录说明如表 1-1 所示。

表 1-1　XFree86 系统目录说明

目　录	描　述
/etc/X11	X 配置文件
/usr/X11R6/bin	X 运行程序
/usr/X11R6/include	X11 程序运行所需的头文件和图形
/usr/X11R6/lib	X 程序共享库
/usr/X11R6/lib/X11	配置文件、应用程序资源文件文档、字体及国际化文件
/usr/X11R6/lib/modules	XFree86 模块（驱动程序、扩展文件和字体文件等）
/usr/X11R6/man	X man 手册
/usr/X11R6/share	各种 X 客户程序共享的资源文件

单机上运行的 XFree86 字体目录主要在/usr/X11R6/lib/X11/fonts 目录中，下面几个工具用于管理字体。

（1）mkfontdir：用来在字体目录下创建字体名数据库。

（2）xlsfonts：用来列出已安装字体。