



Qt/Embedded 在嵌入式 Linux 系统中的应用

北京航空航天大学 徐广毅 张晓林 崔迎炜 蒋文军

摘 要

分析和讨论 Qt/Embedded 的主流版本 3.x 系列的底层实现技术；结合 2.x 版本系列和 3.x 版本系列，在两种不同的硬件平台（Intel PXA255 开发系统与笔者自行设计的 Motorola MC9328 MX1 开发系统）上的移植过程，讨论 Qt/Embedded 的底层设备接口与应用移植技术。

关键词

Qt/Embedded 嵌入式 Linux framebuffer 驱动接口

引 言

随着嵌入式 Linux 应用的不断发展，嵌入式处理器运算能力的不断增强，越来越多的嵌入式设备开始采用较为复杂的 GUI 系统，手持设备中的 GUI 系统发展得非常迅速。传统的 GUI 系统，如 Microwindows 等，由于项目规模较小、功能较为薄弱，缺乏第三方软件开发的支持等诸多原因，在比较高级的手持或移动终端设备（如 PDA、Smart-Phone、车载导航系统）中应用较少。

Qt/Embedded 是著名的 Qt 库开发商 Trolltech 公司开发的面向嵌入式系统的 Qt 版本，开发人员多为 KDE 项目的核心开发人员。许多基于 Qt 的 X Window 程序可以非常方便地移植到 Qt/Embedded 上，与 X11 版本的 Qt 在最大程度上接口兼容，延续了在 X 上的强大功能，在底层彻底摒弃了 X lib，仅采用 framebuffer 作为底层图形接口。Qt/Embedded 类库完全采用 C++ 封装。丰富的控件资源和较好的可移植性是 Qt/Embedded 最为优秀的一方面，使用 X 下的开发工具 Qt Designer 可以直接开发基于 Qt/Embedded 的 UI（用户操作接口）界面。越来越多的第三方软件公司也开始采用 Qt/Embedded 开发嵌入式 Linux 下的应用软件。其中非常著名的 Qt Palmtop Environment (Qtopia) 早期是一个第三方的开源项目，并已经成功应用于多款高档 PDA。Trolltech 公司针对 Smart-Phone 中的应用需求，于 2004 年 5 月底发布了 Qtopia 的 Phone 版本。

1 Qt/Embedded 的实现技术基础分析

横向来看，由于发布的版权问题，Qt/Embedded 采用两种方式进行发布：在 GPL 协议下发布的 free 版与专门针对商业应用的 commercial 版本。二者除了发布方式外，在源码上没有任何区别。纵向看来，当前主流的版本为 Qtopia 的 2.x 系列与最新的 3.x 系列。其中 2.x 版本系

列较多地应用于采用 Qtopia 作为高档 PDA 主界面的应用中；3.x 版本系列则应用于功能相对单一，但需要高级 GUI 图形支持的场合，如 Volvo 公司的远程公共交通信息系统。图 1 为 Qt/Embedded 的实现结构。

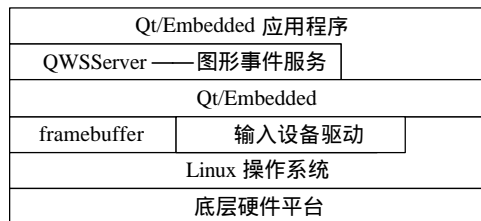


图1 Qt/Embedded的实现结构

3.x 版本系列的 Qt/Embedded 相对于 2.x 版本系列增加了许多新的模块，如 SQL 数据库查询模块等。几乎所有 2.x 版本中原有的类库，在 3.x 版本中都得到极大程度的增强。这就极大地缩短了应用软件的开发时间，扩大了 Qt/Embedded 的应用范围。

在代码设计上，Qt/Embedded 巧妙地利用了 C++ 独有的机制，如继承、多态、模板等，具体实现非常灵活。但其底层代码由于追求与多种系统、多种硬件的兼容，代码补丁较多，风格稍显混乱。

1.1 Qt/Embedded 的图形引擎实现基础

Qt/Embedded 的底层图形引擎基于 framebuffer。framebuffer 是在 Linux 内核架构版本 2.2 以后推出的标准的显示设备驱动接口。采用 mmap 系统调用，可以将 framebuffer 的显示缓存映射为可连续访问的一段内存指针。由于目前比较高级的 ARM 体系的嵌入式 CPU 中大多集成了 LCD 控制模块，LCD 控制模块一般采用双 DMA 控制器组成的专用 DMA 通道。其中一个 DMA 可以自动从一个数据结构队列中取出并装入新的参数，直到整个队列中的 DMA 操作都已完成为止。另外一个 DMA 与画



面缓冲区相关,这部分由两个DMA控制器交替执行,并每次都自动按照预定的规则改变参数。虽然使用了双DMA,但这两个DMA控制器的交替使用对于CPU来说是不可见的。CPU所获得的只是由两个DMA组成的一个“通道”而已。

framebuffer 驱动程序的实现分为两个方面:一方面是对LCD及其相关部件的初始化,包括画面缓冲区的创建和对DMA通道的设置;另外一方面是对画面缓冲区的读写,具体到代码为read、write、lseek等系统调用接口。至于将画面缓冲区的内容输出到LCD显示屏上,则由硬件自动完成。对于软件来说是透明的。当对于DMA通道和画面缓冲区设置完成后,DMA开始正常工作,并将缓冲区中的内容不断发送到LCD上。这个过程是基于DMA对于LCD的不断刷新的。基于该特性,framebuffer驱动程序必须将画面缓冲区的存储空间(物理空间)重新映射到一个不加高速缓存和写缓存的虚拟地址区间中,这样才能够保证应用程序通过mmap将该缓存映射到用户空间后,对于该画面缓存的写操作能够实时的体现在LCD上。

在Qt/Embedded中,QScreen类为抽象出的底层显示设备基类,其中声明了对于显示设备的基本描述和操作方式,如打开、关闭、获得显示能力、创建GFX操作对象等。另外一个重要的基类是QGfx类。该类抽象出对于显示设备的具体操作接口(图形设备环境),如选择画刷、画线、画矩形、alpha操作等。以上两个基类是Qt/Embedded图形引擎的底层抽象。其中所有具体函数基本都是虚函数,Qt/Embedded对于具体的显示设备,如Linux的framebuffer、Qt Virtual Framebuffer做的抽象接口类全都由此继承并重载基类中的虚函数实现。图2为Qt/Embedded中底层图形引擎实现结构。

在图2中,对于基本的framebuffer设备,Qt/Embedded

用QLinuxFbScreen来处理。针对具体显示硬件(如Mach卡、Voodoo卡)的加速特性,Qt/Embedded从QLinuxFbScreen和图形设备环境模板类QGfxRaster<depth,type>继承出相应子类,并针对相应硬件重载相关虚函数。

Qt/Embedded在体系上为C/S结构,任何一个Qt/Embedded程序都可以作为系统中唯一的一个GUI Server存在。当应用程序首次以系统GUI Server的方式加载时,将建立QWSServer实体。此时调用QWSServer::openDisplay()函数创建窗体,在QWSServer::openDisplay()中对QWSDisplay::Data中的init()加以调用;根据QGfxDriverFactory实体中的定义(QLinuxFbScreen)设置关键的QScreen指针qt_screen并调用connect()打开显示设备(dev/fb0)。在QWSServer中所有对于显示设备的调用都由qt_screen发起。至此完成了Qt/Embedded中QWSServer的图形发生引擎的创建。当系统中建立好GUI Server后,其它需要运行的Qt/Embedded程序在加载后采用共享内存及有名管道的进程通信方式,以同步访问模式获得对共享资源framebuffer设备的访问权。

1.2 Qt/Embedded的事件驱动基础

Qt/Embedded中与用户输入事件相关的信号,是建立在对底层输入设备的接口调用之上的。Qt/Embedded中的输入设备,分为鼠标类与键盘类。以3.x版本系列为例,其中鼠标类设备的抽象基类为QWSMouseHandler,从该类又重新派生出一些具体的鼠标类设备的实现类。该版本系列的Qt/Embedded中,鼠标类设备的派生结构如图3所示。

与图形发生引擎加载方式类似的,在系统加载构造QWSServer时,调用QWSServer::openMouse与QWSServer::openKeyboard函数。这两个函数分别调用QMouseDriverFactory::create()与QKbdDriverFactory::create()函数。这时会根据Linux系统的环境变量

QWS_MOUSE_PROTO与QWS_KEYBOARD获得鼠标类设备和键盘类设备的设备类型和设备节点。打开相应设备并返回相应设备的基类句柄指针给系统,系统通过将该基类指针强制转换为对应的具体子类设备指针,获得对具体鼠标类设备和键盘类设备的调用操作。

值得注意的是,虽然几乎鼠标类设备在功能上基本一致,但由于触摸屏和鼠标

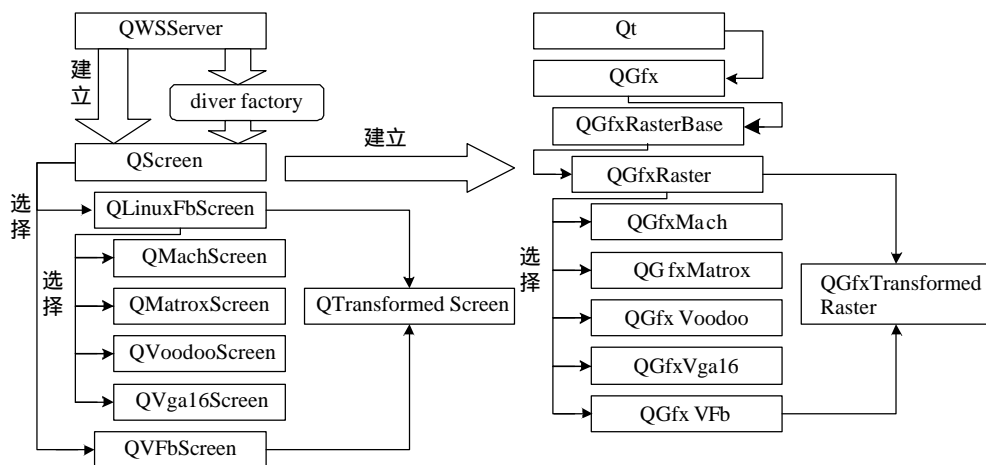


图2 Qt/Embedded 3.x中底层图形引擎实现结构

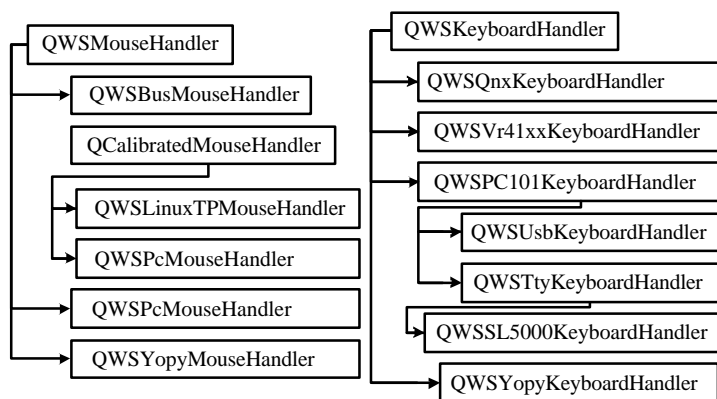


图3 Qt/Embedded 3.x中输入设备抽象派生结构

底层接口并不一样，会造成对上层接口的不一致。举例来讲，从鼠标驱动接口中几乎不会得到绝对位置信息，一般只会读到相对移动量。另外，鼠标的移动速度也需要考虑在内，而触摸屏接口则几乎是清一色的绝对位置信息和压力信息。针对此类差别，Qt/Embedded将同一类设备的接口部分也给予区别和抽象，具体实现在QMouseDriverInterface类中。键盘类设备也存在类似问题，同样引入了QKbdDriverInterface来解决。具体实现此处暂不赘述。

2 Qt/Embedded的移植与应用

针对Qt/Embedded的实现特点，移植该嵌入式GUI系统一般分为以下几个步骤：

设计硬件开发平台，并移植Linux操作系统；

采用静态链接进Linux内核的方式，根据该平台显示设备的显示能力，开发framebuffer驱动程序；

开发针对该平台的鼠标类设备驱动程序，一般为触摸屏或USB鼠标；

开发针对该平台的键盘类设备驱动程序，一般为板载按钮或USB键盘（该部分可选）；

根据framebuffer驱动程序接口，选择并修改Qt/Embedded中的QLinuxFbScreen和QGfxRaster类；

根据鼠标类设备驱动程序，实现该类设备在Qt/Embedded中的操作接口；

根据键盘类设备驱动程序，实现该类设备在Qt/Embedded中的操作接口（该部分可选）；

根据需要选择Qt/Embedded的配置选项，交叉编译Qt/Embedded的动态库；

交叉编译Qt/Embedded中的Example测试程序，在目标平台上运行测试。

framebuffer设备驱动程序提供的接口是标准的，除了注

意endian问题外，配置Qt/Embedded时选择相应的色彩深度支持即可，因此该部分的移植难点就在于framebuffer驱动程序的实现。Qt/Embedded部分的QWSServer打开/dev/中的framebuffer设备后读出相应的显示能力（屏幕尺寸、显示色彩深度），模板QGfxRaster<depth, type>将根据色彩深度在用户空间设备创建出与显示缓存同样大小的缓冲作为双缓冲，并采用正确方式进行显示。

2.1 在PXA255平台上移植和应用

在笔者参与设计的某Smart-Phone开发平台中，GUI系统实现方案采用了Qt/Embedded 2.3.7和Qttopia 1.7.0（基于Qt/Embedded 2.x系列的手持套件），硬件平台采用了基于Intel XScale PXA255处理器的嵌入式开发系统。该开发系统采用640 × 480分辨率的TFT LCD和PXA255内部LCD控制模块作为显示设备，ADS7846N作为外部电阻式触摸屏控制器；另外，采用了五方向按键作为板载键盘。由于该系统采用了ISP1161作为USB Host控制器，较好地支持了USB接口的键盘和鼠标，操作系统为ARM Linux 2.4.19。参考Linux 2.4.19内核目录drivers/input部分，可以按照标准内核中input device接口设计实现触摸屏和键盘，在实现了基于ISP1161的EHCI驱动程序后，移植标准的USB接口的人机界面设备驱动HID和USB键盘、鼠标的驱动程序后，可以获得对于该类设备的调用接口。此过程不属本文讨论范畴，此处暂不赘述。

Qt/Embedded 2.x系列对于输入设备的底层接口与3.x系列不同，触摸屏设备和键盘设备需要根据具体的驱动程序接口在Qt/Embedded中设计实现对应的设备操作类。其中对应于鼠标类设备的实现位于src/kernel/qmouse_qws.cpp中。由于触摸屏在实现原理上存在着A/D量化误差的问题，因此所有的触摸屏接口实现类需要从特殊的QCalibratedMouseHandler继承，并获得校正功能。其具体的实现接口如表1所列。

Qt/Embedded 2.x中对于键盘响应的实现函数位于src/kernel/qkeyboard_qws.cpp中。在qkeyboard_qws.h中，定义了键盘类设备接口的基类QWSKeyboardHandler，移植时需要根据键盘驱动程序从该类派生出实现类，实现键盘事件处理函数processKeyEvent()，并在QWSServer::

表1 Qt/Embedded 2.x中触摸屏类接口

函数接口	描述
QCustomTPanelHandlerPrivate::QCustomTPanelHandlerPrivate (MouseProtocol, QString)	打开触摸屏类设备，连接触摸屏点击事件与处理函数
QCustomTPanelHandlerPrivate::~QCustomTPanelHandlerPrivate()	关闭触摸屏类设备
struct CustomTPdata	驱动程序返回的点击数据结构
void QCustomTPanelHandlerPrivate::readMouseData()	触摸屏点击数据获取



newKeyboardHandler 函数中注册自己的键盘类设备即可。其中对于点击键的键码定义在 Qt/Embedded 的命名空间——src/kernel/qnamespace.h 中。表 2 是具体的实现接口。(以 USB 接口的 101 键键盘为例)

图 4 为笔者在该 Smart-Phone 开发平台上移植 Qt/Embedded 2.3.7 和 Qtopia 1.7.0 后显示的截图。

表 2 Qt/Embedded 2.x 中键盘类接口

函数接口	描述
QWSUsbKeyboardHandler::QWSUsbKeyboardHandler(const QString& device)	打开 USB 键盘, 连接键盘点击信号到 slot 处理函数 readKeyboardData 中
QWSUsbKeyboardHandler::~~QWSUsbKeyboardHandler()	关闭 USB 键盘
struct input_event	驱动程序返回的点击数据结构
void QWSUsbKeyboardHandler::readKeyboardData()	键盘点击 slot 处理函数



图 4 基于 Intel XScale PXA255 的 Smart-Phone 开发平台显示界面

2.2 在 MC9328 平台上移植和应用

在某车载导航辅助系统的开发平台设计中, 采用了 Qt/Embedded 3.3.2 版本作为其 GUI 系统的实现方案。硬件平台采用自行设计的以 Motorola MC9328 MX1 为核心的开发系统。该系统采用 CPU 内部 LCD 控制器和 240 × 320 分辨率的 16 bpp TFT LCD 作为显示设备, 采用 I²C 总线扩展出 16 按键以及 MX1 集成的 ASP 模块和电阻触摸屏。操作系统为 ARM Linux 2.4.18。

Qt/Embedded 3.x 版本系列中与底层硬件接口相关部分的源码位于 src/embedded/ 目录中。该部分包含三类设备的接口: framebuffer、鼠标与键盘。参照该目录中相关设备的具体接口代码, 根据自身硬件平台增添接口即可。

由于系统 LCD 的分辨率为 240 × 320, 物理尺寸较小, 在实现基于该系统的 framebuffer 驱动程序时并没有将其本身与 Linux 字符控制台设备挂靠, 因此 framebuffer 并不具备 TEXT 模式的工作方式。在移植 Qt/Embedded 时, 无需作 framebuffer 设备的工作方式转换。正确配置色彩

显示支持后, Qt/Embedded 能够在 LCD 上显示出正确的图形。由于该平台的显示系统为纵向 320 行, 在设计时考虑到人对于非手持设备的视觉习惯为宽度大于高度的观察方式, 为了符合这种习惯性的观察方式, 在移植 Qt/Embedded 时采用了 Transformed 的旋转图形显示方式, 在软件上实现了显示方向的旋转变换。

鼠标设备接口这一基类

QWSMouseHandler 的实现位于 src/embedded/qmouse_qws.cpp 中。与 2.x 版本系列不同的是, 3.x 中所有的 Linux 触摸屏示例接口代码均实现在 src/embedded/

qmouselinuxtp_qws.cpp 中的 QWSLinuxTPMouseHandler 类中。其中对于不同型号的触摸屏的接口实现代码, 采用不同的宏定义和预编译的方式将它们分隔开。笔者还通过从 QWSLinuxTPMouseHandler 中继承自身触摸屏接口类, 替代原有的 QWSLinuxTPMouseHandlerPrivate 类, 而在 QWSLinuxTPMouseHandler 生成自身触摸屏接口对象的方式, 较好地将移植部分的代码与原有比较混乱的代码分隔开来。3.x 触摸屏类接口如表 3 所列。

3.x 中键盘接口基类位于 src/embedded/qkbd_qws.cpp 中, 为 QWSKeyboardHandler。实现 I²C 总线扩展出的 16 键键盘接口类方式与触摸屏类似, 此处不赘述。需要注意的是, Qt/Embedded 提供了事件过滤器(key event filter)的接口, 在键盘点击事件从 QWSServer 截获并发送到相应的 client 之前会经过函数 QWSServer::KeyboardFilter。在此函数中可以按照自身需求生成新的键盘点击事件, 而后利用 QWSServer::sendKeyEvent() 发送新的点击事件到 client 中。利用该方式可以将各种键盘点击无法输入的 unicode 字符转换出来, 从而可以在较少的按键键盘上实现多 unicode 字符输入法。Qt/Embedded 3.x 键盘接口的移植与鼠标设备接口类似, 此处不赘述。

3 总结

随着嵌入式处理器运算能力的不断提高, 对外设支持不断丰富, 嵌入式 Linux 系统的应用也逐渐增多。Qt/Embedded 延续了 Qt 在桌面系统的所有功能, 丰富的 API

表 3 Qt/Embedded 3.x 中 Linux 触摸屏设备接口

函数接口	描述
QWSLinuxTPMouseHandlerPrivate (QWSLinuxTPMouseHandler *h)	打开触摸屏类设备, 连接触摸屏点击事件与处理函数
~QWSLinuxTPMouseHandlerPrivate()	关闭触摸屏类设备
struct MyTouchP_event	Linux 系统触摸屏驱动程序返回的点击数据格式
void readMouseData()	触摸屏点击数据获取



嵌入式移动数据库与 Agent 技术

南昌大学 李荣鑫

摘要

移动环境中所具有的移动性、频繁的断接性、低带宽、电池电量有限性等特性，决定了移动数据库中的计算环境不同于分布式数据库，给移动数据库的研究提出了许多新的挑战。本文分析移动数据库的特点、体系结构；介绍移动数据库系统中的一些关键性技术，及移动 Agent 在移动数据库中的应用。

关键词 嵌入式移动数据库 移动计算 Agent 技术

随着网络技术的迅速发展和不断渗透，在任何地点和任何时候都能接入网络获取各种信息，必将成为 21 世纪人类的普遍要求；同时，移动通信技术的进步和人们对移动数据处理需求的不断提高，与各种智能通信设备紧密结合的嵌入式移动数据库技术已经得到了学术界、工业界、军事领域、民用部门等各方面的高度重视。移动计算和移动数据库技术将使得这种需求得以实现。


移动数据库是移动计算环境中的分布式数据库，由于移动数据库的应用大都嵌入到诸如掌上电脑、PDA、车载设备等移动通信设备中，故移动数据库有时也称为嵌入式移动数据库。在数据库系统的研究历史中，传统的分布计算与分布式数据库的研究是基于有线网络和固定主机的。这些都采用了一些默认的隐含假设，例如固定连接、对等通信代价、主机节点固定不变等。但进入 20 世纪 90 年代以来，随着移动通信技术和网络技术迅速发展，加之移动计算机和移动通信设备的大量普及，许多计算节点可以在移动过程中与网络建立连接，使得上述假设条件不成立。移动计算环境具有移动性、低带宽、频繁断接性、网络通信的非对称性、电源电力

的有限性等特点，使得传统分布式数据库中的方法和技术不能直接应用于移动数据库。目前，移动数据库的应用与研究正在成为学术界的一个研究热点，有大批学者投入到这一新的研究领域。

1 嵌入式移动数据库的体系结构

在传统的分布式计算系统中，各个计算节点之间是通过固定连接并保持网络的持续连接性的，而移动计算系统改变了这种假设条件。移动计算系统是固定节点和移动节点构成的分布计算系统。移动计算的网络环境具有鲜明的特点^[1,2]：移动性、断接性、带宽多样性、可伸缩性、弱可靠性、网络通信的非对称性、电源能力的局限性等。移动环境中的分布式数据库就是移动数据库。它是传统分布式数据库系统的扩展，可以看作客户与固定服务器节点动态连接的分布式系统。移动数据库系统的结构如图 1 所示^[1]。

其中，移动客户机 MC(Mobile Client)包括便携式电脑、PDA 等；MSS(Mobile Support Station)支持移动计算的固定节点，具有无线通信接口；FH(Fixed Host)没有无

接口和基于组件的编程模型使得嵌入式 Linux 系统中的应用程序开发更加便捷。由于 Qt/Embedded 本身面向高端的手持设备和移动设备，将成为未来嵌入式系统的主流 GUI。

参考文献

- 1 Trolltech Inc. Qt Reference Documentation 2.3.7. 2001
- 2 Trolltech Inc. Qt Reference Documentation 3.3.2. 2004
- 3 Trolltech Inc. Qtopia Documentation 1.7.0. 2003

4 毛德操. 胡希明. 嵌入式系统——采用开源代码和 StrongARM/XScale 处理器. 杭州: 浙江大学出版社, 2003

徐广毅：硕士研究生，主要研究方向为嵌入式 Linux 系统及高级 GUI 系统、实时嵌入式电子飞行信息系统等。张晓林：教授、博士生导师，主要研究方向为通信与信息系统、嵌入式系统、无人飞行器遥控遥测、集成电路设计。

(收修改稿日期：2004-07-08)