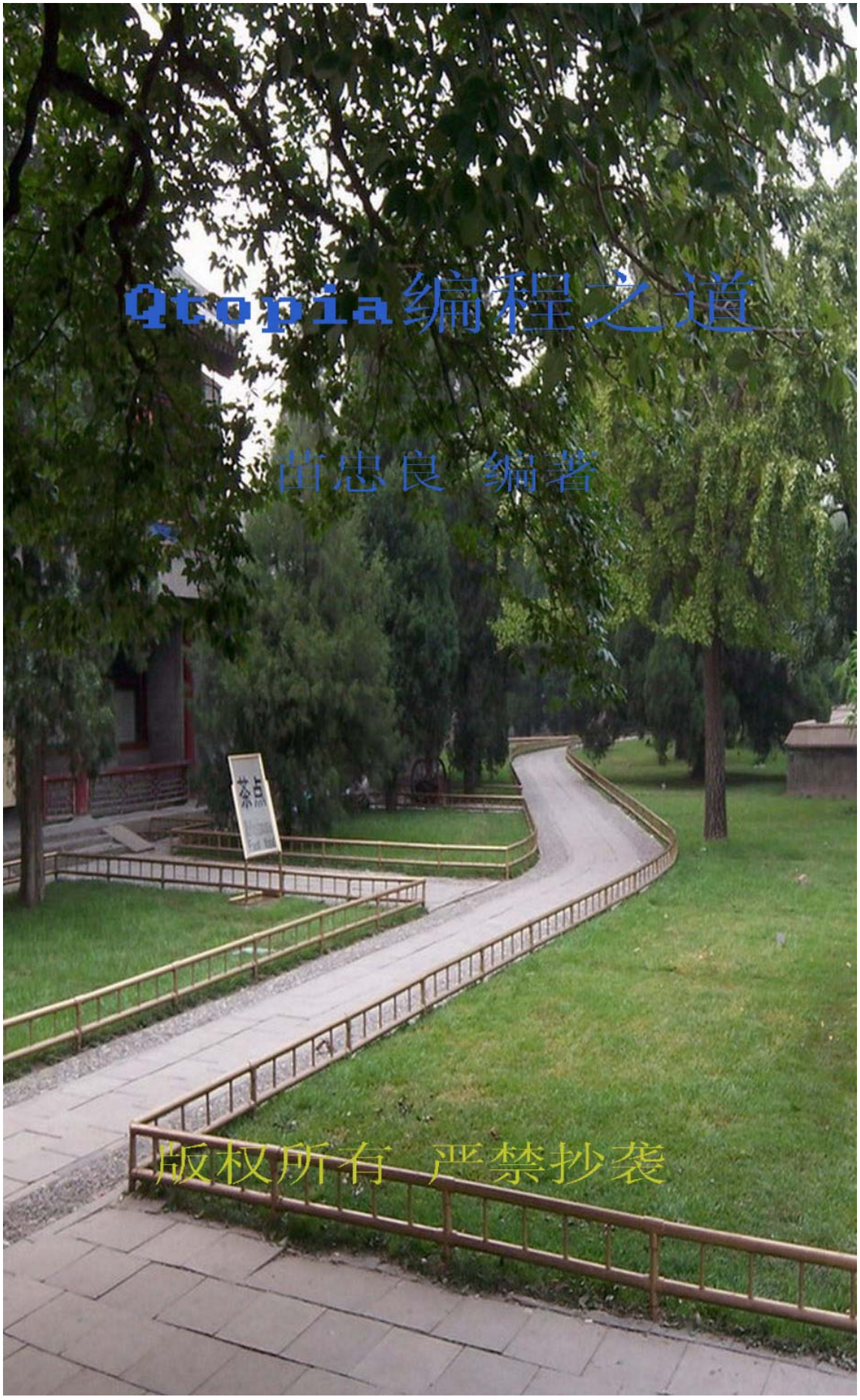

# Qtopia编程之道

苗忠良 编著

献给

我亲爱的爸爸妈妈！

# 简 介

本书是一本面向基于 Linux 利用嵌入式软件研发语言 Qtopia 进行研发的软件研发人员的入门书。面向的读者主要为嵌入式软件研发领域的初、中级读者。希望本书的出版能对在该领域正苦苦探索、希望在软件开发理论和经验方面有所提升的读者有所裨益。

需要声明的是，虽然笔者在撰写本书时足够认真和细心,但因为能力的原因且时间仓促,本书也许还存在这样那样的不足,甚至可能出现错误！敬请读者批评指正!如果读者对本书有什么看法或见解，欢迎来信交流。

# 关于作者

苗忠良：1981 年 6 月生，河南省原阳县人，于 2007 年 1 月在东南大学获得模式识别与智能系统专业硕士学位，于 2004 年 7 月在郑州大学获得自动化专业学士学位。目前在一家企业研究院工作。主要从事嵌入式软件研发技术和无线宽度接入技术等方面的研究。

他的 Email 是：mzl626@163.com。

他的博客地址为：http://miaozl.spaces.live.com，上面发表了一些他对操作系统、程序设计、软件工程等软件研发技术的见解。欢迎登陆交流。

# 前 言

人格心智是一个人能否幸福、快乐、成功的基础，在人类生活的交往过程中有重要的意义，当今社会无情地竞争给人带来的压力越来越大,以一种什么样的人格心态和思维方式来迎接这样的挑战，人格心智是否健康显得尤为重要，一个不会积极思维 的人，生活不会丰富多彩，而且还会有许多麻烦。

**心智决定视野，视野决定格局，格局决定命运，命运决定未来。**

上帝造出来的人都会犯下各式各样的错误。不要只从别人的错误中学习，还要看看别人是怎么做对的。

所谓正确的方法就是寻求可以做到的人，再找出有效的方法来完成它。因而我们必须从成功的典范中学习，而不是一再地错误。

以上是管理学大师德鲁克在《卓有成效的管理者》中一段话，对于软件研发而言，笔者相信也适用之，因此尝试以此作为本书的主旨来阐述相关的内容。尝试探讨软件研发的真谛。

作为一位从事嵌入式软件研发的研发人员，一直以来，都有一种强烈的愿望，希望自己构建出的代码足够完美！但数年的程序设计经验告诉我，编出完美的代码并不是一件很容易的事，受国内环境所限，众多的软件工程师对程序设计往往缺乏总体的了解，对软件的素养一直停留在编程语言的语法阶段，认为写代码就是编程的一切，能够通过调用系统提供的函数来实现所需要的功能就是任务的完成，很少用心去体验代码内在的设计原则和其中蕴含的哲理，更不用说去思考我这样的设计是否已经是最优了！正如用砖头、水泥和钢筋去建造建筑，总以为自己能够将手头的材料组合成貌似需要的模样并能够大致的实现了所需的功能就认为自己已经完成了项目的开发工作！其实不然，建造的建筑是否具有美感暂且不说，但它是否足够牢固，是否经得起地震、飓风？是否为以后的住户的自我改建提供了方便？这也是每个建筑师应该考虑的问题。作为软件，它也存在着这样的问题，写出来的代码是否足够简洁？是否足够强壮？是否有足够的可扩展性？这些概念很多人都懂，但很遗憾的是却很少有人在自己从事研发的时候去考虑！而往往把完成任务了作为自己考虑的第一要素！

如何才能成为一个好的研发人员？笔者自己也一直在苦苦探索，综合前人的经验，笔者认为大致有以下几点：第一，要有扎实的基础，不集跬步，无以成千里，不择细流,无以成江河，扎实的基本功是攀登高峰的基础；第二，要有开阔的视野和思维，大鹏展翅九万里,才能翻动扶摇羊角，只有拥有开阔的视野，对自己对周围的同事和朋友，对业界的发展，对整个社会的发展具有较好的认识，才能更好的把握前进的方向，才能有效的察觉自己的优点和不足，才能在每日三省其身之时获得更好的进步；第三，要乐于探索，坚持不懈，成功往往是百分之九十九的汗水加百分之一的灵感，成功之花需要通过你的辛勤付出才能开的更艳，只有不畏艰难，勇于追求真理的人们才能在科学的道路上走的更高更远；第四，要有良好的品德和哲学素养，软件开发早已经不是一个人两个人所能完成的工作，现代的软件开发更强调团队合作，如何能够让你的团队高效的运转？除了一套完善的制度外还需要的是负责人的个人魅力，良好的道德品质、公平公正的做事风范才能赢得同事的高度信任，彼此间猜疑的团队是没有生命力。另外，细心的读者应该早有体会，世间的万物并不是孤立存在的，彼此间存在着千丝万缕的联系，最重大的发明和发现所蕴含的哲理往往能够从普通的事物上得以体现，千变万化的背后往往体现了最朴素的哲学，以软件为例，设计模式的众多模式背后隐藏的原则不过聊聊数语，闲暇时读读专业外的书籍，看看历史，读读哲学，也许灵感突来，你将走的更远。

在本书的撰写过程中，笔者力求在本书中体现自己参悟到的一些道理，试图从设计背后隐藏的原则着手，为软件研发引入更多的思考因素，尝试说明软件从来就不是单纯的代码编写，优雅的软件往往拥有着深刻的设计哲学。授人以鱼不如授人以渔，慢慢看来，也许你会发现本书的实用价值。当然笔者功力有限，所想所写可能存在着这样那样的错误，我生亦有

## 适用读者

本书适合的读者范围包括：从事嵌入式软件、C/C++等领域的渴望学习的中、高级软件工程师和在校学生。

## 预备知识

本书假设读者已具备如下方面的知识：

- 面向对象程序设计语言（C＋＋、Java）的知识和经验。
- 对象技术的基本概念和知识，如类、对象、封装、继承等。
- Linux 操作系统和嵌入式软件研发语言 Qtopia 等方面的基础知识。

## 本书的组织

本书一共包括四部分共十一章，其中第一章和第二章为第一部分，介绍了 Linux 的基础知识和移动终端的基础知识；第三章、第四章、第五章、第六章、第七章和第八章为第二部分，介绍了 Qtopia 的相关知识；第九章和第十章为第三部分，介绍了人机交互和软件设计的一些重要原则；第十一章为第四部分，介绍了一些从事嵌入式软件研发的重要书籍。便于扩大读者的视野。

## 致谢

首先感谢我的爸爸妈妈，是他们的辛勤培养造就了我今天的成就，没有他们，就没有我今天的一切，衷心的祝福他们永远快乐、健康长寿！

其次，非常感谢我在单位的同事和朋友们，他们都是业界一流的研发工程师，很高兴能够和他们一起工作、 探讨学术话题。同时在本书中笔者也融入了与同事在软件研发过程中交流的一些经验。再次对他们对谢意！

最后，笔者在撰写本书过程中，参考了网上一些朋友的见解， 在这里深表谢意！

学海无涯，在不断学习的道路上，笔者本人也仅是刚刚迈上大道！本书的撰写主要目的是和大家一起分享一些笔者的经验，在交流中一起提高在软件方面的素养！因此笔者采取了在网上公开出版的方式。希望本书能够为在无边黑暗中苦苦摸索的人们带来些许光明。

苗忠良

2007 年 9 月于张江

# 约 定

为了说明的方面，对于本书中采用的一些符号和名词术语做如下的约定：

1. 注释

在本书中在为代码或者命令选项进行注释时，无论是 Linux 平台还是 windows 平台，统一采用“//”的方式进行。

2. 终端提示符

在本书中，常要用到 Linux 终端命令的操作。为了方便起见，终端提示符统一用“#”表示。

3. Qt 与 Qtopia

在本书中，为了说明的方便，默认情况下所说的 Qt 是指 Qt 和 Qtopia 的公共部分，而 Qtopia 则是指 Qtopia 中特有的部分。

# 目 录

# 第 1 章　Linux基础

知之者不如好之者，好之者不如乐之者。

《论语·雍也》

## 1.1 Linux的发展历程

“Hello everybody out there using minix——I’m doing a (free) operating system”在 1991 年的八月，网络上出现了一篇以此为开篇话语的帖子——这是一个芬兰的名为 Linus Torvalds 的大学生为自己开始写作一个类似 minix，可运行在 386 上的操作系统寻找志同道合的合作伙伴。

1991 年 10 月 5 日，Linus Torvalds 在新闻组 comp.os.minix 发布了大约有一万行代码的 Linux v0.01 版本。

到了 1992 年，大约有 1000 人在使用 Linux，值得一提的是，他们基本上都属于真正意义上的 hacker。

1993 年，大约有 100 余名程序员参与了 Linux 内核代码编写/修改工作，其中核心组由 5 人组成，此时 Linux 0.99 的代码有大约有十万行，用户大约有 10 万左右。

1994 年 3 月，Linux1.0 发布，代码量 17 万行，当时是按照完全自由免费的协议发布，随后正式采用 GPL 协议。至此，Linux 的代码研发进入良性循环。很多系统管理员开始在自己的操作系统环境中尝试 linux，并将修改的代码提交给核心小组。由于拥有了丰富的操作系统平台，因而 Linux 的代码中也充实了对不同硬件系统的支持，大大的提高了跨平台移植性。

1995 年，此时的 Linux 可在 Intel、Digital 以及 Sun SPARC 处理器上运行了，用户量也超过了 50 万，相关介绍 Linux 的 Linux Journal 杂志也发行了超过 10 万册之多。

1996 年 6 月，Linux 2.0 内核发布，此内核有大约 40 万行代码，并可以支持多个处理器。此时的 Linux 已经进入了实用阶段，全球大约有 350 万人使用。

1997 年夏，大片《泰坦尼克号》在制作特效中使用的 160 台 Alpha 图形工作站中，有 105 台采用了 Linux 操作系统。

1998 年是 Linux 迅猛发展的一年。1 月，小红帽高级研发实验室成立，同年 RedHat 5.0 获得了 InfoWorld 的操作系统奖项。4 月 Mozilla 代码发布，成为 linux 图形界面上的王牌浏览器。Redhat 宣布商业支持计划，网络了多名优秀技术人员开始商业运作。王牌搜索引擎”Google”现身，采用的也是 Linux 服务器。值得一提的是， Oracle 和 Informix 两家数据库厂商明确表示不支持 Linux，这个决定给予了 Mysql 数据库充分的发展机会。同年 10 月，Intel 和 Netscape 宣布小额投资红帽软件，这被业界视作 Linux 获得商业认同的信号。同月，微软在法国发布了反 Linux 公开信，这表明微软公司开始将 Linux 视作了一个对手来对待。十二月，IBM 发布了适用于 Linux 的文件系统 AFS 3.5 以及 Jikes Java 编辑器和 Secure Mailer 及 DB2 测试版，IBM 的此番行为，可以看作是与 Linux 羞答答地第一次亲密接触。迫于 Windows 和 Linux 的压力，Sun 逐渐开放了 Java 协议，并且在 UltraSparc 上支持 Linux 操作系统。1998 年可说是 Linux 与商业接触的一年。

1999 年，IBM 宣布与 Redhat 公司建立伙伴关系，以确保 Redhat 在 IBM 机器上正确运行。三月，第一届 LinuxWorld 大会的召开，象征 Linux 时代的来临。IBM、Compaq 和 Novell 宣布投资 Redhat 公司，以前一直对 Linux 持否定态度的 Oracle 公司也宣布投资。五月，SGI 公司宣布向 Linux 移植其先进的 XFS 文件系统。对于服务器来说，高效可靠的文件系统是不可或缺的，SGI 的慷慨再一次帮助了 Linux 确立在服务器市场的专业性。7 月 IBM 启动对 Linux 的支持服务和发布了 Linux DB2，从此结束了 Linux 得不到支持服务的历史，这可以视作 Linux 真正成为服务器操作系统一员的重要里程碑。

2000 年初始，Sun 公司在 Linux 的压力下宣布 Solaris8 降低售价。事实上 Linux 对 Sun 造成的冲击远比对 Windows 来得更大。2 月 Red Hat 发布了嵌入式 Linux 的研发环境，Linux 在嵌入式行业的潜力逐渐被发掘出来。在 4 月，拓林思公司宣布了推出中国首家 Linux 工程师认证考试，从此使 Linux 操作系统管理员的水准可以得到权威机构的资格认证，此举大大增加了国内 Linux 爱好者学习的热情。伴随着国际上的 Linux 热潮，国内的联想和联邦推出了“幸

2001 月新年伊始就爆出新闻，Oracle 宣布在 OTN 上的所有会员都可免费索取 Oracle 9i 的 Linux 版本，从几年前的“绝不涉足 Linux 系统”到如今的主动献媚，足以体现 Linux 的发展迅猛。IBM 则决定投入 10 亿美元扩大 Linux 系统的运用，此举犹如一针强心剂，令华尔街的投资者们闻风而动。到了 5 月这个初夏的时节，微软公开反对”GPL”引起了一场大规模的论战。8 月红色代码爆发，引得许多站点纷纷从 windows 操作系统转向 linux 操作系统，虽然是一次被动的转变，不过也算是一次应用普及吧。12 月 Red Hat 为 IBM s/390 大型计算机提供了 Linux 解决方案，从此结束了 AIX 孤单独行无人伴的历史。

2002 年是 Linux 企业化的一年。2 月，微软公司迫于各洲政府的压力，宣布扩大公开代码行动，这可是 Linux 开源带来的深刻影响的结果。3 月，内核研发者宣布新的 Linux 系统支持 64 位的计算机。

2003 年 1 月，NEC 宣布将在其手机中使用 Linux 操作系统，代表着 Linux 成功进军手机领域。5 月之中 SCO 表示就 Linux 使用的涉嫌未授权代码等问题对 IBM 进行起诉，此时人们才留意到，原本由 SCO 垄断的银行、金融领域，份额已经被 Linux 抢占了不少，也难怪 SCO 如此气急败坏了。9 月中科红旗发布 Red Flag Server 4 版本，性能改进良多。11 月 IBM 注资 Novell 以 2.1 亿收购 SuSE，同期 Redhat 计划停止免费的 Linux，顿时业内骂声四起。Linux 在商业化的路上渐行渐远。

2004 年的第 1 月，本着“天下事分久必合，合久必分”之天理，SuSE 嫁到了 Novell，SCO 继续顶着骂名四处强行“化缘”，Asianux，MandrakeSoft 也在五年中首次宣布季度赢利。3 月 SGI 宣布成功实现了 Linux 操作系统支持 256 个 Itanium 2 处理器。4 月里美国斯坦福大学 Linux 大型机系统被黑客攻陷，再次证明了没有绝对安全的操作系统。六月的统计报告显示在世界 500 强超级计算机系统中，使用 Linux 操作系统的已经占到了 280 席，抢占了原本属于各种 Unix 的份额。9 月 HP 开始网罗 Linux 内核代码人员，以影响新版本的内核朝对 HP 有利的方式发展，而 IBM 则准备推出 OpenPower 服务器，仅运行 Linux 系统。

“谁会牵你的手，走过风风雨雨”这句歌词曾经代表着千万 Linuxer 的心，如今，这只可爱的小企鹅终于能独挡一面，在 IBM、HP、Novell、Oracle 等诸多厂商的支持下，迎着风雪傲然前行。

### 1.1.1 开源运动

提到开源，就不得不提到 Richard Stallman——开源软件的始创者和精神领袖（不是偶像崇拜^_^）。他是 GNU（GNU's Not UNIX）工程的发起人，FSF（Free Software Foundation）的缔造者，同时还是大名鼎鼎的 GNU Emacs（一个编辑工具），GCC（c/c 编译器，不用我说了吧，呵呵），GDB（调试工具）和 GNU Make 的笔者。

Stallman 的生平是和开源历史紧密相关的：

1984 年，Stallman 辞去了 MIT AI 实验室的工作，开始编写 GNU 软件。

1985 年，Stallman 成立了自由软件基金会 Free Software Foundation。

1991 年，Linus Torvalds 研发的 linux 加入 GNU，和其他 GNU 软件一起组成了一个完整的自由软件操作系统。

1998 年，FSF 倡导自由软件（Free Software）改名为开源软件（open source software）。Stallman 反对这一做法，他认为 Free 一词，表达的不仅仅是技术，更重要的是思想。

除了技术厉害，Stallman 对自由信念的追求则更让人钦佩。正是因为他的努力，才有今天百花齐放的开源世界。大家想一想，如果没有了开源软件，将会怎样？那很多人恐怕根本用不起计算机，很多企业也没有钱购买服务器系统，相应的网络应用、企业应用恐怕都会大打折扣。

GNU 计划，又称革奴计划，是由 Richard Stallman 在 1983 年 9 月 27 日公研发起的。它的目标是创建一套完全自由的操作系统。Richard Stallman 最早是在 net.unix-wizards 新闻组上公布该消息，并附带一份《GNU 宣言》等解释为何发起该计划的文章，其中一个理由就是要“重

GNU 是“GNU's Not Unix”的递归缩写。Stallman 宣布 GNU 应当发音为 Guh-NOO，与 canoe 发音相同，以避免与 gnu（非洲牛羚，发音与 new 相同）这个单词混淆。

UNIX是一种广泛使用的商业操作系统的名称。由于GNU将要实现UNIX系统的接口标准，因此GNU计划可以分别研发不同的操作系统部件。GNU计划采用了部分当时已经可自由使用的软件，例如TeX排版系统和X Window视窗系统等。不过GNU计划也研发了大批其他的自由软件。

为保证 GNU 软件可以自由地“使用、复制、修改和发布”，所有 GNU 软件都在一份在禁止其他人添加任何限制的情况下授权所有权利给任何人的协议条款，GNU 通用公共许可证（GNU General Public License，GPL）。这个就是被称为“反版权”（或称 Copyleft）的概念。

1985 年Richard Stallman又创立了自由软件基金会（Free Software Foundation）来为GNU计划提供技术、法律以及财政支持。尽管GNU计划大部分时候是由个人自愿无偿贡献，但FSF有时还是会聘请程序员帮助编写。当GNU计划开始逐渐获得成功时，一些商业公司开始介入研发和技术支持。当中最著名的就是之后被Red Hat兼并的Cygnus Solutions。

到了 1990 年，GNU计划已经研发出的软件包括了一个功能强大的文字编辑器Emacs，C语言编译器GCC，以及大部分UNIX系统的程序库和工具。唯一依然没有完成的重要组件就是操作系统的内核(称为HURD)。

1991 年Linus Torvalds编写出了与UNIX兼容的Linux操作系统内核并在GPL条款下发布。Linux之后在网上广泛流传，许多程序员参与了研发与修改。1992 年Linux与其他GNU软件结合，完全自由的操作系统正式诞生。该操作系统往往被称为“GNU/Linux”或简称Linux。（尽管如此GNU计划自己的内核Hurd依然在研发中，目前已经发布Beta版本。）

许多UNIX系统上也安装了GNU软件，因为GNU软件的质量比之前UNIX的软件还要好。GNU工具还被广泛地移植到Windows和Mac OS上 。

开源软件的最著名的官方网站为：http://sourceforge.net，在这里你可以下载到所需的大多数开源软件。

1. windows 下 GNU 的使用

如果想在 windows 下使用 GNU，有两个工具可以为你提供帮助。即 MinGW 和 Cygwin。

- MinGW

MinGW，即 Minimalist GNU For Windows。它是一些头文件和端口库的集合，该集合允许人们在没有第三方动态链接库的情况下使用 GCC（GNU Compiler C）产生 Windows32 程序。

在基本层，MinGW 是一组包含文件和端口库，其功能是允许控制台模式的程序使用微软的标准 C 运行时间库（MSVCRT.DLL）,该库在所有的 NT OS 上有效，在所有的 Windows 95 发行版以上的 Windows OS 有效，使用基本运行时间，你可以使用 GCC 写控制台模式的符合美国标准化组织（ANSI）程序，可以使用微软提供的 C 运行时间扩展。该功能是 Windows32 API 不具备的。下一个组成部分是 w32api 包，它是一组可以使用 Windows32 API 的包含文件和端口库。与基本运行时间相结合，就可以有充分的权利既使用 CRT（C Runtime）又使用 Windows32 API 功能。

- Cygwin

Cygwin 是 Cygwin 公司（http://cygwin.com/）的产品，它提供了 Windows 操作系统下的一个 UNIX 环境，它可以帮助程序研发人员把应用程序从 UNIX/Linux 移植到 Windows 平台，是一个功能强大的工具集。

Cygwin 由两部分组成：

- ✧ cygwin1.dll:它作为 UNIX 的一个仿真层，提供 UNIX API 功能
- ✧ 一组工具：它的功能是负责创建一个 UNIX 或 Linux 的外观界面。Cygwin 动态链接库，即 cygwin1.dll 可以在 Windows CE 以外，Windows 95 以上的所有非 beta 版本的 Windows OS 下工作，如 Windows 98,Windows 2000 等。目前该软件的最新版本的 Cygwin 是 1.1.x 或 1.3.x.可以从 cygwin 公司的网站上直接下载（http://cygwin.com/setup.exe ）

网络管理人员通过 Cygwin 可以很容易地远程登录到任何一台 PC 机，在 UNIX/Linux 外壳（shell）下解决问题。在任何一台 Windows OS 计算机上运行外壳

### 1.1.2 Linux内核

Linux 是一款类 UNIX 的操作系统，除了内核外，Linux 系统中常用的系统程序是由美国自由软件基金会（Free Software Foundation）研发出来的，同时还有不少机构和个人在为 Linux 研发应用程序。

Linux 具有 UNIX 操作系统的程序接口和操作方式，也继承了 UNIX 稳定高效的特点，由于其在 GPL 的版权下发行，逐渐成为一款非常受人欢迎的多用户多任务操作系统，能够在 i386、SPARC、Alpha、Mips、Power PC 等不同的计算机硬盘平台上运行。

Linux 具有以下特色：

- 支持多用户多任务。
- 支持多 CPU。
- RAM 保护模式，程序之间不会相互干扰，保证了系统能长久无错运行。
- 动态加载程序，当程序加载 RAM 执行时，Linux 仅将磁盘中相关的程序模块加载，有效的提升了执行的效率和 RAM 管理。
- 支持多种文件系统，如 NTFS。

Linux 内核的版本号分为主版本号、次版本号和扩展版本号等，根据稳定版本、测试版本和研发版本定义不同版本序列。

稳定的主版本号用偶数表示，例如：2.2、2.4、2.6 等。

紧接着是次版本号，例如：2.6.13、2.6.14、等，次版本号不分奇偶数，顺序递增。每隔 1～2 个月发布一个稳定版本。

然后是升级版本号，例如：2.6.14.3 等，升级版本号部分奇偶数，顺序递增，每月几次发布升级版本号，修正最新的稳定版本的问题。

另外一种是测试版本，在下一个稳定版本发布之前，每个月发布几个测试版本，例如：2.6.12-rc1。通过测试，可以使内核正式发布的时候更加稳定。

还有一类是研发版本。研发版本的主版本号用奇数表示，例如：2.3、2.5。也有次版本号，例如：2.5.32 等。研发版本是不稳定的，适合内核研发者在新的稳定的主版本发布之前使用。

Linux 内核的重要特点是具有很强的可移植性（Portability），支持多种硬件平台，在大多数体系结构上都可以运行。Linux 内核除 32 位体系结构外也能支持 64 位的体系结构。

Linux 内核也具有很好的可伸缩性（Scalability），即可以在超级计算机上运行，也可以在个人计算机或者嵌入式设备上运行。能够支持的最小要求：32 位处理器，可以不带 MMU。

Linux 内核全部源代码遵守 GPL 软件许可，对于 Linux 等自由软件，必须对最终用户开放源代码，但是没有义务向其他任何人开放，在商业 Linux 公司中，通常会要求客户签署最终用户的使用许可。

私有的模块也是允许使用的，只要不被认定为 GPL 许可的代码，就可以按照私有许可使用。但是，私有的驱动不能静态的链接到内核当中，只能作为动态加载的模块使用。

与 Linux 2.4 版本相比，Linux 2.6 内核吸收了一些新技术，在性能、可伸缩性、可用性等方面都有提高，具有许多新特性，内核也有很大修改。

Linux 2.6 内核的重要特性如下：

- 调度器算法

  Linux 2.6 内核的使用了新的调度器算法，它是由 Ingo Molnar 研发的 O(I)调度器算法，通过优先级数组的数据结构来实现。优先级数组可以使每个优先级都有相应的任务队列，还有一个优先级位图。每个优先级对应位图中一位，通过位图可以快速执行最高优先级任务。因为优先级个数是固定的，所以查找的时间也是固定的，不受系统运行任务数的影响。

  新的调度器为每个调度器维护 2 个优先级数组：有效数组和过期数组。有效数组内

这使 O(I)调度器算法在高负载的情况下表现及其出色，并且对多处理器调度有很好的扩展。

- 内核抢占算法

在 Linux2.4 以前的内核版本中，内核空间运行的任务（包括通过系统调用进入内核空间的用户任务）不允许被抢占。在 Linux 2.6 内核中采纳了有 Robert Love 研发的补丁，为了让更重要的用户应用程序可以继续运行，Linux 2.6 内核允许内核没抢占，大大减小了用户交互、多媒体等应用程序的调度延迟。提升了 Linux 在实时和嵌入式情况下的性能。

- 线程模型

Linux 2.6 内核由 Ingo Molnar 重写了线程框架，基于一个 1:1 的线程模型，新的线程模型能够支持 NPTL(Native Posix Threading Library)线程库。NPTL 解决了传统的 Linux 线程库存在的问题，对系统的性能有很大的提升。

- 文件系统

相对于 Linux 2.4，Linux 2.6 对文件系统的支持在很多方面都有大的改进。关键的变化包括对扩展属性以及 POSIX 标准的访问控制的支持。

EXT2/EXT3 文件系统作为多数 Linux 系统缺省安装的文件系统，是在 2.6 中改进最大的一个，最主要的变化是对扩展属性的支持，也即给指定的文件在文件系统中嵌入一些元数据。新的扩展属性子系统的第一个用途就是实现 POSIX 访问控制链表。POSIX 访问控制是标准 UNIX 权限控制的超集，支持更细粒度的访问控制。

Linux 对文件系统曾还进行了大量的改进以兼容其他操作系统。Linux 2.6 对 NTFS 文件系统的支持也进行了重写；同时也支持 IBM 的 JFS（journaling file system），和 SGI 的 XFS。

- 声音

Linux 2.6 内核还添加了新的声音系统：ALSA（Advanced Linux Sound Architecture），新的声音体系结构支持 USB 音频和 MIDI 设备。全双工重放等。

- 总线

Linux 2.6 的 IDE/ATA、SCSI 等存储总线也都被更新。最主要的是重写了 IDE 子系统，解决了许多可扩展性问题以及其他限制。其次是可以象微软的 windows 操作系统那样检查介质的变动，以更好的兼容那些并不完全遵照标准规范的设备。

Linux 2.6 还大大提升了对 PCI 总线的支持，增强或者扩展了 USB、蓝牙（Bluetooth）、红外（IrDA）等外围设备总线。所有的总线设备类型（硬件、无线和存储）都集成到了 Linux 新的设备模型子系统中。

- 电源管理

Linux 2.6 支持高级电源配置管理界面（ACPI，Advanced Configuration and Power Interface）,ACPI 不同于 APM（高级电源管理），拥有这种接口的系统在改变电源状态时需要分别通知每一个兼容的设备。新的内核系统允许子系统跟踪需要进行电源状态转换的设备。

- 网络

Linux 是一种网络性能优越的操作系统，意见可以支持世界上大多数主流网络谢谢，包括 TCP/IP(IPv4/IPv6)、AppleTalk、IPX 等。

在网络硬件驱动方面，利用了 Linux 的设备模型底层的改进和许多设备驱动程序的升级。

在网络安全方面，Linux 2.6 的一个重要改进是提供了对 IPsec 协议的支持，IPsec 是在网络协议层为 IPv4 和 IPv6 提供加密支持的一组协议。由于安全是在协议层提供的，对应用层是透明的。它与 SSL 协议及其他 tunneling/security 协议很相似，但是位于一个低得多的层面。当前内核支持的加密算法包括 SHA、DES 等。

在协议方面，Linux 2.6 还加强了对组播网络的支持。网络组播使得由一点发出的数

- 用户界面层

Linux 2.6 中一个主要的内部改进是人机接口层的大量重写，加入了对近乎所有可接入设备的支持，从触摸屏到盲人用的设备，到各种各样的鼠标。

- 统一的设备模型

Linux 2.6 内核最值得关注的变化是创建了一个统一的设备模型，这个设备模型通过维持大量的数据结构囊括了几乎所有的设备结构和系统。这样做的好处是，可以改进设备的电源管理和简化设备相关的任务管理。

Linux 2.6 内核还引入了 sysfs 文件系统，提供了系统的设备模型的用户空间描述，通常 sysfs 文件系统挂接在/sys 目录下。

关于Linux内核的更详细的知识请参考参考文献[7]。

### 1.1.3 主要发行版

1. RadHat Linux

国内，乃至是全世界的Linux用户所最熟悉、最耳闻能详的发行版想必就是Red Hat了。Red Hat最早由Bob Young和Marc Ewing在 1995 年创建。而公司在最近才开始真正步入盈利时代，归功于收费的Red Hat Enterprise Linux（RHEL，Red Hat的企业版）。而正统的Red Hat版本早已停止技术支持，最后一版是Red Hat 9.0。于是，目前Red Hat分为两个系列：由Red Hat公司提供收费技术支持和更新的Red Hat Enterprise Linux，以及由社区研发的免费的Fedora Core。Fedora Core 1 发布于 2003 年年末，而FC的定位便是桌面用户。FC提供了最新的软件包，同时，它的版本更新周期也非常短，仅六个月。目前最新版本为FC 3，而FC4 也预定将于今年 6 月发布。这也是为什么服务器上一般不推荐采用Fedora Core。 关于Fedora Linux的更多的知识请参考参考文献[4]。

适用于服务器的版本是 Red Hat Enterprise Linux，而由于这是个收费的操作系统。于是，国内外许多企业或空间商选择CentOS。CentOS 可以算是 RHEL 的克隆版，但它最大的好处是免费！菜鸟油目前的服务器便采用的 CentOS 3.4。

Red Hat Linux 的优点是拥有数量庞大的用户，优秀的社区技术支持，许多创新；缺点是免费版（Fedora Core）版本生命周期太短，多媒体支持不佳；官方网站为：http://www.redhat.com/。

2. SUSE Linux

SUSE Linux 是全球 Linux 技术的领导者之一，作为一个开源操作系统软件的解决方案提供商。SUSE Linux 广泛应用于商务和家庭环境。SUSE Linux 独有的 Linux 技术专长及其拥有的全世界最大的开源软件研发团队都给 SUSE Linux 带来了作为当今最完整 Linux 解决方案的赞誉。SUSE Linux 在全世界拥有 500 多人的职员，其办事处遍布欧洲、拉丁美洲和美国，这一切都以对 Linux 社区和开源软件研发的支持为中心。2003 年 11 月，Novell 收购了 SUSE LINUX。

与当前诸多Linux产品相比，SUSE Linux 10.0设计独特，简单易用，并拥有一切Linux入门所需的功能和特性。SUSE Linux 10 的桌面功能包括最新版的Firefox网页浏览器，最新版兼容Windows的OpenOffice.org 2.0 套件，电子邮件和即时信息客户端，图形设计、编辑和管理应用，以及重要的安全工具，包括垃圾屏蔽装置、防病毒软件以及集成的防火墙。在诸多新功能中还包括Beagle桌面搜索引擎以及支持MP3的Amarok。

除了完整的桌面功能之外，SUSE Linux 10 还提供 1,500 多个开源 Linux 应用及程序包，适用于高级网页寄存（Web Hosting）、应用和研发以及家庭网络，用户可以根据自身需要选择安装，其内容可谓丰富多彩。

据悉，与当前诸多 Linux 产品相比，SUSE Linux 10.0 设计独特，简单易用，并拥有一切 Linux 入门所需的功能和特性。SUSE Linux 10 的桌面功能包括最新版的 Firefox 网页浏览器，最新版兼容 Windows 的 OpenOffice.org 2.0 套件，电子邮件和即时信息客户端，图形设计、编辑和管理应用，以及重要的安全工具，包括垃圾屏蔽装置、防病毒软件以及集成的防火墙。在诸多新功能中还包括 Beagle 桌面搜索引擎以及支持 MP3 的 Amarok。

除了完整的桌面功能之外，SUSE Linux 10 还提供 1,500 多个开源 Linux 应用及程序包，适用于高级网页寄存（Web Hosting）、应用和研发以及家庭网络，用户可以根据自身需要选择安装，其内容可谓丰富多彩。

Novell 将向零售版用户提供安装支持以及完整的使用手册，帮助用户以极低的价格获得最完善的 Linux。SUSE Linux 10.0 为个人用户和研发人员提供了一个质优价廉的选择，满足了他们对 Linux 的性能、安全性、可靠性以及核心计算的需求。

SUSE Linux 10 得益于 Novell 的 openSUSE 计划，openSUSE 于 2005 年 8 月 9 日发布是针对 SUSE LINUX 产品的开源计划，旨在全球范围内推广 Linux 的使用。作为该计划的第一个成果，SUSE Linux 10 首次包含了来自全球 Linux 社区研发人员提供的改良代码及纠错程序。此前的测试版吸引了众多 Linux 社区研发人员，正是他们的积极参予和热心奉献重新定义了安装、配置以及使用简易性的标准。“这从根本上改变了这一产品。”Novell 副总裁兼 Linux 开源平台及服务事业部总经理 David Patrick 对这社区的奉献精神表达了自己的敬意，“Linux 社区帮助扩展了 SUSE Linux 产品的可用性，所以我们将以最快速度发布 SUSE Linux 10。”

作为操作系统SUSE Linux可以取代 Windows ，也可以作为第二个操作系统附属于 Windows 系统，这样他们可以先体验一下Linux，而不影响现有软件的安装。SUSE Linux 10 的零售版为最终用户提供安全和便利的Linux产品，包括可安装的媒体、用户手册，以及 90 天安装支持服务。完整的SUSE Linux 10 系统包市场售价 60 美元。

来自 IDC 的统计资料，Windows Server 的销售额一直以超过 20%的速度增长，但是，Linux 也正在从最初的学术应用和因特网应用范围进入到企业应用。目前，Linux 的出货量增长飞快，每年增长大约 60%。

作为著名的 Linux 操作系统制造商，Novell 携其 Suse Linux 已在美国市场称雄。同时，还强力进军巴西和印度市场，并在中国获得了一些企业应用。作为面向中国的全球 Linux 产品唯一完全本地化的计划，openSUSE 计划的中文网站 openSUSE.org.cn 也已经正式开通，这显示出 Novell 对中国市场的重视。

SUSE Linux Enterprise 10 平台具备了良好的操作性、低廉的支出费用而且极具互动性，是所有 Novell 公司下一代企业 Linux 产品的基础，利用该平台，商务企业可更好地应对不断加剧的竞争、螺旋式上升的成本、不断加剧的系统安全威胁，以及满足客户严格的要求。

除了 Xen 虚拟技术外，SUSE Linux Enterprise 10 还包括许多 Novell 发起的 Linux 技术创新。如：SUSELinux Enterprise 的存储管理集成了 Oracle Cluster File System2, Enterprise Volume Manager 和 Heartbeat 2 群集服务，提供全面开源、高可用性的存储解决方案。Xgl 是 Linux 桌面系统的新核心技术，它使用硬件加速，可以实现 Linux 系统下真正意义上的 3D 图形效果。

SUSE Linux的优点是专业，易用的YaST软件包管理系统；缺点是FTP发布通常要比零售版晚 1~3 个月；官方网站为http://www.suse.com/。关于SUSE Linux使用的更多的知识请参考参考文献[3]。

3. Debian Linux

Debian 最早由 Ian Murdock 于 1993 年创建。可以算是迄今为止，最遵循 GNU 规范的 Linux 系统。Debian 系统分为三个版本分支（branch）：stable, testing 和 unstable。截至 2005 年 5 月，这三个版本分支分别对应的具体版本为：Woody, Sarge 和 Sid。其中，unstable 为最新的测试版本，其中包括最新的软件包，但是也有相对较多的 bug，适合桌面用户。testing 的版本都经过 unstable 中的测试，相对较为稳定，也支持了不少新技术（比如 SMP 等）。而 Woody 一般只用于服务器，上面的软件包大部分都比较过时，但是稳定和安全性都非常的高。

为何有如此多的用户痴迷于 Debian 呢？apt-get / dpkg 是原因之一。dpkg 是 Debian 系列特有的软件包管理工具，它被誉为所有 Linux 软件包管理工具（比如 RPM）中最强大的软件包管理工具！配以 apt-get，利用 dpkg 在 Debian 上安装、升级、删除和管理软件变得非常容易。许多 Debian 的用户都开玩笑的说，Debian 将他们养懒了，因为只要简单得敲一下”apt-get upgrade && apt-get update”，机器上所有的软件就会自动更新了……。

Debian Linux的优点是遵循GNU规范，100%免费，优秀的网络和社区资源，强大的 apt-get ； 缺点是安装相对不易， stable 分支的软件极度过时； 官方网站为 http://www.debian.org。关于Debian Linux更多的知识请参考参考文献[5]。

4. Ubuntu Linux

简单而言，Ubuntu Linux 就是一个拥有 Debian 所有的优点，以及自己所加强的优点的近乎完美的 Linux 操作系统。:) Ubuntu 是一个相对较新的发行版，但是，它的出现可能改变了许多潜在用户对 Linux 的看法。也许，从前人们会认为 Linux 难以安装、难以使用，但是，Ubuntu 出现后，这些都成为了历史。Ubuntu 基于 Debian Sid，所以这也就是笔者所说的，Ubuntu 拥有 Debian 的所有优点，包括 apt-get。然而，不仅如此而已，Ubuntu 默认采用的 GNOME 桌面系统也将 Ubuntu 的界面装饰的简易而不失华丽。当然，如果你是一个 KDE 的拥护者的话，Kubuntu 同样适合你！

Ubuntu 的安装非常的人性化，只要按照提示一步一步进行，安装和 Windows 同样简便！并且，Ubuntu 被誉为对硬件支持最好最全面的 Linux 发行版之一，许多在其他发行版上无法使用，或者默认配置时无法使用的硬件，在 Ubuntu 上轻松搞定。并且，Ubuntu 采用自行加强的内核（kernel），安全性方面更上一层楼。并且，Ubuntu 默认不能直接 root 登陆，必须从第一个创建的用户通过 su 或 sudo 来获取 root 权限（这也许不太方便，但无疑增加了安全性，避免用户由于粗心而损坏系统）。Ubuntu 的版本周期为六个月，弥补了 Debian 更新缓慢的不足。

Ubuntu Linux的优点是人气颇高的论坛提供优秀的资源和技术支持，固定的版本更新周期和技术支持，可从Debian Woody直接升级；缺点是还未建立成熟的商业模式。官方网站为：http://www.ubuntulinux.org。关于Debian Linux更多的知识请参考参考文献[6]。

## 1.2 常用工具

在Linux系统研发过程中，用户最常用的一个工具就是shell，shell是Linux与用户之间通信的媒介。这种通信方式以交互方式或Shell 脚本方式执行。Shell 脚本是放在文件中的一串Shell和操作系统命令，shell 脚本命令的详细说明 请参考文档[10]。

常用的 shell 有 Bash 和 konsole,笔者推荐读者使用 konsole,它能够载一个终端上打开多个会话，可以通过“shift+方向键”来切换会话。极大的为用户提供了方便。Konsole 的笔者为 Lars Doelle.目前的维护者为 Kurt V.Hindenburg，当前的最高版本为 1.6。

Konsole 的使用技巧如下：

- 按下 Ctrl+Alt+N 将启动新的标准会话选择文本时按住 Ctrl 和 Alt，Konsole 就会选择列。
- 双击可以选择整个单词第二次点击后，移动鼠标而不松开鼠标键，就可以选择更多的单词。
- 三次点击鼠标可以选择整行。
- 第三次点击后，移动鼠标而不松开鼠标键，就可以选择更多的行。
- 按住 Shift 键，再按向左或向右方向键， 就可以在 Konsole 的各个会话间切换
- 用 Ctrl+Alt+S 快捷键可以将当前 Konsole 会话重新命名
- 在标签上按下鼠标中键即可移动会话
- 按住 Shift 键，再按 Page Up 或 Page Down 键， 可以按页面大小卷动以查看屏幕上以前的记录
- 按住 Shift 键，再按向上或向下方向键， 可以按行卷动以查看屏幕上以前的记录
- 按住 Shift 键再按 Insert 键，可以插入剪贴板里的东西
- 选择文本时按住 Ctrl，Konsole 就会忽略行之间的间隔

需要说明的是,在利用 shell 进行文件操作时,有的文件夹或者文件名可能会很长,完全逐字输入比较麻烦,在输入命令的任何时刻,可以按“tab”键，当这样做时，系统将试图补齐此时已输入的命令，如果系统发现有多个文件或文件夹以用户已输入的字母开头，这时候系统会自动补齐公共首字符串，当用户再次按下“tab”键时，系统会显示出所有以公共首字符串开头的文件或文件夹列表。

下面介绍下常用的 shell 命令，需要说明的是在本书中只作常用的介绍，因为本书不是帮助文档性质，更详细的用法请参考各命令的帮助文档。查看帮助文档的方法为：

＃commandNmae --help。

1. cp

该命令的功能是将给出的文件或目录拷贝到另一文件或目录中，就如同 DOS 下的 copy 命令一样，功能非常强大。

用法：cp [选项]... [-T] 源 目的
或：cp [选项]... 源... 目录
或：cp [选项]... -t 目录 源...
主要的选项如下：
-f,--force //强制复制，不再提示
-R,-r,--recursive //复制目录及目录内的所有项目
常用的操作方式如下：
＃cp –rf ... 源... 目录

2. mv

用户可以使用 mv 命令来为文件或目录改名或将文件由一个目录移入另一个目录中。该命令如同 DOS 下的 ren 和 move 的组合。

用法：mv [选项]... [-T] 源 目的
或：mv [选项]... 源... 目录
或：mv [选项]... -t 目录 源...
常用的操作方式如下：
#mv filename_1 fileName_2 //为文件重命名

3. rm

在 linux 中创建文件很容易，系统中随时会有文件变得过时且毫无用处。用户可以用 rm 命令将其删除。该命令的功能为删除一个目录中的一个或多个文件或目录，它也可以将某个目录及其下的所有文件及子目录均删除。对于链接文件，只是删除了链接，原有文件均保持不变。

用法：rm [选项]... 目录...
主要的选项如下：
-f,--force //强制删除，不再提示
-R,-r,--recursive //删除目录及目录内的所有项目
常用的操作方式如下：
#rm –rf directoryName //删除整个目录

4. mkdir

若目录不是已经存在则创建目录。

用法：mkdir [选项] 目录...
主要的选项如下：
-m, --mode=模式 设定权限<模式> (类似 chmod)，而不是 rwxrwxrwx 减 umask
-p, --parents 需要时创建上层目录，如目录早已存在则不当作错误
常用的操作方式如下：
＃mkdir folderName //创建文件夹

5. cd

改变工作目录。该命令将当前目录改变至 directory 所指定的目录。为了改变到指定目录，用户必须拥有对指定目录的执行和读 权限。

用法：cd [-L|-P] [dir]
常用的操作方式如下：
＃cd .. //改变到父目录
#cd - //改变到上次进入的目录

6. pwd

在 Linux 层次目录结构中，用户可以在被授权的任意目录下利用 mkdir 命令创建新目录，也可以利用 cd 命令从一个目录转换到另一个目录。然而，没有提示符来告知用 户目前处于哪一个目录中。要想知道当前所处的目录，可以使用 pwd 命令，该命令显示整个路径名。

用法：pwd [-LP]

常用的操作方式如下：

#pwd　　//显示当前目录

7. ls

对于每个目录，该命令将列出其中的所有子目录与文件。对于每个文件，ls 将输出其文件名以及所要求的其他信息。默认情况下，输出条目按字母顺序排序。当未给出目录名或是文件名时，就显示当前目录的信息。

用法：ls [选项]... [文件]...

主要的选项如下：

-l　　//使用较长格式列出信息

常用的操作方式如下：

＃ls　　//列出当前目录的所有子目录与文件

#ls –l　　//列出当前目录的所有子目录与文件的详细信息

8. tar

tar 可以为文件和目录创建档案。利用 tar，用户可以为某一特定文件创建档案（备份文件），也可以在档案中改变文件，或者向档案中加入新的文件。tar 最初被用来在磁带上创建档案，现在，用户可以在任何设备上创建档案。利用 tar 命令，可以把一大堆的文件和目录全部打包成一个文件，这对于备份文件或将几个文件组合成为一个文件以便于网络传输是非常有用的。

用法：tar [OPTIONS…] [FILE] …

主要的选项如下：

-A,--catenate, -concatentate　　//向一个档案文件中追加文件。

-c,--create　　//创建一个归档文件。

-d,--diff,--compare　　//比较档案文件和文件系统的差异

--delete　　//从档案文件中删除文件

-r,--append　　//在档案文件的尾部追加文件

-t,--list　　//列出档案文件的内容

-u,update　　//向档案文件中追加新文件

-x,--extract,--get　　//从档案文件中抽取文件

主要的压缩工具选项如下：

-j,--bzip2　　//利用 bzip2 来进行压缩解压

-z,--gzip　　//利用 gzip 来进行压缩解压

常用的操作方式如下：

＃tar -cf archive.tar foo bar ….　　//为 foo 和 bar 创建档案文件 archive.tar

＃tar tvf archive.tar　　//列出 archive.tar 中的所有文件

＃tar –xf　　//从 archive.tar 中抽取所有文件

9. grep

在指定的文件或者标准输入中按指定的模式搜索。缺点是 grep 命令一次只能搜索一个指定的模式；但类似的命令 egrep 可以检索扩展的正则表达式(包括表达式组和可选项)；而 fgrep 命令可以检索固定字符串，fgrep 不能识别正则表达式，是快速搜索命令。

用法：grep [OPTION]... PATTERN [FILE] ...

-E,--extended-regexp　　//每个模式作为一个扩展的正则表达式对待。

-F,--fixed-strings　　//每个模式作为一组固定字符串对待（以新行分隔）。

-i,--ignore-case　　//比较时不区分大小写。

常用的操作方式如下：

#grep –i ‘hello world’ menu.h main.c

10. find

在目录结构中搜索文件，并执行指定的操作。此命令提供了相当多的查找条件，功能很强大。

用法：find [路径...] [表达式]

- name “string” //查找文件名匹配所给字串的所有文件，字串内可用通配符*、?、

常用的操作方式如下：

#find –name “string”

11. chmod

chmod 命令可以改变文件夹或者文件的用户控制权限。其中用“r”表示只读，用“w”表示只写，用“x”表示可执行，而“r”“w”“x”分别对于八进制模式下的“4”“2”“1”。每一文件或目录的访问权限都有三组，每组用三位表示，分别为文件属主的读、写和执行权限；与属主同组的用户的读、写和执行权限；系统中其他用户的读、写和执行权限。件被创建时，文件所有者自动拥有对该文件的读、写和可执行权限，以便于对文件的阅读和修改。用户也可根据需要把访问权限设置为需要的任何组合。

用法：chmod [选项]... 模式[,模式]... 文件...

或：chmod [选项]... 八进制模式 文件...

或：chmod [选项]... --reference=参考文件 文件...

主要的选项如下：

-f, --silent, --quiet //去除大部份的错误信息

-R, --recursive //以递归方式更改所有的文件及子目录

常用的操作方式如下：

#chmod +x filename //将 filename 文件属性设为可执行文件

#chmod –R 777 fileFolders //将 fileFolders 设为可被所有用户完全控制

12. ln

该命令在文件之间创建链接。这种操作实际上是给系统中已有的某个文件指定另外一个可用于访问它的名称。对于这个新的文件名，我们可以为之指定不同的访问权限，以控制对信息的共享和安全性的问题。 如果链接指向目录，用户就可以利用该链接直接进入被链接的目录而不用打一大堆的路径名。而且，即使我们删除这个链接，也不会破坏原来的目录。

用法：ln [OPTION]... [-T] 目标 链接名

或：ln [OPTION]... 目标

或：ln [OPTION]... 目标... 目录

或：ln [OPTION]... -t 目录 目标...

主要的选项如下：

-f,--force //强制建立链接，不再提示

-s,--symbolic //创建符号链接而不是硬链接

常用的操作方式如下：

＃ln –sf directoryNmae/filename filename

13. df

显示文件驻留的文件系统的信息，默认为所有文件系统的信息

用法：df [选项]... [文件]...

常用的操作方式如下：

＃df -h

相近的命令有 fdisk。

14. su

这个命令非常重要。它可以让一个普通用户拥有超级用户或其他用户的权限，也可以让超级用户以普通用户的身份做一些事情。普通用户使用这个命令时必须有超级用户或其他用户的口令。如要离开当前用户的身份，可以打 exit。

用法：su [OPTION]... [-] [USER [ARG]...]

常用的操作方式如下：

＃su usreName

15. logname

打印当前用户的名称。

用法：logname [选项]

常用的操作方式如下：

# logname

16. free

用法：free [-b|-k|-m|-g] [-l] [-o] [-t] [-s delay] [-c count] [-V]

主要的选项如下：

-b,-k,-m,-g 按字节, KB, MB, 或者 GB 方式显示输出

-t 显示 RAM + swap 总和

-s 按 delay 给出的秒值定时更新

-c 更新次数

常用的操作方式如下：

＃free

## 1.3 编译与调试

无论是在 Linux 还是在 Unix 环境中，make 都是一个非常重要的编译命令。不管是自己进行项目研发还是安装应用软件，我们都经常要用到 make 或 make install。利用 make 工具，我们可以将大型的研发项目分解成为多个更易于管理的模块，对于一个包括几百个源文件的应用程序，使用 make 和 makefile 工具就可以简洁明快地理顺各个源文件之间纷繁复杂的相互关系。而且如此多的源文件，如果每次都要键入 gcc 命令进行编译的话，那对程序员来说简直就是一场灾难。而 make 工具则可自动完成编译工作，并且可以只对程序员在上次编译后修改过的部分进行编译。因此，有效的利用 make 和 makefile 工具可以大大提高项目研发的效率。同时掌握 make 和 makefile 之后，您也不会再面对着 Linux 下的应用软件手足无措了。

但令人遗憾的是，在许多讲述 Linux 应用的书籍上都没有详细介绍这个功能强大但又非常复杂的编译工具。在这里我就向大家详细介绍一下 make 及其描述文件 makefile。

1. Makefile 文件

Make 工具最主要也是最基本的功能就是通过 makefile 文件来描述源程序之间的相互关系并自动维护编译工作。而 makefile 文件需要按照某种语法进行编写，文件中需要说明如何编译各个源文件并连接生成可执行文件，并要求定义源文件之间的依赖关系。makefile 文件是许多编译器--包括 Windows NT 下的编译器--维护编译信息的常用方法，只是在集成研发环境中，用户通过友好的界面修改 makefile 文件而已。

在 UNIX 系统中，习惯使用 Makefile 作为 makfile 文件。如果要使用其他文件作为 makefile，则可利用类似下面的 make 命令选项指定 makefile 文件：

$ make -f Makefile.debug

例如，一个名为 prog 的程序由三个 C 源文件 filea.c、fileb.c 和 filec.c 以及库文件 LS 编译生成，这三个文件还分别包含自己的头文件 a.h 、b.h 和 c.h。通常情况下，C 编译器将会输出三个目标文件 filea.o、fileb.o 和 filec.o。假设 filea.c 和 fileb.c 都要声明用到一个名为 defs 的文件，但 filec.c 不用。即在 filea.c 和 fileb.c 里都有这样的声明：

```
#include "defs"
```

那么下面的文档就描述了这些文件之间的相互联系:

```
------------------------------------------------------
#It is a example for describing makefile
prog : filea.o fileb.o filec.o
cc filea.o fileb.o filec.o -LS -o prog
filea.o : filea.c a.h defs
cc -c filea.c
fileb.o : fileb.c b.h defs
cc -c fileb.c
filec.o : filec.c c.h
cc -c filec.c
------------------------------------------------------
```

这个描述文档就是一个简单的 makefile 文件。

从上面的例子注意到，第一个字符为 # 的行为注释行。第一个非注释行指定 prog 由三个目标文件 filea.o、fileb.o 和 filec.o 链接生成。第三行描述了如何从 prog 所依赖的文件建立可执行文件。接下来的 4、6、8 行分别指定三个目标文件，以及它们所依赖的.c 和.h 文件以及 defs 文件。而 5、7、9 行则指定了如何从目标所依赖的文件建立目标。

当 filea.c 或 a.h 文件在编译之后又被修改，则 make 工具可自动重新编译 filea.o，如果在前后两次编译之间，filea.C 和 a.h 均没有被修改，而且 test.o 还存在的话，就没有必要重新编译。这种依赖关系在多源文件的程序编译中尤其重要。通过这种依赖关系的定义，make 工具可避免许多不必要的编译工作。当然，利用 Shell 脚本也可以达到自动编译的效果，但是，Shell 脚本将全部编译任何源文件，包括哪些不必要重新编译的源文件，而 make 工具则可根据目标上一次编译的时间和目标所依赖的源文件的更新时间而自动判断应当编译哪个源文件。

Makefile 文件作为一种描述文档一般需要包含以下内容:

- 宏定义
- 源文件之间的相互依赖关系
- 可执行的命令

Makefile 中允许使用简单的宏指代源文件及其相关编译信息，在 Linux 中也称宏为变量。在引用宏时只需在变量前加$符号，但值得注意的是，如果变量名的长度超过一个字符，在引用时就必须加圆括号（）。

下面都是有效的宏引用：

```
$(CFLAGS)
$2
$Z
$(Z)
```

其中最后两个引用是完全一致的。

需要注意的是一些宏的预定义变量，在 Unix 系统中，$*、$@、$?和$<四个特殊宏的值在执行命令的过程中会发生相应的变化，而在 GNU make 中则定义了更多的预定义变量。关于预定义变量的详细内容，

宏定义的使用可以使我们脱离那些冗长乏味的编译选项，为编写 makefile 文件带来很大的方便。

```
------------------------------------------------------------
# Define a macro for the object files
OBJECTS= filea.o fileb.o filec.o
# Define a macro for the library file
LIBES= -LS
# use macros rewrite makefile
prog: $(OBJECTS)
cc $(OBJECTS) $(LIBES) -o prog
……
------------------------------------------------------------
```

此时如果执行不带参数的 make 命令，将连接三个目标文件和库文件 LS；但是如果在 make 命令后带有新的宏定义：

```
make "LIBES= -LL -LS"
```

则命令行后面的宏定义将覆盖 makefile 文件中的宏定义。若 LL 也是库文件，此时 make 命令将连接三个目标文件以及两个库文件 LS 和 LL。

在 Unix 系统中没有对常量 NULL 作出明确的定义，因此我们要定义 NULL 字符串时要使用下述宏定义：

```
STRINGNAME=
```

2. Make 命令

在 make 命令后不仅可以出现宏定义，还可以跟其他命令行参数，这些参数指定了需要编译的目标文件。其标准形式为：

```
target1 [target2 …]:[:][dependent1 …][;commands][#…]
[(tab) commands][#…]
```

方括号中间的部分表示可选项。Targets 和 dependents 当中可以包含字符、数字、句点

和"/"符号。除了引用，commands 中不能含有"#",也不允许换行。

在通常的情况下命令行参数中只含有一个":"，此时 command 序列通常和 makefile 文件中某些定义文件间依赖关系的描述行有关。如果与目标相关连的那些描述行指定了相关的 command 序列，那么就执行这些相关的 command 命令，即使在分号和(tab)后面的 aommand 字段甚至有可能是 NULL。如果那些与目标相关连的行没有指定 command，那么将调用系统默认的目标文件生成规则。

如果命令行参数中含有两个冒号"::"，则此时的 command 序列也许会和 makefile 中所有描述文件依赖关系的行有关。此时将执行那些与目标相关连的描述行所指向的相关命令。同时还将执行 build-in 规则。

如果在执行 command 命令时返回了一个非"0"的出错信号，例如 makefile 文件中出现了错误的目标文件名或者出现了以连字符打头的命令字符串，make 操作一般会就此终止，但如果 make 后带有"-i"参数，则 make 将忽略此类出错信号。

Make 命本身可带有四种参数：标志、宏定义、描述文件名和目标文件名。其标准形式为：

Make [flags] [macro definitions] [targets]

Unix 系统下标志位 flags 选项及其含义为：

-f file 指定 file 文件为描述文件，如果 file 参数为"-"符，那么描述文件指向标准输入。如果没有"-f"参数，则系统将默认当前目录下名为 makefile 或者名为 Makefile 的文件为描述文件。在 Linux 中， GNU make 工具在当前工作目录中按照 GNUmakefile、makefile、Makefile 的顺序搜索 makefile 文件。

-i 忽略命令执行返回的出错信息。

-s 沉默模式，在执行之前不输出相应的命令行信息。

-r 禁止使用 build-in 规则。

-n 非执行模式，输出所有执行命令，但并不执行。

-t 更新目标文件。

-q make 操作将根据目标文件是否已经更新返回"0"或非"0"的状态信息。

-p 输出所有宏定义和目标文件描述。

-d Debug 模式，输出有关文件和检测时间的详细信息。

Linux 下 make 标志位的常用选项与 Unix 系统中稍有不同，下面我们只列出了不同部分：

-c dir 在读取 makefile 之前改变到指定的目录 dir。

-I dir 当包含其他 makefile 文件时，利用该选项指定搜索目录。

-h help 文挡，显示所有的 make 选项。

-w 在处理 makefile 之前和之后，都显示工作目录。

通过命令行参数中的 target ，可指定 make 要编译的目标，并且允许同时定义编译多个目标，操作时按照从左向右的顺序依次编译 target 选项中指定的目标文件。如果命令行中没有指定目标，则系统默认 target 指向描述文件中第一个目标文件。

通常，makefile 中还定义有 clean 目标，可用来清除编译过程中的中间文件，例如：

```
#clean:
#rm -f *.o
```

运行 make clean 时，将执行 rm -f *.o 命令，最终删除所有编译过程中产生的所有中间文件。

隐含规则

在 make 工具中包含有一些内置的或隐含的规则，这些规则定义了如何从不同的依赖文件建立特定类型的目标。Unix 系统通常支持一种基于文件扩展名即文件名后缀的隐含规则。这种后缀规则定义了如何将一个具有特定文件名后缀的文件（例如.c 文件），转换成为具有另一种文件名后缀的文件（例如.o 文件）：

```
.c:.o
$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ $<
```

系统中默认的常用文件扩展名及其含义为：

.o 目标文件

.c C 源文件

.f FORTRAN 源文件
.s 汇编源文件
.y Yacc-C 源语法
.l Lex 源语法

在早期的 Unix 系统系统中还支持 Yacc-C 源语法和 Lex 源语法。在编译过程中，系统会首先在 makefile 文件中寻找与目标文件相关的.C 文件，如果还有与之相依赖的.y 和.l 文件，则首先将其转换为.c 文件后再编译生成相应的.o 文件；如果没有与目标相关的.c 文件而只有相关的.y 文件，则系统将直接编译.y 文件。

而 GNU make 除了支持后缀规则外还支持另一种类型的隐含规则--模式规则。这种规则更加通用，因为可以利用模式规则定义更加复杂的依赖性规则。模式规则看起来非常类似于正则规则，但在目标名称的前面多了一个 % 号，同时可用来定义目标和依赖文件之间的关系，例如下面的模式规则定义了如何将任意一个 file.c 文件转换为 file.o 文件：

```
%.c:%.o
$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ $<
```

下面将给出一个较为全面的示例来对 makefile 文件和 make 命令的执行进行进一步的说明，其中 make 命令不仅涉及到了 C 源文件还包括了 Yacc 语法。本例选自"Unix Programmer's Manual 7th Edition, Volume 2A" Page 283-284

下面是描述文件的具体内容：

```
------------------------------------------------------
#Description file for the Make command
#Send to print
P=und -3 | opr -r2
#The source files that are needed by object files
FILES= Makefile version.c defs main.c donamc.c misc.c file.c \
dosys.c gram.y lex.c gcos.c
#The definitions of object files
OBJECTS= vesion.o main.o donamc.o misc.o file.o dosys.o gram.o
LIBES= -LS
LINT= lnit -p
CFLAGS= -O
make: $(OBJECTS)
cc $(CFLAGS) $(OBJECTS) $(LIBES) -o make
size make
$(OBJECTS): defs
gram.o: lex.c
cleanup:
-rm *.o gram.c
install:
@size make /usr/bin/make
cp make /usr/bin/make ; rm make
#print recently changed files
print: $(FILES)
pr $? | $P
touch print
test:
make -dp | grep -v TIME>1zap
/usr/bin/make -dp | grep -v TIME>2zap
diff 1zap 2zap
rm 1zap 2zap
lint: dosys.c donamc.c file.c main.c misc.c version.c gram.c
$(LINT) dosys.c donamc.c file.c main.c misc.c version.c \
gram.c
rm gram.c
arch:
ar uv /sys/source/s2/make.a $(FILES)
```

---

通常在描述文件中应象上面一样定义要求输出将要执行的命令。在执行了 make 命令之后，输出结果为：

```
$ make
cc -c version.c
cc -c main.c
cc -c donamc.c
cc -c misc.c
cc -c file.c
cc -c dosys.c
yacc gram.y
mv y.tab.c gram.c
cc -c gram.c
cc version.o main.o donamc.o misc.o file.o dosys.o gram.o \
-LS -o make
13188+3348+3044=19580b=046174b
```

最后的数字信息是执行"@size make"命令的输出结果。之所以只有输出结果而没有相应的命令行，是因为"@size make"命令以"@"起始，这个符号禁止打印输出它所在的命令行。

描述文件中的最后几条命令行在维护编译信息方面非常有用。其中"print"命令行的作用是打印输出在执行过上次"make print"命令后所有改动过的文件名称。系统使用一个名为 print 的 0 字节文件来确定执行 print 命令的具体时间，而宏$?则指向那些在 print 文件改动过之后进行修改的文件的文件名。如果想要指定执行 print 命令后，将输出结果送入某个指定的文件，那么就可修改 P 的宏定义：

```
make print "P= cat>zap"
```

在 Linux 中大多数软件提供的是源代码，而不是现成的可执行文件，这就要求用户根据自己系统的实际情况和自身的需要来配置、编译源程序。

关于make和Makefile的更详细信息，请参考参考文献[11]。

### 1.3.1 交叉编译

嵌入式系统是专用计算机系统，它对系统的功能、可靠性、成本、体积、功耗等方面都有着严格的要求。由于嵌入式硬件上的限制，一般不能安装发行版的 Linux 系统。这样对研发而言并不方便。

在当前的嵌入式软件研发方式上，为了研发的方便，普遍采用了 Host/Target 的方式进行研发。在 PC 端编译好目标端版本后，运用文件系统生成工具如 mkcramfs,生成相应的文件系统。然后通过一定的方式将生成的文件系统烧到目标端。

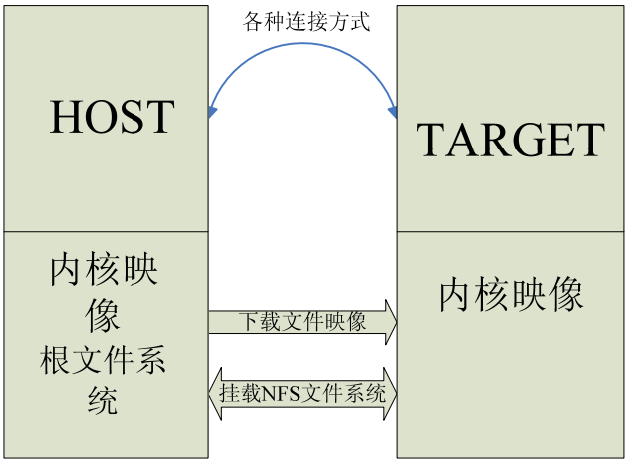


图1. HOST/TARGET 研发模型

### 1.3.2 编译工具链

Linux 软件从一开始就使用 GNU 的工具链，这些 GNU 的工具和软件都是开源的，可以免费从 http://www.gnu.org 或者其镜像上下载和编译，但是软件包总会呈现这样那样的版本匹配问题，笔者自己也曾遇见在低版本的 GCC 下编译通过但在高版本的 GCC 下却无法编辑通过的情况。

设置工具链在主机机器上创建一个用于编译将在目标上运行的内核和应用程序的构建环境 — 这是因为目标硬件可能没有与主机兼容的二进制执行级别。

工具链由一套用于编译、汇编和链接内核及应用程序的组件组成。 这些组件包括：

- BINUTILS：是二进制程序处理工具。它们包括诸如 ar、as、objdump、objcopy 等实用工具。
- GCC（GNU Compiler Collection）：为编译器，不但能够支持 C/C++语言的编译，而且能够支持 Fortran、Java、Ada 等编程语言。
- GLIBC：应用程序的函数库软件包，可以编译生成静态库和共享库，完整的 GCC 需要支持 glibc。
- GDB 是调试工具，可以读取可执行程序的符号表，对程序进行源码调试。

构建工具链建立了一个交叉编译器环境。本地编译器编译与本机同类的处理器的指令。交叉编译器运行在某一种处理器上，却可以编译另一种处理器的指令。重头设置交叉编译器工具链可不是一项简单的任务：它包括下载源代码、修补补丁、配置、编译、设置头文件、安装以及很多很多的操作。另外，这样一个彻底的构建过程对内存和硬盘的需求是巨大的。如果没有足够的内存和硬盘空间，那么在构建阶段由于相关性、配置或头文件设置等问题会突然冒出许多问题。

工具链的版本匹配是个大问题，但可以利用工具进行匹配测试，Crosstool 就是这样一个工具，Crosstool 的维护者为 Dan Kegel。官方网页为 http://www.kegel/crosstool。

### 1.3.3 GDB调试

在应用程序的研发中，调试的一个不可避免的环节，Linux 的最常用的调试器为 GNU 的 GDB（GNU Debugger）,该软件最早由 Richard Stallman 编写！GDB 对 C 和 C＋＋都有很好的支持，同时也能够支持对 Ada、Objective-C、Pascal 等语言的调试。除了 Linux 系统外，GDB 也能够在 UNIX 和 Windows 上运行。

使用者可以在程序运行时观察程序的内部结构和内存的使用情况，GDB是一种基于命令行工作模式的程序。目前最新的GDB版本为GDB 6.7(2007 年 9 月 11 日发布),但由于最新版本往往可能不会足够稳定，因此建议使用GDB 6.6（2006 年 12 月 18 日发布）,GDB的官方网站为http://www.gnu.org/software/gdb/gdb.html。关于GDB调试更多的知识请参考参考文献[12]。

目前 Linux 上已经有 GDB 的多个图形用户界面的前端，最著名的有 KDbg，KDbg 主要有 Johannes Sixt 完成，KDbg 的主页为 http://members.nextra.at/ohsixt/kdbg.html。

以下为 GDB 所提供的一些功能：

- 启动程序，通过设置运行环境和参数来运行指定的程序。
- 让程序在指定的断点处停止执行。
- 对程序做出相应的调整，这样就能在纠正一个错误后继续调试。

需要说明的是，GDB 调试的是可执行文件，如果想让 GDB 调试编译后生成的可执行文件，在利用 gcc 编译时需要带上 g 或者 gdb 选项。在 Qtopia 中，应该在 pro 文件的 config 选项中添加 debug 选项，方法为 CONFIG+=debug。

1. 运行 GDB

当需要利用 GDB 进行调试时，应首先设置好相应的环境变量，需要说明的是，对于 Qtopia 而言，如果编译生成的文件被安装到与 QTEDIR 不同的目录下，调试时的 QTEDIR 环境变量路径应设为安装目录。

启动 GDB 调试应用程序的命令如下：

＃gdb programName

然后在系统的(gdb)提示符后输入 run 命令就可以启动应用程序了。

常用的 GDB 命令如下：

| 命令 | 说明 |
|---|---|
| file | 指定要调试的可执行程序 |
| kill | 终止正在调试的可执行程序 |
| next | 执行一行源代码但不进入函数内部 |
| list | 列出上下文源代码 |
| step | 执行一行源代码并进入函数内部 |
| run | 执行当前的可执行程序 |
| quit | 结束 GDB 调试 |
| watch | 检查一个变量的值 |
| print | 打印表达式的值 |
| break N | 在指定的第 N 行源代码设置断点 |
| info break | 显示当前断点信息 |
| info func | 显示所有的函数名 |
| info local | 显示当前函数中的局部变量信息 |
| info prog | 显示当前被调试程序的执行状态 |
| info var | 显示所有的全局和静态变量名称 |
| make | 在不退出 GDB 的情况下运行 make 进行编译 |
| shell | 在不退出 GDB 的情况下运行 shell 命令 |
| continue | 继续执行正在被调试的程序 |

表1 GDB 常用命令

需要说明的是，当当前源代码并非单个文件时，设置断点的方式为 break filename.*:N。如果需要删除断点，首先利用 info break 获得当前的断点信息。然后利用 delete M 即可删除 M 断点。这里的 M 为利用 info break 显示出来的断点的标识值。

## 1.4 文件系统

所有的计算机应用程序都需要存储和检索信息，信息以一种单元的形式，也即文件，存储在磁盘或其他外部介质上。文件的构造、命名、存取、使用、保护和实现方法都是操作系统设计的主要内容，通常将操作系统处理文件的部分成为文件系统（File System），文件系统是操作系统用于明确磁盘或分区上的文件的方法和数据结构；即在磁盘上组织文件的方法。也指用于存储文件的磁盘或分区，或文件系统种类。

### 1.4.1 网络文件系统

网络文件系统（NFS,Network File System）最早是 Sun 研发的一种文件系统。NFS 允许一个系统在网络上共享目录和文件。通过使用 NFS,用户和程序可以像访问本地文件一样访问远端系统上的文件，这极大地简化了信息共享。

Linux 系统支持 NFS,并且可以配置启动 NFS 网络服务。

NFS 文件系统的优点如下：

- 本地工作站使用更少的磁盘空间，因为通常的数据可以存放在一台机器上而且可以通过网络访问。
- 用户可以通过网络访问共享目录，而不必在计算机上为每个用户都创建工作目录。
- 软驱、CDROM 等存储设备可以在网络上面共享使用。这可以减少整个网络设备上

的移动介质设备的数量。

- NFS 至少有一台服务器和一台（或者更多）客户机两个主要部分。客户机远程访问存放在服务器上的数据，需要配置启动 NFS 等相关服务。

网络文件系统的优点正好适合嵌入式 Linux 系统研发。目标版没有足够的存储空间，Linux 内核挂载网络根文件系统可以避免使用本地存储介质，快速建立 Linux 系统。这样就可以方便的运行和调试应用程序。

### 1.4.2 EXT2/EXT3

EXT2(The Second Extended Filesystem)和 EXT3(The Third Extended Filesystem)是 Linux 内核自己的文件系统。

EXT2 发布于 1993 年一月，它是有 Remy Card、Theodore TSo 和 Stephen Tweedie 编写的，它是 EXT 文件系统重写的版本。

EXT2 与传统的 Unix 文件系统有许多共性，都有块（block）、节点（incode）和目录（directory）的概念。尽管没有实现访问控制列表（ACL）、碎片、恢复删除文件和压缩功能，但是都预留了空间。另外，版本兼容机制可以让文件系统添加新的特性（如日志），同时保持最大程度兼容。

在启动的适合，大多数系统要检查文件系统的连续性（执行 e2fsck 命令）。EXT2 文件系统的超级块包含了几个字段，用来表示是否需要执行 fsck。如果文件系统没有卸载干净，或者超出最大挂载数，或者超出最大检查间隔周期，就会执行 fsck。

EXT2 使用的特点兼容机制很复杂。它允许在文件系统中安全地添加许多特点，不会牺牲老版本文件系统代码的兼容性。它的特点兼容机制是从 EXT2 版本 1 中引入的，原始的版本 0 并不支持。它有 3 格 32 位字段，一个是兼容的特点（COMPAT），一个是只读兼容的特点（RO_COMPAT）,一个是不兼容的特点（INCOMPAT）。

EXT2 元数据操作有异步和同步两种方式，据说异步元数据写操作比 FFS 同步元数据方案快，但是可靠性差一些。这两种方法都可以被相应的 fsck 程序处理。

EXT2 文件系统的磁盘布局会导致各种局限性。当前内核代码的实现也会导致其他一些局限性。许多局限性在文件系统第一次创建的适合就确定了，这取决于块大小的选择。节点与数据块的比例在创建的适合就确定了。因此增大节点数的唯一办法就是增大文件系统尺寸。

EXT2 的日志功能是 Stephen Tweedie 研发的。日志功能可以避免元数据污损并且需要 e2fsck 检查，而且不需要改变 EXT2 的磁盘布局。总之，日志是用来存储被修改全部元数据块的正规文件，优先于写到文件系统，这意味折可以对存在的 EXT2 文件系统创建日志，而不需要进行数据转换。

当修改文件系统的时候，事务会存储在日志中，在系统崩溃的适合可以完成或者没有完成事务处理。如果崩溃时事务处理完成，日志的块可以代表有效的文件系统状态，并且复制到文件系统。如果崩溃时事务处理没有完成，就不能保证事务处理的块的连续性（这就意味着所代表文件系统修改会丢失）。

EXT3 文件系统是 1999 年 9 月发布的。最早是 Stephen Tweedie 为 2.2 内涵版本写的。

EXT3 是 EXT2 文件系统的改进版，添加了日志等功能。EXT3 使用了 EXT2 文件系统的实现，还添加了事务处理的功能。日志功能通过块设备日志层（Journaling Block Device Layer,JBD）完成。

JBD 不是 EXT3 文件系统所特有的，它是专门为块设备添加日志功能而设计的。EXT3 文件系统代码会把执行的修改（提交事务）通知 JBD。日志支持事务的启动和停止。在系统崩溃的适合，日志可以快速重新执行事务以保持分区的连续性。事务处理对文件系统的单个原子更新操作。JBD 可以在块设备上处理外部日志。

EXT3 的数据模式分为 3 种：

- 写回模式（writeback mode）

对于这种模式的数据，EXT3 根本不做日志，在 XFS、JFS 和 ReiserFS 文件系统中，它缺省地提供了简单的元数据日志。崩溃重启可能引起正在写的数据出错。在这种模式下，EXT3 文件系统的性能最好。

- 有序模式（ordered mode）

对于这种模式的数据，EXT3 仅正式地做元数据日志，但是逻辑上把元数据和数据块组成一个事务单元。在向磁盘上写元数据之前，先写相关的数据块。这种模式性能比写回模式略微慢一点，但是比下面的日志模式快很多。

- 日志模式（journal mode）

对于这种模式的数据，EXT3 将对全部数据和元数据做日志处理。所有新的数据先写到日志区，然后写到它最终的位置。遇到崩溃时间，日志可以重做，保持数据和元数据的连续性。这种模式是最慢的，除了数据需要做同时从磁盘读出并且写回操作的情况。

EXT3 文件系统完全兼容 EXT2，EXT2 分区可以挂载成 EXT2 格式，EXT2 分区可以通过 tune2fs 命令转换成 EXT3 格式。

## 1.5 Linux的安装

对于使用 Linux 的用户而言，常常因为某种需要，同时也需要使用 Windows 操作系统。

### 1.5.1 虚拟机方式

虚拟机（VM）是支持多操作系统并行运行在单个物理服务器上的一种系统，能够提供更加有效的底层硬件使用。在虚拟机中，中央处理器芯片从系统其它部分划分出一段存储区域，操作系统和应用程序运行在“保护模式”环境下。如果在某虚拟机中出现程序冻结现象，这并不会影响运行在虚拟机外的程序操作和操作系统的正常工作。

虚拟机具有四种体系结构。第一种为“一对一映射”，其中以 IBM 虚拟机最为典型。第二种由机器虚拟指令映射构成，其中以 Java 虚拟机最为典型。Unix 虚拟机模型和 OSI 虚拟机模型可以直接映射部分指令，而其它的可以直接调用操作系统功能。

在真实计算机系统中，操作系统组成中的设备驱动控制硬件资源，负责将系统指令转化成特定设备控制语言。在假设设备所有权独立的情况下形成驱动，这就使得单个计算机上不能并发运行多个操作系统。虚拟机则包含了克服该局限性的技术。虚拟化过程引入了低层设备资源重定向交互作用，而不会影响高层应用层。通过虚拟机，用户可以在单个计算机上并发运行多个操作系统。

每个虚拟机由一组虚拟化设备构成，其中每个虚拟机都有对应的虚拟硬件。客户操作系统和应用程序可以运行在虚拟机上，而不需要提供任何交互作用的网络适配器的支持。虚拟服务器只是物理以太网中的一种软件仿真设备。

目前最流行的虚拟机软件有 Virtual PC 是与 VMWare，在笔者撰写本书时，Virtual PC 的最新版本为 Virtual PC 2007，VMWare 的最新版本为 VMware Workstation ACE Edition 6。

微软在一份有关Virtual PC 2007 应用程序兼容性白皮书中透露了Virtual PC 2007 的部分更新情况:

- 为Windows Vista优化:在性能、系统资源占用和稳定性有所改进。
- 改进基于 Windows Virtual Server 2005 R2 的性能
- 提供 64 位宿主操作系统支持:Virtual PC 2007 可以在 64 位版Windows Vista上运行。
- 增加了 64 位的音频驱动

在 VMware Workstation 的参考手册中，显示 VMware Workstation ACE Edition 6 的更新情况如下：

- 集成了虚拟调试器，使研发人员可以轻松在虚拟机上利用 Visual Studio(仅 Windows)和 Eclispe（windows 和 Linux）来测试、运行和调试程序。
- 使虚拟机可以在后台运行
- 增强了对以太网适配器的支持
- 增加了 64 位的音频驱动
- 支持 USB 2.0 设备
- 增加了多种宿主端和客户端操作系统

Virtual PC是与vmware类似的虚拟机软件。两者的主要区别如下：

- VMWare没有模拟显卡，要通过vmware-tools才能用上高分辨率和真彩色，否则只能用VGA。而Virtual PC模拟了一个比较通用的显卡：S3 Trio 32/64(4M)。从这一点看，Virtual PC比VMWare通用，但显示性能不如VMWare。
- Virtual PC 的网络共享方式与 VMWare 不同。VMWare 是通过模拟网卡实现网络共享的，而 Virtual PC 是通过在现有网卡上绑定 Virtual PC emulated switch 服务实现网络共享的。对于 win2000/xp 等操作系统，如果网线没插或没有网卡的时候，要安装 Microsoft 的 loopback 软网卡，才能实现网络共享。在 Virtual PC 的 global setting 里，当有网卡并插好网线的时候，将 Virtual switch 设成现实的网卡；当没有网卡或网线没插的时候，将 Virtual switch 设成 ms loopback 软网卡，即可实现网络共享。

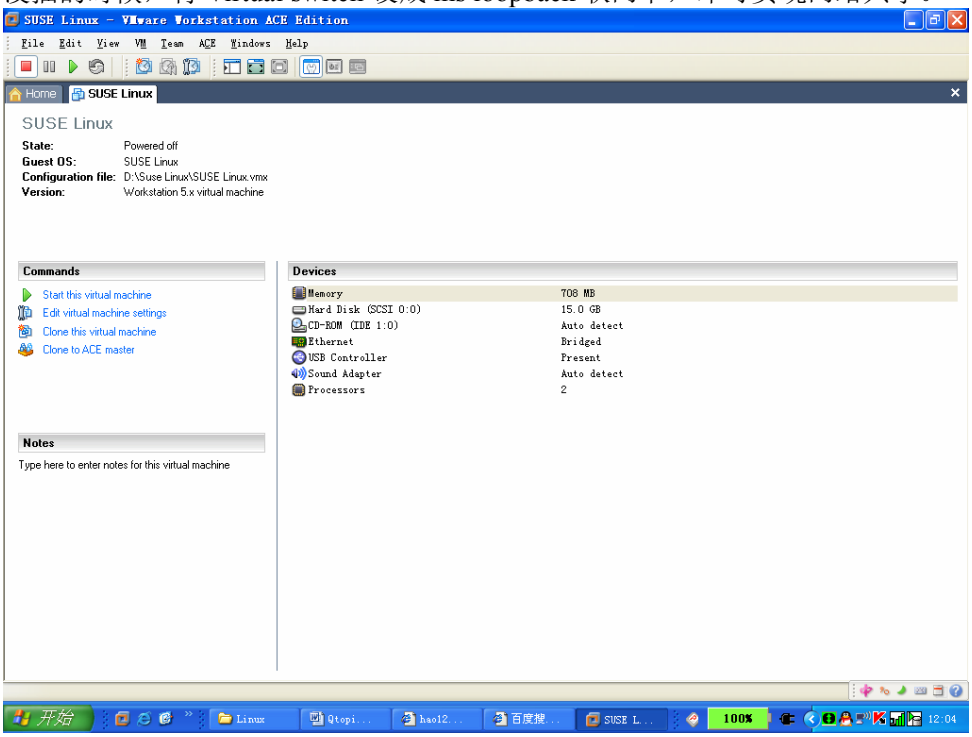


图2. VMware Workstation ACE Edition 6

下面以 SUSE Linux 10.1 为例，介绍在 VMware Workstation ACE Edition 6 下客户端操作系统的安装过程，其他的操作系统的安装方法类似：

- 从光驱处插入 SUSE Linux 10.1 的安装光盘(如果是映象文件，从 VMware Workstation 菜单中选择“VM”->“Settings”->“Handware”->“CD-ROM”->“use ISO Image”)。
- 创建一个新的虚拟机开始安装 SUSE Linux 10.1。
- 默认虚拟机配置为典型配置。
- 从客户端操作系统类中选择 Linux，在版本下拉列表中选择 SUSE Linux。
- 设置虚拟机名，设置虚拟机的安装位置。
- 默认网络连接为桥连接。
- 设置分配的硬盘大小，默认为 8G，在笔者的计算机上设为 15G。

这样一个虚拟机就建好了，然后系统会提示你虚拟机构建成功，接着按 VMware Workstation 提供的虚拟电源键，就可以一步步的安装 SUSE Linux 10.1 了，关于如何安装 SUSE Linux 10.1，由于 Linux 的安装方法已经大大简化，笔者在这里就不再详细介绍，需要说明的是，对研发人员而言，最好在安装 Linux 时，选择完全安装，这样就避免了以后在进行项目编译时因为相关工具没有安装而出现不必要的问题。

另外，为了客户端操作系统可以更好的运行，在 VMware Workstation 还可以配置客户端操作系统的运行内存。需要说明的是，配置操作需要在客户端操作系统运行前进行，下图即为配置客户端操作系统的运行内存的操作界面。

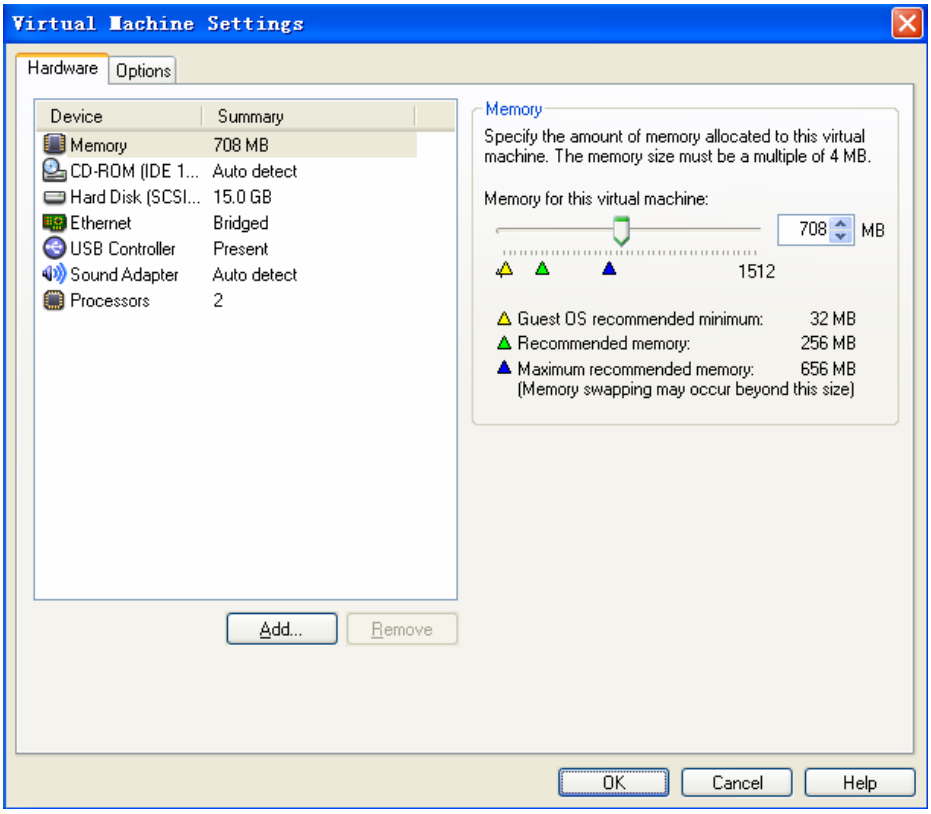


图3. VMware Workstation 设置内存

为了增强虚拟机的图形和鼠标性能，VMware Workstation 为需要的用户提供了另外的 VMware Tools。如果客户端操作系统是 windows，VMware Tools 会自动安装，如果是 linux，安装后，VMware Tools 的安装文件会被挂载到光驱中(是虚拟方式，此时光驱并没有光盘)，进入光驱的挂载点，把文件复制出来安装即可。如下为在 Linux 图形界面方式下挂载 windows 下目录的必要工具。

VMware Tools 的安装方法如下：

- 在客户端操作系统已经运行的情况下，从 VMware Workstation 菜单中选择“VM”->“install VMware Tools”。
- 在/media/VMware_Tools 下就会出现两个文件: VMwareTools-6.0.0-45731.i386.rpm 和 VMwareTools-6.0.0-45731.tar.gz。
- 解压 VMwareTools-6.0.0-45731.tar.gz 到特定目录，如/opt/vmware-tools-distrib。
- 在/opt/vmware-tools-distrib 下运行脚本 vmware-install.pl。按照提示信息操作即可安装好 VMware Tools。

如果希望挂载特定的 windows 下的目录，这时候可以开始执行图形界面操作，首先将当前目录转到/mnt 下。查看是否存在 hgfs 目录，以确信 VMware Tools 已经正确安装；然后从 VMware Workstation 菜单中选择“VM”->“Settings”->“Options”->“shared folders”；然后选择添加特定的挂载目录，按提示操作即可完成 windows 下目录的挂载，需要说明的是 windows 下目录在 Linux 下的挂载路径为/mnt/hgfs/winFolderName。

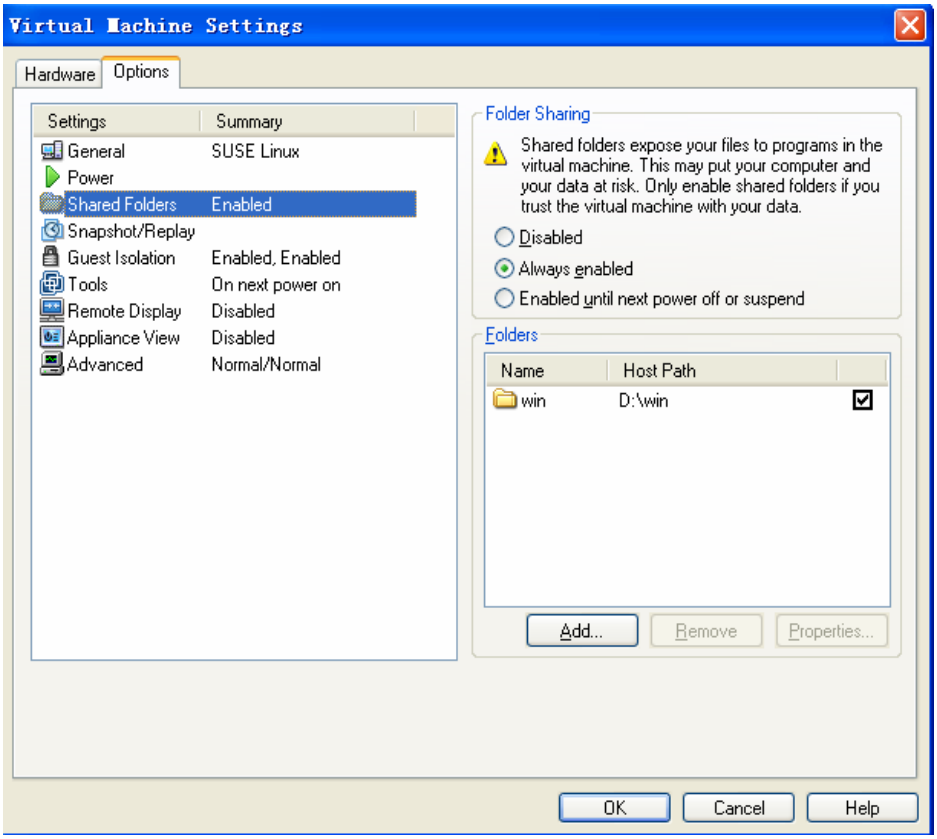


图4.　VMware Workstation 挂载 win 下的目录

### 1.5.2 双系统方式

由于通过虚拟机方式安装 Linux 存在运行速度慢，在嵌入式软件开发中存在无法利用串口烧写等因素，双系统方式则成为一个更加方便的选择。但双系统方式也存在着某些不足。

第一，因为安装双系统后，默认情况下，作为 bootloader 的 GRUB 会写到 linux 分区中，如果不幸因某种原因在 windows 系统下对 linux 分区进行了删除等误操作，将会导致 windows 和 linux 都无法启动的境地。此时就需要对 bootloader 进行恢复。此时的最简单的恢复办法是利用事先刻录的 dos 工具光盘进行 windows 的 bootloader 恢复。至于 linux 则只能重装了。

第二，如果由于某种原因，希望卸载 linux 系统，这时候的操作应为，进入 windows 系统，然后在命令行方式下键入 fdisk /mbr 来重写主引导记录，接着在成功重新启动 windows 系统后，磁盘管理器直接删除 linux 所在分区。

### 1.5.3 Samba服务

Samba 是 SMB 的一种实现方法，主要用来实现 Linux 系统的文件和打印服务。Linux 用户通过配置使用 Samba 服务器可以实现与 Windows 用户的资源共享。守护进程 smbd 和 nmbd 是 Samba 的核心，smbd 是建立对话，验证客户和提供文件系统和打印服务的基础；nmbd 则实现了网络浏览的功能，它的任务是向局域网广播 Samba 服务器所提供的服务，nmbd 使得 Samba 服务器显示在 windows 操作系统的网络邻居中，允许用户浏览可以通过 Samba 使用的资源。

启动 Samba 服务之后，我们在 windows 下网络邻居中通过搜索 IP 就可以找到 Samba 服务器,这时候该 Samba 已经完全可以提供局域网内网络共享服务了。在 root 权限下启动 Samba 的方法如下：

```
#cd /etc/init.d
#./smb start
```

如果 Samba 服务器没有直接显示出来，是因为我们还没有启动 nmbd 服务。在 root 权限下启动 nmbd 服务的方法如下：

#cd /etc/init.d

#./nmb start

在 SUSE Linux 下，提供 Samba 服务的文件位于/etc/samba 下，下面简单介绍下常用的两个文件 smbpasswd 和 smb.conf,其中 smbpasswd 为 Samba 服务的用户名和密码管理文件，smb.conf 为 Samba 服务的主要配置文件。

1. smbpasswd

linux 系统用户跟 samba 用户是相互联系而又相互独立的：首先 samba 用户必须是 linux 用户，是 linux 系统的组，而 member 是属于该组的 linux 系统用户,添加 samba 用户的前提是该组和成员在系统中已经存在；其次对于同一个用户来说，samba 密码和系统密码不必一定相同，换句话说 samba 的用户只是用来提供网络共享服务的，在这里与系统用户没有什么大的关系。

smbpasswd 的用法如下：

smbpasswd [选项] [用户名]

其中[用户名]为可选项，常用选项如下：

-a 添加用户

-x 删除用户

2. smb.conf

Samba 主要的共享方式有两种：share 和 user。可以通过 smb.conf 文件中的 security 选项配置，如：

在 smb.conf 中的[global]设置

security = share

guest ok = yes

guest account = miaozla

表示采用的是 share 共享方式，用户不需要输入帐号和密码就可以登陆 samba 服务器，可以指定用户登陆的默认帐号，这里用户就是通过默认的 miaozla 来登陆的。用户登陆之后，可以看到所有系统共享文件和 miaozla 所共享的文件。其他用户共享的文件能够看得到，但是不一定能够使用或者进的去。如果不设置 guest account 的话，则用户就以默认的 nobody 进行登陆，只能够使用系统设置的共享文件。其他用户设置的文件不一定能够使用。

Samba 有三种典型配置：pub（不需要密码，且可读写及删除文件）、read-only（不需要密码，但只可以读取文件）、user1（需要密码，可读写及删除文件），如果要将共享目录设为不需要密码可以读写的配置，以/opt 为共享目录为例时 smb.conf 的内容如下：

```
……
[opt]
inherit acls = yes
path = /opt/
read only = no                    //可以读写
comment = all users               //备注
guest ok = yes                    //是否需要进行用户鉴权
case sensitive = no
msdfs proxy = no
guest account = miaozla
hide files = eclipse/gnome/icecream/kde3/moneyplex/planmaker/        //隐藏文件
sesam/sourcenav/textmaker/tuxbase/vmware-tools-distrib/
```

关于 smb.conf 的更详细说明参考/usr/share/doc/packages/samba/examples/smb.conf.SUSE。

如果你改变了 smb.conf 这个配置文件，这个改变直到你使用/sbin/service smb restart 命令重启 Samba 守护进程后才会生效。

在 smb.confsmbpasswd 和 smb.conf 设置时，如果习惯利用图形界面工具，在 KDE 桌面环境下时，可以启动 KDE 控制中心，进入“Internet 和网络”->“Samba”进行相关设置即可。但令人遗憾的是，图形界面下的设置常常无法正常操作，往往需要通过直接修改文件的方式进行。需要说明的是，在设置 Samba 服务时，如果需要 windows 下用户对文件进行完全控制，

＃chmod　-R 777 SambaDirectory

下图为设置 Samba 服务的图形界面。

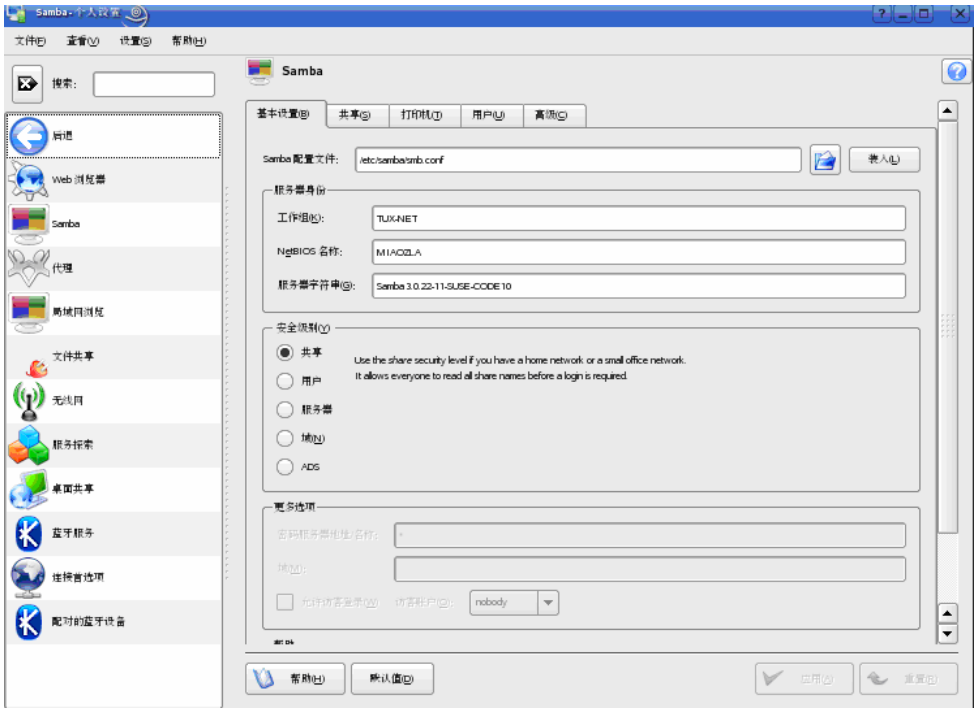


图5.　设置 Samba 服务

关于Samba服务的更详细信息，请参考参考文献[15]。

# 第 2 章　移动终端基础

**独学而无友，则孤陋而寡闻。**

《礼记·杂记》

## 2.1 PDA的发展历史

个人数码助理(PDA, Personal Digital Assistant)。PDA 最初是用于个人信息管理（PIM,Personal Information Management），替代纸笔，帮助人们进行一些日常管理，主要为日程安排、通讯录、任务安排、便笺。随着科技的发展，PDA 逐渐融合计算、通信、网络、存储、娱乐、电子商务等多功能，成为人们移动生活中不可缺少的工具。

其最大特点是 PDA 是一个开放的系统，就像电脑的操作系统一样，人们可以根据自己的需要，安装不同的软件，实现不同的功能。

一台 PDA 最基本的功能当然是日常个人信息管理（PIM），常用的 4 大天王：日历、联系人、任务、便笺。

- 日历：日程管理，例如你计划 9 月 10 日上午 9 点到 11 点召开部门经理会议。
- 联系人：也就是我们说的通讯录，里面有很详细的条目记录人员信息，有点像名片。
- 任务：字面意思很好理解，辞海解释为：需要去做的事情
- 便笺：就是可以随手记录的纸片

安装相应的同步软件就可以和 PC 上的 PIM 程序同步了，最常用的 PC PIM 程序就是微软 office 里面带的 outlook 了。所谓“同步”，就是有 2 个资料库，最初资料内容完全相同。如这 2 个资料库各自的资料内容经过不同修改，删除等系列处理，为了让这 2 个资料库资料内容仍保持一致，就必须执行一个让双方资料内容一致的操作，这个操作就是叫“同步”。而当这 2 个资料库进行同步后，资料内容一致后，称作“已同步”或者“同步状态”。

经常换手机的人可能最头痛的事是如何转移电话本，一般 SIM 卡容量只能保存 200 个号码。但如果你用的是具有 PDA 功能的手机，手机丢了都没有关系，因为包括联系人这些记录在手机和 PC 同步的时候已经保存到 PC 里面，只需要再同步一下到新的 PDA 手机上即可。这也是我多年来选择手机的最低条件：必须能和 outlook 同步。

不过需要提醒的是，对于 PDA 来说，虽然经常被叫做掌上电脑，但如果你以为它能实现电脑所有功能，那会非常失望。它仅仅是因为自身的便携，让你能够随时随地利用时间，说通俗一点，就是让你在坐车或者排队时能够消磨时光。所以在目前技术条件下，企图让 PDA 取代电脑是不现实的。PDA 应该作为电脑的辅助工具。对于相同软件 PC 版本和 PPC 版本来说，PC 版本功能应该强于 PPC，正确的使用方式是尽量在 PC 上完成相应工作，外出时再同步到 PDA 上，这样才能事倍功半。

## 2.2 手机的发展历史

第一代手机（1G）是指模拟的移动电话，也就是在 20 世纪八九十年代香港美国等影视作品中出现的大哥大。最先研制出大哥大的是美国摩托罗拉公司的 Cooper 博士。由于当时的电池容量限制和模拟调制技术需要硕大的天线和集成电路的发展状况等等制约，这种手机外表四四方方，只能成为可移动算不上便携。很多人称呼这种手机为“砖头”或是 黑金刚等。

这种手机有多种制式，如 NMT，AMPS，TACS，但是基本上使用频分复用方式只能进行语音通信，收讯效果不稳定，且保密性不足，无线带宽利用不充分。此中手机类似于简单的无线电双工电台，通话是锁定在一定频率，所以使用可调频电台就可以窃听通话。

第二代手机也是最常见的手机。通常这些手机使用 PHS，GSM 或者 CDMA 这些十分成熟的标准，具有稳定的通话质量和合适的待机时间。在第二代中为了适应数据通讯的需求，一

些中间标准也在手机上得到支持，例如支持 MMS 业务的 GPRS 和上网业务的 WAP 服务，以及各式各样的 Java 程序等。

第三代手机能够处理图像、音乐、视频流等多种媒体形式，提供包括网页浏览、电话会议、电子商务等多种信息服务。为了提供这种服务，无线网络必须能够支持不同的数据传输速度，也就是说，在室内、室外和行车的环境中能够分别支持至少 2 兆字节／每秒、384 千字节／每秒以及 144 千字节／每秒的传输速度。

未来的手机将偏重于安全和数据通讯。一方面加强个人隐私的保护，另一方面加强数据业务的研发，更多的多媒体功能被引入进来，手机将会具有更加强劲的运算能力，成为个人的信息终端，而不是仅仅具有通话和文字消息的功能。手机会更加智能化，微型化，安全化，多功能化。

未来的手机是个信息终端，以信息为中心！

## 2.3 主流操作系统

目前主流的手机操作系统主要有以下几种：Symbian、linux、Windows Mobile 和 palm，其中 Symbian 作为几家最大的手机巨头共同支持的操作系统，目前在市场上暂居着领导地位，而 linux 作为一个开源的操作系统，一直收到业界的喜爱，在进入嵌入式领域后，linux 操作系统发展十分迅猛。而作为 Microsoft 公司在移动通信领域的战略性产品，Windows Mobile 有着先天的优势，得到了多家手机厂商的一致看好。最好 palm，随着 PDA 的逐渐退出市场，因为其先天的设计不足，目前已经逐渐式微，现在简要介绍下以上几种主流的操作系统。

### 2.3.1 Symbian

Symbian OS 前身是 Psion 公司为移动设备推出的 EPOC（Electronic Piece of Cheese）操作系统。1998 年 6 月，诺基亚(NOKIA)、摩托罗拉(MOTOROLA)、爱立信(ERICSSON)和 Psion 在英国伦敦共同投资成立 Symbian 公司。这个公司继承了 Psion 公司 EPOC 操作系统软件的使用授权，承接 Psion 公司在移动设备的系统研发方面的所有经验。其目标是设定无线移动设备的标准操作系统，主要致力于无线移动设备的核心技术研发以及无线信息传输的应用程序，提供无线设备系统的研发技术及应用程式研发工具，并积极地建立无线核心技术上一套标准架构，使无线网路环境和其服务系统可以应用於企业或个人。由于手机巨头的参与，与当时的 Palm 和 WIndows CE 相比，Symbian 在在通信方面的优势得天独厚。

1999 年 3 月，EPOC Release 5.0 发布，也称 Symbian OS 5.0。为了能够适合不同产品需要，EPOC 采用内核和用户界面分离的方式，允许研发者自行定制和设计具有不同的风格和增强的特性的用户界面。在核心功能不变的情况下，能够根据不同硬件配置，采用不同的用户界面和接口，发展出不同的产品系列。

Symbian 预测从 2000 年起，2～3 年内，结合无线通讯、个人资讯和移动上网的设备（WID）将会迅速占领市场，而将产品规划成 3 大类别：Keypad Handheld Device（Communicator）（具备键盘的通讯器）、Smart Phone（普通智能手机）、及 Pen based PDA（手写输入设备）。并以此在 2000 年发布的 Symbian 6.0 核心技术上研发了 Crystal（Series80 原型）、Smart phone（Series60 原型）、Quartz（Symibna OS 7.0 起开始改称为 UIQ）3 种图形界面。

Symbian OS 6.0 与 5.0 版本相比，增加了对 GPRS、WAP 1.2、蓝牙技术的支持，配备安全性功能（SSL，HTTPS，WTLS），采用了 16bit Unicode，PersonalJava 3.0 和 JavaPhone 1.0

2002 年 2 月，Symbian OS 7.0 公布，增加了对 J2ME MIDP v2.0 以及多种通信网络的支持，改进了一些一些安全及认证功能。由于 Symbian OS 最初出发点是作为手机的操作系统，以完成通话的基本功能为主，设计之初并未考虑加入更多类似 PC 的功能。与同期微软的 Pocket PC Phone 2002 和 Smartphone 2002 相比，与 PDA、PC、互联网互通和扩展性方面还是有所欠缺的。

2004 年 2 月，Symbian OS 8.0 问世，其中 8.0a 采用 EKA1 核心，主要为了保持对旧程序的兼容性；8.0b 采用 EKA2 核心，提供更强的即时处理功能。后续 8.1 也有 a、b2 个版本针对

2005 年初，Symbian OS 9.1 发布，引入了新的系统安全模型，使用新型 ARM 处理器，而且使用 EKA2 内核，相对之前采用 EKA1 内核的机型有着众多的优越性，EKA1 不支持可写数据段，而 EKA2 全面改进了 EKA1 的任务调度算法，能够更好利用最新的 CPU 特性，拥有性能更强的编译器，完全支持实时性。实时性是支持某些高带宽、高优先级的任务对系统的基本要求，如 VoIP 网络电话、高速率的视频在线点播等，都是今后手机发展的方向。但正是因为这些改变，导致采用 Symbian OS 9.1 的机器均无法使用以前版本的应用程序，同样以前 OS 的机器也没法使用基于 Symbian OS 9.1 研发的程序，需要软件厂商重新编译。

2006 年 7 月，Symbian OS 9.3 发布。该版本减少了开机时间以及应用程序的启动时间；改善了软件研发环境,提供了更好的软件工具研发包；提升了对本地 WIFI 网络技术支持；支持 USB 2.0 On-The-Go 技术；提供了 Firmware Over-The-Air（能够通过无线网络更新自己的软件）功能；HSDPA 支持；支持 Java JSR 248。

目前市场上存在 4 种不同用户界面及接口的 Symbian OS 产品，分别是 Series 60、Series 80、Series 90 和 UIQ。

### 2.3.2 Linux

相对于前面几种 PDA 的兴旺，采用 Linux 操作系统的 PDA 可谓人丁稀少。早期有 Sharp 发布过几款带键盘的掌上电脑，现在主要是摩托罗拉力推的智能手机系列。

Linux 系统与之前操作系统相比，具有开放式架构、授权费用低、研发资源丰富等优点。制造商根据实际情况，有针对性地研发适合自己设备的 Linux 操作系统。这样既能让自己的产品有特色，又能避免受制于人，还能够最大限度地满足用户多方面的应用。

摩托罗拉从 2003 年售出 Symbian 股份以来，专心致力于 Linux 操作系统的手机，并于当年推出第一款 A760，采用 MontaVista 的 Linux 系统、32MB 记忆体，与 intel 的 200MHz PXA262 处理器。随后又推出 A768、A728、A780、E680 多款产品。

特别是今年发布的 A1200，更是凭借优雅的外形，大受市场好评。

除了摩托罗拉，三星、松下也推出过 Linux 系统的手机。Palm 新一代操作系统据说也要采用 Linux 平台，Nokia 也莫名其妙的于 2005 年发布了 采用 Linux 系统，支持 WiFI 连接的掌上电脑 770。这一切都表明众多厂家对其市场的试探和期待。

但与前面几种操作系统具有统一的标准不同，每一个制造商都还在使用自己在 Linux 内核基础上研发出来的独有的操作系统。这种情况直接导致了在市场上出现了各种各样的系统应用和软件，互相不能通用，资源不能共享。虽然 2006 年 6 月摩托罗拉、NEC、NTT Docomo、松下、三星和沃达丰等六家公司宣布将合作研发手机用的开放性 Linux 软件平台，但该计划目前仅有一个简单的架构规划而已，谁知道未来又会怎样呢？

### 2.3.3 Windows Mobile

按照 Microsoft 官方的说法：Windows Mobile 将熟悉的 Windows 体验扩展到了移动环境中，所以您可以立即使用它投入工作。目前包括 3 类产品：Pocket PC、Smartphone 和 Media Centers。

1997 年，第一代微软移动设备操作系统名称为 Windows CE 1.0 发布；1998 年，升级版本为 Windows CE 2.0（设备称为 Palm-Size PC）；2000 年，新的操作系统改名为 Pocket PC 2000（也简称为 Pocket PC 或者 Windows CE 3.0；2001 年底，Pocket PC 2002 问世；2002 年，推出增加手机功能的 Pocket PC Phone 2002；同年，主要针对 Symiban 系统的 Smartphone 2002 诞生。2003 年，微软将 Pocket PC 2003 和 Smart Phone 2003 统一改称为 Windows Mobile 2003，依然包括 Windows Mobile 2003 for Pocket PC、Windows Mobile 2003 for Pocket PC Phone Edition 和 Windows Mobile 2003 for Smartphone。2005 年，微软没有延续年号的命名方法，采用操作系统所采用的 Windows CE 内核版本命名，将新的操作系统称为 Windows

Mobile 5.0。依然包括 Windows Mobile 5.0 for Pocket PC、Windows Mobile 5.0 for Pocket PC Phone 和 Windows Mobile 5.0 for Smartphone。

2005 年 5 月 11 日，在微软公司一年一度的移动嵌入式研发者大会上，用来替代 Windows Mobile 2003 SE 新版本 Windows Mobile 5.0（包含 Pocket PC 和 Smartphone 2 个版本）正式发布。

与之前版本相比，Windows Mobile 5.0 最重大的改进在于将数据储存由原来的 RAM 改在 ROM 中，这样即使断电也不会丢失数据，提高了设备稳定性。

Pocket PC 屏幕底端增设的 2 个功能软键，给操作带来便利，也与手机操作方式相近。

新版 MSN 集成 MSN Messenger 和 MSN Hotmail，可以直接收取 Hotmail 邮件。Outlook 支持接收附件，而且可以在联系人中的个人信息里添加个性化图片和声音。

新版 Office 软件中，用户可以用 Excel Mobile 查看、创建图表，用 Word Mobile 编辑带有图形的文件，同时保证与其在 PC 上创建的文件格式一致。首次增加 PowerPoint Mobile，终于可以不借助第三方软件，直接观看 PowerPoint 幻灯片了。

其它改进还有：

- 网络支持

支持更高带宽的 3G 网络，支持 Wi-Fi 的 Smartphone 平台以及对现有支持 BlueTooth 技术的改进等将使合作伙伴在通过多种网络进行整合移动服务时拥有更大的灵活性。

- 安全性

为了让 Windows Mobile 平台中已涵盖的诸如 BlueTooth 授权、虚拟专用网络上端到端编密码等安全功能更加完善，Windows Mobile 5.0 经过了大量的建模测试，通过了微软可信赖计算严格的全面安全审核，且已经达到 FIPS-140-2 标准——美国 ZF 对于 IT 产品安全要求标准。

- Windows Media Player 10 Mobile

Windows Media Player 10 Mobile 让用户能够享受到大量的受保护的数字音乐、视频和录制的电视文件；这些文件可以便捷地与 PC 同步，或是通过互联网以及运营商的音乐中心进行下载。Windows Media Player 10 Mobile 同时还能提供与用户播放列表、唱片等的同步。另外，合作伙伴还可以加入数字版权管理技术来帮助其完善特定的媒体商业模式。

- 图片和视频

图片与视频功将带有先进的脉冲模式和计时器功能，而这种功能目前仅用于高端数码相机。

- 扩展存储

对硬盘和 USB2.0 的支持将帮助人们在移动设备上方便而快速的存储、与 PC 同步大量的信息，如完整的数码照片，媒体库等。

其中重要的改进还有：支持 USB 2.0 及内置硬盘；支持设备制造商选项以把微软的 Voice 命令工具集成到他们的设备中（Pocket PC 以及 Smartphone)，这就允许用户通过在联系人应用程序中说出一个人的名字来实现拨号功能；新的 GPS 管理器，它将简化到 GPS 接收器的连接。

与以前的版本相比，Windows Mobile 5.0 强调了稳定和网络通讯，更适合智能手机（Pocket PC Phone 和 Smartphone）使用。各大厂商纷纷推出的 Windows Mobile 5.0 产品也以智能手机为主。

这个新版本相对以前做了很大的改变，重新设计了 Kernel，支持的进程数从 32 个扩展到 3.2 万，每个进程的地址空间从 32MB 扩展到 2GB。很多系统模块(如文件系统和设备管理器）将运行在 kernel 模式，OAL 也从 kernel 独立出来，driver 可以运行在 kernel 模式和 user 模式。这种架构将有助于研发更复杂的设备，并保证设备的稳定性。以前基于 5.0 的 BSP，必须要做一定的修改才能移植到 6.0 上。代号为“Photon”的 Windows Mobile 6.0 也将基于 CE 6.0 来研发，已于 2007 年初推出。主要增加了 Office 方面的处理功能。

### 2.3.4 Palm

1992 年，杰夫•霍金斯（Jeff Hawkins）和唐娜•杜宾斯基（Donna Dubinsky）创立 Palm 公司。看到“牛顿”推出后潜在的市场需求，杰夫•霍金斯将兴趣放在 PDA 产品上。经过市场调

1996 年，第一个真正意义上的具有 Palm 操作系统（Palm OS1.0 ）的 PalmPilot 1000 诞生。PalmPilot 1000 功能虽然十分简陋，但是它已经具备了今天 Palm 的很多特征，从外型、按钮布局到手写识别系统。内建的日历、地址簿、日程表、记事本，直到今天一直都是所有 PDA 或者 PIM 程序必备的。截止到 2000 年初，PalmPilot 总共售出了 600 万台，大大超出原先的估计，一个新的时代诞生了。

1997 年 2 月，Palm OS 2.0 发布。与 1.0 相比，主要改进在增加了 TCP/IP 支持，能使 Palm Pilot 在 TCP/IP 网络上通信。

1998 年 4 月，Palm OS 3.0 随着工业史上的又一个里程碑式的产品——Palm III 一起发布。Palm OS 3.0 第一次支持红外线，使得与其它设备交换数据成为可能。而且底层的稳定性和易用性方面有了很大的改善，各种应用软件开始大量出现，Palm 迈入黄金时期。

1999 年年初，Palm 最经典的产品 Palm V 发布。出色的外型设计，颇具质感的铝合金外壳，内置可方便充电的锂电池，超轻超薄的机身，都让每一个看到的人爱不释手。

1999 年 7 月，由于与管理层意见不合，杰夫•霍金斯（Jeff Hawkins）和唐娜•杜宾斯基（Donna Dubinsky）离开原来公司，并成立了一家新的企业 Handspring，继续生产 Palm OS，但比 Palm 便宜的 PDA。1999 年 9 月，发布了第一款产品－－Visor Delux，5 种颜色炫丽的外型，搭配略为透明的外壳，如同苹果电脑那般，深受年轻时尚人群喜爱。Visor 系列采用了 Springboard 扩展技术，扩大了各方面应用。

2000 年，Sony 获得 Palm 授权，开始生产 Palm OS 设备，命名为“Clie”，第一款产品为 PEG-S300，凭着在家用娱乐领域的经验，Sony 频繁推出设计各异的新品。截止到 04 年 9 月最后一款 PEG-VZ90，一共推出 32 款产品，极大扩充了 Palm 产品线。Clie 相对于 Palm 单一商务功能来说，增强了多媒体应用，例如内置音频解码器，摄像头，高分辨率屏幕等。而时尚外形、记忆棒插槽、Jog 滚轮也具有强烈的 Sony 风格。

2000 年，Palm OS 设备占据掌上市场 90％以上的份额，这时在人们心中，PDA 等于 Palm。不过，2000 年，微软也发布了其 PDA 历史上具有划时代意义的 OS：Windows CE 3.0，也就是我们熟悉的 Pocket PC。凭借这个系统，微软开始慢慢蚕食 Palm 原来的市场份额了。

2000 年 2 月，配备 Palm OS 3.5 的第一款彩色产品 Palm IIIc 问世，Palm OS 3.5 的最大特点就是支持彩色。

2002 年 10 月，Palm OS 5.0 舍弃原来的 Motorola DragonBall CPU，开始采用更高频率芯片，并支持 320x320 和 320x480 高分辨率和 BlueTooth 以及 WiFi 连接功能，但依然没有实现多任务功能。产品主要为高端的 Tungsten 系列和低端的 Zire 系列。

2003 年 6 月，Palm 收购 Handspring，并继承推出大受好评的智能手机产品 Treo 600 以及 Treo650。

2004 年 6 月，Sony 由于日益萎缩的传统掌上电脑市场宣布放弃除日本以外的海外市场，并在同年 9 月发布最后一款 PEG-VZ90 后，将重心转移到 Symbian UIQ 产品上。这也表明了智能手机在逐步威胁传统掌上电脑的地位。

2003 年 10 月，负责 Palm OS 的 PalmSource 有限公司正式宣布从 Palm 股份有限公司分拆出来将成为一家独立上市、主营针对智能移动设备操作系统的软件公司。

由于下一代 Palm OS 迟迟不能推出（从 03 年就持续跳票），Palm 新品推出缓慢，基本处于停滞阶段。从 04 年 10 月发布 Treo650 以后，再也没有一个重量级的产品出现，这就给了其它竞争对手追赶的机会。2004 年下半年，微软的 Windows Mobile 设备已经超过 Palm 出货量。

2005 年 9 月，Palm 公司宣布其新款智能手机 Treo 700 W 将采用来自微软的操作系统。同月，PalmSource 公司被日本公司 Acess 以 3.24 亿美元收购，标志着一个时代的终结。

## 2.4 研发语言

在嵌入式领域，由于其先天的对实时性等方面的要求，C 及其衍生版本一直是嵌入式软件开发

### 2.4.1 J2ME

J2ME 是 Sun 公司针对嵌入式、消费类电子产品推出的研发平台，与 J2SE 和 J2EE 共同组成 Java 技术的三个重要的分支。J2ME 实际上是一系列规范的集合，由 JCP 组织制定相关的 Java Specification Request（JSR）并发布，各个厂商会按照规范在自己的产品上进行实现，但是必须要通过 TCK 测试，这样确保兼容性。

J2ME 平台是由配置（Configuration）和简表（Profile）构成的。配置是提供给最大范围设备使用的最小类库集合，在配置中同时包含 Java 虚拟机。简表是针对一系列设备提供的研发包集合。在 J2ME 中还有一个重要的概念是可选包（Optional Package），它是针对特定设备提供的类库，比如某些设备是支持 BlueTooth 的，针对此功能 J2ME 中制定了 JSR82（Bluetooth API）提供了对 BlueTooth 的支持。

J2ME 中有两个最主要的配置，分别是 Connected Limited Devices Configuration（CLDC）和 Connected Devices Configuration（CDC）。其中 CLDC（Connected Limited Device Configuration）1.0(JSR 30)的标准化自 1999 年 10 月开始，旨在定义一个“最小公分母”式的 Java 平台, CLDC 1.1(JSR 139)的标准化工作自 2001 年 9 月开始，加入了一系列特性以提升 J2ME 平台和完整的 J2SE 平台的兼容性，最显著的特性是加入了对浮点数的支持。

而 MIDP（Mobile Information Device Profile）1.0（JSR 137）的标准化自 1999 年 9 月开始，旨在基于 CLDC 的成果特别针对无线通信设备添加特性和 API。MIDP 2.0（JSR 118）的标准化工作从 2001 年 8 月开始，加入了一系列重要的新特性。但市面上的 MIDP 平台仍然处于混乱状态。研发者必须在执行时期侦测各种专属 API 和 Optional Package 的存在，了解决上述问题，进一步提高 MIDP 应用程序的可移植性，Sun Microsystems 以 MIDP 2.0 规格为核心，于 2003 年 7 月发布了 JTWI（Java Technology for the Wireless Industry）规范。有效确保 MIDP 软件的可移植性。该规范是无线 Java 的框架集成，包括了 CLDC1.0/1.1、MIDP2.0、WMA（Wireless Messaging API）1.1（JSR 120）、以及 MMA（Mobile Media APIs）1.1（JSR 135）等具体规范，从而使得支持 JTWI 规范标准的 Java 手机，可以同时支持 Java 游戏、SMS、MMS、WAP、Email、多媒体、电子商务等多种功能。

但 2007 年 10 月，Sun 副总裁，Java 之父 James Gosling 说“移动版全部工作已经越来越向 Java 标准版靠拢”。移动版的 Java 最终将被标准版 Java 取代。

### 2.4.2 Brew

BREW的全称是无线二进制运行环境（Binary Runtime Environment for Wireless），是一套完整的端到端的解决方案，由美国Qualcomm公司于 2001 年 1 月推出。目前的最新版本为 BREW 3.1。于 2006 年 11 月 9 日发布。

BREW 3.0 客户端软件能大大扩展移动电话的多媒体功能，通过其新增的对移动存储介质与串行接口的支持特性，使 BREW 手机能够和外接键盘或者个人电脑等设备轻松连接。除多媒体增强特性外，BREW 3.0 客户端软件还提供了强大的组群管理特性，使运营商能够灵活的对其消费者市场进行细分（包含普通商业用户和大企业用户），以提供量身定制的无线应用组合。另外，BREW 3.0 还使研发商可充分利用双屏手机的外部显示屏。

此外，BREW 提供了一个高效率、低成本、可扩展的应用程序执行环境（AEE，Application Exection Environment），专为研发可无缝植入任何手持设备的应用程序而定制。制造商和研发人员可以随时对运行环境进行扩展，提供应用程序需要的各种附加性能模块。同时作为软件接口层，BREW 位于无线设备的芯片系统软件和应用程序之间，使最终用户可以无线下载程序到支持 BREW 的设备上运行。BREW 具有较高的闪存和 RAM 利用效率。

BREW 平台是一个开放的并能为应用程序研发商、设备制造商、网络运营商提供具体利益的端到端的解决方案，利用 BREW 可以有效减少技术和市场间的间隔，将产品快速推向市

安装BREW SDK对操作系统的最低要求为Microsoft Windows NT 4.0 SP3、Windows 2000 SP2或Windows XP，要使用BREW SDK研发应用程序，还需要安装Microsoft Visual C++ 6.0或者Microsoft VC++.NET。

### 2.4.3 Qtopia

Qtopia最初是sourceforge.net上的一个开源项目，全称是Qt Palmtop Environment，是构建于Qte之上一个类似桌面系统的应用环境，包括了PDA和手机等掌上系统常见的功能如电话簿、日程表等。现在Qtopia 已经成为了Trolltech的又一个主打产品，为基于Linux操作系统的PDA和手机提供了一个完整的图形环境。

版本4之前，Qte和Qtopia是不同的两套程序，Qte是基础类库，Qtopia是构建于Qte之上的一系列应用程序。但从版本 4 开始，Trolltech 将 Qte 并入了 Qtopia，并推出了新的Qtopia4。在该版中，原来的 Qte 被称为 Qtopia Core，作为嵌入式版本的核心，既可以与Qtopia配合，也可以独立使用。原来的Qtopia则被分成几层，核心的应用框架和插件系统被称为 Qtopia Platform，上层的应用程序则按照不同的目标用户分为不同的包，如 Qtopai PDA，Qtopia Phone。

# 第 3 章　Qtopia基础

**骐骥一跃，不能十步；驽马十驾，功在不舍。**

《荀子·劝学》

Qt 是由挪威 TrollTech 公司开放的跨平台 C++图形用户界面研发工具,也是该公司的一个标志性产品,分为商业版和免费版两种,软件工程师可以利用 Qt 编写应用程序,并可在 Windows、Linux、Unix、以及 Mac OS X 和嵌入式 Linux 等不同平台上进行本地化运行。目前，Qt 已被成功地应用倒全球数以千计的商业应用程序中，此外，Qt 还是开源 KDE 桌面环境的基础，TrollTech 公司在 1995 年推出的 Qt 的第一个商业版本，直到现在 Qt 已经被世界各地跨平台的研发人员所使用，而 Qt 的功能也得到了不断的完善和提高。Qt 以工具开发包的形式提供给研发者，这些开放包包括了图形设计器 Qt Makefile 制作工具 qmake、国际化工具和 Qt 的 C++类库等，

Qt 的一个显著特点是跨平台特性。它能够在多个平台（Unix、Linux、Windows）上运行，并对不同平台的私有 API 进行了封装，如文字处理、网络协议、进程处理、线程、数据库访问等。从某种意义上讲，Qt 雷同于 MicroSoft 的 MFC 和 Borland 的 VCL，都是 C++的一个研发库。不同的是它封装了不同操作系统的访问细节，能够实现跨平台的应用；其次，由于 Qt 是基于 C++构造的，因此 Qt 具有面向对象编程的所有优点，这有利于研发人员快速的切换到 Qt 平台上进行研发，有效的降低了学习成本；第三，Qt 的易用而且运行速度非常快，这主要得益于它的实现方式，Qt 是一个 GUI 仿真工具包，它是由不同平台的底层绘图函数仿真不同平台的风格，较好的提升浪运行速度。

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负责的 GUI 系统，Qtopia 是著名的 Qt 库研发商 Trolltech 公司研发的、面向嵌入式系统的 Qt 版本，是一个专门为小型设备提供图形用户界面的应用框架和窗口系统，也是完整的自包含的 C＋＋ GUI 和基于 Linux 的嵌入式平台研发工具，它为研发者提供了丰富的窗口部件，能够支持窗口部件的定制，可以为用户提供非常丰富的图形界面。被看作是 Qt 的嵌入式版本。

Qtopia 与 Qt 一样，同样采用的是 Server/Client 结构。具有丰富的控件资源和较好的可移植性。需要说明的是 Qtopia 应用程序目前只能在 Linux 平台上研发和运行。

Trolltech 公司基于 GNU 公共发布许可第 2 版（GPL,GNU General Public License,version 2.0）发布 Qtopia。截至到笔者撰写本书时，Qtopia 的最高版本为 4.2.4。Qtopia 4 包括三个系列：Qtopia Core、Qtopia platform、Qtopia phone Edition，其中 Qtopia Core 提供了整个 Qtopia 产品家族的基础；Qtopia platform 构建于 Qtopia Core 之上能够为 Linux 设备提供丰富的用户体验；Qtopia Phone Edition 为基于 Linux 的移动终端提供了丰富的应用和用户接口，它提供了预集成的应用模块，并允许制造商和研发商在保持对分支和用户体验完全控制的同时提供富有特色的移动终端产品。截至到笔者撰写本书时，Qtopia Phone Edition 的最新版本为 4.2.4。除以上三个系列外，Qtopia PDA Edition 也曾是 Qtopia 的早期版本的一个系列，但 Trolltech 公司已经停止了对 Qtopia PDA Edition 的升级，目前 Qtopia PDA Edition 的最高版本为 2.2.0。

在本书中采用的是 Qtopia Open Source Edition 4.2.4，相对商业版的 Qtopia Phone Edition，Qtopia Open Source Edition 包含了除安全执行环境（SXE,Safe eXecution Environment）以外的所有 Qtopia Phone Edition 的特性。需要说明的是 Qtopia Open Source Edition 4.2.4 只能在基于 kernel 2.6 及以上内核的 Linux 平台上安装，笔者采用的 Linux 平台为 SUSE Linux 10.1。

需要约定的是，在本书中对于 Qt 和 Qtopia 共有的特性，笔者统一以 Qt 的特性说明，只在 Qtopia 中才具有的特性，笔者以 Qtopia 说明。

## 3.1 Qtopia的安装

首先登陆 trolltech 的官方网站 http://trolltech.com，找到 Qtopia Open Source Edition 的下载地址：http://trolltech.com/developer/downloads/qtopia/qtopia-source-gpl2.2，下载 qtopia-opensource-src-4.2.4.tar.gz 到 Linux 特定目录 QTOPIADIR。在笔者的机器上，QTOPIADIR 为/opt/qtopia-opensource-4.2.4。

### 3.1.1 安装Qtopia

然后打开终端 konsole，将当前目录切换到/opt 下。执行以下命令：

tar zxvf qtopia-opensource-src-4.2.4.tar.gz

得到解压后目录 qtopia-opensource-4.2.4。然后将当前目录切换到 QTOPIADIR，执行以下命令：

＃./configure

＃./make

＃./make install

即可完成对 Qtopia 的安装。

需要说明的是 konsole 是到目前为止 Linux 上所提供的最优秀的终端之一，在启动多个会话时，可以通过组合键“Shift+方向键”来切换当前会话。

### 3.1.2 环境变量

Qtopia 在安装完成后，会将生成的映象文件放在一个特定的目录，默认情况下该目录为 QTOPIADIR /scripts，在启动 Qtopia 模拟器前，除了 QTOPIADIR 外，还必须正确设置如下的几个环境变量：

- PATH
- LD_LIBRARY_PATH
- QMAKESPEC

方法如下：

＃export PATH= QTOPIADIR/bin:$PATH

＃export LD_LIBRARY_PATH= QTOPIADIR/lib:$ LD_LIBRARY_PATH

＃export QMAKESPEC= QTOPIADIR/qtopiacore/qt/mkspecs/qws/linux-g++

通过上面的方式设置的环境变量只是暂时的，和 windows 下的设置意义不同，为了能够在环境启动后设置的环境变量能够自动生效，有两种方法：

- 在 etc/profile 中添加以上设置，这样 Linux 在启动时就会自动导出设置的内容。
- 创建脚本文件，将有关设置添加到脚本文件中，然后将该脚本添加到某 Linux 启动时能够调用的文件中，或者在 Linux 启动后，在 shell 中运行该脚本。

需要说明的是，在 Qtopia 中 QMAKESPEC 是交叉编译的一个很重要的环境变量，为了根据不同的硬件平台来产生不同的 Makefile 文件进行编译，这就要求 Qtopia 必须能够获得处理器和编译器的相关信息，在实际的设计中，Qtopia 利用 qmake.conf 来描述处理器和编译器的相关信息，QMAKESPEC 则承担了指明配置文件 qmake.conf 路径的功能。

下面是当 QMAKESPEC 配置为 QTOPIADIR/qtopiacore/qt/mkspecs/qws/linux-g++时的 qmake.conf 文件内容：

```
MAKEFILE_GENERATOR          = UNIX
TEMPLATE                    = app
CONFIG                      += qt warn_on release incremental link_prl
QT                          += core gui
QMAKE_INCREMENTAL_STYLE     = sublib
QMAKE_CFLAGS                = -m32
QMAKE_LFLAGS                = -m32
```

include(../common/g++.conf)
include(../common/linux.conf)
load(qt_config)

g++.conf 主要描述了编译器的配置信息，下面是 g++.conf 的内容：

```
QMAKE_CC        = gcc
QMAKE_LEX       = flex
QMAKE_LEXFLAGS      +=
QMAKE_YACC      = yacc
QMAKE_YACCFLAGS         += -d
QMAKE_YACCFLAGS_MANGLE  += -p $base -b $base
QMAKE_YACC_HEADER       = $base.tab.h
QMAKE_YACC_SOURCE       = $base.tab.c
QMAKE_CFLAGS        += -pipe
QMAKE_CFLAGS_DEPS   += -M
QMAKE_CFLAGS_WARN_ON  += -Wall -W
QMAKE_CFLAGS_WARN_OFF += -w
QMAKE_CFLAGS_RELEASE  += -O2
QMAKE_CFLAGS_DEBUG+= -g
QMAKE_CFLAGS_SHLIB  += -fPIC
QMAKE_CFLAGS_STATIC_LIB+= -fPIC
QMAKE_CFLAGS_YACC  += -Wno-unused -Wno-parentheses
QMAKE_CFLAGS_HIDESYMS    += -fvisibility=hidden
QMAKE_CXX       = g++
QMAKE_CXXFLAGS      += $$QMAKE_CFLAGS
QMAKE_CXXFLAGS_DEPS     += $$QMAKE_CFLAGS_DEPS
QMAKE_CXXFLAGS_WARN_ON    += $$QMAKE_CFLAGS_WARN_ON
QMAKE_CXXFLAGS_WARN_OFF   += $$QMAKE_CFLAGS_WARN_OFF
QMAKE_CXXFLAGS_RELEASE    += $$QMAKE_CFLAGS_RELEASE
QMAKE_CXXFLAGS_DEBUG  += $$QMAKE_CFLAGS_DEBUG
QMAKE_CXXFLAGS_SHLIB    += $$QMAKE_CFLAGS_SHLIB
QMAKE_CXXFLAGS_STATIC_LIB += $$QMAKE_CFLAGS_STATIC_LIB
QMAKE_CXXFLAGS_YACC    += $$QMAKE_CFLAGS_YACC
QMAKE_CXXFLAGS_HIDESYMS         += $$QMAKE_CFLAGS_HIDESYMS
-fvisibility-inlines-hidden
QMAKE_LINK      = g++
QMAKE_LINK_SHLIB= g++
QMAKE_LFLAGS      +=
QMAKE_LFLAGS_RELEASE    +=
QMAKE_LFLAGS_DEBUG+=
QMAKE_LFLAGS_SHLIB  += -shared
QMAKE_LFLAGS_PLUGIN     += $$QMAKE_LFLAGS_SHLIB
QMAKE_LFLAGS_SONAME     += -Wl,-soname,
QMAKE_LFLAGS_THREAD     +=
QMAKE_RPATH     = -Wl,-rpath,
include(unix.conf)
```

unix.conf 主要描述了调试配置信息，下面是 unix.conf 的内容：

```
QMAKE_SEPARATE_DEBUG_INFO = (test -z \"$(DESTDIR)\" || cd \"$(DESTDIR)\" ;
targ=`basename $(TARGET)`; objcopy --only-keep-debug \"\$\$targ\" \"\$\$targ.debug\" &&
objcopy --strip-debug \"\$\$targ\" && objcopy --add-gnu-debuglink=\"\$\$targ.debug\"
\"\$\$targ\" && chmod -x \"\$\$targ.debug\" ) ;

QMAKE_INSTALL_SEPARATE_DEBUG_INFO = test -z "$(DESTDIR)" || cd
\"$(DESTDIR)\" ; $(INSTALL_FILE) `basename $(TARGET)`.debug
$(INSTALL_ROOT)/\$\$target_path/
```

linux.conf 描述了编译过程中用到的头文件路径、库路径、编译工具配置等信息，下面是 linux.conf 的内容：

```
QMAKE_CFLAGS_THREAD     += -D_REENTRANT
```

```
QMAKE_CXXFLAGS_THREAD       += $$QMAKE_CFLAGS_THREAD
QMAKE_INCDIR                =
QMAKE_LIBDIR                =
QMAKE_INCDIR_X11            = /usr/X11R6/include
QMAKE_LIBDIR_X11            = /usr/X11R6/lib
QMAKE_INCDIR_QT             = $$[QT_INSTALL_HEADERS]
QMAKE_LIBDIR_QT             = $$[QT_INSTALL_LIBS]
QMAKE_INCDIR_OPENGL         = /usr/X11R6/include
QMAKE_LIBDIR_OPENGL         = /usr/X11R6/lib
QMAKE_LIBS                  =
QMAKE_LIBS_DYNLOAD          = -ldl
QMAKE_LIBS_X11              = -lXext -lX11 -lm
QMAKE_LIBS_X11SM            = -lSM -lICE
QMAKE_LIBS_NIS              = -lnsl
QMAKE_LIBS_OPENGL           = -lGLU -lGL
QMAKE_LIBS_OPENGL_QT        = -lGL
QMAKE_LIBS_THREAD           = -lpthread
QMAKE_MOC                   = $$[QT_INSTALL_BINS]/moc
QMAKE_UIC                   = $$[QT_INSTALL_BINS]/uic
QMAKE_AR                    = ar cqs
QMAKE_RANLIB                =
QMAKE_TAR                   = tar -cf
QMAKE_GZIP                  = gzip -9f
QMAKE_COPY                  = cp -f
QMAKE_COPY_FILE             = $(COPY)
QMAKE_COPY_DIR              = $(COPY) -r
QMAKE_MOVE                  = mv -f
QMAKE_DEL_FILE              = rm -f
QMAKE_DEL_DIR               = rmdir
QMAKE_STRIP                 = strip
QMAKE_STRIPFLAGS_LIB += --strip-unneeded
QMAKE_CHK_DIR_EXISTS        = test -d
QMAKE_MKDIR                 = mkdir -p
QMAKE_INSTALL_FILE          = install -m 644 -p
QMAKE_INSTALL_PROGRAM = install -m 755 -p
```

### 3.1.3 启动模拟器

在安装完成后，为了能够在 PC 上启动模拟器，必须进行以下几个方面的工作：

- 设置环境变量
- 为模拟器设置适当的皮肤
- 启动 QVFB
- 启动移动终端仿真器
- 启动 Qtopia

在 Qtopia 42.4 版本中，Qtopia 通过脚本 runqtopia 来完成了必要的启动工作，如果需要启动模拟器，可以将当前目录切换到 QTOPIADIR/scripts，然后执行以下命令：

```
# ./runqtopia
```

即可启动 Qtopia Phone 的模拟器了。

## 3.2 研发环境

由于历史的因素，在Linux早期平台上，所能使用的研发环境大多都是基于字符型界面的。著名的编辑器有vim[13]、emacs[14]等。虽然能够支持的功能足够强大，但对于习惯了Windows下的图形用户界面的软件研发人员来说，毕竟总有些遗憾之处！所幸的是，今天的

linux同样提供了许多优秀的图形界面开发工具供研发人员使用。下面从软件分析和设计、集成开发环境、源代码分析工具、差异比较工具等方面介绍几款常用的开发工具，另外Linux下的调试工具有很多，其中Kdbg是一个基于gdb的KDE应用程序，作为Linux平台下一个优秀的代码调试工具，可以在图形界面下直观地提供变量检查、断点设置等功能，弥补了Gdb在调试商的不足。这里就不在详细介绍。

### 3.2.1 软件分析和设计

另外，在软件设计过程中，尤其是在利用面向对象技术进行研发时，一款优秀的软件设计和分析工具是必不可少的，由于 UML 在行业中的事实标准地位，其中最著名的 UML 工具有 Rational Rose 和 Borland Together，另外，笔者采用的一款开源的 StarUML 也非常不错。但令人遗憾的是，以上三种 UML 工具都只能在 windows 环境下运行。在 linux 环境下，目前常用的 UML 工具为 UMbrello。

1. Rational Rose

Rational Rose 最初是由 Tational 公司推出的一套完全的，具有能满足所有建模环境需求的建模环境，它允许开发人员、项目经理、系统工程师和分析师在软件开发周期内将需求和系统的体系结构转换为代码，消除浪费的消耗，对需求和系统的体系结构进行可视化、理解和精炼，

Rational Rose 能够支持 C＋＋、Visual Basic、Java、Oracle、CORBA 或者数据定义语言（Data Defition Language）。

其最新版本为Rational Rose 2007,官方网站为http://www.rational.com。关于Rational Rose使用的更多知识请参考参考文档[29]。

2. Borland Together

Borland Together 是一个跨平台的 UML 建模解决方案，可为创建 UML 图表提供高级建模环境，致力于通过公共的可视化语言来消除架构师、分析师和开发人员及其他相关人员沟通中存在的鸿沟。

Borland Together 支持业务过程建模，并通过模型到模型间的转换支持模型驱动架构（MDA）的设计和开发。

Borland Together 由著名的开发工具提供商 Borland 公司开发，分为 Together Archutect、Together Designer 和 Together Developer 等版本。

3. StarUML

StarUML 是一个开源软件，可以快速、灵活、可扩展地利用 UML/MDA 在 windows 下进行软件的分析和设计，支持 UML 2.0 标准，支持常用的 Java、C＋＋、C＃等语言，支持代码生成和反向工程。目的即为提供一个可代替 Rational Rose 和 Borland Together 等的开源建模工具和平台。

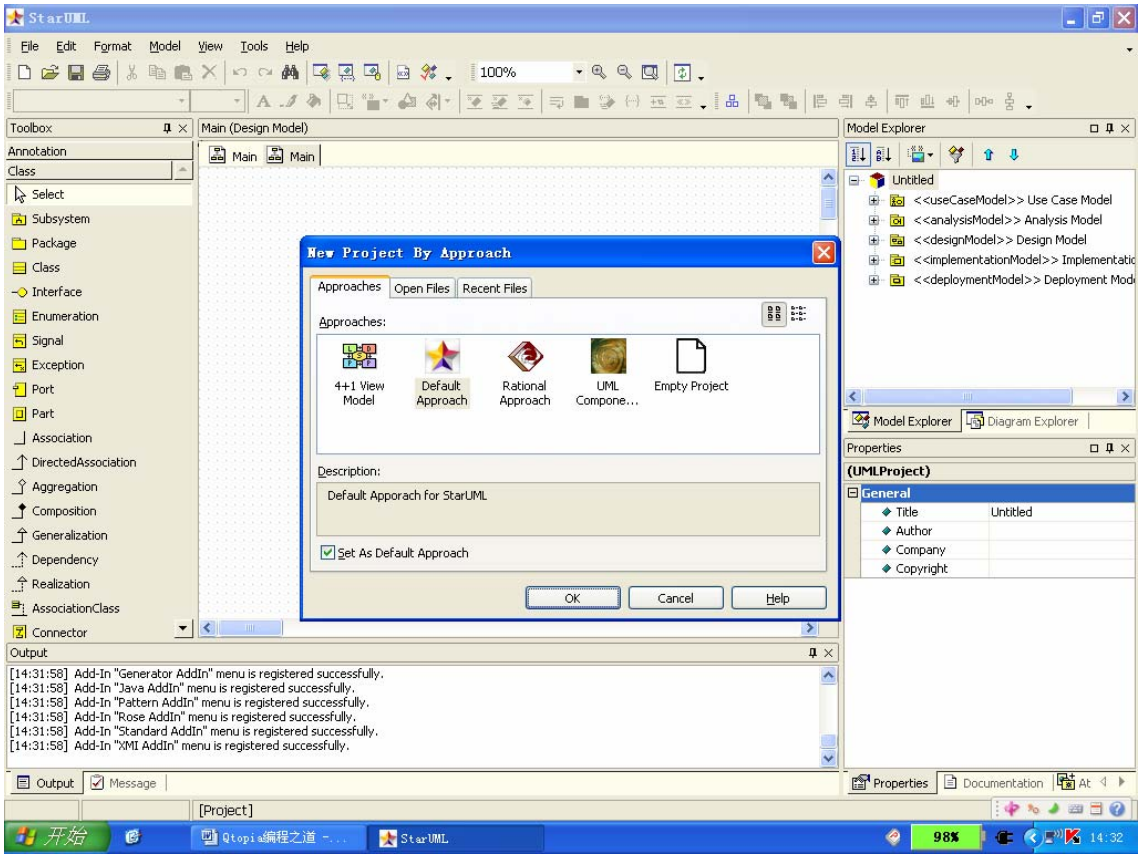


图6. StarUML 5

StarUML 的前身为 Plastic，从 1996 年开始开发，在 1998 年，Plastic 转变为 UML 建模工具，2005 年 Plastic 改名为 StarUML。目前的最高版本为 StarUML 5.0。

StarUML 和其核心模块在 GPL 下发布。但允许进行特定的商业库和组件扩展，其官方网站为 http://staruml.sourceforge.net。

4. UMbrello

UMbrello 是一个基于 KDE 桌面环境的 UML 建模工具，其最新版本 2007 年 11 月发布的 UMbrello 1.5.8，官方网站为 http://staruml.sourceforge.net。

### 3.2.2 集成开发环境

当前的 Linux 平台上，已经拥有两款足够优秀的集成开发环境 KDevelop 和 Eclipse。由于 Eclipse 的跨平台特性和超强的开放性，笔者在进行软件研发时，毫不犹豫的投入了 Eclipse 的怀抱。同时 KDevelop 作为一款同样优秀的集成开发环境同样值得称道。

1. Eclipse

Eclipse 是目前最著名的一个开源集成开放环境，前身是 IBM 公司研发的 Visual Age for Java，后来 IBM 为了更好的对抗 Borland 公司 Java Builder 和 Sun 公司的 NetBeans，将 Visual Age for Java 贡献给了开源社区，并继续投入大量资金进行研发。由于 Eclipse 良好的可扩展性以及 IBM 公司的强大号召力，使得 Eclipse 逐渐发展成为一款极为优秀的集成研发环境。

就 Eclipse 项目本身而言，它专注于为高度集成的工具研发提供一个全功能的、具有商业品质的工业平台，目前主要由三个子项目组成，包括 Eclipse 项目、Eclipse 工具项目和 Eclipse 技术项目等，具体包括四个部分组成--Eclipse Platform、JDT、CDT 和 PDE。JDT 支持 Java 研发、CDT 支持 C 研发、PDE 用来支持插件研发，Eclipse Platform 则是一个开放的可扩展 IDE，提供了一个通用的研发平台。它提供建造块和构造并运行集成软件研发工具的基础。Eclipse Platform 允许工具建造者独立研发与他人工具无缝集成的工具从而无须分辨一个工具功能在哪里结束，而另一个工具功能在哪里开始。

Eclipse SDK（软件研发者包）是 Eclipse Platform、JDT 和 PDE 所生产的组件合并，它们可以一次下载。这些部分在一起提供了一个具有丰富特性的研发环境，允许研发者有效地建造可以无缝集成到 Eclipse Platform 中的工具。Eclipse SDK 由 Eclipse 项目生产的工具和来自其它开放源代码的第三方软件组合而成。Eclipse 项目生产的软件以 CPL 发布，第三方组件有各自自身的许可协议。

目前 Eclipse 的最高版本为 3.3，能够支持的操作系统平台包括 windows、Linux、Mac OS X 等，通过安装相应的插件，Eclipse 3.3 能够支持 Java、C/C++、PHP 等众多程序设计语言，在官方网站：http://www.eclipse.org/可以很容易的下载到支持 C/C++的基于 windows 平台的 eclipse-cpp-europa-win32.zip 和基于 linux 平台的 eclipse-cpp-europa-linux-gtk.tar 工具包。

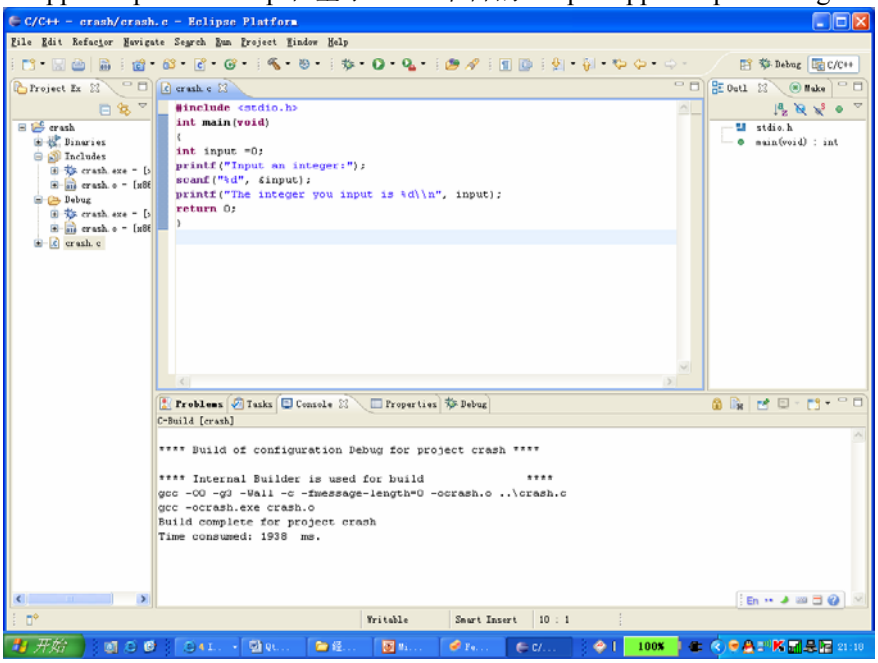


图7. Eclipse 3.3

另外，需要说明的是，如果你机器上没有 Java 5 JRE，则从 Sun 官方网站上下载 Java 5 JRE。先编译 Java 5 JRE，然后将 Eclipse 3.3 解压到特定目录。出于方便的需要，笔者建议用户为 eclipse 的可执行文件创建快捷方式。

在 Windows 平台下，创建快捷方式的方法如下：在 eclipse.exe 文件上点击右键，选择“发送到——→桌面快捷方式”菜单。

在 Linux 平台下，创建快捷方式的方法如下：在 Linux 桌面上点击邮件，选择“Create Launcher”菜单。在如下图所示的对话框的“Name”选项中输入“eclipse 3.3”,在“Command”选项中输入“eclipse”，在“Type”选择中默认选择“Application”，倾向于完美的话可以在“Icon”中选择一个自己喜欢的图标，然后单击“OK”。就可以在 Linux 桌面上看到自己为 Eclipse 创建的快捷方式了。在笔者的笔记本电脑上采用的 Linux 操作系统为 SUSE Linux 10.1 版本。

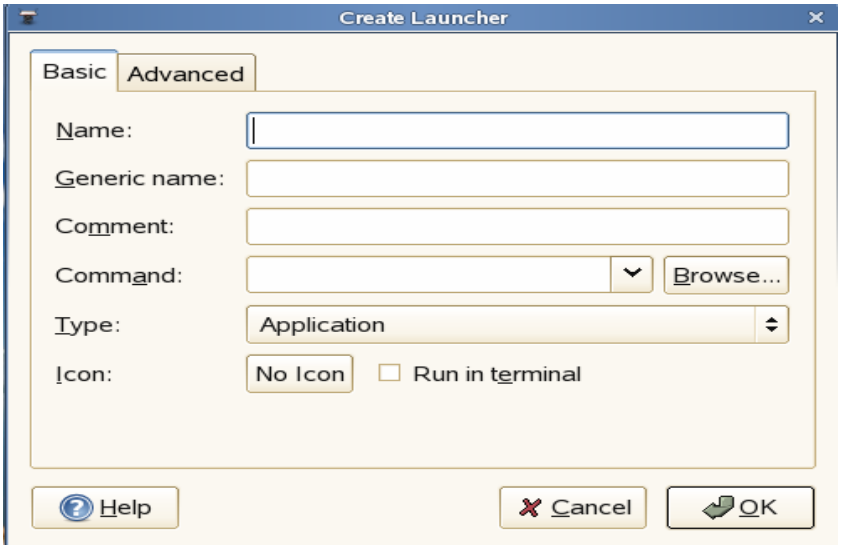


图8.　Linux 下创建快捷方式

打开 Eclipse 3.3 后然后创建一个 C＋＋的工程，点击 projectName 右键选择 Properties 选项。在 C/C++ Gereral 目录下选择 Paths and symbols,在里面添加 Lib 和 Include 路径。

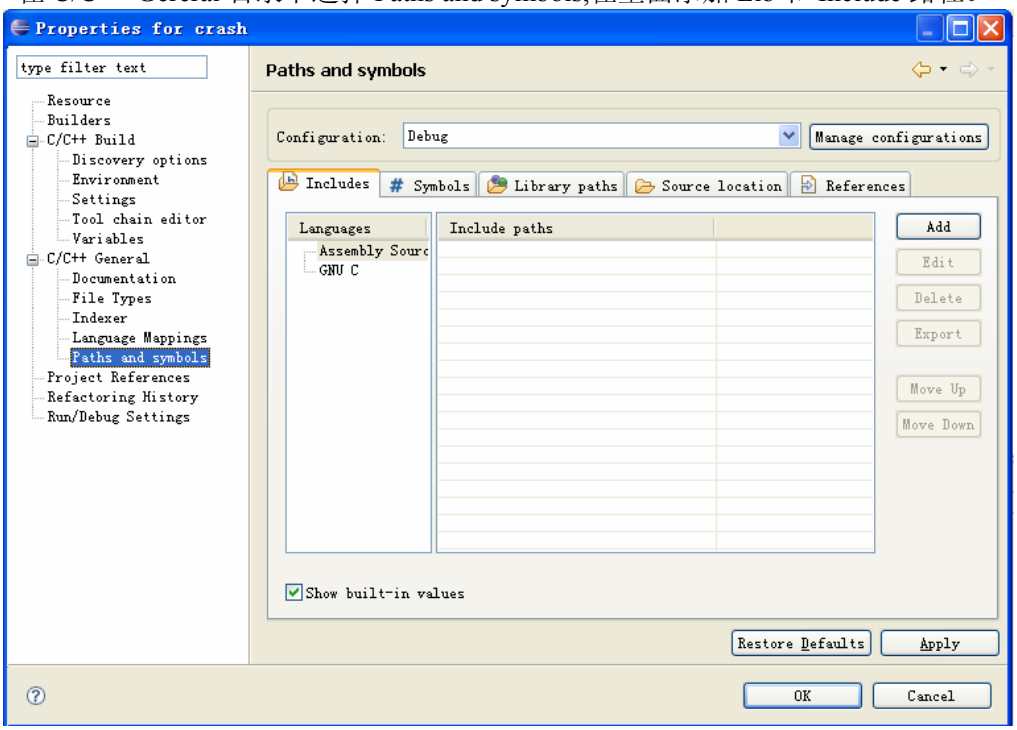


图9.　添加 include 和 lib 路径

点击 projectName 右键选择 Make target,选择 Create 创建编译命令，编译时采用 Build 选项选择正确的命令编译。

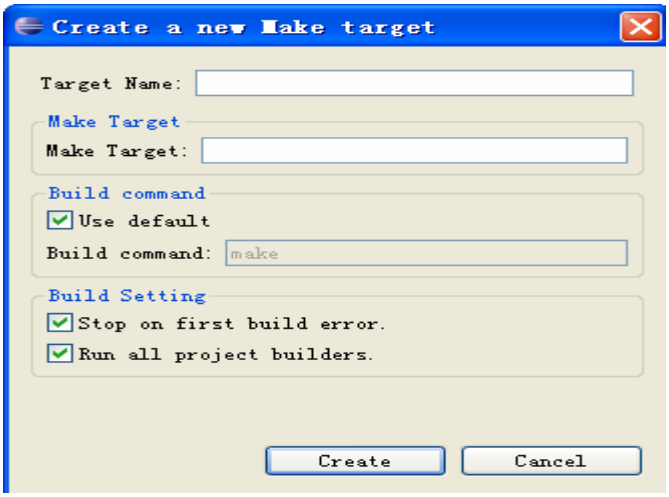


图10. 创建 Make Target

点击 projectName 右键选择 Debug as,如果是本地应用程序，则采用 Local C/C++ Application 方式进行调试，如果应用程序不在本地，则采用 Open Debug Dialog 方式进行调试，在 Dialog 中可以设置应用程序路径、环境变量、Debug 工具等。

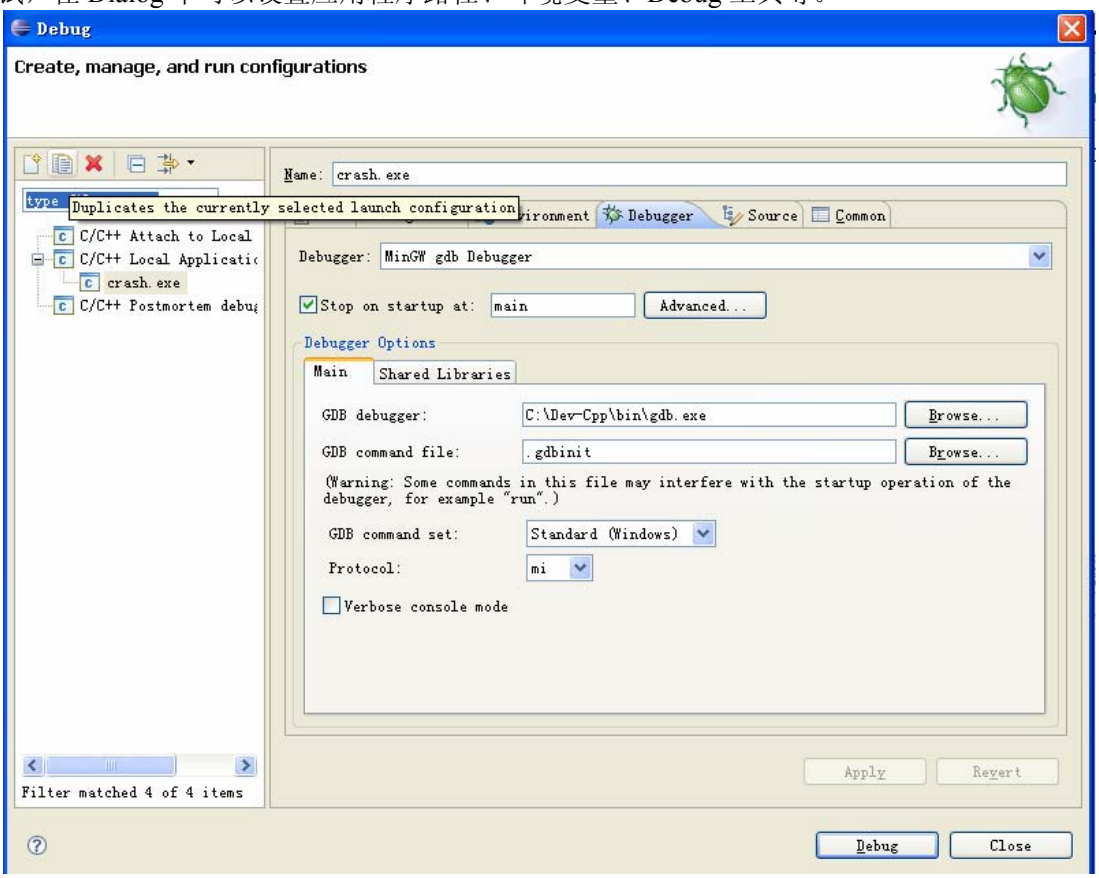


图11. 配置 Debug 对话框

2. KDevelop

KDevelop-Project诞生于 1998 年，其目的是为KDE提供一个易用的集成开发环境，此后，KDevelop IDE采用GPL进行发布， 它支持很多程序设计语言。如C/C++、Java、Qt等。目前的最新版本为 3.5.0，其官方网站为http://www.kdevelop.org。

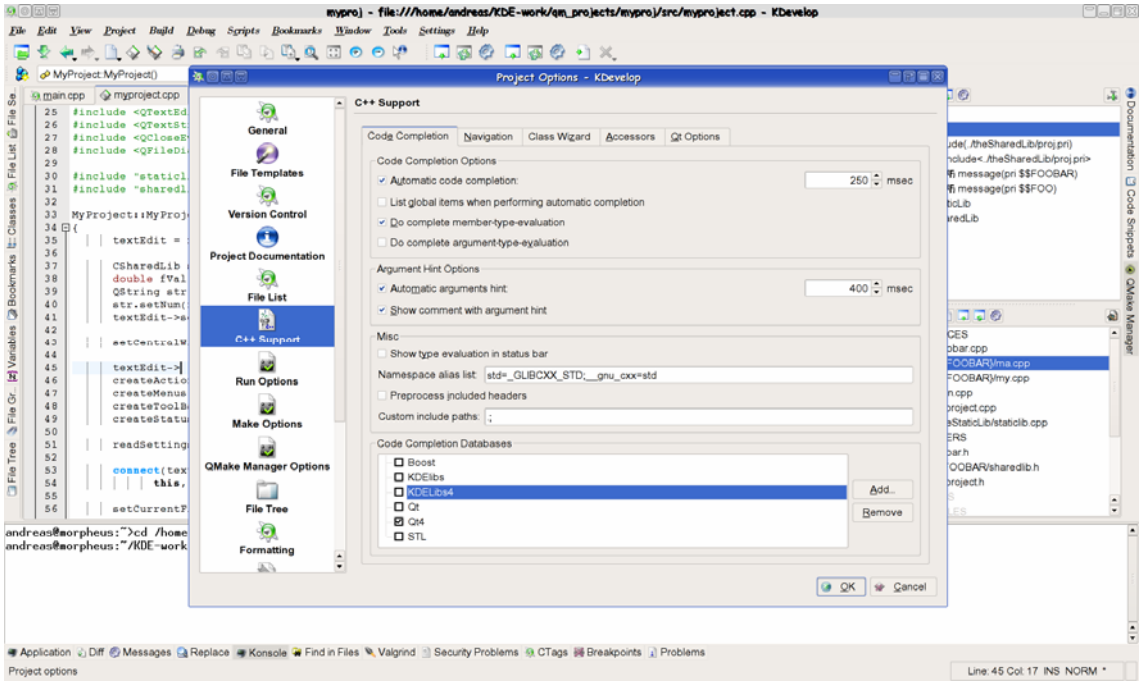


图12. Kdevelop 3.4

### 3.2.3 源代码分析

在进行软件开发，尤其是大规模的软件开发时，常常需要对他人的代码进行学习，这时候就十分需要一款足够强大的源代码分析工具。

在当前的 linux 环境中，最常用的两种源代码分析工具为 Kscope 和 Source Navigator，其中 Source Navigator 功能最强，在代码导航方面甚至超过了 windows 最著名的 Source Insight，但由于长久没人维护原因，常常无法在较新的 linux 发行版上安装。而 Kscope 则是 linux 环境下较新的一个源代码分析工具，遗憾的是尽管有 Graphviz 的支持，Kscope 对代码间的调用关系支持的仍然不够好。但以足以应付大多数工程，笔者的笔记本上就安装了 Kscope，类似于 Source Insight 的操作方式深得笔者之心。

1. KScope

Kscope 作为 Cscope 的前端，为大规模 C/C++工程如 Linux kernel 等提供了一个优秀的原代码编辑和分析环境需要说明的是 KScope 无意代替当前 Linux/KDE 下的集成开发环境如 KDevelop，KScope 并不提供原代码编译/调试的功能。

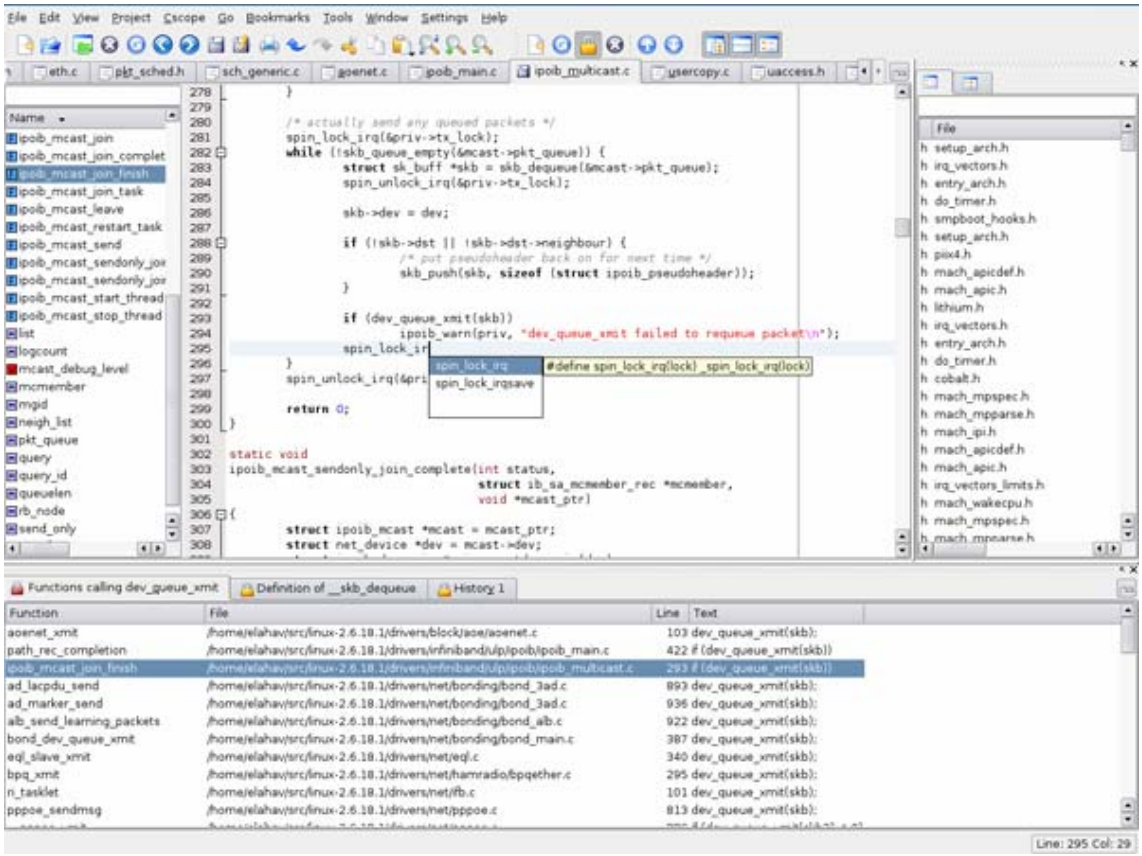

图13. Kscope 1.6

Kscope 提供的主要功能有：

- 多编辑窗口(基于 kate)
- 工程管理
- 大多数 Cscope 查询的前端
- Tag list
- 调用树
- 会话管理

KScope 的作者为 Elad Lahav，另外 Albert Yosher 和 Gabor Fekete 在代码维护方面也做了不少的工作，其最新版本为 1.6.0，Kscope 的官方网站为 http://kscope.sourceforge.net。

安装方法为，首先从 Kscope 的官方网站上下载最新的源代码 kscope-1.6.0.tar.gz，直接解压安装：

＃tar zxvf kscope-1.6.0.tar.gz
#cd kscope-1.6.0
#./configure
#make
#make install

需要说明的是，Kscope 是基于 KDE 桌面环境的工具，再安装 Kscope 前，应保证你的 Linux 上已经安装了 KDE 和 Qt 的开发包，另外，Kscope 对 Cscope、Ctags 和 dot 存在依赖关系，为了能够更好的发挥 Kscope 的功能，需要安装以上三个开发包。下面对 Cscope、Ctags 和 dot 进行说明。

Cscope 为一个源代码浏览工具，最初由贝尔实验室开发，最初作为AT＆T Unix发行版的一部分发布，在 2000 年 4 月，才有Santa Cruz Operation公司（被Caldera公司并购）在BSD的license下开源。其最新版本为 15.6，其官方网站为：http://cscope.sourceforge.net/。

安装方法为，首先从 Kscope 的官方网站上下载最新的源代码 kscope-1.6.0.tar.gz，直接解压安装：

＃tar zxvf cscope-15.6.tar.gz
#cd cscope-15.6
#./configure
#make
#make install

Ctags 能够为源代码产生索引（或 tag）文件，在 linux 下广泛使用，其最新版本为 5.7，其官方网站为 http://ctags.sourceforge.net。安装方法为，首先从 Ctags 的官方网站上下载最新的源代码 ctags-5.7.tar.gz ，直接解压安装：

＃tar zxvf ctags-5.7.tar.gz
#cd ctags-5.7
#./configure
#make
#make install

所谓 dot，是指 Graphviz，它是一个图表程序，如果要使用 Kscope 提供的图表功能，需要 graphviz 的支持。graphviz 的最新版本为 2.14，其官方网站为：http://www.graphviz.org。安装方法为，首先从 graphviz 的官方网站上下载最新的源代码 kscope-1.6.0.tar.gz，直接解压安装：

＃tar zxvf graphviz-2.14.tar.gz
#cd graphviz-2.14
#./configure
#make
#make install

2. Source Navigator

Source Navigator 是 RedHat 出品的一款 Linux 环境下的代码阅读工具，代码导航功能比 SourceInsight 比强，但速度稍慢。在安装过程中，要给所有的文件建立索引，花费的时间很长。其最新版本为 2004 年发布的 5.2b2 版，已经很久没有更新了，其官方网站为 http://sourcenav.sourceforge.net。于 2000 年在 GPL 下开源。

3. Source Insight

Source Dynamics 公司成立于 1994 年。Source Insight 软件最初作为一个内部的编辑工具，协助一些全球最成功和最复杂的软件引用程序的研发。其最新版本为 3.5.0058，其官方网站为 http://www.sourceinsight.com。

Source Insight 是一个革新的面向项目研发的程序编辑器和代码浏览器，它拥有内置的对 C/C++、C#和 Java 等程序的分析。Source Insight 能分析你的源代码并在你工作的同时动态维护它自己的符号数据库，并自动为你显示有用的上下文信息。Source Insight 不仅仅是一个强大的程序编辑器，它还能显示 reference trees，class inheritance diagrams 和 call trees。Source Insight 提供了最快速的对源代码的导航和任何程序编辑器的源信息。就将 Source Insight 应用到你的项目研发过程中并切实感受它为你项目研发的效率带来的变化。

Source Insight 适用于大型和要求严格的程序研发项目。 事实上，Source Insight 正在和曾经被用于研发一些迄今为止全球最大型和最成功的商业软件产品。

SourceInsight3.5 通过详细的交叉参考提供了代码导航功能，并提供了一个支持语法着色的“聪明”的编辑器，以及多种图形化的反向工程视图。能够有效的帮助反向研发人员跟踪敏感信息，理解已有工程的设计结构。

它的使用非常简单：先选择 Project 菜单下的 new，新建一个工程，输入工程名，接着把欲读的源代码加入（可以整个目录加），该软件会自动分析我们添加的源代码。分析完后，就可以进行阅读了。其强大的功能体现在：

- 代码关联

SourceInsight 具备超强的代码分析能力，其会将工程中的所有代码进行关联性检查，建立纵横复杂的代码关联，提供类似超文本的代码视图能力。例如，在 SourceInsight 中，如果想看某一变量、函数或宏的定义，只需把光标定位于该变量、函数或宏，然后点击工具条上的跳转，该变量、函数或宏的定义就显示出来；对于函数，我们还可以跳转到引用它的地方。

例如在下面图中代码的"msg_rmid"宏上，右击"Jump to Definition"：

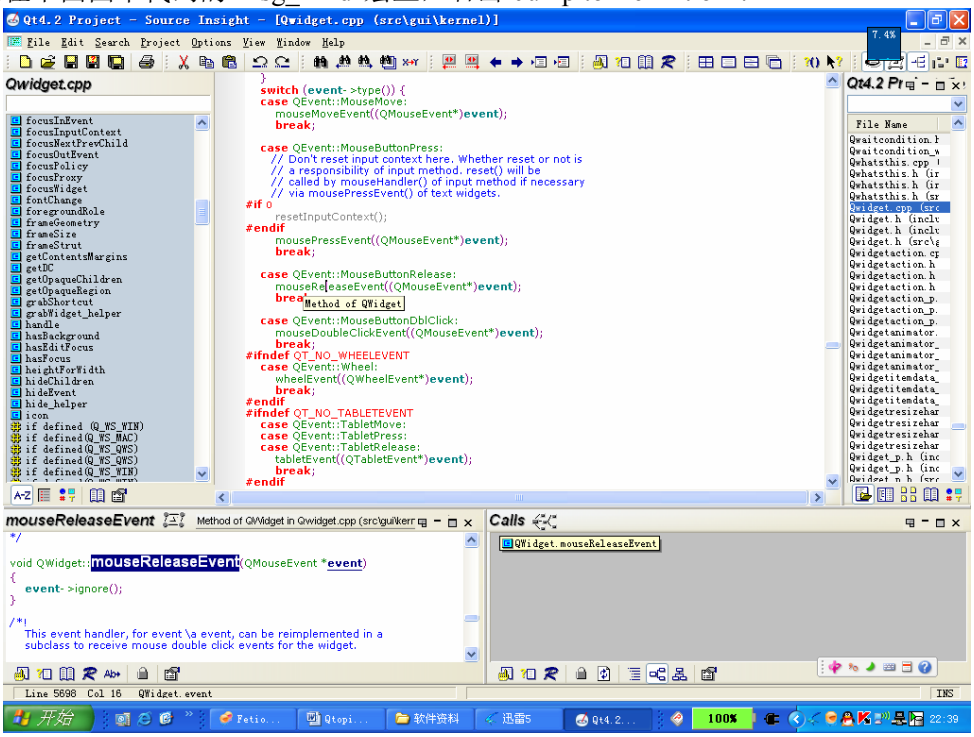


图14. SourceInsight3.5

➢ 变量/数据结构/函数视图方式：

一般的阅读软件只能以文件的方式视图，但是 SourceInsight 会自动分析代码，并以树的形式列出变量、数据结构和函数，非常类似于 Visual C++的类视图方式，如下图：

➢ 具有类似于 IE 浏览器的在代码中前进与返回功能，自由穿梭于复杂的代码

### 3.2.4 差异比较

当前可用的代码差异比较工具很多，在 windows 下著名的用 Beyond Compare 和 winMerge，在 linux 下著名的有 KDiff3 和 vimdiff。其中 Beyond Compare 和 KDiff3 最受研发人员的欢迎，Vimdiff 则比较适合于文件之间的快速比较。

1. Beyond Compare

Beyond Compare 是一个综合的比较工具。可比较的对象包括文本文件、文件夹、zip 压缩文件、FTP 站点，等等。您可以使用它管理您的源代码、保持目录同步、比较程序的输出结果、以及验证所刻录的光盘副本的精确度。其最新版本为 2.5.2，官方网站为 http://www.scootersoftware.com。

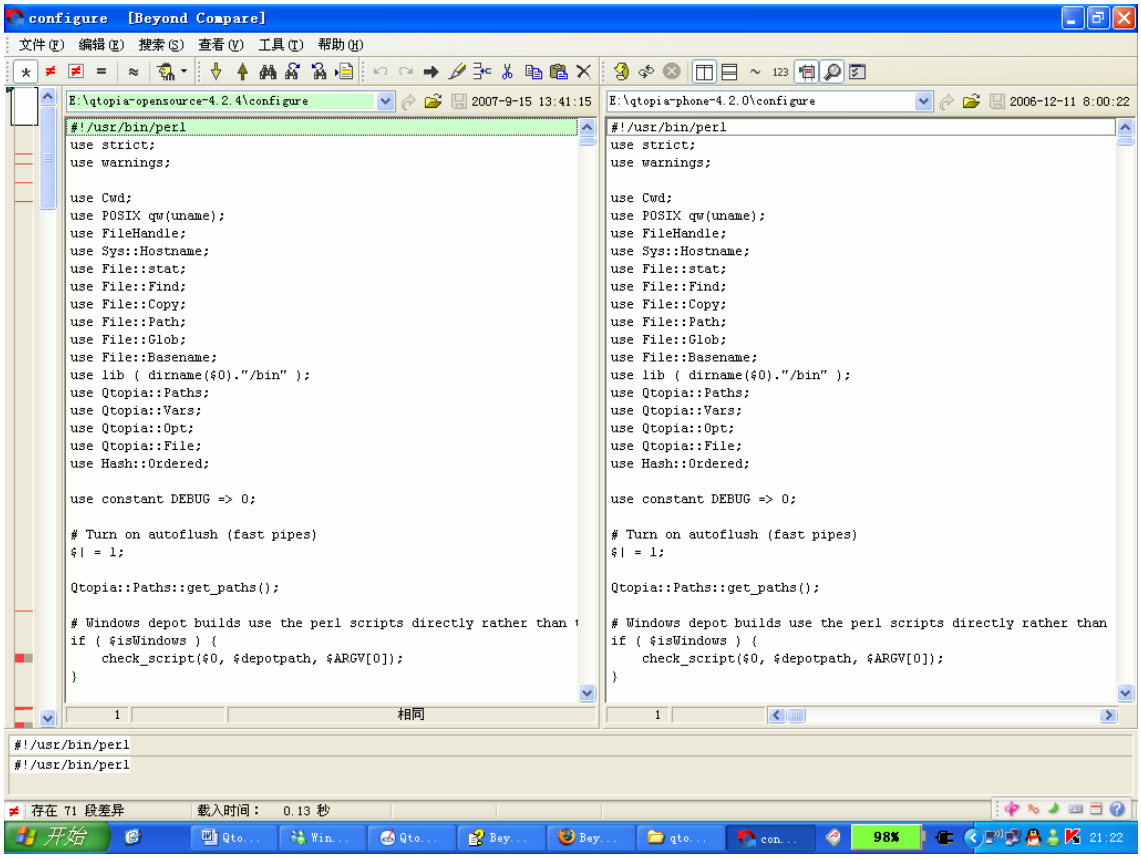


图15. Beyond Compare 2

虽然 Beyond Compare 能够支持自动化功能，但 Beyond Compare 的主要目标还是帮助您详尽的分析差异之处，并且对它们进行详尽的处理。 软件内部包含了许多文件和文件夹命令操作。

Beyond Compare 把它比较的信息区分为文件或文件夹。 Beyond Compare 并不仅限于进行本地文件和文件夹的比较，它还可以对局域网和 FTP 上的文件夹和文件进行比较。

在 Beyond Compare 对文件的显示方面，带有文件夹性质的文件，比如 ZIP 文件和 CAB 文件，会被以文件夹进行处理。 Beyond Compare 也能够生成类似文件夹状态的文件，叫做 "快照文件"。 快照文件是文件夹在某一特定时间所包含内容的影像，但快照文件并不包含文件夹的实际子文件夹或文件。

Beyond Compare 的主要部分是分两侧窗口显示 文件夹查看器 和分两侧窗口显示的 文件查看器。 另外也有很多 插件查看器 用来对特定格式文件进行比较。 这些插件对注册用户来说是免费的，您可以访问官方网站获得。

2. KDiff3

KDiff3 是一款用来对文件或目录进行比较/合并的工具，在比较时它可以同时针对两个或者三个文件/目录而进行。通过比较，它将文件/目录的差异按行加以显示。同 时，KDiff3 提供有自动化的合并工具，方便使用者进行有关合并的操作。此外，KDiff3 支持 Unicode 编码，集成了编辑器，可以自动合并版本控制历史。虽然 KDiff3 主要为 KDE 桌面而开发，但是仍然可以运行于其他的 Linux 环境。甚至对于 Windows、Mac OS X，KDiff3 也有相应的版本。KDiff3 的作者为 Joachim Eibl，其最新版本为 0.9.92。官方网站为 http://kdiff3.sourceforge.net。

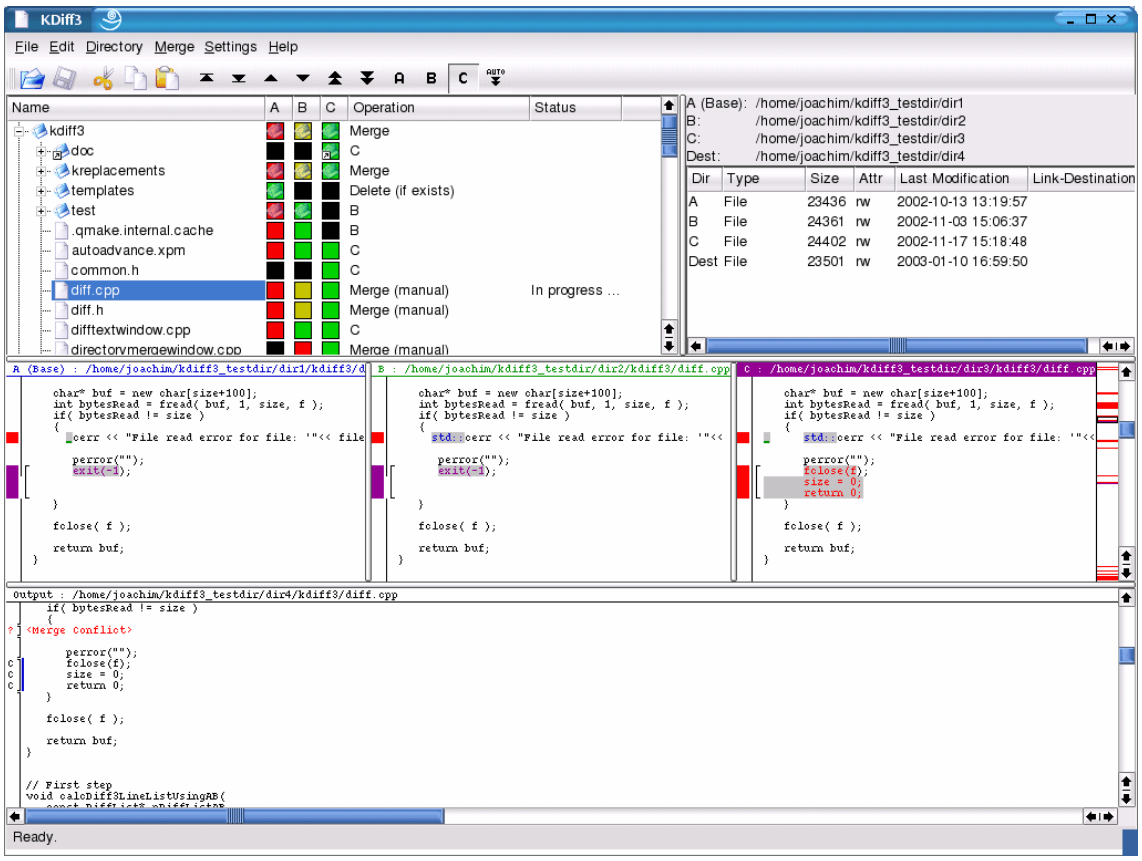


图16. KDiff3

3. vimdiff

当远程工作在 Unix/Linux 平台上的时候，恐怕最简单而且到处存在的就是命令行工具，比如 diff。可惜 diff 的功能有限，使用起来也不是很方 便。作为命令行的比较工具，我们仍然希望能拥有简单明了的界面，可以使我们能够对比较结果一目了然；我们还希望能够在比较出来的多处差异之间快速定位，希 望能够很容易的进行文件合并。而 Vim 提供的 diff 模式，通常称作 vimdiff，就是这样一个能满足所有这些需求，甚至能够提供更多的强力工具。VIM 的作者 Bram Moolenaar ，其最新版本为 7.1，其官方网站为 http://www.vim.org。

Vimdiff 的基本用法就是：

# vimdiff FILE_LEFT FILE_RIGHT

或者

# vim -d FILE_LEFT FILE_RIGHT

比较适合需要快速比较两个文件时使用。

## 3.3 Qt工具

为了编程的方便，在 qt 中，提供了 Qt Designer、Qt Assistant、Qt Linguist 和 qmake 等几个工具，分别针对不同的用途，其中 Qt Designer 是利用 Qt 组件设计或构建 GUI 的工具，Qt Assistant 是个帮助文档性质的工具，Qt Linguist 是为应用程序的本地化提供的图形界面工具，Qmake 是一个能够帮助简化跨平台的研发项目的编译过程的工具。

### 3.3.1 Qt Designer

Qt Designer 是利用 Qt 组件设计或构建 GUI 的工具，除了布局功能外，还能提供信号与槽连接功能，但遗憾的是 Qt Designer 并不是一个真正意义上的所见即所得的工具，因为它无法

实现对组件对象的函数的重载和添加，更不用说继承了！

由于Qt Designer对嵌入式软件研发来说用处不大，这里就不再过多的介绍了，更详细的信息请参考文献[38]。

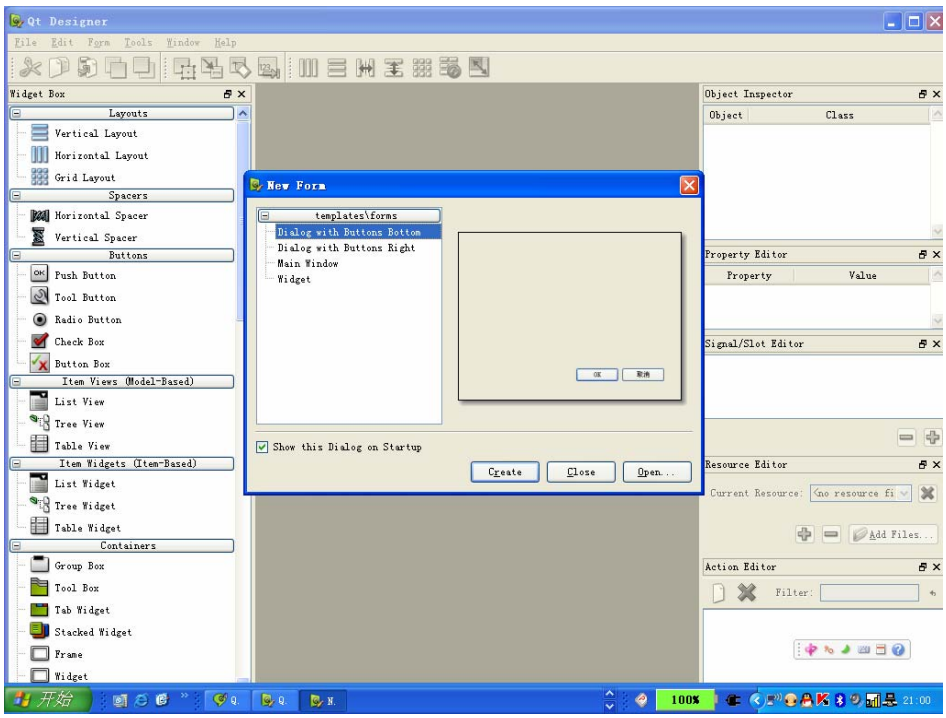


图17. Qt Designer 界面

### 3.3.2 Qt Assistant

Qt Assistant是个帮助文档性质的工具，如果研发人员需要查找库中的某些内容时，Qt Assistant会是个不错的助手。它提供了很清晰的结构，对如何开始编写一个Qt程序、Qt的常识、Qt的API文档、Qt的核心特征及关键技术、Qt的工具等都有详细的说明，这有利于研发人员更快捷的找到所需要的信息。不过令人遗憾的是，其中的部分内容还没有做到和代码的同步，对部分关键点缺乏清晰的说明，一定程度上阻碍了研发人员对Qt的理解。关于Qt更详细的信息请参考参考文献[38]。

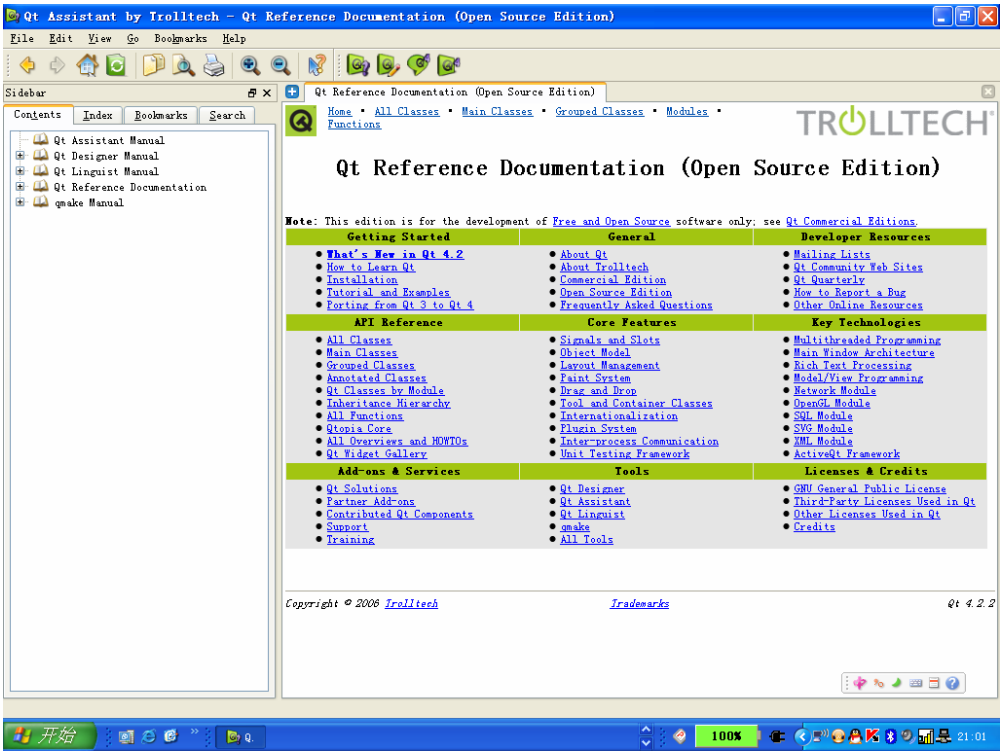


图18. Assistant 界面

### 3.3.3 Qt Linguist

Qt为应用程序的本地化提供了强有力的支持，下图就是Qt为应用程序的本地化提供的图形界面工具 Qt Linguist。Qt Linguist 对 Qt 的另两个本地化工具 lupdate 和 lrelease 进行了封装。

关于本地化的更详细信息请参考 3.2.3 节。

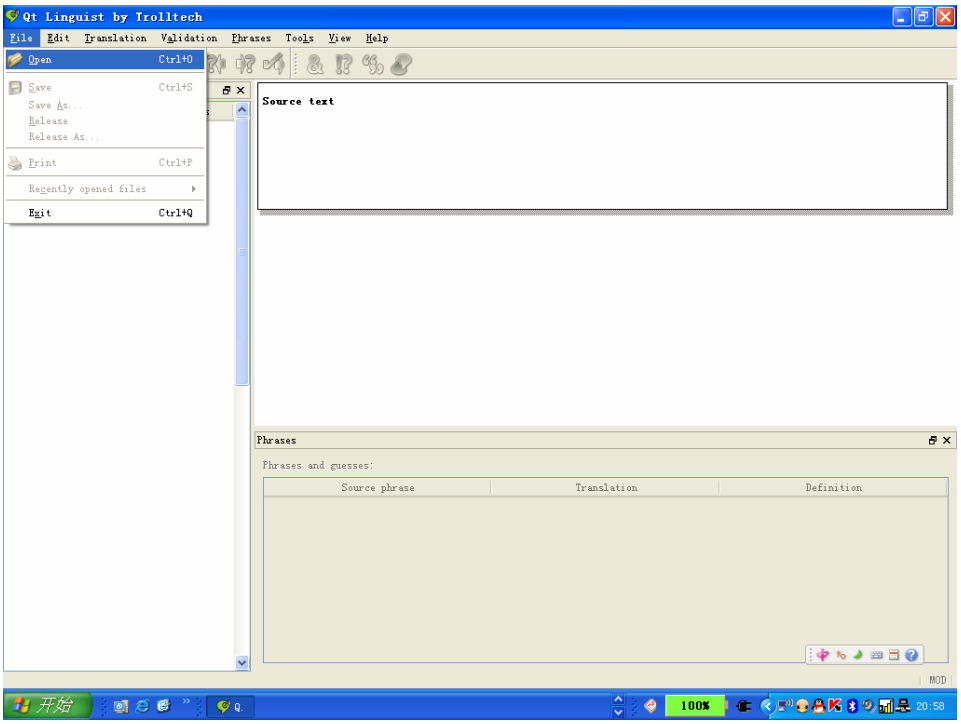


图19. Qt Linguist 界面

### 3.3.4 Qmake

Qmake 工具能够帮助简化跨平台的研发项目的编译过程，qmake 能够根据源代码目录信息自动产生工程文件，并能够根据工程文件自动产生 makefile 文件。但需要说明的是 qmake 产生的工程文件往往比较简单，更复杂的工程文件往往需要研发人员对自动生成的工程文件再修改。

下面介绍下qmake的常用用法,更详细的信息请参考文献[38]。

1. qmake 的用法

qmake [mode] [options] files

- 操作模式

qmake 支持两种操作模式：makefile 和 project.

默认情况下，qmake 会读取工程文件信息来生成 Makefile 文件，如果显式指定 -project，qmake 则会根据当前目录信息生成工程文件。

- 选项

作为一个研发人员一般明白，警告信息往往可能意味着程序可能存在错误风险，因此往往希望在编译调试过程中，编译器能暴露出足够多的警告信息，qt 通过 qmake 可是设定警告信息提示等级。选项如下：

- -Wall
  Qmake 报告所有警告信息
- -Wnone
  Qmake 不报告任何警告信息
- Wparser
  Qmake 会警告在分析工程文件时产生的潜在问题信息
- Wlogic
  Qmake 会提示在分析工程文件时产生的潜在问题信息

关于工程文件的语法信息，请参考 5.2.1 节。

# 第 4 章　Qtopia核心技术

**路漫漫其修远兮，吾将上下而求索。**

《离骚》

标准 C＋＋对象模型为面向对象编程提供了非常有效的实时支持，但美中不足的是它的静态特性在一些领域中表现的并不够灵活。GUI 对实时性和灵活性都有着很高的要求。Qt 通过其改进的对象模型在保持 C＋＋执行速度的同时提供了所需要的灵活性。

Qt 相对于标准 C＋＋增添的特性有：

- 能够强有力的支持对象间通信信号与槽机制
- 支持可查询和可设计的动态对象属性机制
- 事件和事件过滤器
- 基于上下文的字符串国际化
- 能够支持多任务的定时器
- 支持按层检索的对象树
- 受保护指针
- 动态类型转换

Qt 是基于单根 QObject 的架构，其众多特性都是基于 QObject 提供的，然而如对象通信机制和动态属性系统等，则还需要 Qt 的元对象系统的支持。

```
class Q_CORE_EXPORT QObject//QtopiaCore 类型类声明
{
    Q_OBJECT                                        //使用元对象系统声明
    Q_PROPERTY(QString objectName READ objectName WRITE setObjectName)
    Q_DECLARE_PRIVATE(QObject)
public:
    explicit QObject(QObject *parent=0);
    virtual ~QObject();
    virtual bool event(QEvent *);                   //事件
    virtual bool eventFilter(QObject *, QEvent *);  //事件过滤器
#ifdef qdoc
    static QString tr(const char *sourceText, const char *comment = 0, int n = -1);//国际化
    static QString trUtf8(const char *sourceText, const char *comment = 0, int n = -1);
    virtual const QMetaObject *metaObject() const;
    static const QMetaObject staticMetaObject;
#endif
#ifdef QT_NO_TRANSLATION
    static QString tr(const char *sourceText, const char *, int)
        { return QString::fromLatin1(sourceText); }
    static QString tr(const char *sourceText, const char * = 0)
        { return QString::fromLatin1(sourceText); }
#ifndef QT_NO_TEXTCODEC
    static QString trUtf8(const char *sourceText, const char *, int)
        { return QString::fromUtf8(sourceText); }
    static QString trUtf8(const char *sourceText, const char * = 0)
        { return QString::fromUtf8(sourceText); }
#endif
#endif //QT_NO_TRANSLATION
    QString objectName() const;
    void setObjectName(const QString &name);        //设置对象名
    inline bool isWidgetType() const { return d_ptr->isWidget; }//判断是否为一个 Widget
    inline bool signalsBlocked() const { return d_ptr->blockSig; }
```

```
    bool blockSignals(bool b);
    QThread *thread() const;
    void moveToThread(QThread *thread);
    int startTimer(int interval);                                   //启动定时器
    void killTimer(int id);
#ifndef QT_NO_MEMBER_TEMPLATES
    template<typename T>
    inline T findChild(const QString &name = QString()) const
    { return qFindChild<T>(this, name); }
    template<typename T>
    inline QList<T> findChildren(const QString &name = QString()) const
    { return qFindChildren<T>(this, name); }
#ifndef QT_NO_REGEXP
    template<typename T>
    inline QList<T> findChildren(const QRegExp &re) const
    { return qFindChildren<T>(this, re); }
#endif
#endif
#ifdef QT3_SUPPORT
    QT3_SUPPORT QObject *child(const char *objName, const char *inheritsClass = 0,
                               bool recursiveSearch = true) const;
    QT3_SUPPORT QObjectList queryList(const char *inheritsClass = 0,
                                      const char *objName = 0,
                                      bool regexpMatch = true,
                                      bool recursiveSearch = true) const;
#endif
    inline const QObjectList &children() const { return d_ptr->children; }
    void setParent(QObject *);                                      //设定父类
    void installEventFilter(QObject *);                             //安装事件过滤器
    void removeEventFilter(QObject *);                              //移去事件过滤器
    static bool connect(const QObject *sender, const char *signal,
            const QObject *receiver, const char *member, Qt::ConnectionType =
#ifdef QT3_SUPPORT
                                    Qt::AutoCompatConnection
#else
                                    Qt::AutoConnection
#endif
            );                                                      //信号与槽连接
    inline bool connect(const QObject *sender, const char *signal,
                                    const char *member, Qt::ConnectionType type =
#ifdef QT3_SUPPORT
                                    Qt::AutoCompatConnection
#else
                                    Qt::AutoConnection
#endif
            ) const;
    static bool disconnect(const QObject *sender, const char *signal,
            const QObject *receiver, const char *member);           //断开信号与槽连接
    inline bool disconnect(const char *signal = 0,
            const QObject *receiver = 0, const char *member = 0)
            { return disconnect(this, signal, receiver, member); }
    inline bool disconnect(const QObject *receiver, const char *member = 0)
            { return disconnect(this, 0, receiver, member); }
    void dumpObjectTree();
    void dumpObjectInfo();
#ifndef QT_NO_PROPERTIES
```

```
        bool setProperty(const char *name, const QVariant &value);//设置对象属性
        QVariant property(const char *name) const;                    //获取对象属性
        QList<QByteArray> dynamicPropertyNames() const;
    #endif // QT_NO_PROPERTIES
    #ifndef QT_NO_USERDATA
        static uint registerUserData();
        void setUserData(uint id, QObjectUserData* data);
        QObjectUserData* userData(uint id) const;
    #endif // QT_NO_USERDATA
    Q_SIGNALS:
        void destroyed(QObject * = 0);
    public:
        inline QObject *parent() const { return d_ptr->parent; }          //获取父对象
        inline bool inherits(const char *classname) const       //判断该类是否继承于某个类
            { return const_cast<QObject *>(this)->qt_metacast(classname) != 0; }
    public Q_SLOTS:
        void deleteLater();
    protected:
        QObject *sender() const;
        int receivers(const char* signal) const;
        virtual void timerEvent(QTimerEvent *);
        virtual void childEvent(QChildEvent *);
        virtual void customEvent(QEvent *);
        virtual void connectNotify(const char *signal);
        virtual void disconnectNotify(const char *signal);
    #ifdef QT3_SUPPORT                                          //判断是否需要与 QT3 兼容
    public:
        QT3_SUPPORT_CONSTRUCTOR QObject(QObject *parent, const char *name);
        inline QT3_SUPPORT void insertChild(QObject *o)
            { if (o) o->setParent(this); }
        inline QT3_SUPPORT void removeChild(QObject *o)
            { if (o) o->setParent(0); }
        inline QT3_SUPPORT bool isA(const char *classname) const
            { return qstrcmp(classname, metaObject()->className()) == 0; }
        inline QT3_SUPPORT const char *className() const
    { return metaObject()->className(); }
        inline QT3_SUPPORT const char *name() const
    { return objectName().latin1_helper(); }
        inline QT3_SUPPORT const char *name(const char *defaultName) const
            { QString s = objectName(); return s.isEmpty()?defaultName:s.latin1_helper(); }
    inline QT3_SUPPORT void setName(const char *name)
{ setObjectName(QLatin1String(name)); }
    protected:
        inline QT3_SUPPORT bool checkConnectArgs(const char *signal,
              const QObject *,const char *member)
            { return QMetaObject::checkConnectArgs(signal, member); }
        static inline QT3_SUPPORT QByteArray normalizeSignalSlot(const char *signalSlot)
            { return QMetaObject::normalizedSignature(signalSlot); }
    #endif
    protected:
        QObject(QObjectPrivate &dd, QObject *parent = 0);
    protected:
        QObjectData *d_ptr;
        static const QMetaObject staticQtMetaObject;
        friend struct QMetaObject;
        friend class QApplication;
        friend class QApplicationPrivate;
```

```
    friend class QCoreApplication;
    friend class QCoreApplicationPrivate;
    friend class QWidget;
    friend class QThreadData;
private:
    Q_DISABLE_COPY(QObject)
    Q_PRIVATE_SLOT(d_func(), void _q_reregisterTimers(void *))
};
```

## 4.1 父子化机制

Qt 的父子化机制是在 QObject 类中实现的，作为就会 QWidget 与 QLayout 的基类。当我们父化一个对象时（一个窗口小部件、布局或其他种类），该父类添加这个对象到他的子级列表中。

对象的子级对象链表是通过 QObjectList 来实现的，定义如下：

```
typedef QList<QObject*> QObjectList;
```

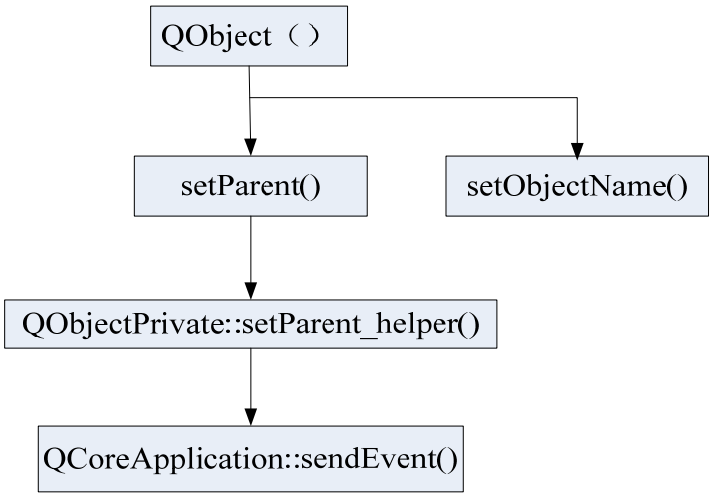


图20. 对象的父化过程

下面是 QObject 的构造函数的处理过程：

```
QObject::QObject(QObject *parent, const char *name): d_ptr(new QObjectPrivate)
{
    Q_D(QObject);
    ::qt_addObject(d_ptr->q_ptr = this);
    d->threadData = QThreadData::current();
    d->threadData->ref();
    if (parent && parent->d_func()->threadData != d->threadData)
    {
        qWarning("QObject: Cannot create children for a parent that is in a
        different thread.");
        parent = 0;
    }
    setParent(parent);                              //父化 parent 对象
    setObjectName(QString::fromAscii(name));        //设置对象名
}
```

下面是 setParent()函数的处理过程：

```
void QObject::setParent(QObject *parent)
{
```

```
    Q_D(QObject);
    Q_ASSERT(!d->isWidget);
    d->setParent_helper(parent);
}
```

下面是 QObjectPrivate 类的 setParent_helper 的处理过程：

```
void QObjectPrivate::setParent_helper(QObject *o)
{
    Q_Q(QObject);
    if (o == parent)
        return;
    if (parent)
    {
        QObjectPrivate *parentD = parent->d_func();
        if (parentD->wasDeleted && wasDeleted
            && parentD->currentChildBeingDeleted == q)
        {
            // don't do anything since QObjectPrivate::deleteChildren() already
            // cleared our entry in parentD->children.
        }
        else
        {
            const int index = parentD->children.indexOf(q);
            if (parentD->wasDeleted)
            {
                parentD->children[index] = 0;
            }
            else
            {
                parentD->children.removeAt(index);
                if (sendChildEvents && parentD->receiveChildEvents)
                {
                    QChildEvent e(QEvent::ChildRemoved, q);
                    QCoreApplication::sendEvent(parent, &e);
                }
            }
        }
    }
    parent = o;
    if (parent)
    {
        // object hierarchies are constrained to a single thread
        if (threadData != parent->d_func()->threadData)
        {
            qWarning("QObject::setParent: New parent must be in the same thread
            as the previous parent");
            parent = 0;
            return;
        }
        parent->d_func()->children.append(q);   //追加对象到父对象的子级列表
        if(sendChildEvents && parent->d_func()->receiveChildEvents)
        {
```

```
            if (!isWidget)
            {
                QChildEvent e(QEvent::ChildAdded, q);
                QCoreApplication::sendEvent(parent, &e);
#ifdef QT3_SUPPORT
                QCoreApplication::postEvent(parent,                    new
                QChildEvent(QEvent::ChildInserted, q));
#endif
            }
        }
    }
}
```

如果研发需要获得子级对象，可以调用 children()得到 child 的对象列表:

```
QObjectList &children() const
{
   return d_ptr->children;
}
```

当一个对象被析构时，它将遍历子级列表并删除所有子级对象。然后从父对象哪里将自己清除。

```
QObject::~QObject()
{
    Q_D(QObject);
    if (d->wasDeleted)
    {
#if defined(QT_DEBUG)
        qWarning("QObject: Double deletion detected");
#endif
        return;
    }
    d->wasDeleted = true;
    d->blockSig = 0; // unblock signals so we always emit destroyed()
    if (!d->isWidget)
    {
        QObjectPrivate::clearGuards(this);
     }
    emit destroyed(this);
    QConnectionList *list = ::connectionList();
    if (list)
    {
        QWriteLocker locker(&list->lock);
        list->remove(this);                 //从信号与槽的连接列表中移去自身
    }
    if (d->pendTimer)
    {
         if (d->threadData->eventDispatcher)
         d->threadData->eventDispatcher->unregisterTimers(this);
    }
    d->eventFilters.clear();
    if (!d->children.isEmpty())
        d->deleteChildren();                //删除子级对象
    {
```

```
            QWriteLocker locker(QObjectPrivate::readWriteLock());
            ::qt_removeObject(this);
            if (d->postedEvents > 0)
                QCoreApplication::removePostedEvents(this);
        }
        if (d->parent)                          // 从父对象中移去自身
            d->setParent_helper(0);
        d->threadData->deref();
        delete d;
        d_ptr = 0;
    }
```

父子化机制简化了内存管理，减少了内存泄漏风险。我们必须删除唯一的对象是我们用 new 创建的并且没有被父化的对象。

## 4.2 元对象系统

Qt 的元对象系统是一个基于标准 C＋＋的扩展，能够使 C＋＋更好的适应真正的组件 GUI 编程。它为 qt 提供了支持对象间通信的信号与槽机制、实时类型信息和动态属性系统等方面的功能。

元对象系统在 Qt 中主要有以下三部分构成：

- 单根架构中的单根 QObject，为利用元对象系统提供了方便。
- Q_OBJECT 宏，当在基类为 QObject 的类的声明中添加 Q_OBJECT 宏后，该类就能支持元对象的特性了。
- 元对象编译器 moc，能够支持元对象的编译，当编译时，编译器会检查所有类的声明中是否包含了 Q_OBJECT 宏，如果包含，则调用 moc 进行编译。其编译过程是 qt 自动运行的，不需要研发人员的干预。

除了最常用的信号与槽机制外，元对象系统还提供了以下特性：

- QObject::metaObject()，能够返回与类相关的元对象。
- QMetaObject::className() 能够在不需要实时类型信息(RTTI,Run-Time Type Information)支持的情况下实时返回字符串类型的类名。这在调试时非常有用。
- QObject::inherits()能够判断一个类是否继承于另一个类。
- QObject::tr()和 QObject::trUtf8()能够为国际化提供字符串翻译。
- QObject::setProperty()和 QObject::property()能够根据名字动态的设置或获取属性。
- QObject::qobject_cast()能够支持动态类型转换。

### 4.2.1 信号与槽机制

程序交互时所用 GUI 应用程序关心的问题，通过将某种用户事件（如鼠标事件、按键事件等）和程序事件（启动应用、退出应用等）联系起来，使用户能够基于图形界面来控制程序，其他 GUI 工具包大都使用回调函数来创建程序交互，但回调函数非常复杂，容易混淆，又难以理解；而 Qt 采用的交互方式不同于其他的 GUI 工具包，它基于 QObject 类内建了特有的信号与槽机制，通过信号和槽将事件和处理函数联系起来。

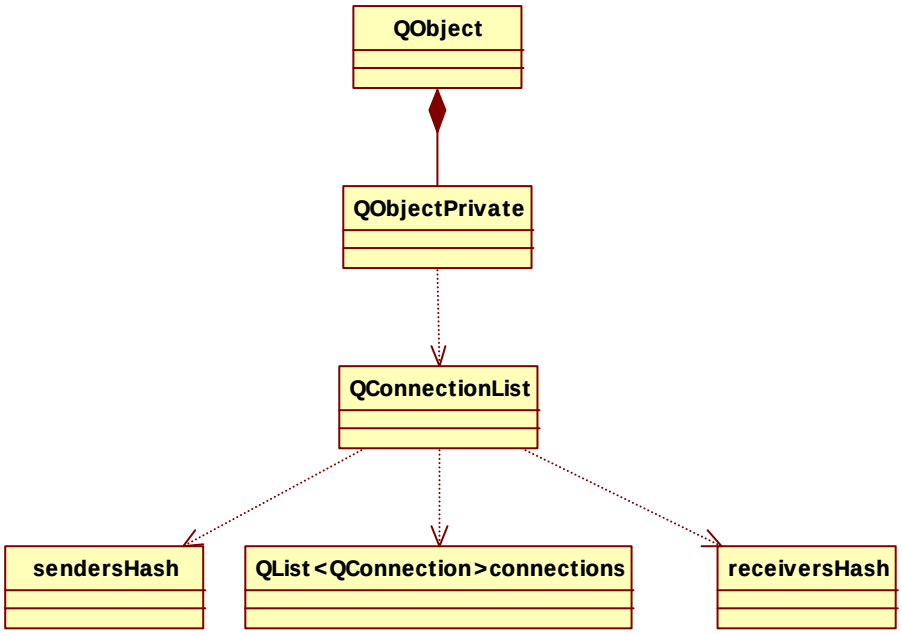


图21. 信号与槽机制的类图

事实上信号和槽都是累的成员函数，槽更是标准函数，如果需要，可以像调用其他函数那样调用他们。一个信号可以于多个槽或信号连接，一个槽可以接收多个信号，当多个槽连接到同一个信号时，如果信号被触发，这些槽的执行顺序将是任意的，这在程序研发中需要特别注意，不能够在这些槽间设计某种关联。下图为信号与槽链接的创建过程：

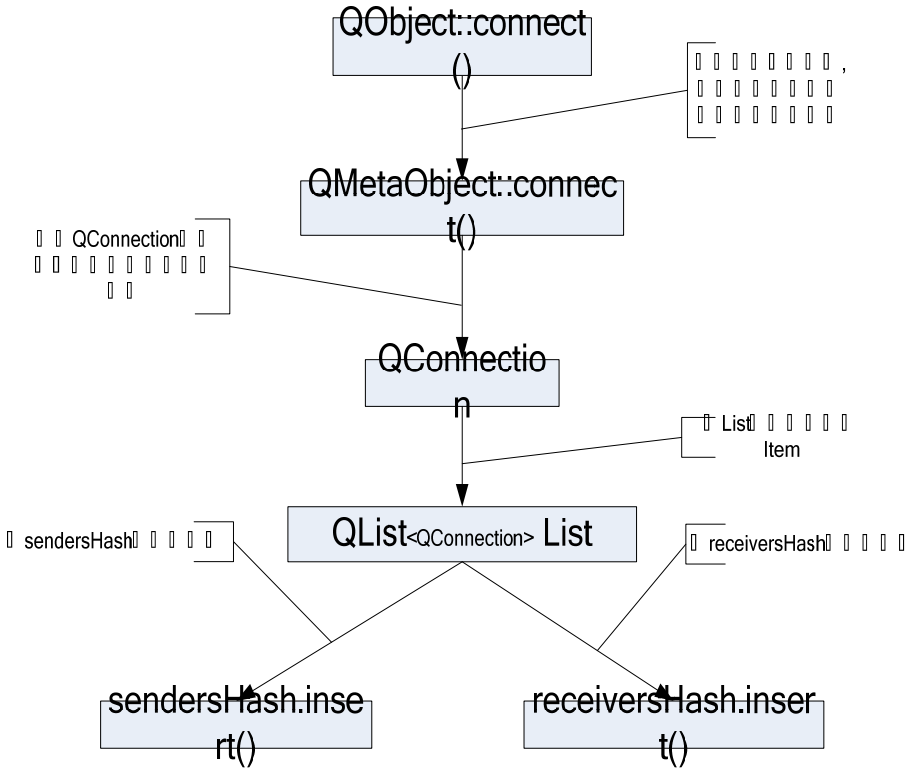


图22. 创建信号与槽连接过程

当创建了一个信号与槽的连接后，在 Qt 中信号与槽的关系是通过一个 QConnection 来保持的，QConnection 结构体的定义如下：

```
struct QConnection
{
    QObject     *sender;        //发送对象
    int         signal;         //信号索引
    QObject     *receiver;      //接收对象
    int         method;         //槽索引
    uint        refCount:30;    //计数
    uint        type:2;         //连接类型，
    int         *types;         //数据类型
};
```

其中，连接类型有三种：0 表示“auto”，当接收者和发送者处于同一个线程时，其行为等价于“direct”，否则其行为等价于“queued” ，默认的连接类型为“auto”；1 表示“direct”，即信号被发射后，槽函数直接被执行；2 表示“queued”，即信号发射后，要进入事件队列中排队，当控制返回到接收者所在的线程时，槽函数被执行。需要注意的是，数据类型“types”表示发射的信号所带参数的类型，如果连接类型为“queued”，必须保证数据类型“types”为 Qt 元对象系统已知的数据类型，因为 Qt 需要将该参数存在一个事件中以放在事件队列中排队，否则可能发生错误。

QConnection 的保存、添加和移除是通过 QConnectionList 类来实现的，QConnectionList 类的定义如下:

```
class QConnectionList
{
public:
    QReadWriteLock                                  lock;
    typedef QMultiHash<const QObject *, int>    Hash;
    //发送对象和接收对象的哈希表
    Hash                                            sendersHash, receiversHash;
    //在 connections 中 QConnection 已经被移去的索引的列表
    QList<int>                                      unusedConnections;
    typedef QList<QConnection>                      List;
    // QConnection 列表
    List                                            connections;
    void remove(QObject *object);
    //向 connections 中添加连接
    void addConnection(QObject *sender, int signal,QObject *receiver,
                                int method,int type = 0, int *types = 0);
    //从 connections 中移去连接
    bool removeConnection(QObject *sender, int signal,
                                QObject *receiver, int method);
};
```

下面为 connect()函数对信号与槽的处理过程：

```
bool QObject::connect(const QObject *sender, const char *signal,
                      const QObject *receiver, const char *method,Qt::ConnectionType type)
{
    {
        const void *cbdata[] = { sender, signal, receiver, method, &type };
        if (QInternal::activateCallbacks(QInternal::ConnectCallback, (void **) cbdata))
            return true;
    }
    #ifndef QT_NO_DEBUG
        bool warnCompat = true;
```

```
        #endif
        if (type == Qt::AutoCompatConnection)                    //判断连接类型
        {
            type = Qt::AutoConnection;
            #ifndef QT_NO_DEBUG
                warnCompat = false;
            #endif
        }
        if (sender == 0 || receiver == 0 || signal == 0 || method == 0)
        {
            qWarning("QObject::connect: Cannot connect %s::%s to %s::%s",
                     sender ? sender->metaObject()->className() : "(null)",
                     (signal && *signal) ? signal+1 : "(null)",
                    receiver ? receiver->metaObject()->className() : "(null)",
                     (method && *method) ? method+1 : "(null)");
            return false;
        }
        QByteArray tmp_signal_name;
        if (!check_signal_macro(sender, signal, "connect", "bind")) //检查信号宏
            return false;
        const QMetaObject *smeta = sender->metaObject();          //获得发送对象的元对象
        ++signal; //skip code
        int signal_index = smeta->indexOfSignal(signal);          //获取信号索引
        if (signal_index < 0)
        {
            // check for normalized signatures
             tmp_signal_name = QMetaObject::normalizedSignature(signal).prepend(*(signal -
            1));
            signal = tmp_signal_name.constData() + 1;
            signal_index = smeta->indexOfSignal(signal);
            if (signal_index < 0)
            {
                err_method_notfound(QSIGNAL_CODE, sender, signal, "connect");
                err_info_about_objects("connect", sender, receiver);
                return false;
            }
        }
        QByteArray tmp_method_name;
        int membcode = method[0] - '0';
        if (!check_method_code(membcode, receiver, method, "connect"))//检查接收函数类型
            return false;
        ++method; // skip code
        const QMetaObject *rmeta = receiver->metaObject();        //获取接收对象的元对象
        int method_index = -1;
        switch (membcode)
{
            case QSLOT_CODE:                                      //接收函数为槽函数
                method_index = rmeta->indexOfSlot(method);    //获取槽索引
                break;
            case QSIGNAL_CODE:                                    //接收函数为信号函数
                method_index = rmeta->indexOfSignal(method); //获取信号索引
                break;
        }
        if (method_index < 0)
{
            // check for normalized methods
```

```
                tmp_method_name = QMetaObject::normalizedSignature(method);
                method = tmp_method_name.constData();
                switch (membcode)
                {
                    case QSLOT_CODE:
                        method_index = rmeta->indexOfSlot(method);
                        break;
                case QSIGNAL_CODE:
                        method_index = rmeta->indexOfSignal(method);
                        break;
                }
            }
            if (method_index < 0)
            {
                err_method_notfound(membcode, receiver, method, "connect");
                err_info_about_objects("connect", sender, receiver);
                return false;
            }
            if (!QMetaObject::checkConnectArgs(signal, method))             //检查参数兼容性
            {
                qWarning("QObject::connect: Incompatible sender/receiver arguments"
                        "\n\t%s::%s --> %s::%s",sender->metaObject()->className(), signal,
                        receiver->metaObject()->className(), method);
                return false;
            }
            int *types = 0;
            if (type == Qt::QueuedConnection                                //连接类型为“queued”
            //检查信号所带参数是否为 Qt 元系统的已知类型
        && !(types = ::queuedConnectionTypes(smeta->method(signal_index).parameterTypes())))
            return false;
#ifndef QT_NO_DEBUG
            {
                QMetaMethod smethod = smeta->method(signal_index);
                QMetaMethod rmethod = rmeta->method(method_index);
                if (warnCompat)
{
                    if(smethod.attributes() & QMetaMethod::Compatibility)
{
                        if (!(rmethod.attributes() & QMetaMethod::Compatibility))
qWarning("QObject::connect: Connecting from COMPAT signal (%s::%s)", smeta->className(),
signal);
}
else
if(rmethod.attributes() & QMetaMethod::Compatibility && membcode != QSIGNAL_CODE)
{
qWarning("QObject::connect: Connecting from %s::%s to COMPAT slot (%s::%s)",
                                smeta->className(), signal, rmeta->className(), method);
                        }
                }
            }
#endif
            QMetaObject::connect(sender, signal_index, receiver, method_index, type, types);
            const_cast<QObject*>(sender)->connectNotify(signal - 1);
            return true;
}
```

下面为元对象系统的 connect()函数的处理过程：

```
bool QMetaObject::connect(const QObject *sender, int signal_index,
```

```
                const QObject *receiver, int method_index, int type, int *types)
{
    QConnectionList *list = ::connectionList();
    if (!list)
        return false;
    QWriteLocker locker(&list->lock);           //加锁
    //向 QConnectionList 中添加连接
    list->addConnection(const_cast<QObject *>(sender), signal_index,
       const_cast<QObject *>(receiver), method_index, type, types);
    return true;
}
```

下面为 QConnectionList 对添加信号与槽链接的处理过程：

```
void QConnectionList::addConnection(QObject *sender, int signal,
     QObject *receiver, int method,int type, int *types)
{
    QConnection c = { sender, signal, receiver, method, 0, 0, types };
    c.type = type; // don't warn on VC++6
    int at = -1;
    //判断是否有可用的“unused”位置可供连接插入
    for (int i = 0; i < unusedConnections.size(); ++i)
    {
        if (!connections.at(unusedConnections.at(i)).refCount)
            {
                // reuse an unused connection
                at = unusedConnections.takeAt(i);
                connections[at] = c;
                break;
            }
     }
     if (at == -1)
    {
        // 添加新连接
        at = connections.size();
        connections << c;
     }
    sendersHash.insert(sender, at);          //向 sendersHash 中加入信息
    receiversHash.insert(receiver, at);      //向 receiversHash 中加入信息
}
```

有时，研发人员可能需要断开信号和槽之间的连接，disconnect()函数提供了这样的功能。下面为元对象的 disconnect()的处理过程：

```
bool QMetaObject::disconnect(const QObject *sender, int signal_index,
                                const QObject *receiver, int method_index)
{
    if (!sender)
        return false;
    QConnectionList *list = ::connectionList();
    if (!list)
        return false;
    QWriteLocker locker(&list->lock);                                //加锁
    return list->removeConnection(const_cast<QObject*>(sender), signal_index,
                 const_cast<QObject*>(receiver), method_index);//移去连接
```

```
}
```

下面为 QConnectionList 类的 removeConnection()函数的处理过程：

```
bool QConnectionList::removeConnection(QObject *sender, int signal,
                                       QObject *receiver, int method)
{
    bool success = false;
    Hash::iterator it = sendersHash.find(sender);
    while (it != sendersHash.end() && it.key() == sender)
    {
         const int at = it.value();              //获取在 connections 中的 at
        QConnection &c = connections[at];        //获取连接
        if (c.receiver&& (signal < 0 || signal == c.signal)
        && (receiver == 0|| (c.receiver == receiver && (method < 0 || method ==
        c.method))))
        {
            if (c.types)
            {
                if (c.types != &DIRECT_CONNECTION_ONLY)
                    qFree(c.types);
                    c.types = 0;
            }
            it = sendersHash.erase(it);              //擦去发送对象
            Hash::iterator x = receiversHash.find(c.receiver);
            const Hash::iterator xend = receiversHash.end();
            while (x != xend && x.key() == c.receiver)
            {
                if (x.value() == at)
                {
                     x = receiversHash.erase(x);     //擦去接收对象
                    break;
                }
                else
                {
                    ++x;
                 }
            }
            uint refCount = c.refCount;
            memset(&c, 0, sizeof(c));                //c 清零
            c.refCount = refCount;
            unusedConnections << at;    //将索引添加到 unusedConnections 中
            success = true;
        }
        else
        {
             ++it;
         }
    }
    return success;
}
```

### 4.2.2 动态属性机制

Qt 提供了一个与一些编译器厂商提供的属性系统类似的复杂的属性系统，但作为一个与编译器和平台独立的库，Qt 不能依赖非标准编译器提供的类似于__prpterty 或[proterty]的特性，它是基于元对象系统来在所支持的平台上提供属性系统的。

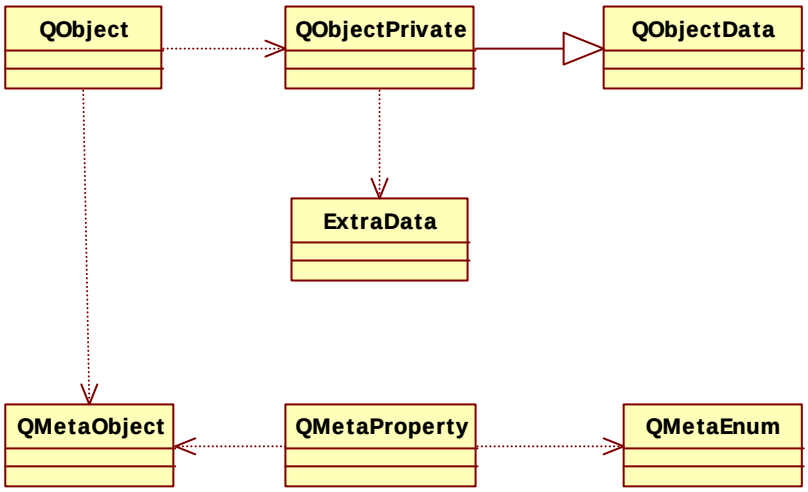


图23. 属性系统依赖关系图

Qt 通过在 QObjectPrivate 类的 ExtraData 类型的 extraData 数据成员中提供了对属性的存储，ExtraData 的声明如下：

```
struct ExtraData
{
    #ifndef QT_NO_USERDATA
        QVector<QObjectUserData *> userData;
    #endif
    QList<QByteArray> propertyNames;
    QList<QVariant> propertyValues;
 };
```

Qt 通过 Q_PROPERTY()宏来在继承于 QObject 的类的声明中声明一个属性，通过 Q_PROPERTY()宏定义的属性为动态属性，动态属性即在实时运行中能被添加或移去的属性。下面是 Q_PROPERTY()的语法：

```
Q_PROPERTY(type name
    READ getFunction
    [WRITE setFunction]
    [RESET resetFunction]
    [DESIGNABLE bool]
    [SCRIPTABLE bool]
    [STORED bool])
```

从 Q_PROPERTY()的语法可以看的出来，属性表面上和一个数据成员有点类似，但属性有如下几个特征和数据成员不同：

- 一个读取函数，这是必须的。
- 一个写入函数，这是可选的，如 QWidget::rect 就没有写入函数。
- 一个“stored”属性，大多数属性都能被存储，但虚属性不能。“stored”属性以为着该属性具有持久性。
- 一个重设函数，能够将属性重设为默认值。
- 一个“designable”属性，意味着该属性在 GUI 构造器中可显。

属性可以在不知道所用类的任何内容的情况下，通过 Qobject 类的泛型函数如 QObject::setProperty()和 QObject::property()被读写，通过调用 QMetaObject::propertyCount()函数可以返回所有可用属性的数量。

```
bool QObject::setProperty(const char *name, const QVariant &value)
{
    Q_D(QObject);
    const QMetaObject* meta = metaObject();//获取对象的元对象系统
    if (!name || !meta)
        return false;
    int id = meta->indexOfProperty(name);   //获取属性的索引
    if (id < 0)
    {
       if (!d->extraData)
           d->extraData = new QObjectPrivate::ExtraData;
       const int idx = d->extraData->propertyNames.indexOf(name);
        if (value.isNull())
        {
          if (idx == -1)
              return false;
          d->extraData->propertyNames.removeAt(idx);
          d->extraData->propertyValues.removeAt(idx);
       }
        else
        {
          if (idx == -1)
            {
                d->extraData->propertyNames.append(name);
                d->extraData->propertyValues.append(value);
            }
            else
            {
                d->extraData->propertyValues[idx] = value;
            }
        }
        QDynamicPropertyChangeEvent ev(name);
        QCoreApplication::sendEvent(this, &ev);
        return false;
    }
   QMetaProperty p = meta->property(id);
#ifndef QT_NO_DEBUG
   if (!p.isWritable())
     qWarning("%s::setProperty: Property \"%s\" invalid,"
" read-only or does not exist", metaObject()->className(), name);
#endif
    return p.write(this, value);
}
```

### 4.2.3 软件本地化

为了能够让不同的国家的用户都能够使用研发的软件，必须对所研发的软件进行软件的本地化。在 Qt 中，Qt 对 Unicode 4.0 中不需要特殊处理的所有字符提供了支持，能够支持英

文等拉丁字符，中文、日文、韩文等象形字符，阿拉伯字符，希腊字符，俄文等斯拉夫字符，希伯来字符等。由于都是基于Unicode 4.0的，文本处理方式是类似的。所以Qt软件的本地化可以通过Qt的元对象系统提供的统一方式来实现。

Qt通过Qt的文本引擎对文本的写入进行了完全的封装。除了当Qt需要处理阿拉伯文或者希伯来文（自右向左书写）或者需要获得字符宽度时。写入系统对研发人员来说是都透明的。通过QFontMetrics::width()可以获得字符或字符串的宽度。

1. 预处理

对于引用的文本信息，为了能够进行本地化，需要对文本信息通过 QObject:: tr()进行封装，当所属的类并非继承于 QObject 时，可以直接利用 QCoreApplication::translate()来进行封装。如果需要对函数外面的文本信息进行本地化，可以利用 QT_TR_NOOP() 和 QT_TRANSLATE_NOOP()来对文本信息进行封装。

通过在编译时选上QT_NO_CAST_ASCII宏，可以使const char * 向Qstring的自动类型转换失效，但这样以来，就会使所写的程序显得笨重。

如果你引用的文本信息包含了Latin1以外的字符，可以考虑利用QObject::trUtf8()来代替QObject::tr()，因为 QObject::tr()依赖于 QTextCodec::codecForTr(),这就使 QObject::tr()比QObject::trUtf8()更加脆弱。

当所引用的文本信息中采用了快捷键时，翻译系统就无法将快捷键信息提取出来，为了在本地化软件中使用同样的快捷键，Qt提供了如下的正确方法：

```
exitAct=new QAction(tr(“&Exit”),this);
exitAct->setShortcut(tr(“Ctrl+E”));
```

但当字符串是动态生成的时，简单的利用tr()来封装字符串进行本地化变得不可行，可行的方式有两种：

- 利用qt提供的arg()。
- 对字符串进行分解，调用append()函数来追加字符串。

2. 生成翻译文件

翻译文件包含了应用程序中所有用户可见的文本和快捷键，翻译文件的创建方法如下：

- 在moduleName.pro中添加相应的选项：TRANSLATIONS+=moduleName.ts。
- 运行lupdate moduleName.pro，将用户可见的文本信息和加速键信息从代码中提取出来，创建或更新moduleName.ts文件。
- 用Qt Linguist打开moduleName.ts文件，在相应的位置输入本地化翻译字符串。
- 进行保存。然后运行“File”菜单下的“Release”或“Release As”子菜单，或在shell中运行lrelease moduleName.ts，得到相应的moduleName.qm文件。

需要说明的是，生成的qm文件名应该与应用程序名保持一致。 在翻译字符串时如果需要换行，可以通过按回车键实现。

3. 导出与安装

在实际的商用版本中，往往需要根据用户的习惯将软件切换到其所需要的语言，因此就需要设置特定的目录来存储特定语言的翻译文件(qm 文件)，以便在设置所用语言时能够导出不同的qm文件。

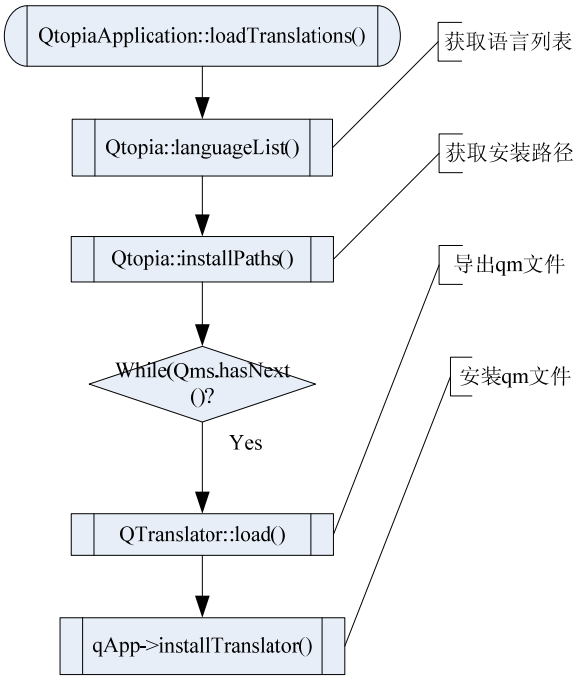


图24. qm 文件的导出与安装

下面是 loadTranslations()函数导出 qm 文件的处理过程：

```
void QtopiaApplication::loadTranslations( const QStringList& qms )
{
    QStringList qmList( qms );
    QStringList langs = Qtopia::languageList();        //获取语言信息
    QStringList qpepaths = Qtopia::installPaths();
    for (QStringList::ConstIterator qit = qpepaths.begin(); qit!=qpepaths.end(); ++qit)
    {
        for (QStringList::ConstIterator it = langs.begin(); it!=langs.end(); ++it)
        {
            QString lang = *it;
            //Optimisation: Assumption the source is en_US => don't load translations
            if ( lang == QLatin1String("en_US")
            {
                qLog(I18n) << "Loading of en_US requested, skipping loading of
                translation files";
                return;
            }
            QTranslator * trans;
            QString tfn;
            QMutableStringListIterator qmit( qmList );
            while ( qmit.hasNext() )
            {
                trans = new QTranslator(qApp);
                tfn = *qit + QLatin1String("i18n/") + lang + QLatin1String("/") +
                qmit.next() + QLatin1String(".qm");    //获取 qm 文件信息
                qLog(I18n) << "Loading" << tfn;
                if ( trans->load( tfn ))
                {
                    qApp->installTranslator( trans );   //按照 qm 文件
                    qmit.remove();
                }
                else
```

```
                    {
                        qLog(I18n) << "Cannot load " << tfn;
                        delete trans;
                    }
                }
            }
        }
    }
```

需要说明的是 qApp 是 QCoreApplication 的单子实例，下面是 qApp 的定义：

```
#define qApp (static_cast<QApplication *>(QCoreApplication::instance()))
```

下面是 installTranslator 函数对 qm 文件的安装处理过程：

```
void QCoreApplication::installTranslator(QTranslator *translationFile)
{
    if (!translationFile)
        return;
    if (!QCoreApplicationPrivate::checkInstance("installTranslator"))
        return;
    QCoreApplicationPrivate *d = self->d_func();
    d->translators.prepend(translationFile);
#ifndef QT_NO_TRANSLATION_BUILDER
    if (translationFile->isEmpty())
        return;
#endif
    QEvent ev(QEvent::LanguageChange);
    QCoreApplication::sendEvent(self, &ev);          //发送语言变更事件
}
```

当用户变更显示语言时，会在系统信道上发出 language(QString)消息，其中参数 QString 表示所采用的语言，服务器端收到该消息后，会对环境变量 LANG 重新进行设置，然后重启系统以响应用户需求。下面时服务器端的处理过程：

```
void QtopiaApplication::systemMessage( const QString &msg, const QByteArray &data)
{
    ……
    else if ( msg == QLatin1String("language(QString)") )
    {
        if ( type() == GuiServer ) {
            QString l;
            stream >> l;
            l += QLatin1String(".UTF-8");
            QString cl = getenv("LANG");
            if ( cl != l )
            {
                if ( l.isNull() )
                    unsetenv( "LANG" );
                else
                    setenv( "LANG", l.toLatin1(), 1 );
                restart();
            }
        }
    ……
}
```

## 4.3 事件处理机制

Qt 的事件处理机制主要是基于 QEvent 类来实现的，QEvent 类是其他事件类的基类。当

一个事件产生时，Qt就会构造一个QEvent子类的实例来表述该事件，然后将该事件发送到相应的对象上进行处理。

Qt 的主事件循环能够从事件队列中获取本地窗口系统事件，然后判断事件类型，并将事件分发给特定的接收对象。主事件循环通过调用 QCoreApplication::exec()启动，随着QCoreApplication::exit()结束，本地的事件循环可用利用 QEventLoop 构建。

作为事件分发器的 QAbstractEventDispatcher 管理着 Qt 的事件队列，事件分发器从窗口系统或其他事件源接收事件，然后将他们发送给 QCoreApplication 或 QApplication 的实例进行处理或继续分发。QAbstractEventDispatcher 为事件分发提供了良好的保护措施。

一般来说，事件是由触发当前的窗口系统产生的，但也可以通过使用QCoreApplication::sendEvent()和 QCoreApplication::postEvent()来手工产生事件。需要说明的是 QCoreApplication::sendEvent()会立即发送事件，QCoreApplication::postEvent()则会将事件放在事件队列中分发。如果需要在一个对象初始化完成之际就开始处理某种事件，可以将事件通过 QCoreApplication::postEvent()发送。

通过接收对象的 event()函数可以返回由接收对象的事件句柄返回的事件，对于某些特定类型的事件如鼠标（触笔）和键盘事件，如果接收对象不能处理，事件将会被传播到接收对象的父对象。需要说明的是接收对象的 event()函数并不直接处理事件，而是根据被分发过来的事件的类型调用相应的事件句柄进行处理。

下面是 5 种自定义事件处理的方式：

- 重载 paintEvent()、 mousePressEvent()等事件，这是最普通、最简单、最有效的方法。
- 重载 QCoreApplication::notify()函数，能够对事件处理进行完全控制。但是这样以来，每次只能有一个子类被激活。
- 在QCoreApplication::instance()也即在qApp上安装事件过滤器，这样就可处理所有部件（widget）上的所有事件，这和重载 QCoreApplication::notify()函数一样有效。
- 重载 QObject::event()函数，可是研发人员在特定部件的事件过滤器前处理 Tab 事件。
- 在特定对象上安装事件过滤器，这样就可以处理除 Tab 和 Shift-Tab 以外的所有事件。

### 4.3.1 事件过滤器

当一个对象上安装了事件过滤器时，所有的发送到该对象的事件都会被事件过滤器截获，如果在同一个对象上安装了多个事件过滤器，那么随后的事件过滤器将会激活第一个事件过滤器。需要说明的是，如果删除了安装了事件过滤器的对象，一定要确保返回真，否则Qt 仍会发送事件到该对象，这样就会引起程序崩溃。

```
void QObject::installEventFilter(QObject *obj)
{
    Q_D(QObject);
    if (!obj)
        return;
    QWriteLocker locker(QObjectPrivate::readWriteLock());
    // clean up unused items in the list
    d->eventFilters.removeAll((QObject*)0);
    d->eventFilters.removeAll(obj);
    d->eventFilters.prepend(obj);
}
```

在需要时为对象安装事件过滤器在程序研发是个很重要的技巧，可以极大的简化程序设计，但如果对事件处理存在疏漏，往往也会出现意想不到的 Bug。尤其是在含有 List 选项的界面对 Tab 键等的处理，如果忽略处理，很容易造成焦点丢失等类似的 Bug，这是十分需要注意的地方。另一个需要注意的地方是，在事件过滤器中对事件进行处理后一定要返回TURE，而且如果不手工添加处理代码，事件过滤器不会自动区分事件的类型，这样在进行按键处理时很容易出现意想不到的问题。

下面是一个选项列表界面的事件过滤器的处理过程：

```
bool SettingOptions::eventFilter(QObject *o ,QEvent *e)
{
    int index=pOptionsListBox->currentItem();
    if(e->type==QEvent::KeyPress)                    //对按键事件类型的判断
    {
    QKeyEvent *k = (QKeyEvent*)e;
    switch ( k->key() )
    {
        case Key_Context3:
        case Key_Select:
            //do sth for this event
            return TRUE;                             //对事件处理完成后返回 TURE
        case Key_Context4:
            //do sth for this event
            return TRUE;
        case Key_Left:
        case Key_Up:
                if(index==0)
                  {
                    pOptionsListBox->setCurrentItem(index);
                  }
                else
                  {
                  pOptionsListBox->setCurrentItem(index-1);
                  }
        return TRUE;
        case Key_Right:
        case Key_Down:
          index=pOptionsListBox->currentItem();
                if(index==((int)pOptionsListBox->count()-1))
                  {
                    pOptionsListBox->setCurrentItem(index);
                  }
                   else
                  {
                  pOptionsListBox->setCurrentItem(index+1);
                  }
        return TRUE;
        case Key_Tab:
        pOptionsListBox->setFocus();                 //Tab 键的处理
        return TRUE;
        default:
            break;
    }
}
     return QWidget::eventFilter( o, e );;
}
```

### 4.3.2 客户端事件处理

在Qt中，events即对象，当一个事件发生时，Qt通过构造一个适当的QEvent子类来创建一个事件对象来表述事件，并将它通过调用特定对象的实例的event()函数将事件传递给它。所以向一个对象传递事件的时候，经过的第一个函数就是event()。

event()函数并不处理事件，根据 event 的类型，它调用相应的 event handler 来进行处理。

当需要对传递给特定对象的事件进行特定处理时，就需要对该对象安装 eventFilter，在事件被传递给 event()前，对事件进行处理。

当对象需要创建或调用事件时，可以利用sendEvent()和postEvent()，前者会立即发送事件，后者会将事件添加到事件队列中，当主事件循环再次运行时，派发出去。

```
bool QWidget::event(QEvent *event)
{
    Q_D(QWidget);
    // ignore mouse events when disabled
    if (!isEnabled()) {
        switch(event->type()) {
        case QEvent::TabletPress:
        case QEvent::TabletRelease:
        case QEvent::TabletMove:
        case QEvent::MouseButtonPress:
        case QEvent::MouseButtonRelease:
        case QEvent::MouseButtonDblClick:
        case QEvent::MouseMove:
        case QEvent::ContextMenu:
#ifndef QT_NO_WHEELEVENT
        case QEvent::Wheel:
#endif
            return false;
        default:
            break;
        }
    }
    switch (event->type()) {
    case QEvent::MouseMove:
        mouseMoveEvent((QMouseEvent*)event);
        break;
    case QEvent::MouseButtonPress:
#if 0
        resetInputContext();
#endif
        mousePressEvent((QMouseEvent*)event);
        break;
    case QEvent::MouseButtonRelease:
        mouseReleaseEvent((QMouseEvent*)event);
        break;
    case QEvent::MouseButtonDblClick:
        mouseDoubleClickEvent((QMouseEvent*)event);
        break;
#ifndef QT_NO_WHEELEVENT
    case QEvent::Wheel:
        wheelEvent((QWheelEvent*)event);
        break;
#endif
#ifndef QT_NO_TABLETEVENT
    case QEvent::TabletMove:
    case QEvent::TabletPress:
    case QEvent::TabletRelease:
        tabletEvent((QTabletEvent*)event);
        break;
#endif
#ifdef QT3_SUPPORT
    case QEvent::Accel:
        event->ignore();
```

```
            return false;
#endif
        case QEvent::KeyPress: {                            //按键事件
            QKeyEvent *k = (QKeyEvent *)event;
            bool res = false;
            if (!(k->modifiers() & (Qt::ControlModifier | Qt::AltModifier))) {
                if (k->key() == Qt::Key_Backtab
//按 tab 键事件
                        || (k->key() == Qt::Key_Tab && (k->modifiers() & Qt::ShiftModifier)))
                    res = focusNextPrevChild(false);
                else if (k->key() == Qt::Key_Tab)
                    res = focusNextPrevChild(true);
                if (res)
                    break;
            }
            keyPressEvent(k);
#ifdef QT_KEYPAD_NAVIGATION
            if (!k->isAccepted() && QApplication::keypadNavigationEnabled()
                && !(k->modifiers() & (Qt::ControlModifier | Qt::AltModifier |
Qt::ShiftModifier))) {
                if (k->key() == Qt::Key_Up)
                    res = focusNextPrevChild(false);
                else if (k->key() == Qt::Key_Down)
                    res = focusNextPrevChild(true);
                if (res) {
                    k->accept();
                    break;
                }
            }
#endif
#ifndef QT_NO_WHATSTHIS
            if (!k->isAccepted()
                && k->modifiers() & Qt::ShiftModifier && k->key() == Qt::Key_F1
                && d->whatsThis.size()) {

QWhatsThis::showText(mapToGlobal(inputMethodQuery(Qt::ImMicroFocus).toRect().center()
), d->whatsThis, this);
                k->accept();
            }
#endif
        }
            break;
        case QEvent::KeyRelease:
            keyReleaseEvent((QKeyEvent*)event);
            // fall through
        case QEvent::ShortcutOverride:
            break;
        case QEvent::InputMethod:
            inputMethodEvent((QInputMethodEvent *) event);
            break;
        case QEvent::PolishRequest:
            ensurePolished();
            break;
        case QEvent::Polish: {
            style()->polish(this);
            setAttribute(Qt::WA_WState_Polished);
            if                                              (!testAttribute(Qt::WA_SetFont)
```

```
&& !QApplication::font(this).isCopyOf(QApplication::font()))
                d->resolveFont();
            if (!QApplication::palette(this).isCopyOf(QApplication::palette()))
                d->resolvePalette();
#ifdef QT3_SUPPORT
            if(d->sendChildEvents)
                QApplication::sendPostedEvents(this, QEvent::ChildInserted);
#endif
        }
            break;
        case QEvent::ApplicationWindowIconChange:
            if (isWindow() && !testAttribute(Qt::WA_SetWindowIcon))
                d->setWindowIcon_sys();
            break;
        case QEvent::FocusIn:
            focusInEvent((QFocusEvent*)event);
            break;
        case QEvent::FocusOut:
            focusOutEvent((QFocusEvent*)event);
            break;
        case QEvent::Enter:
#ifndef QT_NO_STATUSTIP
            if (d->statusTip.size()) {
                QStatusTipEvent tip(d->statusTip);
                QApplication::sendEvent(const_cast<QWidget *>(this), &tip);
            }
#endif
            enterEvent(event);
            break;
        case QEvent::Leave:
#ifndef QT_NO_STATUSTIP
            if (d->statusTip.size()) {
                QString empty;
                QStatusTipEvent tip(empty);
                QApplication::sendEvent(const_cast<QWidget *>(this), &tip);
            }
#endif
            leaveEvent(event);
            break;
        case QEvent::HoverEnter:
        case QEvent::HoverLeave:
            update();
            break;
        case QEvent::Paint:
            // At this point the event has to be delivered, regardless
            // whether the widget isVisible() or not because it
            // already went through the filters
            paintEvent((QPaintEvent*)event);
            break;
        case QEvent::Move:
            moveEvent((QMoveEvent*)event);
            break;
        case QEvent::Resize:
            resizeEvent((QResizeEvent*)event);
            break;
        case QEvent::Close:
            closeEvent((QCloseEvent *)event);
```

```
            break;
        case QEvent::ContextMenu:
            switch (data->context_menu_policy) {
            case Qt::PreventContextMenu:
                break;
            case Qt::DefaultContextMenu:
                contextMenuEvent(static_cast<QContextMenuEvent *>(event));
                break;
            case Qt::CustomContextMenu:
emit customContextMenuRequested(static_cast<QContextMenuEvent *>(event)->pos());
                break;
#ifndef QT_NO_MENU
            case Qt::ActionsContextMenu:
                if (d->actions.count()) {
QMenu::exec(d->actions, static_cast<QContextMenuEvent *>(event)->globalPos());
                    break;
                }
#endif
            default:
                event->ignore();
                break;
            }
            break;
#ifndef QT_NO_DRAGANDDROP
        case QEvent::Drop:
            dropEvent((QDropEvent*) event);
            break;
        case QEvent::DragEnter:
            dragEnterEvent((QDragEnterEvent*) event);
            break;
        case QEvent::DragMove:
            dragMoveEvent((QDragMoveEvent*) event);
            break;
        case QEvent::DragLeave:
            dragLeaveEvent((QDragLeaveEvent*) event);
            break;
#endif
        case QEvent::Show:
            showEvent((QShowEvent*) event);
            break;
        case QEvent::Hide:
            hideEvent((QHideEvent*) event);
            break;
        case QEvent::ShowWindowRequest:
            if (!isHidden())
                d->show_sys();
            break;
        case QEvent::ApplicationFontChange:
            d->resolveFont();
            break;
        case QEvent::ApplicationPaletteChange:
            if (!(windowType() == Qt::Desktop))
                d->resolvePalette();
            break;
        case QEvent::ToolBarChange:
        case QEvent::ActivationChange:
        case QEvent::EnabledChange:
```

```
        case QEvent::FontChange:
        case QEvent::StyleChange:
        case QEvent::PaletteChange:
        case QEvent::WindowTitleChange:
        case QEvent::IconTextChange:
        case QEvent::ModifiedChange:
        case QEvent::MouseTrackingChange:
        case QEvent::ParentChange:
        case QEvent::WindowStateChange:
            changeEvent(event);
            break;
        case QEvent::WindowActivate:
        case QEvent::WindowDeactivate: {
#ifdef QT3_SUPPORT
            windowActivationChange(event->type() != QEvent::WindowActivate);
#endif
            if (isVisible() && !palette().isEqual(QPalette::Active, QPalette::Inactive))
                update();
            QList<QObject*> childList = d->children;
            for (int i = 0; i < childList.size(); ++i) {
                QWidget *w = qobject_cast<QWidget *>(childList.at(i));
                if (w && w->isVisible() && !w->isWindow())
                    QApplication::sendEvent(w, event);
            }
            break; }
        case QEvent::LanguageChange:
#ifdef QT3_SUPPORT
            languageChange();
#endif
            // fall through
        case QEvent::LocaleChange:
            changeEvent(event);
            {
                QList<QObject*> childList = d->children;
                for (int i = 0; i < childList.size(); ++i) {
                    QObject *o = childList.at(i);
                    QApplication::sendEvent(o, event);
                }
            }
            update();
            break;
        case QEvent::ApplicationLayoutDirectionChange:
            d->resolveLayoutDirection();
            break;
        case QEvent::LayoutDirectionChange:
            if (d->layout)
                d->layout->invalidate();
            update();
            changeEvent(event);
            break;
#if defined(Q_WS_X11) || defined(Q_WS_QWS)
        case QEvent::UpdateRequest: {
            extern void qt_syncBackingStore(QWidget *widget);
            qt_syncBackingStore(this);
            break; }
#endif
        case QEvent::UpdateLater:
```

```
            update(static_cast<QUpdateLaterEvent*>(event)->region());
            break;
        case QEvent::WindowBlocked:
        case QEvent::WindowUnblocked:
            {
                QList<QObject*> childList = d->children;
                for (int i = 0; i < childList.size(); ++i) {
                    QObject *o = childList.at(i);
                    if (o != qApp->activeModalWidget()) {
                        if (qobject_cast<QWidget *>(o) && static_cast<QWidget
*>(o)->isWindow()) {
                            // do not forward the event to child windows,
                            // QApplication does this for us
                            continue;
                        }
                        QApplication::sendEvent(o, event);
                    }
                }
            }
            break;
#ifndef QT_NO_TOOLTIP
        case QEvent::ToolTip:
            if (d->toolTip.size()
                && (window()->testAttribute(Qt::WA_AlwaysShowToolTips) ||
isActiveWindow())) {
                QToolTip::showText(static_cast<QHelpEvent*>(event)->globalPos(),
d->toolTip, this);
            } else {
                event->ignore();
            }
            break;
#endif
#ifndef QT_NO_WHATSTHIS
        case QEvent::WhatsThis:
            if (d->whatsThis.size())
                QWhatsThis::showText(static_cast<QHelpEvent *>(event)->globalPos(),
d->whatsThis, this);
            else
                event->ignore();
            break;
        case QEvent::QueryWhatsThis:
            if (d->whatsThis.isEmpty())
                event->ignore();
            break;
#endif
#ifndef QT_NO_ACCESSIBILITY
        case QEvent::AccessibilityDescription:
        case QEvent::AccessibilityHelp: {
            QAccessibleEvent *ev = static_cast<QAccessibleEvent *>(event);
            if (ev->child())
                return false;
            switch (ev->type()) {
#ifndef QT_NO_TOOLTIP
            case QEvent::AccessibilityDescription:
                ev->setValue(d->toolTip);
                break;
#endif
```

```
#ifndef QT_NO_WHATSTHIS
            case QEvent::AccessibilityHelp:
                ev->setValue(d->whatsThis);
                break;
#endif
            default:
                return false;
            }
            break; }
#endif
        case QEvent::EmbeddingControl:
            d->topData()->frameStrut.setCoords(0 ,0, 0, 0);
            data->fstrut_dirty = false;
            break;
#ifndef QT_NO_ACTION
        case QEvent::ActionAdded:
        case QEvent::ActionRemoved:
        case QEvent::ActionChanged:
            actionEvent((QActionEvent*)event);
            break;
#endif
        case QEvent::KeyboardLayoutChange:
            {
                changeEvent(event);
                // inform children of the change
                QList<QObject*> childList = d->children;
                for (int i = 0; i < childList.size(); ++i) {
                    QWidget *w = qobject_cast<QWidget *>(childList.at(i));
                    if (w && w->isVisible() && !w->isWindow())
                        QApplication::sendEvent(w, event);
                }
                break;
            }

        default:
            return QObject::event(event);
        }
        return true;
}
```

### 4.3.3 服务器端事件处理

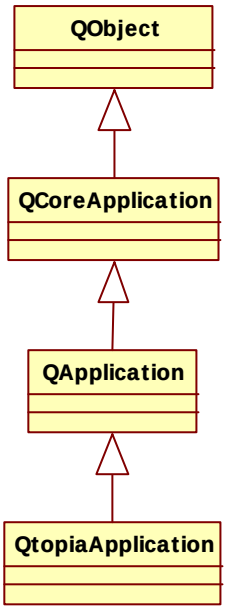


图25. 服务器端的继承关系

在启动事件处理时，有必要调用 QCoreApplication::exec()，这样主事件循环就可以接收到窗口系统的事件并将事件分发给相应的应用部件。下面是 QCoreApplication::exec()的处理过程：

```
int QCoreApplication::exec()
{
    if (!QCoreApplicationPrivate::checkInstance("exec"))
        return -1;
    QThreadData *threadData = self->d_func()->threadData;
    if (threadData != QThreadData::current())
    {
        qWarning("%s::exec: Must be called from the main thread",
        self->metaObject()->className());
        return -1;
    }
    if (!threadData->eventLoops.isEmpty())
    {
        qWarning("QCoreApplication::exec: The event loop is already running");
        return -1;
    }
    threadData->quitNow = false;
    QEventLoop eventLoop;
    self->d_func()->in_exec = true;
    int returnCode = eventLoop.exec();
    threadData->quitNow = false;
    if (self)
    {
        self->d_func()->in_exec = false;
        emit self->aboutToQuit();
        sendPostedEvents(0, QEvent::DeferredDelete);
    }
    return returnCode;
}
```

下面是 QApplication 的 notify()处理过程：

```
bool QApplication::notify(QObject *receiver, QEvent *e)
```

```
{
    Q_D(QApplication);
    // no events are delivered after ~QCoreApplication() has started
    if (QApplicationPrivate::is_app_closing)
        return true;
    if (receiver == 0)
{                                   // serious error
        qWarning("QApplication::notify: Unexpected null receiver");
        return true;
    }
    d->checkReceiverThread(receiver);
#ifdef QT3_SUPPORT
if (e->type() == QEvent::ChildRemoved && receiver->d_func()->postedChildInsertedEvents)
d->removePostedChildInsertedEvents(receiver, static_cast<QChildEvent *>(e)->child());
#endif // QT3_SUPPORT
    // capture the current mouse/keyboard state
    if(e->spontaneous()) {
        if(e->type() == QEvent::KeyPress || e->type() == QEvent::KeyRelease) {
            Qt::KeyboardModifier modif = Qt::NoModifier;
            switch(static_cast<QKeyEvent*>(e)->key()) {
            case Qt::Key_Shift:
                modif = Qt::ShiftModifier;
                break;
            case Qt::Key_Control:
                modif = Qt::ControlModifier;
                break;
            case Qt::Key_Meta:
                modif = Qt::MetaModifier;
                break;
            case Qt::Key_Alt:
                modif = Qt::AltModifier;
                break;
            case Qt::Key_NumLock:
                modif = Qt::KeypadModifier;
                break;
            default:
                break;
            }
            if(modif != Qt::NoModifier) {
                if(e->type() == QEvent::KeyPress)
                    QApplicationPrivate::modifier_buttons |= modif;
                else
                    QApplicationPrivate::modifier_buttons &= ~modif;
            }
        } else if(e->type() == QEvent::MouseButtonPress
                    || e->type() == QEvent::MouseButtonRelease) {
            QMouseEvent *me = static_cast<QMouseEvent*>(e);
            if(me->type() == QEvent::MouseButtonPress)
                QApplicationPrivate::mouse_buttons |= me->button();
            else
                QApplicationPrivate::mouse_buttons &= ~me->button();
        }
    }
    // User input and window activation makes tooltips sleep
    switch (e->type()) {
    case QEvent::ActivationChange:
    case QEvent::KeyPress:
```

```
        case QEvent::KeyRelease:
        case QEvent::FocusOut:
        case QEvent::FocusIn:
        case QEvent::MouseButtonPress:
        case QEvent::MouseButtonRelease:
        case QEvent::MouseButtonDblClick:
            d->toolTipWakeUp.stop();
            d->toolTipFallAsleep.stop();
            break;
        default:
            break;
        }
        bool res = false;
        if (!receiver->isWidgetType()) {
            res = d->notify_helper(receiver, e);
        } else switch (e->type()) {
#if defined QT3_SUPPORT && !defined(QT_NO_SHORTCUT)
        case QEvent::Accel:
            {
                if (d->use_compat()) {
                    QKeyEvent* key = static_cast<QKeyEvent*>(e);
                    res = d->notify_helper(receiver, e);
                    if (!res && !key->isAccepted())
res = d->qt_dispatchAccelEvent(static_cast<QWidget *>(receiver), key);
                        // next lines are for compatibility with Qt <= 3.0.x: old
                        // QAccel was listening on toplevel widgets
if (!res && !key->isAccepted() && !static_cast<QWidget *>(receiver)->isWindow())
res = d->notify_helper(static_cast<QWidget *>(receiver)->window(),e);
                }
                break;
            }
#endif //QT3_SUPPORT && !QT_NO_SHORTCUT
        case QEvent::ShortcutOverride:
        case QEvent::KeyPress:
        case QEvent::KeyRelease:
            {
                if (!receiver->isWidgetType()) {
                    res = d->notify_helper(receiver, e);
                    break;
                }
                QWidget* w = static_cast<QWidget*>(receiver);
                QKeyEvent* key = static_cast<QKeyEvent*>(e);
#if defined QT3_SUPPORT && !defined(QT_NO_SHORTCUT)
                if (d->use_compat() && d->qt_tryComposeUnicode(w, key))
                    break;
#endif
                if (key->type()==QEvent::KeyPress) {
#ifndef QT_NO_SHORTCUT
                    // Try looking for a Shortcut before sending key events
                    if (res = qApp->d_func()->shortcutMap.tryShortcutEvent(w, key))
                        return res;
#endif
                    qt_in_tab_key_event = (key->key() == Qt::Key_Backtab
                                           || key->key() == Qt::Key_Tab
                                           || key->key() == Qt::Key_Left
                                           || key->key() == Qt::Key_Up
                                           || key->key() == Qt::Key_Right
```

```
                                                || key->key() == Qt::Key_Down);
                    }
                    bool def = key->isAccepted();
                    while (w) {
                        if (def)
                            key->accept();
                        else
                            key->ignore();
                        res = d->notify_helper(w, e);
                        if ((res && key->isAccepted()) || w->isWindow() || !w->parentWidget())
                            break;
                        w = w->parentWidget();
                    }
                    qt_in_tab_key_event = false;
                }
                break;
            case QEvent::MouseButtonPress:
            case QEvent::MouseButtonRelease:
            case QEvent::MouseButtonDblClick:
            case QEvent::MouseMove:
                {
                    QWidget* w = static_cast<QWidget *>(receiver);

                    QMouseEvent* mouse = static_cast<QMouseEvent*>(e);
                    QPoint relpos = mouse->pos();

                    if (e->spontaneous()) {

                        if (e->type() == QEvent::MouseButtonPress) {
                            QWidget *fw = w;
                            while (fw) {
                                if (fw->isEnabled() && (fw->focusPolicy() & Qt::ClickFocus)) {
                                    fw->setFocus(Qt::MouseFocusReason);
                                    break;
                                }
                                if (fw->isWindow())
                                    break;
                                fw = fw->parentWidget();
                            }
                        }
                        if (e->type() == QEvent::MouseMove && mouse->buttons() == 0) {
                            d->toolTipWidget = w;
                            d->toolTipPos = relpos;
                            d->toolTipGlobalPos = mouse->globalPos();
                           d->toolTipWakeUp.start(d->toolTipFallAsleep.isActive()?20:700, this);
                        }
                    }
                    bool eventAccepted = mouse->isAccepted();
                    QPointer<QWidget> pw = w;
                    while (w) {
    QMouseEvent me(mouse->type(), relpos, mouse->globalPos(), mouse->button(),
mouse->buttons(),
                                                mouse->modifiers());
                        me.spont = mouse->spontaneous();
                        // throw away any mouse-tracking-only mouse events
                        if (!w->hasMouseTracking()
      && mouse->type() == QEvent::MouseMove && mouse->buttons() == 0)
```

```
{
                        QReadWriteLock *lock = QObjectPrivate::readWriteLock();
                        if (lock)
                            lock->lockForRead();
                        for (int i = 0; i < d->eventFilters.size(); ++i) {
                            register QObject *obj = d->eventFilters.at(i);
                            if (lock)
                                lock->unlock();
                            if (obj && obj->eventFilter(w, w == receiver ? mouse : &me)) {
                                lock = 0;
                                break;
                            }
                            if (lock)
                                lock->lockForRead();
                        }
                        if (lock)
                            lock->unlock();
                        res = true;
                    } else {
                        w->setAttribute(Qt::WA_NoMouseReplay, false);
                        res = d->notify_helper(w, w == receiver ? mouse : &me);
                        e->spont = false;
                    }
                    eventAccepted = (w == receiver ? mouse : &me)->isAccepted();
                    if (res && eventAccepted)
                        break;
                    if (w->isWindow() || w->testAttribute(Qt::WA_NoMousePropagation))
                        break;
                    relpos += w->pos();
                    w = w->parentWidget();
                }
                mouse->setAccepted(eventAccepted);
                if (e->type() == QEvent::MouseMove) {
                    if (!pw)
                        break;

                    w = static_cast<QWidget *>(receiver);
                    relpos = mouse->pos();
                    QPoint diff = relpos - w->mapFromGlobal(d->hoverGlobalPos);
                    while (w) {
                        if (w->testAttribute(Qt::WA_Hover) &&
                             (!qApp->activePopupWidget()   ||   qApp->activePopupWidget()
                            == w->window()))
                        {
                            QHoverEvent he(QEvent::HoverMove, relpos, relpos - diff);
                            d->notify_helper(w, &he);
                        }
                        if (w->isWindow() || w->testAttribute(Qt::WA_NoMousePropagation))
                            break;
                        relpos += w->pos();
                        w = w->parentWidget();
                    }
                }
                d->hoverGlobalPos = mouse->globalPos();
            }
            break;
#ifndef QT_NO_WHEELEVENT
```

```
    case QEvent::Wheel:
        {
            QWidget* w = static_cast<QWidget *>(receiver);
            QWheelEvent* wheel = static_cast<QWheelEvent*>(e);
            QPoint relpos = wheel->pos();
            bool eventAccepted = wheel->isAccepted();
            if (e->spontaneous())
            {
                QWidget *fw = w;
                while (fw)
                {
                    if (fw->isEnabled() && (fw->focusPolicy() & Qt::WheelFocus) ==
                    Qt::WheelFocus)
                    {
                        fw->setFocus(Qt::MouseFocusReason);
                        break;
                    }
                    if (fw->isWindow())
                        break;
                    fw = fw->parentWidget();
                }
            }
            while (w)
            {
                QWheelEvent we(relpos, wheel->globalPos(), wheel->delta(),
                wheel->buttons(),wheel->modifiers(), wheel->orientation());
                we.spont = wheel->spontaneous();
                res = d->notify_helper(w, w == receiver ? wheel : &we);
                eventAccepted = ((w == receiver) ? wheel : &we)->isAccepted();
                e->spont = false;
                if ((res && eventAccepted)
                    || w->isWindow() || w->testAttribute(Qt::WA_NoMousePropagation))
                    break;
                relpos += w->pos();
                w = w->parentWidget();
            }
            wheel->setAccepted(eventAccepted);
        }
        break;
#endif
    case QEvent::ContextMenu:
        {
            QWidget* w = static_cast<QWidget *>(receiver);
            QContextMenuEvent *context = static_cast<QContextMenuEvent*>(e);
            QPoint relpos = context->pos();
            bool eventAccepted = context->isAccepted();
            while (w) {
                QContextMenuEvent ce(context->reason(), relpos, context->globalPos());
                ce.spont = e->spontaneous();
                res = d->notify_helper(w, w == receiver ? context : &ce);
                eventAccepted = ((w == receiver) ? context : &ce)->isAccepted();
                e->spont = false;
                if ((res && eventAccepted)
                    || w->isWindow() || w->testAttribute(Qt::WA_NoMousePropagation))
                    break;
                relpos += w->pos();
                w = w->parentWidget();
```

```
                }
                context->setAccepted(eventAccepted);
            }
            break;
#ifndef QT_NO_TABLETEVENT
        case QEvent::TabletMove:
        case QEvent::TabletPress:
        case QEvent::TabletRelease:
            {
                QWidget *w = static_cast<QWidget *>(receiver);
                QTabletEvent *tablet = static_cast<QTabletEvent*>(e);
                QPoint relpos = tablet->pos();
                bool eventAccepted = tablet->isAccepted();
                while (w)
                {
                    QTabletEvent te(tablet->type(), relpos, tablet->globalPos(),
                        tablet->hiResGlobalPos(), tablet->device(), tablet->pointerType(),
                        tablet->pressure(), tablet->xTilt(), tablet->yTilt(),
                        tablet->tangentialPressure(),tablet->rotation(), tablet->z(),
                        tablet->modifiers(), tablet->uniqueId());
                    te.spont = e->spontaneous();
                    res = d->notify_helper(w, w == receiver ? tablet : &te);
                    eventAccepted = ((w == receiver) ? tablet : &te)->isAccepted();
                    e->spont = false;
                    if ((res && eventAccepted)
                        || w->isWindow()|| w->testAttribute(Qt::WA_NoMousePropagation))
                        break;
                    relpos += w->pos();
                    w = w->parentWidget();
                }
                tablet->setAccepted(eventAccepted);
                qt_tabletChokeMouse = tablet->isAccepted();
            }
            break;
#endif // QT_NO_TABLETEVENT

#if !defined(QT_NO_TOOLTIP) || !defined(QT_NO_WHATSTHIS)
        case QEvent::ToolTip:
        case QEvent::WhatsThis:
        case QEvent::QueryWhatsThis:
            {
                QWidget* w = static_cast<QWidget *>(receiver);
                QHelpEvent *help = static_cast<QHelpEvent*>(e);
                QPoint relpos = help->pos();
                bool eventAccepted = help->isAccepted();
                while (w)
                {
                    QHelpEvent he(help->type(), relpos, help->globalPos());
                    he.spont = e->spontaneous();
                    res = d->notify_helper(w, w == receiver ? help : &he);
                    e->spont = false;
                    eventAccepted = (w == receiver ? help : &he)->isAccepted();
                    if ((res && eventAccepted) || w->isWindow())
                        break;
                    relpos += w->pos();
                    w = w->parentWidget();
                }
```

```
                help->setAccepted(eventAccepted);
            }
            break;
#endif
#if !defined(QT_NO_STATUSTIP) || !defined(QT_NO_WHATSTHIS)
        case QEvent::StatusTip:
        case QEvent::WhatsThisClicked:
            {
                QWidget *w = static_cast<QWidget *>(receiver);
                while (w) {
                    res = d->notify_helper(w, e);
                    if ((res && e->isAccepted()) || w->isWindow())
                        break;
                    w = w->parentWidget();
                }
            }
            break;
#endif
#ifndef QT_NO_DRAGANDDROP
        case QEvent::DragEnter:
{
                QWidget* w = static_cast<QWidget *>(receiver);
                QDragEnterEvent *dragEvent = static_cast<QDragEnterEvent *>(e);
#ifdef Q_WS_MAC
                QWidget *currentTarget = QDragManager::self()->currentTarget();
                if (currentTarget)
{
                    // Assume currentTarget did not get a leave
                    QDragLeaveEvent event;
                    QApplication::sendEvent(currentTarget, &event);
                }
#endif
                while (w)
{
                    if (w->isEnabled() && w->acceptDrops())
{
                        res = d->notify_helper(w, dragEvent);
                        if (res && dragEvent->isAccepted())
{
                            QDragManager::self()->setCurrentTarget(w);
                            break;
                        }
                    }
                    if (w->isWindow())
                        break;
                    dragEvent->p = w->mapToParent(dragEvent->p);
                    w = w->parentWidget();
                }
            }
            break;
        case QEvent::DragMove:
        case QEvent::Drop:
        case QEvent::DragLeave:
            {
                QWidget * w = QDragManager::self()->currentTarget();
                if (!w)
                {
```

```
    #ifdef Q_WS_MAC
                        if (e->type() == QEvent::DragLeave)
                            break;
                        // Assume that w did not get an enter.
                        QDropEvent *dropEvent = static_cast<QDropEvent *>(e);
    QDragEnterEvent                                         dragEnterEvent(dropEvent->pos(),
dropEvent->possibleActions(),dropEvent->mimeData(),
dropEvent->mouseButtons(),dropEvent->keyboardModifiers());
                        QApplication::sendEvent(receiver, &dragEnterEvent);
                        w = QDragManager::self()->currentTarget();
                        if (!w)
    #endif
                            break;
                    }
                    if (e->type() == QEvent::DragMove || e->type() == QEvent::Drop)
    {
                        QDropEvent *dragEvent = static_cast<QDropEvent *>(e);
                        QWidget *origReciver = static_cast<QWidget *>(receiver);
                        while (origReciver && w != origReciver)
    {
                            dragEvent->p = origReciver->mapToParent(dragEvent->p);
                            origReciver = origReciver->parentWidget();
                        }
                    }
                    res = d->notify_helper(w, e);
                    if (e->type() != QEvent::DragMove)
    {
                        QDragManager::self()->setCurrentTarget(0, e->type() == QEvent::Drop);
                    }
    Else
    {
                        QDragMoveEvent *moveEvent = static_cast<QDragMoveEvent *>(e);
    moveEvent->rect.setTopLeft(static_cast<QWidget                *>(receiver)->mapFrom(w,
moveEvent->rect.topLeft()));
                    }
                }
                break;
    #endif
            default:
                res = d->notify_helper(receiver, e);
                break;
            }
            return res;
    }
```

下面是 QtopiaApplication::notify()函数的处理过程：

```
bool QtopiaApplication::notify(QObject* o, QEvent* e)
{
    bool r = QApplication::notify(o,e);
    if (e->type() == QEvent::Show && o->isWidgetType()
        && ((QWidget*)o)->testAttribute(Qt::WA_ShowModal))
    {
        QMessageBox *mb = qobject_cast<QMessageBox*>(o);
        if (mb)
            hideMessageBoxButtons( mb );
    }
#if defined(QTOPIA_KEYPAD_NAVIGATION)
    if ( e->type() == QEvent::KeyPress || e->type() == QEvent::KeyRelease )
```

```
    {
        QKeyEvent *ke = (QKeyEvent*)e;
        QWidget *w = qobject_cast<QWidget*>(o)
        if (!ke->isAccepted())
        {
            if ( ke->key() == Qt::Key_Hangup || ke->key() == Qt::Key_Call || ke->key() ==
Qt::Key_Flip )
            {
                // Send some unaccepted keys back to the server for processing.
                QtopiaIpcEnvelope
e(QLatin1String("QPE/System"),QLatin1String("serverKey(int,int)"));
                e << ke->key() << int(ke->type() == QEvent::KeyPress);
            }
        }
        if (!ke->isAccepted()
                || (w && !w->hasEditFocus() && isSingleFocusWidget(w)
                    && !Qtopia::mousePreferred()))
        {
            if (w && ke->key() == Qt::Key_Back && e->type() == QEvent::KeyPress)
            {
                w = w->window();
                qLog(UI) << "Handling Back for" << w;
                if (QDialog *dlg = qobject_cast<QDialog*>(w))
                {
                    if (isMenuLike(dlg))
                    {
                        qLog(UI) << "Reject dialog" << w;
                        dlg->reject();
                      } else
                    {
                        qLog(UI) << "Accept dialog" << w;
                        dlg->accept();
                    }
                }
                else
                {
                    qLog(UI) << "Closing" << w;
                    w->close();
                }
                r = true;
            }
        }
    }
    else if (e->type() == QEvent::InputMethod)
    {
        // Try to keep the back button sensible when IM is composing.
        bool setBack = false;
        QSoftMenuBar::StandardLabel label = QSoftMenuBar::NoLabel;
        if (QLineEdit *le = qobject_cast<QLineEdit*>(o))
        {
            setBack = true;
            QInputMethodEvent *ime = (QInputMethodEvent*)e;
            if (mousePreferred)
            {
                label = QSoftMenuBar::Back;
            }
            else
```

```
                {
                    if (ime->commitString().length() + ime->preeditString().length() > 0
                        || le->cursorPosition() > 0)
                        label = QSoftMenuBar::BackSpace;
                    else if (le->text().length() == 0)
                        label = QSoftMenuBar::RevertEdit;
                }
            }
            else if (QTextEdit *te = qobject_cast<QTextEdit*>(o))
            {
                setBack = true;
                QInputMethodEvent *ime = (QInputMethodEvent*)e;
                if (mousePreferred)
                {
                    label = QSoftMenuBar::Back;
                }
                else
                {
                    if (ime->commitString().length() + ime->preeditString().length() > 0
                        || te->textCursor().position() > 0)
                        label = QSoftMenuBar::BackSpace;
                    else if (te->document()->isEmpty())
                        label = QSoftMenuBar::Cancel;
                }
            }
            if (setBack)
                ContextKeyManager::instance()->setStandard(qobject_cast<QWidget*>(o),
Qt::Key_Back, label);
        }
        else if (e->type() == QEvent::ChildAdded)
        {
#if QT_VERSION >= 0x040300
#error "Fix QCalendarWidget in keypad mode (see below)"
#else
            QCalendarWidget* w = qobject_cast<QCalendarWidget*>(o);
            if ( w )
            {
                QWidget *prevMonth = w->findChild<QWidget*>("qt_calendar_prevmonth");
                QWidget *nextMonth = w->findChild<QWidget*>("qt_calendar_nextmonth");
                QWidget *monthButton = w->findChild<QWidget*>("qt_calendar_monthbutton");
                QWidget *yearButton = w->findChild<QWidget*>("qt_calendar_yearbutton");
                QWidget *yearEdit = w->findChild<QWidget*>("qt_calendar_yearedit");
                if ( prevMonth && nextMonth && monthButton && yearButton && yearEdit )
                {
// This is the fix to go into QCalendarWidget
#ifdef QTOPIA_KEYPAD_NAVIGATION
                    prevMonth->hide();
                    nextMonth->hide();
                    monthButton->setFocusPolicy(Qt::NoFocus);
                    yearButton->setFocusPolicy(Qt::NoFocus);
                    //yearEdit->setFocusPolicy(Qt::NoFocus);
                    CalendarTextNavigation::ensureNavigationFor(w);
#endif
// end-of-fix
                }
            }
#endif
```

```
#endif
    }
    /* Work around a QComboBox bug and Key_Hangup */
    if (e->type() == QEvent::ChildAdded)
    {
        QComboBox *w = qobject_cast<QComboBox*>(o);
        if ( w )
        {
            w->installEventFilter(this);
        }
    }
    return r;
}
```

## 4.4 套接字通信

信号与槽机制为Qt提供了对象间通信的能力，而套接字通信为Qt提供了进程之间的通信能力，QCopChannel类继承于QObject类，它通过send(const QString& channel, const QString& msg)来发送消息;通过static void registerChannel(const QString& ch, QWSClient *cl);向服务器注册信道;通过receive(const QString& msg, const QByteArray &data)来接收消息。

另外，Qt还同时提供了QcopEnvelope类、QtopiaIpcAdaptor类、QtopiaServiceRequest类、QtopiaIpcEnvelope类对QCop消息进行了封装。为了方便用户使用来发送消息，接收消息的时使用QCopChannel类。

进程间通信实际上是通过csocket来实现的。

套接字通信对于处理不同应用程序间的交互和冲突事不可或缺的内容。比如利用浏览器下载Java文件时，就需要利用套接字发送消息以启动Java应用进行处理。

### 4.4.1 创建信道

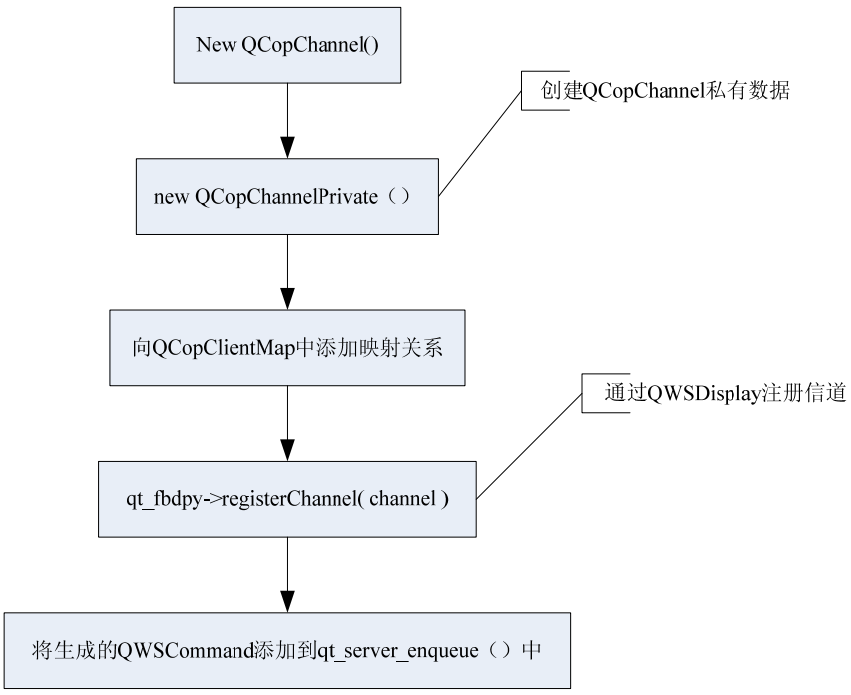


图26. 创建信道流程图

QcopClientMap 的定义如下：

```
typedef QMap<QString, QList< QPointer<QCopChannel> > > QCopClientMap;
```

下面为 QcopChannel 类的构造过程：

```
QCopChannel::QCopChannel(const QString& channel, QObject *parent) :QObject(parent)
{
    init(channel);
}
```

下面为 init()函数的处理过程：

```
void QCopChannel::init(const QString& channel)
{
    d = new QCopChannelPrivate;
    d->channel = channel;
    if (!qt_fbdpy)
    {
        qFatal("QCopChannel: Must construct a QApplication "
            "before QCopChannel");
        return;
    }
    {
        QMutexLocker locker(qcopClientMapMutex());
        if (!qcopClientMap)
        qcopClientMap = new QCopClientMap;              //创建客户端 Map
        // do we need a new channel list ?
        QCopClientMap::Iterator it = qcopClientMap->find(channel);
        if (it != qcopClientMap->end())
        {
            it.value().append(this);
            return;
        }
        it = qcopClientMap->insert(channel, QList< QPointer<QCopChannel> >());
        it.value().append(QPointer<QCopChannel>(this));
    }
    // inform server about this channel
    qt_fbdpy->registerChannel(channel);                 //注册信道
}
```

### 4.4.2 注册信道

下面为 registerChannel()函数的处理过程：

```
void QCopChannel::registerChannel(const QString& ch, QWSClient *cl)
{
     if (!qcopServerMap)
        qcopServerMap = new QCopServerMap;  //创建服务器端Map
        // do we need a new channel list ?
    QCopServerMap::Iterator it = qcopServerMap->find(ch);
    if (it == qcopServerMap->end())
          it = qcopServerMap->insert(ch, QList<QWSClient*>());
        // If the channel name contains wildcard characters, then we also
        // register it on the server regexp matching list.
    if (containsWildcards( ch ))
    {
        QCopServerRegexp item(ch, cl);
        if (!qcopServerRegexpList)
            qcopServerRegexpList = new QCopServerRegexpList;
```

```
        qcopServerRegexpList->append( item );
     }
    // If this is the first client in the channel, announce the channel as being
created.
    if (it.value().count() == 0)
    {
        QWSServerSignalBridge* qwsBridge = new QWSServerSignalBridge();
        connect(qwsBridge, SIGNAL(newChannel(QString)), qwsServer,
        SIGNAL(newChannel(QString)));
        qwsBridge->emitNewChannel(ch);
        delete qwsBridge;
    }
    it.value().append(cl);
}
```

### 4.4.3 发送消息

下面是 QcopChannel 的 send()函数的处理过程：

```
bool QCopChannel::send(const QString& channel, const QString& msg,
                       const QByteArray &data)
{
    if (!qt_fbdpy)
    {                    // qt_fbdpy 为QWSDisplay类的全局对象
        qFatal("QCopChannel::send: Must construct a QApplication "
               "before using QCopChannel");
        return false;
    }
    qt_fbdpy->sendMessage(channel, msg, data);
    return true;
}
```

下面是QWSDisplay的sendMessage()函数的处理过程：

```
void QWSDisplay::sendMessage(const QString &channel, const QString &msg,
                  const QByteArray &data)
{
    QWSQCopSendCommand com;
    com.setMessage(channel, msg, data);     //向com中写入有关信息
    qt_fbdpy->d->sendCommand(com);
}
```

下面是sendCommand()函数的处理过程：

```
void QWSDisplay::Data::sendCommand(QWSCommand & cmd)
{
#ifndef QT_NO_QWS_MULTIPROCESS
    if  (csocket)
        cmd.write(csocket);                 //向csocket中写入cmd
    else
#endif
                        //添加到static QList<QWSCommand*> outgoing链表中
    qt_server_enqueue(&cmd);
}
```

在实际的软件研发中，通常采用QtopiaServiceRequest类来发送消息，如在

Addressbook应用中，为了拨打选中的号码，需要向Dialer应用发送如下消息：

```
QtopiaServiceRequest req( "Dialer", "dial(QString,QUniqueId)" );
req << cit.value();              //输入信道名
req << cnt.uid();                //输入信息消息
req.send();                      //发送消息
```

下面是执行应用程序时系统发送的消息：

```
void Qtopia::execute( const QString &app, const QString& document )
{
    if ( document.isNull() ) {
        QtopiaServiceRequest e( "Launcher", "execute(QString)" );
        e << app;
        e.send();
    } else {
        QtopiaServiceRequest e( "Launcher", "execute(QString,QString)" );
        e << app << document;
        e.send();
    }
}
```

### 4.4.4 接收消息

下面是 QcopChannel 的 receive()函数的处理过程：

```
void QCopChannel::receive(const QString& msg, const QByteArray &data)
{
    emit received(msg, data);                //发射信号
}
```

当应用程序接收到收到消息的信号后，可以在一个槽函数中进行处理，如消息控制中心通过“QPE/System”来获得收到短信的情况：

```
connect(&channel, SIGNAL(received(const QString&,const QByteArray&)),
            this, SLOT(sysMessage(const QString&,const QByteArray&)) );
```

下面是MessageControl类的sysMessage()对系统消息的处理过程：

```
void MessageControl::sysMessage(const QString& message, const QByteArray &data)
{
    QDataStream stream( data );
    if ( message == "newMmsCount(int)")          //新彩信消息数
    {
         int count;
         stream >> count;
         if (count != mmsCount)
        {
             mmsCount = count;
             doNewCount(true,false,false);
         }
    }
    else
    if( message == "newSystemCount(int)" )       //新系统消息数
    {
        int count;
        stream >> count;
        if( count != systemCount )
```

```
            {
                systemCount = count;
                doNewCount( true, true, true);
            }
        }
    }
```

### 4.4.5 系统信道

在 Qt 中，提供了如下几种常用的系统信道供用户使用：

| 信道 | 描述 |
|---|---|
| QPE/QtopiaApplication | 服务器信道 |
| QPE/System | 系统信道 |
| QPE/SysMessages | 系统消息信道 |
| QPE/Mail | 邮件信道 |
| QPE/LauncherSettings | Launcher 设置信道 |
| Qtopia/PowerStatus | 电量状况信道 |
| QPE/PIM | 个人信息管理信道 |
| QPE/QStorage | 存储状况信道 |
| QPE/InputMethod | 输入法信道 |
| QPE/QuickLauncher | QuickLauncher 信道 |
| QPE/TaskBar | 任务栏信道 |

表2 系统信道

1. 服务器信道

“QPE/QtopiaApplication”为服务器信道，负责处理应用程序的状况如 fastAppHiding、available、closing、notBusy、appRaised、raise 等。

为了隐藏应用程序，QtopiaApplication 在其 hideOrQuit()函数中将发送以下消息，其中消息参数为应用程序名。

```
QtopiaIpcEnvelope
e(QLatin1String("QPE/QtopiaApplication"), QLatin1String("fastAppHiding(QString)") );
e << d->appName;
```

当应用程序完全启动后，QtopiaApplication 将会在其 initApp（）函数中发送以下消息给服务器，其中消息参数为应用程序名、应用程序 ID：

```
QtopiaIpcEnvelope env("QPE/QtopiaApplication","available(QString,int)");
env << QtopiaApplication::applicationName() << ::getpid();
```

当服务器需要杀掉应用程序时，QtopiaApplication 会在其 tryQuit()函数中发生以下消息给服务器，其中消息参数问应用程序名：

```
QtopiaIpcEnvelope
e(QLatin1String("QPE/QtopiaApplication"), QLatin1String("closing(QString)") );
e << d->appName;
```

2. 系统信道

“QPE/System”为系统信道，负责处理跟系统状况相关的消息如 applyStyle、shutdown、quit、close、forceQuit、restart、language、timeChange、volumeChange 等等，关于应用程序之间交互的消息也大多通过系统信道进行。

当语言设置发送变化时 LanguageSettings 在 newLanguageSelected()函数中发送以下消息，其中消息参数为语言名。

```
QtopiaIpcEnvelope e("QPE/System","language(QString)");
e << chosenLanguage;
```

当背景设置发生变化时，QExportedBackground 在 setExportedBackground()函数中发送以下消息。

```
QtopiaIpcEnvelope e("QPE/System", "backgroundChanged()");
```

当时间相关的信息发生变化时，SetDateTime 在 storeSettings()函数中发送以下消息。
QtopiaIpcEnvelope setTimeZone( "QPE/System", "timeChange(QString)" );//时区发生变化
setTimeZone << tz->currentZone();
QtopiaIpcEnvelope setClock( "QPE/System", "clockChange(bool)" );//AM/PM、时刻发生变化
setClock << ampmCombo->currentIndex();

3. 系统消息信道
“QPE/SysMessages”为系统消息信道，负责处理跟系统消息有关的消息如 ackMessage、collectMessages 等。
4. 邮件信道
“QPE/Mail”为邮件信道，负责处理邮箱的变化消息 mailboxChanged 等。
5. Launcher 设置信道
“QPE/LauncherSettings”为 Launcher 设置信道，负责处理跟 Launcher 相关的消息如 setTabView、setTabBackground、setTextColor、setFont 等。
6. 电量状况信道
“Qtopia/PowerStatus”为电量状况信道，负责处理背景光相关的消息 brightnessChanged 等。
7. 个人信息管理信道
“QPE/PIM”为个人信息管理信道，负责处理跟个人信息管理相关的消息如 addedAppointment、removedAppointment、updatedAppointment、reloadAppointments、addedTask、updatedTask、removedTask、addedContact、updatedContact、removedContact、updatePersonalId 等。
8. 存储状况信道
“QPE/QStorage”为存储状况信道，负责处理跟文件系统相关的消息 updateStorage 等。在 SD 卡等插拔时会使用到。
9. 输入法信道
“QPE/InputMethod”为输入法信道，负责处理跟输入法相关的消息如 loadInputMethods、unloadInputMethods、changeInputMethod、activateMenuItem、showInputMethod、hideInputMethod、inputMethodHint、inputMethodPasswordHint 等。
当需要隐藏输入法时，QtopiaApplication 在 hideInputMethod()发送以下消息。
QtopiaChannel::
send( QLatin1String("QPE/InputMethod"), QLatin1String("hideInputMethod()") );
当需要显示输入法时，QtopiaApplication 在 showInputMethod ()发送以下消息。
QtopiaChannel::
send( QLatin1String("QPE/InputMethod"), QLatin1String("showInputMethod()") );
10. QuickLauncher 信道
“QPE/QuickLauncher”为 QuickLauncher 信道，负责处理跟应用程序启动相关的消息如 available、exited、crashed、running 等。
QuickLauncher 启动后，需要通知应用程序发射器（launcher）QuickLauncher 已经可用，需要发送以下消息。
QtopiaIpcEnvelope env("QPE/QuickLauncher", "available(int)");
env << ::getpid();
QuickLauncher 崩溃时，需要发送以下消息。
QtopiaIpcEnvelope env("QPE/QuickLauncher", "crashed(int)");
11. 任务栏信道
“QPE/TaskBar”为任务栏信道，负责处理跟任务栏相关的消息如 message、reloadApplets、setLed 等。

## 4.5 虚拟帧缓存

Qtopia Core 能够直接对帧缓存（framebuffer）进行写入，这样就可以免除了对 X 窗口系

虚拟帧缓存技术通过模拟一个使用共享内存的帧缓存来实现。并通过 qvfb 工具将帧缓存内容在桌面上显示出来，这就是 Qtopia 模拟器的基本原理。Qvfb 工具实现了所谓的皮肤切换功能，能够改变显示内容的观感，可以调用下面的函数来判断是否使用了 qvfb：

```
static bool isQVFb()
{
    static bool result, test = true;
    if (test)
    {
        test = false;
        QByteArray display_spec(getenv("QWS_DISPLAY"));
        result = display_spec.startsWith("QVFb");
    }
    return result;
}
```

需要说明的是虚拟帧缓存仅仅是一个研发工具，没有考虑其中的安全问题，因此在实际的产品中应该避免使用。具体的做法是在使用 configure 配置产品的库时应该除去 qvfb 选项。

虚拟帧缓存在启动后会通过 QApplication::GuiServer 标识符来或使用-qws 选项构造 QApplication 对象来启动应用。

Qtopia Core 在默认情况下显示的模拟器分辨率为 640*480，可以通过 QWS_DISPLAY 环境变量来调整分辨率,可以通过下面提供的方式来获得 QWS_DISPLAY 的信息：

```
QByteArray display_spec(getenv("QWS_DISPLAY"));
```

## 4.6 布局管理

Qt 布局系统提供了一个简单而有效的方法来管理子部件的布局。其中最大的好处在于当子部件的字体大小、文本信息或者其他内容发生改变时，或者发生了子部件的显示、隐藏、移去等事件时，父部件会自动感知、自动刷新区域。需要说明的是一个布局管理器的子部件的父部件并不是布局管理器，而是被布局管理器安装在其上的部件。

利用布局管理器对部件进行管理，虽然存在着一定的欠灵活性，但对于需要自适应伸缩的对话框等而言，则是不可缺少的神兵利器！可以有效的避免因为字体变化、语种变化带来的显示问题。

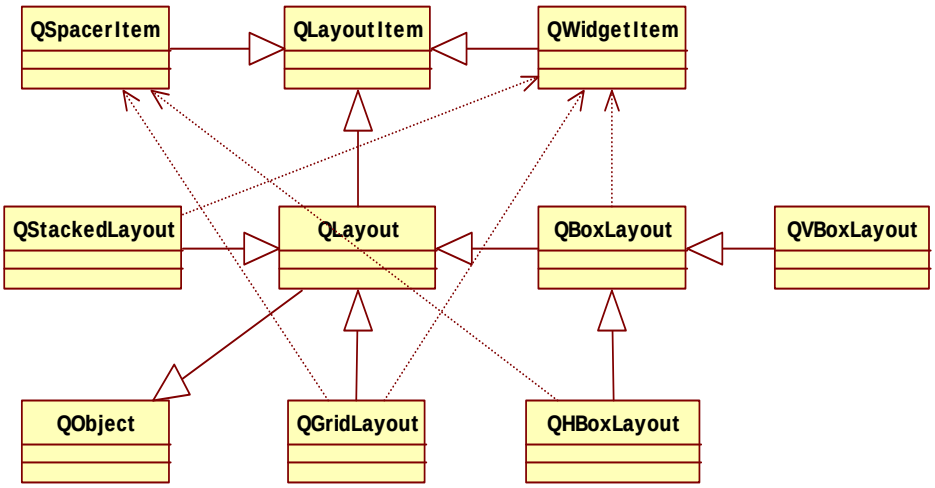


图27. 布局管理类图

1. 布局管理器

在 Qt 中，为了更好的完成部件的布局，Qt 提供了几个内建的布局管理器如 QHBoxLayout、QVBoxLayout、QGridLayout 来管理部件，如果需要更负责的布局，可以对这几种部件进行组合。

当向一个布局管理器中添加部件时，布局管理器会按以下几个步骤进行处理：

- 所有的部件将会根据 QWidget::sizePolicy()来初始化空间分配。
- 如果部件存在伸缩因子，那么在分配空间时，部件将会根据伸缩因子进行调整。伸缩因子定义了部件间的间隔。
- 如果分配的空间小于其最小值或大于其最大值，那么部件会自动设置其空间为最小值或最大值。

下面为向 QboxLayout 的 addWidget()函数添加 Qwidget 的处理过程：

```
void QBoxLayout::addWidget(QWidget *widget, int stretch, Qt::Alignment alignment)
{
    insertWidget(-1, widget, stretch, alignment);       //插入 Qwidget
}
```

下面为 QBoxLayout 的 insertWidget()函数插入 Qwidget 的处理过程：

```
void QBoxLayout::insertWidget(int index, QWidget *widget, int stretch,
                              Qt::Alignment alignment)
{
    Q_D(QBoxLayout);
    if (!checkWidget(this, widget))                 //检查 widget 是否为空 ,代码中注释有误
        return;
    addChildWidget(widget);
    if (index < 0)                                  // index 为负数，将 Qwidget 追加到尾部
        index = d->list.count();
    QWidgetItem *b = new QWidgetItem(widget);   //创建 QWidgetItem
    b->setAlignment(alignment);                     //设置排列方式
    QBoxLayoutItem *it = new QBoxLayoutItem(b, stretch);   //创建 QBoxLayoutItem
    d->list.insert(index, it);                      //  将 QboxLayoutItem 添加到相应位置
    invalidate();
}
```

通过以上分析可以看出，为了向布局管理器中添加 Item，首先要创建一个相应的 Item，然后将其插入到 Item 的列表中。

2. 自定义布局管理器

通过继承 QLayout，研发人员可以自定义自己需要的布局管理器，为了自定义自己的布局管理器，需要做如下的几点工作：

- 为布局管理器创建一个数据结构来存储条目。
- AddItem()来向布局管理器中添加条目。
- SetGeometry()来执行布局。
- sizeHint()来设置布局的最佳大小。
- itemAt()来设置如何迭代条目。
- takeAt()来设置如何移去条目。

如果一个部件没有包含任何子部件或者使用手工布局（调用 QWidget::setGeometry()设置部件的几何布局），可以通过以下几种方式来改变部件的行为：

- 重载 QWidget::sizeHint()以返回部件的最佳大小。
- 重载 QWidget::minimumSizeHint()以返回部件的最小大小。

  调用 QWidget::setSizePolicy()来定制部件的大小。

需要说明的是在 Qwidget 中已经包含了一个布局管理器。如果希望向 Qwidget 中添加期望的布局管理器，必须首先删除已有的布局管理器。

## 4.7 着色系统

Qt 的着色（paint）系统主要基于 QPainter、QPaintDevice、QPaintEngine 等类，可以使软件研发人员能够利用相同的 API 在屏幕上和打印设备上着色。

其中 QPaintEngine 类对底层的设备类型进行了封装，为 QPainter 提供了着色的接口，QPaintDevice 是一个抽象的二维空间，使研发人员可以利用 QPainter 来在 QPaintDevice 上执行着色操作。

为了使用 QPainter 创建用户图形，需要采取一些措施避免干扰 Qt 自己的绘图功能，可以考虑将绘图代码放置在 drawEvent()函数中，通过调用 QPainter::begin()和 QPainter::end（）函数来启动和停止绘图。

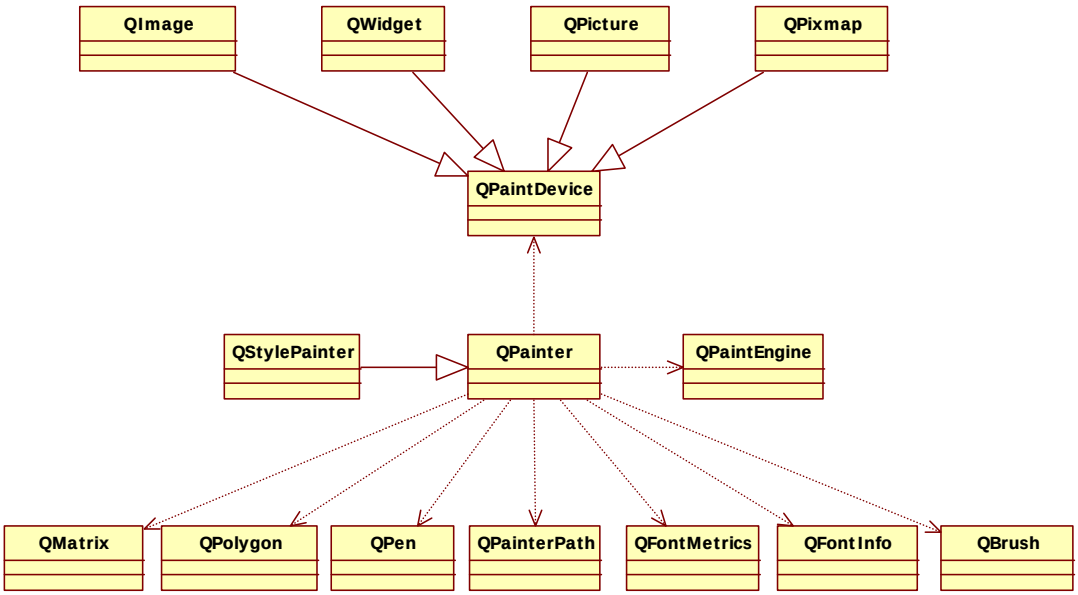


图28. 着色系统类图

1. 画

从简单的点、线到复杂的矢量路径，QPainter 类为大多数 GUI 提供了高度优化的方法来着色。其中点由 QPaint 来表述，线由 QLine 来表述，矩形由 QRect 来表述，区域由 QRegion 来表述，多边形由 QPolygon 来表述。而矢量图形则有 QPainterPath 来表述，QPainterPath 提供了一个容器可以使图形边缘被构造和复用。

当 QPainter 在画线和轮廓时，会使用到 QPen 类，一个画笔有风格、宽度、画刷等多种属性，画笔的画刷由 QBrush 来表述。

除了线和轮廓以外，QPainter 同样也能够画文本和图像。

当画文本时，字体由 QFont 来表述，如果没有安装相应的字体，Qt 将会采用最接近的已安装的字体，字体的属性信息可以由 QFontInfo 来表述。除此之外，字体的测量则由 QFontMetrics 来表述，这在需要获得字符串的长度等信息时非常有用；而当前的窗口系统的可用字体信息则由 QFontDatabase 来表述。

在研发过程中，常常需要自定义字体的颜色，控件背景的颜色等，这时候就要用到 QPalette，下面为 E2Button 采用的方法：

```
QPalette pal = palette();
pal.setColor(QPalette::Window, QColor(0, 0, 0, 0));
setPalette(pal);
```

其中 ColorRole 类型有 WindowText、Button、Light、Midlight、Dark、Mid、Text、BrightText、ButtonText、Base、Window、Shadow、Highlight、HighlightedText、Link、LinkVisited、AlternateBase、NoRole、NColorRoles = NoRole、Foreground = WindowText、Background = Window 等，可根据开发需要设置。

需要说明的是，为了进行复杂的操作，QPainter提供了多个图像裁剪函数，QPainter::scale(qreal sx, qreal sy)函数能够支持图像的缩放，其中sx表示水平方向的缩放系统，sy表示垂直方向的缩放系数；QPainter::shear ( qreal sh, qreal sv )函数能够支持图像的剪切，其中sx、sy表示要剪切的坐标系统；QPainter::rotate ( qreal angle )函数能够支持图像的旋转，其中angle表示图像旋转角度；QPainter::translate ( qreal dx, qreal dy )函数能够支持图像的平移，其中dx表示水平方向平移的像素数，dy表示垂直方向平移的像素数。QPainter::setWindow ( const QRect & rectangle )能够设置绘图设备的区域。

另外，需要提到的是 Qmatrix 是 Qt 内置的一个支持 2D 变换的类。能够支持对候选系统(coordinate system)的平移（Translation）、伸缩（Scaling）、剪切（Shearing）和旋转（Rotation）等操作，尤其在进行图像的渲染时特别有用。QPainter 对图像的变换也是基于 Qmatrix 进行的。通过使用 Qmatrix::map()函数，Qmatrix 还能够对图像原语如：点、线、多边形、区域和画笔路径等进行映射。下图为 QPainter 为伸缩图像的处理过程。

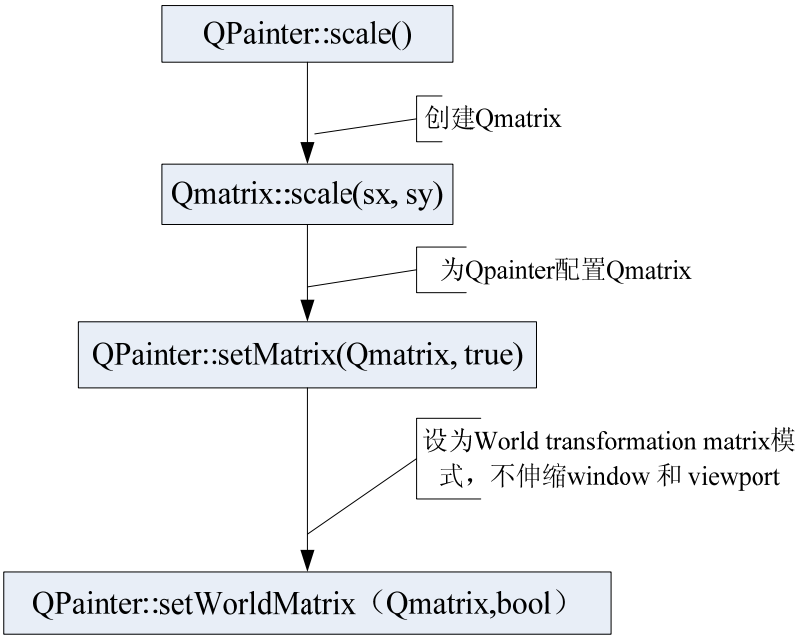


图29. QPainter 的图像伸缩过程

下面为 Qpainter 的 scale()函数的处理过程：

```
void QPainter::scale(qreal sx, qreal sy)
{
#ifdef QT_DEBUG_DRAW
    if (qt_show_painter_debug_output)
        printf("QPainter::scale(), sx=%f, sy=%f\n", sx, sy);
#endif
    QMatrix m;                  //创建 QMatrix
    m.scale(sx, sy);            //设置 Qmatrix 的伸缩参宿
    setMatrix(m, true);         //为 QPainter 配置 QMatrix
}
```

下面为 Qpainter 的 setMatrix ()函数的处理过程：

```
void QPainter::setMatrix(const QMatrix &matrix, bool combine)
{
    //设为 World transformation matrix 模式,只伸缩图像，不涉及窗口和显示区域
    setWorldMatrix(matrix, combine);
}
```

下面为 Qpainter 的 setWorldMatrix ()函数的处理过程：

```
void QPainter::setWorldMatrix(const QMatrix &matrix, bool combine)
{
```

```
    Q_D(QPainter);
    if (!isActive())                    //判断 QPaintEngine 是否激活
    {
        qWarning("QPainter::setMatrix: Painter not active");
        return;
    }
    if (combine)                        //判断伸缩模式
        d->state->worldMatrix = matrix * d->state->worldMatrix;
    else
        d->state->worldMatrix = matrix;
    if (!d->state->WxF)
        setMatrixEnabled(true);
    else
        d->updateMatrix();
}
```

图像的其他操作如平移、剪切和旋转等也有类似的处理过程。下图是图像变换过程的类图：

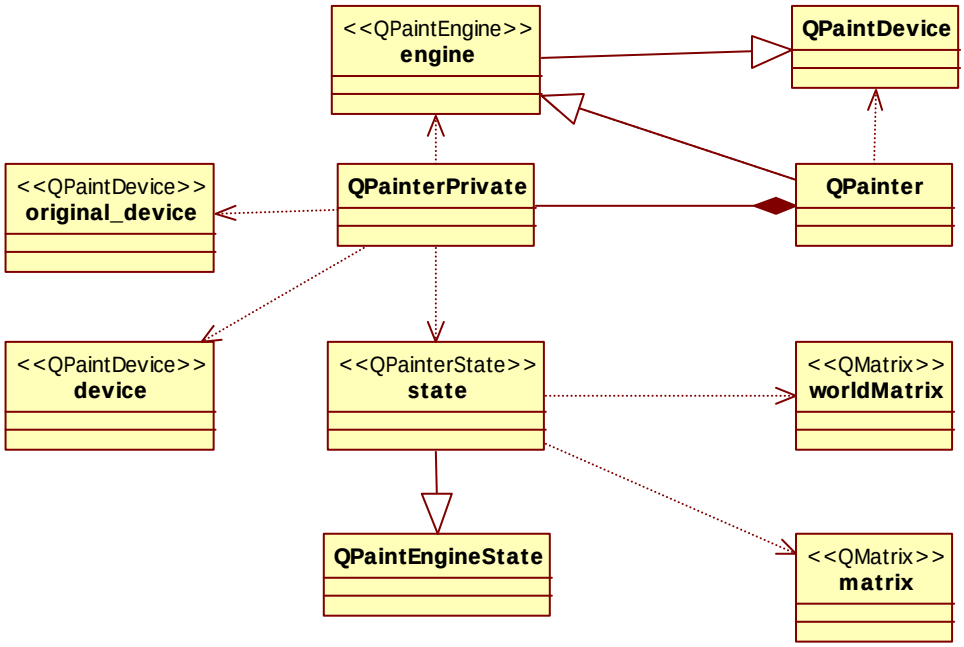


图30. 图像变换系统类图

从中可以看出，Qt 的图像变换是主要是基于 Qmatrix 实现并由 Qpainter 在 QpaintDevice 上显示的。认真分析 Qmatrix 代码可以发现，Qmatrix 实际上包含了一个 3*3 的矩阵，其中 dx 和 dy 参数表示水平和垂直方向上的平移偏量，m11 和 m22 参数表示水平和垂直方向上的伸缩系数，m21 和 m12 表示水平和垂直方向上的剪切系数。其单位矩阵条件下，m11、m22 为 1，其他参数为 0。下面就是图像像素位置的变换公式：

x' = m11*x + m21*y + dx

y' = m22*y + m12*x + dy

对于一个点(x, y)来说，其变换后的对应点就为(x', y')。对图像平移来说，这是最简单的变换，与点(x, y)相对应的点为(1+dx,1+dy);对图形伸缩来说，与点(x, y)相对应的点为(m11*x, m22*y);对图像剪切来说，由参数 m21 和 m12 控制；对图像旋转来说就比较复杂，需要由伸缩参数和剪切参数共同控制。如果需要为目标图像还原，可以基于 Qmatrix 的逆矩阵 QMatrix::inverted()进行操作得到。

2. 填充

外形可以使用 QBrush 来填充，一个画刷有颜色和风格等多种属性，画刷的填充模式有 Qt::BrushStyle 来描述。在 Qt 中，颜色由 QColor 来表述，QColor 是一个独立于平台和设备的类，它可以支持 RGB、HSV、CMYK 等多种颜色模式。

当创建一个新的部件时，最好使用部件提供的调色板中的颜色，调色板由 QPalette 来表述。

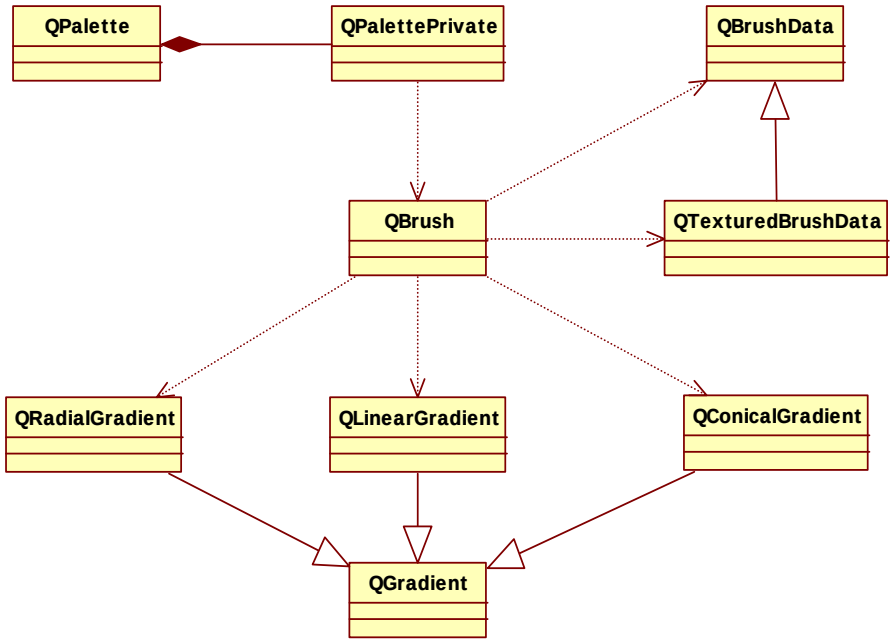


图31. 填充系统的类图

当创建一个画刷时，首先画刷会根据提供的颜色参数和画刷的风格选择不同的填充像素，默认情况下 QBrushData。为下面为 QBrush 初始化的处理过程：

```
void QBrush::init(const QColor &color, Qt::BrushStyle style)
{
    switch(style) {
    case Qt::NoBrush:
        d = nullBrushInstance();
        d->ref.ref();
        return;
    case Qt::TexturePattern:
        d = new QTexturedBrushData;
        static_cast<QTexturedBrushData *>(d)->setPixmap(QPixmap());
        break;
    case Qt::LinearGradientPattern:
    case Qt::RadialGradientPattern:
    case Qt::ConicalGradientPattern:
        d = new QGradientBrushData;
        break;
    default:
        d = new QBrushData;
        break;
    }
    d->ref = 1;
    d->style = style;
    d->color = color;
    d->hasTransform = false;
```

}

3. 着色设备

QPaintDevice 类是所有可以着色类的基类，QPainter 可以在所有着色设备上绘画。下面为 Qt 的着色设备的类图：

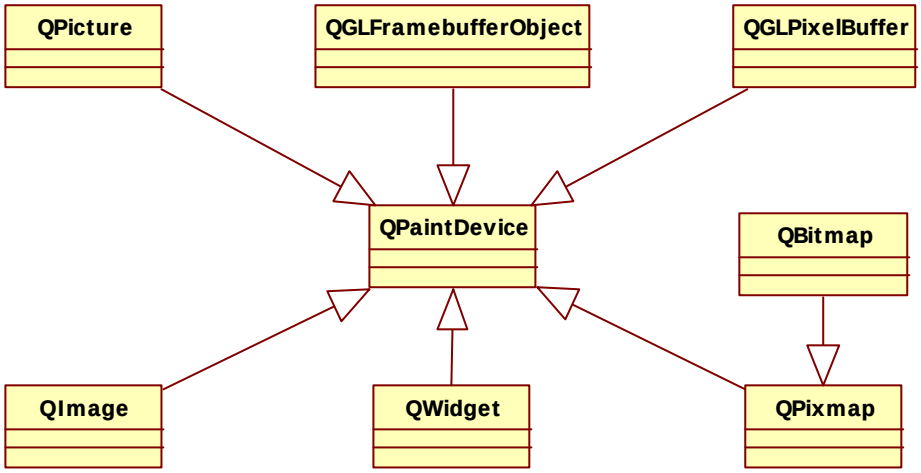


图32. 着色设备类图

4. 读写图像文件

Qt 提供了四个类 QPixmap、QImage 、QPicture、QBitmap 对静态图像文件的读写提供了支持。提供了 QMovie 对动画文件的读取提供了支持。

QImage 主要用于 I/O 和直接的像素存取和操作等。并针对这些操作进行了优化；Qimage 支持以下几种图像格式： Format_Mono 、 Format_MonoLSB 、 Format_Indexed8 、 Format_RGB32、Format_ARGB32、Format_ARGB32_Premultiplied、Format_RGB16。由于 Qimage 继承于 QPaintDevice，可以使用 Qpainter 在 Qimage 上直接着色，需要说明的是除了渲染文本信息（QFont 依赖于 GUI）外，着色操作是在另一个线程中执行的，为了在另一个线程中渲染文本信息，需要将文本信息作为 QpainterPath 为 GUI 线程俘获。

QPixmap 则主要用于显示图像，可以容易的利用 QLabel 和 QAbstractButton 的子类来进行此类操作。需要说明的是 QPixmap 的像素数据主要由窗口系统进行管理，仅能通过 QPainter 的函数和将 QPixmap 转换为 QImage 来存取数据。所以在进行图像操作时通常的做法是当导出数据或进行各种操作时，通常采用 QImage 来进行，如果需要显示，则再转换为 QPixmap 来实现。

QBitmap 则继承于 QPixmap，确保空对象以外的图像的深度为 1。当在一个 Qbitmap 对象上进行着色时，需要用到 Qcolor 对象 Qt::color0 和 Qt::color1。利用 Qt::color0 可以将像素值设为 0，利用 Qt::color1 可以将像素值设为 1。QBitmap 提供了 transformed(const QMatrix &matrix) 函数来提供变换后的副本。Qmatrix 支持平移、伸缩、剪切和旋转等操作

QPicture 则表示一个用于记录和重放 QPainter 命令的着色设备。独立于解决方案，能够在多种设备（SVG、PDF、PS、SCREEN）上显示，结果相同。

除此之外，Qt 还提供了 QImageReader、QImageWriter 对图像的读写过程进行更多的控制，其中 QImageReader 为从文件或其他设备读取图像提供了一个独立的接口。利用 QImageReader 的 setScaledSize()函数用户可以设置显示区域的大小；利用 setClipRect()可以设置图像的剪切区域等，基于对图像格式的潜在支持，可以有效的节省内存和提高图像的导出速度。而 QImageWriter 支持格式特定选项的设置，如 gamma level、compression level 和 quality 等。为例存储图像，需要为创建的 QimageWriter 构造函数提高文件名或设备指针、图像格式等信息。

Qt 支持 BMP（Windows Bitmap）、GIF（Graphic Interchange Format）、JPEG（Joint Photographic Experts Group）、MNG（Multiple-image Network Graphics）、PNG（Portable

Network Graphics）、PBM（Portable Bitmap）、PGM（Portable Graymap）、PPM（Portable Pixmap）、XBM（X11 Bitmap）、XPM（X11 Pixmap）等格式。

5. 风格定制

在 Qt 中，QStyle 对 GUI 的观感进行了封装，它可以从 QStyleOption 来获得渲染图形要素所需要的所有的信息，Qt 内建的部件可以使用 QStyle 来执行几乎所有的着色操作。

除了 Qpainter，着色系统同时还提供了 QStylePainter 来进行着色操作，利用 QStylePainter 可以很方便的画 QStyle 元素。

## 4.8 插件系统

插件(Plug-in)是一种遵循一定规范的应用程序接口编写出来的程序。

Qt 提供了两个 API 来创建插件：

- 一个高层的 API，用来扩展 Qt 自身，如自定义数据库驱动、图像格式、文本编解码、自定义风格等。
  一个底层的 API，用来扩展 Qt 应用程序。

### 4.8.1 Qt扩展

为了扩展 Qt，可以通过子类化一个适当的插件基类来实现。下面是常用的插件基类：

| 基类 | 默认路径 |
|---|---|
| QAccessibleBridgePlugin | plugins/accessiblebridge |
| QAccessiblePlugin | plugins/accessible |
| QDecorationPlugin | plugins/decorations |
| QIconEnginePlugin | plugins/iconengines |
| QImageIOPlugin | plugins/imageformats |
| QInputContextPlugin | plugins/inputmethods |
| QKbdDriverPlugin | plugins/kbddrivers |
| QMouseDriverPlugin | plugins/mousedrivers |
| QPictureFormatPlugin | plugins/pictureformats |
| QScreenDriverPlugin | plugins/gfxdrivers |
| QSqlDriverPlugin | plugins/sqldrivers |
| QStylePlugin | plugins/styles |
| QTextCodecPlugin | plugins/codecs |

表3 常用的插件基类

为了在应用程序运行时能够获得插件所在的路径，必须了解系统对插件的搜索过程。在 Qt 中，当应用程序运行时，Qt 将应用程序执行路径作为 pluginsbase，因而会首先查找应用程序执行路径，Qt 的当前执行路径可以通过调用 QCoreApplication::applicationDirPath()来获得，Qt 也会查找相应的插件路径，插件路径可以通过调用 QLibraryInfo::location(QLibraryInfo::PluginsPath)获得，需要说明的是，如果你将插件放在了自定义的路径，为了让系统能够搜索到该路径，需要调用 QCoreApplication::addLibraryPath()来添加相应的路径，或者通过调用 QCoreApplication::setLibraryPaths()来设置搜索的路径。除此之外，也可以通过修改 qt.conf 的方式来实现路径配置。利用 qt.conf 的好处在于，它可以覆盖 Qt 库中的硬编码路径。qt.conf 的信息可以通过 QlibraryInfo:: findConfiguration()来获得。

如下是假设定义了一个名为 MyStyle 的插件时的插件实现过程，为了实现一个插件，必须重载 keys()和 create()等函数：

```
class MyStylePlugin : public QStylePlugin
 {
 public:
      QStringList keys() const;
      QStyle *create(const QString &key);
  };
```

下面是 keys()的定义：

```
 QStringList MyStylePlugin::keys() const
 {
      return QStringList() << "MyStyle";
 }
```

下面是 create ()的定义：

```
 QStyle *MyStylePlugin::create(const QString &key)
 {
      if (key.toLower() == "mystyle")
           return new MyStyle;
      return 0;
 }
```

插件 MyStylePlugin 的声明：

```
 Q_EXPORT_PLUGIN2(MyStylePlugin)
```

对于数据库驱动、图形格式、字体编码和大多数常用的插件，并不需要显式的声明，根据需要，Qt 会自行查找并创建他们，但是风格插件例外，为了显式创建一个风格插件，可以如下定义：

```
QApplication::setStyle(QStyleFactory::create("MyStyle"));
```

### 4.8.2 Qt应用程序扩展

不仅 Qt 自身可以通过插件的方式进行扩展，Qt 应用程序也可以通过类似的方式利用插件进行扩展。Qt 应用程序可以利用 QpluginLoader 来进行扩展。

为了使一个应用程序可以被插件扩展，需要以下步骤：

- 定义一个接口集，以便与插件通信。
- 利用 Q_DECLARE_INTERFACE()通知 Qt 的元对象系统声明了一个接口。
- 利用 QPluginLoader 来导出插件。
- 利用 qobject_cast()来测试插件是否实现了给定的接口。

插件的编写涉及到以下几个步骤：

- 声明一个继承于 QObject 的插件类，提供插件需要提供的接口函数。
- 利用 Q_INTERFACES()来通知 Qt 的元对象系统声明了一个接口。
- 利用 Q_EXPORT_PLUGIN2()来定义插件。
- 通过适当的 Pro 文件来编译插件。

下面是插件接口的定义：

```
class FilterInterface
 {
 public:
     virtual ~FilterInterface() {}
     virtual QStringList filters() const = 0;
     virtual QImage filterImage(const QString &filter, const QImage &image,
                                QWidget *parent) = 0;
 };
 Q_DECLARE_INTERFACE(FilterInterface,
                     "com.trolltech.PlugAndPaint.FilterInterface/1.0")
```

下面是插件的定义：

```
#include <QObject>
```

```
#include <QStringList>
#include <QImage>
#include <plugandpaint/interfaces.h>
 class ExtraFiltersPlugin : public QObject, public FilterInterface
 {
     Q_OBJECT
     Q_INTERFACES(FilterInterface)
 public:
     QStringList filters() const;
     QImage filterImage(const QString &filter, const QImage &image,
                        QWidget *parent);
 };
```

QpluginLoader 提供了实时导出插件的方法，但 QpluginLoader 只对单个插件的导出提供了支持，当需要导出多个同类的插件时，使用 QpluginLoader 则显得不便，为此 Qt 利用 QpluginManager 来封装 QpluginLoader，为插件管理提供了更加便捷的方法。

下面以输入法为例来介绍插件管理器的使用方法：

```
void InputMethods::loadInputMethods()
{
     ……
     loader = new QPluginManager( "inputmethods" );          //构建插件管理器
     ……
     foreach ( QString name, loader->list() )                //搜索可用插件
     {
          qLog(Input) << "Loading IM: "<<name;
          QObject *instance = loader->instance(name);        //获取输入法
          if ( !instance ){
               qLog(Input) << "Missing loader instance";
               continue;
          }
          QtopiaInputMethod *plugin = qobject_cast<QtopiaInputMethod*>(instance);
            ……
}
```

在导出插件时，如果应用程序是按照静态的方式进行编译，可以利用 Q_IMPORT_PLUGIN()来导出。为了在编译静态版本的时候使用插件，需要的应用程序的 Pro 文件中进行如下的配置：

- 添加 CONFIG += static
- 在应用程序中利用 Q_IMPORT_PLUGIN()来导出插件。
- 在选项 LIBS 中设置插件。

## 4.9 拖放

拖放为在应用程序内部或应用程序之间传递信息提供了一个简单的可视机制，通常拖动是通过鼠标或触笔的方式进行的。在 Qt 中，拖放的数据由 QMimeData 进行了封装。

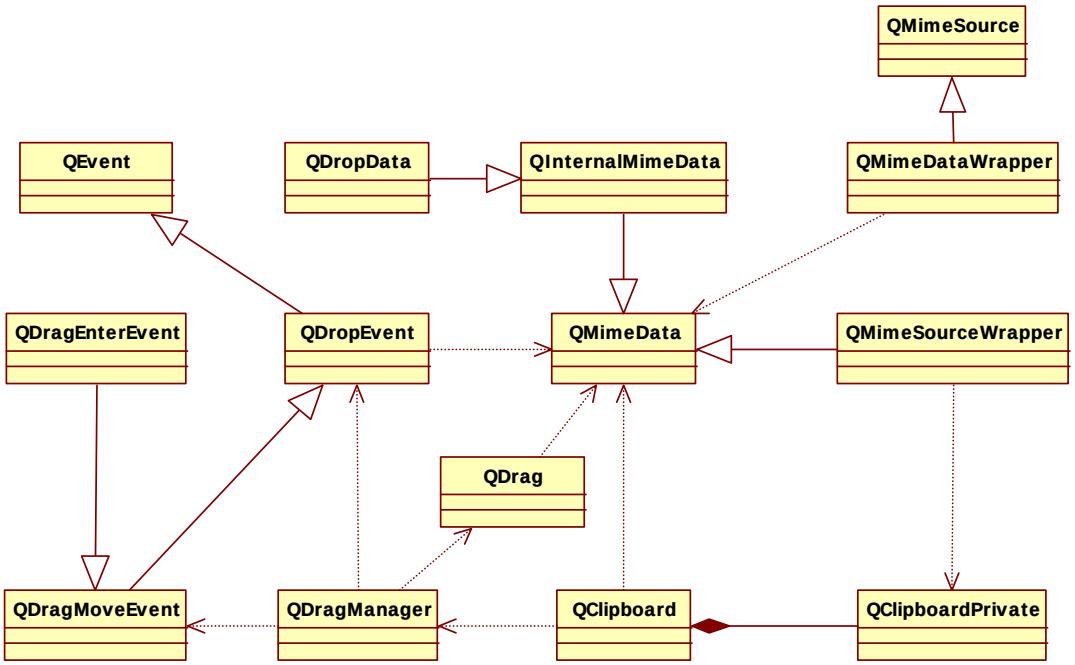


图33. 拖放系统类图

为了进行拖动，需要构建一个QDrag的对象，然后调用其start()函数，下面是一个利用鼠标或触笔进行拖动的例子：

```
void DragWidget::mousePressEvent(QMouseEvent *event)
 {
      if (event->button() == Qt::LeftButton
          && iconLabel->geometry().contains(event->pos()))
     {
          QDrag *drag = new QDrag(this);                    //创建 QDrag 对象
          QMimeData *mimeData = new QMimeData;     //创建 QMimeData 对象
          mimeData->setText(commentEdit->toPlainText());
          drag->setMimeData(mimeData);
          drag->setPixmap(iconPixmap);
          Qt::DropAction dropAction = drag->start();       //调用 start()函数
          ...
      }
 }
```

如果需要区别鼠标事件是拖动事件还是点击时间，则需要重载mousePressEvent()函数记录下鼠标或触笔的起始位置。

```
void DragWidget::mousePressEvent(QMouseEvent *event)
 {
      if (event->button() == Qt::LeftButton)
          dragStartPosition = event->pos();       //起始位置
 }
```

然后再 mouseMoveEvent()函数中进行如下处理：

```
void DragWidget::mouseMoveEvent(QMouseEvent *event)
 {
      if (!(event->buttons() & Qt::LeftButton))
          return;
      if ((event->pos() - dragStartPosition).manhattanLength()   //判断拖拽还是点击
           < QApplication::startDragDistance())
```

```
            return;
        QDrag *drag = new QDrag(this);
        QMimeData *mimeData = new QMimeData;
        mimeData->setData(mimeType, data);
        drag->setMimeData(mimeData);
        Qt::DropAction dropAction = drag->start(Qt::CopyAction | Qt::MoveAction);
        ...
    }
```

为了接收拖到部件上的信息，部件需要调用 setAcceptDrops(true)函数,并且重载 dragEnterEvent() 和 dropEvent() 事件处理函数。

下面是 dragEnterEvent()函数的处理过程：

```
void dropWidget::dragEnterEvent(QDragEnterEvent *event)
 {
     if (event->mimeData()->hasFormat("text/plain"))
         event->acceptProposedAction();
    …
 }
```

下面是针对文本信息的 dragEnterEvent ()函数的处理过程：

```
void dropWidget::dragEnterEvent(QDragEnterEvent *event)
 {
     if (event->mimeData()->hasFormat("text/plain"))
         event->acceptProposedAction();
 }
```

下面是针对文本信息的 dropEvent ()函数的处理过程：

```
void dropWidget::dropEvent(QDropEvent *event)
 {
     textBrowser->setPlainText(event->mimeData()->text());
     mimeTypeCombo->clear();
     mimeTypeCombo->addItems(event->mimeData()->formats());
     event->acceptProposedAction();
 }
```

除了拖放以外，剪切板 QClipboard 也是在应用程序内部或应用程序之间传递信息的一种重要方式，如下提供了获取剪切板的方式：

```
clipboard = QApplication::clipboard();
```

剪切板也同样使用了 QmimeData 来表述传递的数据信息，并通过 setText()、setImage()、setPixmap()等函数对数据进行了封装，更便于用户的使用，需要说明的是剪切板有三种模式 Clipboard、 Selection、FindBuffer。下面为在 Clipboard 模式下设置文本信息的操作方法：

```
clipboard->setText(lineEdit->text(), QClipboard::Clipboard);
```

当剪切板上的数据发生变化时，QClipboard 会发出 dataChanged()信号，可以通过如下的方式来监听剪切板上的数据变化：

```
connect(clipboard, SIGNAL(dataChanged()), this, SLOT(updateClipboard()));
```

下面为 updateClipboard()的处理过程：

```
void ClipWindow::updateClipboard()
 {
     QStringList formats = clipboard->mimeData()->formats();
     QByteArray data = clipboard->mimeData()->data(format);
     ...
  }
```

在Linux平台下，如果发生了鼠标或触笔对数据的选择事件，Qclipboard会发出了 selectionChanged()信号。

更丰富的信息请参看参考文献[38]。

## 4.10 多线程

Qt 提供了一个与平台独立的多线程实现机制，这样研发人员就可以容易的研发出可移植的多线程程序，充分发挥多核计算机的优势。另外，当执行一项很耗时的操作时，用户也可以同时从事其他的操作。

在 Qt 早期的版本中，对线程的支持支持可选的功能，需要进行相应的配置方能进行多线程编程，在 Qt 4 中，对多线程编程进行支持变为默认的功能。

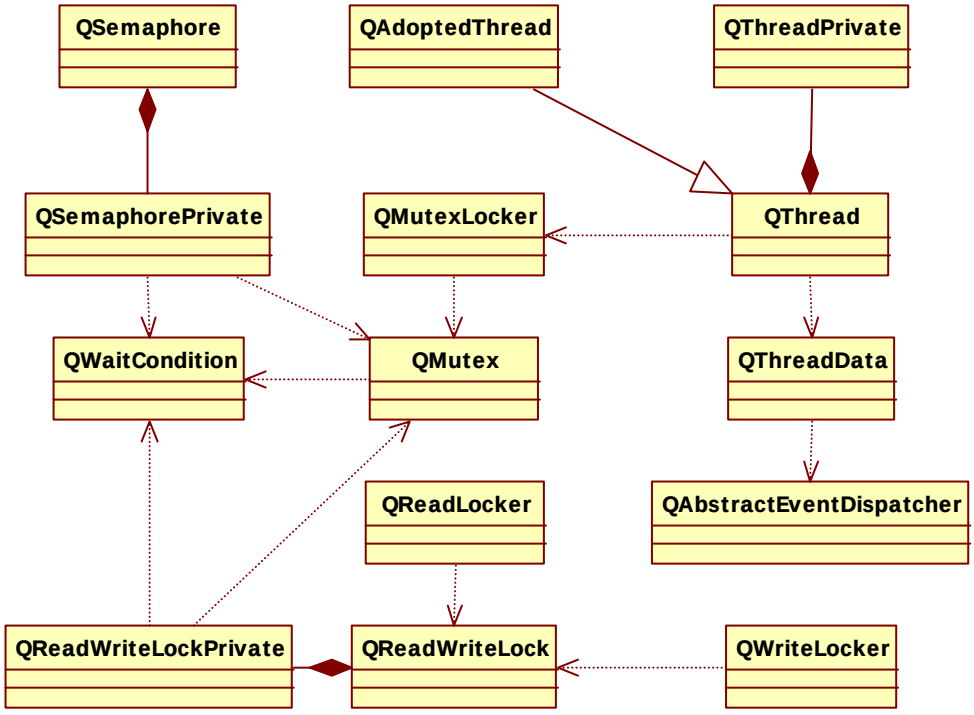


图34. 多线程类图

线程的优先级有以下几个级别：IdlePriority、LowestPriority、LowPriority、NormalPriority、HighPriority、HighestPriority、TimeCriticalPriority、InheritPriority 等。

为了创建一个新线程，需要继承 Qthread 并重载其 run()函数，处理过程如下：

```
class newThread : public QThread
 {
      Q_OBJECT

 protected:
      void run();
 };

 void newThread::run()
 {
      ...
 }
```

然后创建一个 newThread 的对象，调用其 start()函数，run()函数所封装的代码将会在新的线程中执行，start()函数的处理过程如下：

```
void *QThreadPrivate::start(void *arg)
{
     pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
```

```
        pthread_cleanup_push(QThreadPrivate::finish, arg);
        QThread *thr = reinterpret_cast<QThread *>(arg);
        QThreadData *data = QThreadData::get2(thr);
        pthread_once(&current_thread_data_once, create_current_thread_data_key);
        pthread_setspecific(current_thread_data_key, data);
        data->ref();
        data->quitNow = false;
        createEventDispatcher(data);
        emit thr->started();
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
        pthread_testcancel();
        thr->run();                                    //调用 run()函数
        pthread_cleanup_pop(1);
        return 0;
    }
```

需要说明的是，QCoreApplication::exec()总是在主线程中被调用，在GUI应用中，主线程因为是唯一的允许执行 GUI 相关操作的线程也被称为 GUI 线程，在创建新的线程前，必须构建 QApplication (或 QCoreApplication)对象。

Qt 除了提供了 Qthread 用于启动新线程外，还提供了如下的类来支持多线程编程。其中 QMutex、QReadWriteLock、QSemaphore、QWaitCondition 为线程提供了同步。

QMutex 提供了互斥锁的功能，如果一个线程试着去请求一个已经被上锁的共享数据，那么该线程将会被设为睡眠直到当前拥有共享数据的线程为共享数据解锁为止。

QreadWriteLock 类似于 Qmutex，但提供了更多的便利，它对共享数据的操作区分“读”和“写”，允许多个用户同时读取数据。

QSemaphore 是 Qmutex 的泛化，与 Qmutex 智能保护一个共享资源不同，Qsemaphore 能够对多个共享数据提供保护。

QwaitCondition 为一个线程在条件满足时唤醒另一个线程提供了支持。

每个线程都有它自己的事件循环，初始线程利用 QCoreApplication::exec()来启动其实际循环，其他线程则利用 QThread::exec()来启动事件循环。

关于多线程的更多信息，请参考参考文献[37]。

## 4.11 文件管理

文件管理是 Qtopia 提供的一个重要功能，它主要有以下几个类 QStorageMetaInfo、QfileSystem、Qfile、QfileInfo、QfileMonitor、QfileSystemFilter、QDir 等来实现。

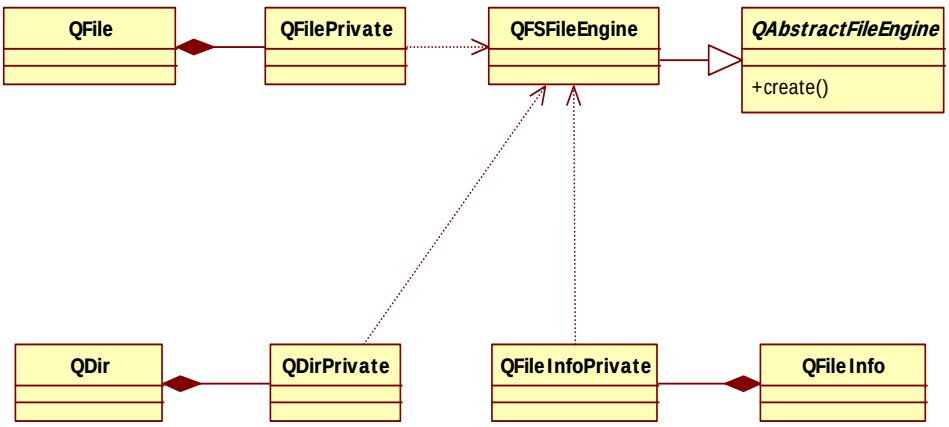


图35. 文件管理的类图

在文件管理系统的设计中，Qt采用了GOF的工程模式进行设计，获得了很好的效果，上

图即为基于工厂模式的文件管理系统的设计架构。

1. 文件

最常用的文件操作是读取文件。QFile 提供了读写文件的接口，能够支持对文本文件、二进制文件和资源的读取，Qfile 通常和 QtextStream 和 QDataStream 一起使用。下面是 Qassistant 读取文件列表的处理过程：

```
void Index::readDocumentList()
{
    QFile f( docListFile );
    if ( !f.open(QFile::ReadOnly ) )
        return;
    QDataStream s( &f );
    s >> docList;
}
```

而 QfileInfo 提供了获得文件信息的接口。能够支持对文件名、存储路径、存取权限、文件属性（目录，符号链接？）、文件大小、修改日期等文件信息的获取。

2. 目录

Qdir 提供了对目录结构及其内容操作的接口，对路径名、关于路径和文件名的存取信息提供了支持。和 QfileInfo 配合，可以利用 QDir::entryInfoList()获取文件信息列表，利用 QFileInfo::size()可以获得文件的大小，这样很容易就可以获得整个文件夹的大小等信息。

```
int dirSize(const Qstring &path)
{
    Qdir dirPath(path);
    QstringList::Iterator it;
    int size=0;
    QStringList files=dir.entryList("*",Qdir::Files);
    it=files.begin();
    while(it!=files.end())
    {
        size+=QfileInfo(dir,*it).size();
        ++it;
    }
    QstringList dirs=dir.entryList(QDir::Dirs);
    it=dirs.begin();
    while(it!=dirs.end())
    {
        if*(*it!="."&&*it!="..")
        size+=dirSize(dirPath+"/"+*it);
        ++it;
    }
    return size;
}
```

需要说明的是利用 QDir::entryInfoList(const QStringList &nameFilters, Filters filters,SortFlags sort)获取文件信息列表时，需要注意其过滤器参数的选择。如文件名过滤器有 QDir::Executable、QDir::Readable、QDir::Writable、QDir::Hidden、QDir::NoSymLinks 等等。

3. 文件系统

QfileSystem 描述了单个挂载点的信息，如可用空间、磁盘大小、块大小以及读写权限等，QstorageMetaInfo 描述了 Linux 文件系统的挂载信息。每个挂载点由 QfileSystem 来描述，能够检测到挂载点的挂载、卸载变化。

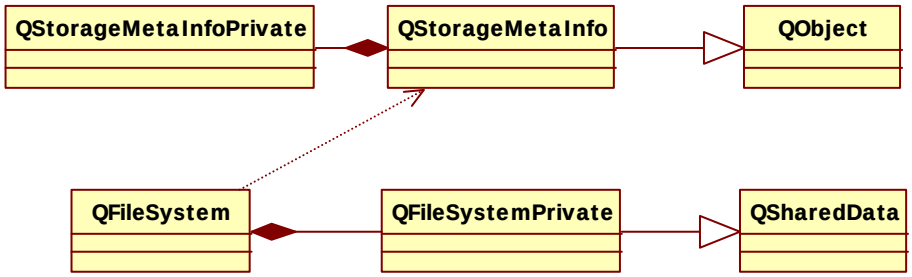


图36. 文件系统类图

下面是 DataView 类中获得文件系统的容量和可用空间的方法：

```
void DataView::fileSystemMetrics(const QFileSystem *fs, long *avail, long *total)
{
    long mult = 0;
    long div = 0;
    if ( fs->blockSize() )
    {
        mult = fs->blockSize() / 1024;
        div = 1024 / fs->blockSize();
    }
    if ( !mult ) mult = 1;
    if ( !div ) div = 1;
    *total = fs->totalBlocks() * mult / div;
    *avail = fs->availBlocks() * mult / div;
}
```

当 Linux 文件系统的挂载信息发生变化时，QstorageMetaInfo 会发出 disksChanged ()信号。

在应用程序模块中，如果需要获取挂载变化信息，就需要如下建立与 QstorageMetaInfo 的 disksChanged ()信号的连接：

```
connect( storage, SIGNAL(disksChanged()), this, SLOT(updatePaths()) );
```

4. 正则表达式

如果需要读取某类信息，如文件名称或者电子邮件地址等，就需要对信息进行验证，获得的信息是否合法，这就要用到正则表达式，Qt 利用 QValidator 和 QRegExp 对正则表达式提供了支持。下面为 Qtopia 在设置收件人时对邮件地址的验证过程：

```
QStringList MailMessage::mailRecipients() const
{
    QStringList mailRecipients;
    QRegExp re( "^[\\+\\*#\\d\\-\\(\\)\\s]*$" );                    //设置非法字符
    QStringList allRecipients = to() + cc() + bcc();
    for( QStringList::Iterator it = allRecipients.begin() ;
        it != allRecipients.end() ; ++it )
        if( !re.exactMatch( *it ) )                                 //检查匹配情况
            mailRecipients.append( *it );
    return mailRecipients;
}
```

关于正则表达式的更多信息，请参考参考文档[16]。

# 第 5 章　编译与配置

**知人者智，自知者明。**

《老子》

## 5.1 编译系统

为了定义编译过程，编译系统提供了如下的配置文件：

| 文件 | 描述 |
|---|---|
| configure | 编译系统的初始接入点，用来建立编译树 |
| bin/qtopiamake | 由 configure 调用，是外部工程的初始接入点 |
| bin/*.pm | 为编译系统中的 perl 脚本提供支持 |
| Makefile | 编译系统的第二个切入点，由 qtopiamake 创建 |
| Makefile.target | 由 qmake 为每个工程创建，不可以直接调用 |
| *.pro | 在 qmake 创建 Makefile.target 时读入 |
| *.prf | 特征（feature）文件 |
| *.pri | 混杂系统文件，用来包含.pro 和.prf 文件 |

表4 编译文件

### 5.1.1 环境变量

编译系统在编译和运行 configure、qtopiamake、Makefile 等脚本时会适当的设置一些环境变量。

- QTOPIADIR
  设置编译树的位置。
- QTOPIA_DEPOT_PATH
  设置 Qtopia 源文件树的位置。

下面是 configure、Makefile 设置的一些环境变量：

| 环境变量 | 备注 |
|---|---|
| QMAKEFEATURES | 设为$PROJECT_ROOT/features:$QTOPIA_DEPOT_PATH/src/build |
| QTDIR | 对于 host 机设为$QTOPIADIR/qtopiacore/host；对于 Target 机设为$QTOPIADIR/qtopiacore/target |
| PREFIX | 设为$QTOPIADIR/image/Qtopia |
| DPREFIX | 设为$QTOPIADIR/dimage/Qtopia |

表5 环境变量

### 5.1.2 编译模式

Qtopia 有多种编译模式，可以根据不同的需要进行选择。主要的编译模式如下：

- quicklaunch
- non-quicklaunch
- singleexec (quicklaunch)
- singleexec (non-quicklaunch)

但最基本的模式有两种：quicklaunch 和 single-exec 编译。

1. single-exec

当采用 single-exec 方式进行编译时，qpe 和所有的应用程序都被静态链接到一个单一的二进制文件中，可以有效的减小生成的二进制文件的大小。但是和普通意义上的静态编译不同的是，在生成的该二进制文件中，对不同的应用程序而言，库文件只有唯一的一份拷贝；而且应用程序可以作为单独的进程运行，从而可以防止因某个应用程序发生错误而导致整个系统发生崩溃。同时由于是静态链接，应用程序的启动速度会相对加快。这对于硬件配置较低的终端而言，是个不错的选择。

采用 single-exec 方式需要注意的一个问题是，当编译第三方组件时，要预防可能存在的命名冲突。

为了进行 single-exec 编译。需要在 Qtopia 的 configure 中添加 config=single-exec 选项。

2. quicklaunch

quicklaunch 模式即动态编译模式。

为了方便在不同的编译模式下切换，Qtopia 提供了如下使用方式的两个宏：

```
QTOPIA_ADD_APPLICATION(QTOPIA_TARGET, MyMainClass)
QTOPIA_MAIN
```

其中 QTOPIA_ADD_APPLICATION()的第一个参数是一个与模块的工程文件中 TARGET 值一致的字符串；QTOPIA_ADD_APPLICATION()的第二个参数是所定义的模块的主类名，如地址薄模块的主类名为：AddressbookWindow，在 addressbook.h 中定义。

如果你准备在编译中使用这些宏，你必须在工程文件中添加配置 CONFIG=qtopia_main，如果不准备利用 quicklaunch 或者 singleexec 编译应用程序，你需要添加 no_quicklaunch 或者 no_singleexec 选项。

例如，游戏并不侧重启动速度，这时候就不用采用 quicklauncher 的方式编译，但是需要利用 singleexec 编译，这时候的配置选项可以设为 CONFIG+=qtopia_main no_quicklaunch。

关于编译模式的配置选项如下：

| 值 | 描述 |
|---|---|
| no_install | 是目标工程不被安装，主要用在 stub 和 subdirs 类型的工程中。 |
| qtopia_main | 提醒编译系统应用程序使用了 QTOPIA_ADD_APPLICATION 和 QTOPIA_MAIN。 |
| no_quicklaunch | 体现编译系统不进行 quicklaunch 模式编译。仅在 qtopia_main 配置后可用。 |
| no_singleexec | 体现编译系统不进行 singleexec 模式编译。仅在 qtopia_main 配置后可用。 |
| singleexec_main | 提醒编译系统应用 singleexec 模式编译，即使不支持 quicklaunch 模式编译，这是必须确保应用程序可以被服务器调用。 |
| no_dest | 可以阻止 DESTDIR 的修改引起的编译变化，确保工程输出位于编译路径。 |
| no_tr | 如果你的工程没有启用 tr()宏，通过设置该选项，可以阻止对 i18n 文件的处理。 |
| no_auto_translatables | 设置该选项，可以阻止自动添加 FORMS, HEADERS and SOURCES to TRANSLATABLES. |
| no_pkg | 可以阻止包被隐式创建 |
| build_all_dirs | 在 subdirs 工程中使用，可以不依赖 PROJECTS 变量设置特定的编译路径 |

表6 编译配置

### 5.1.3 调试方法

除了 GDB 的调试方法外，在 Qt 中也常常会利用一些全局的调试函数和调试宏来进行调试。

调试函数如下：

- qDebug()，用来进行自定义的调试输出。
- qWarning()，用来报告应用程序中警告性和可恢复性的错误。
- qCritical()，用来报告关键性错误信息和系统错误。
- qFatal()，在程序退出前报告致命性的错误。

调试宏如下：

- Q_ASSERT(cond),其中“cond”是一个布尔表达式。
- Q_ASSERT_X(cond, where, what),为 Q_ASSERT(cond)的扩展函数，其中“cond”是一个布尔表达式，“where”表示位置，“what”表示一个消息。
  Q_CHECK_PTR(ptr),其中“ptr”表示一个指针。判断是否成功分配内存或者是否为一个空指针。

### 5.1.4 编译步骤

- configure
  - 配置 Qt 和 Qtopia Core。
  - 创建$QTOPIADIR/src/config.pri。
  - 准备编译树。
  - 为所有的*.pro 文件创建 Makefile 文件。
  - 创建$QTOPIADIR/include 并确保 libs 被处理。
  - 建立工程根：
    - $QTOPIADIR/src – Qtopia
    - $QTOPIADIR/src/qtopiadesktop - Qtopia Desktop
    - $QTOPIADIR/etc/themes – Themes
    - $QTOPIADIR/src/plugins/designer - Designer plugins
- Makefile
  - 执行储存转发
  - 建立环境变量/overrides/etc
  - 确保 Makefile.target 被更新
  - 运行 Makefile.target
- .qmake.cache
  - 每个工程根都有一个.qmake.cache
  - 被 qmake 自动包含
  - 包含 tree_config.pri
- tree_config.pri
  - 每个工程根都有一个 tree_config.pri
  - 设置 config 值
  - 为编译树添加适当的工程描述符
- default_pre.prf
  - 在主工程文件导入前被导入
  - 导入 functions.prf
  - 导入 config.prf
- functions.prf
  - 建立自定义函数

- 定义 qtopia_project()函数

➢ config.prf
- 导入 config.pri
- 设置静态 config
- 导入 projects.pri

➢ last_minute_config.prf
- 在 pro 文件被导入前最后一次设置 config
- 在 qtopia_project()内导入

➢ [project].pro
- 这是实际的工程的 pro 文件
- 必须调用 qtopia_project()，否则将会出错

➢ qtopia_project()
- 设置默认并为编译的工程配置编译系统

➢ projects.pri
- 设置 PROJECTS 变量
- 设置编译细节
- 启用可用的应用程序

➢ default_post.prf
- 确保 CONFIG 变量顺序正确

## 5.2 应用程序配置

在实际的项目中，除了 Qtopia 已经提供的应用外，常常需要添加新的应用，这时候就设计到如何向项目中添加新应用的问题。默认情况下，Qtopia 会将生成的文件目录放置在 QTOPIADIR/image 下。下面是 Qtoipia 启动后，对相应的 image 下的目录的搜索过程：

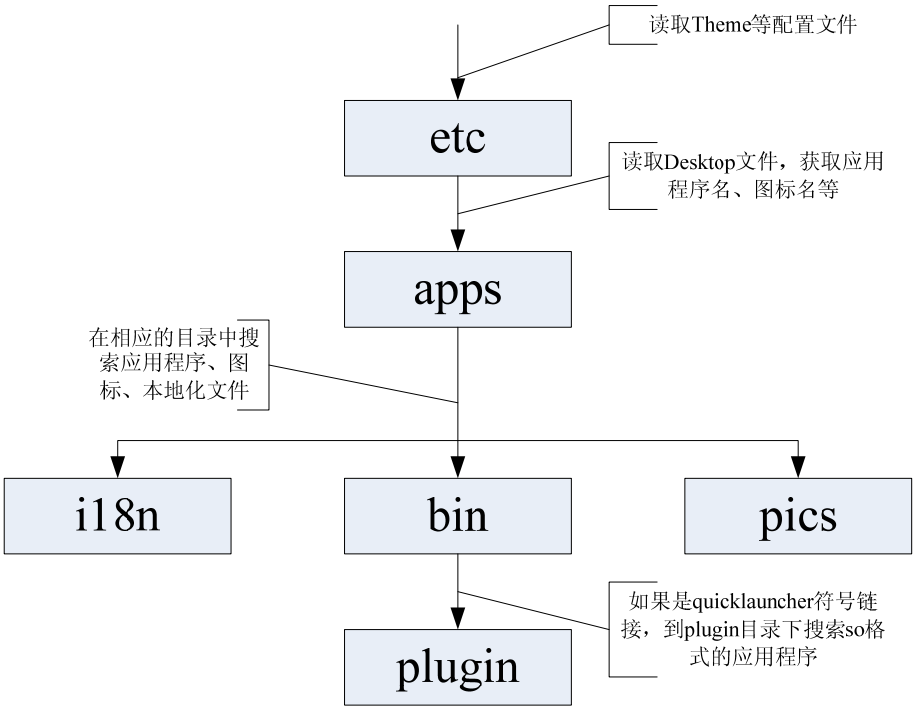


图37. Qtopia 启动后目录搜索过程

在默认的情况下，启动 Qtopia 模拟器的脚本为的 QTOPIADIR/scripts/runqtopia，启动

QVFB 的脚本为 QTOPIADIR/scripts/runqvfb。默认皮肤微 Greenphone，默认的 home 路径为 /opt/qtopia-phone-4.2.0/home。

向项目中添加新应用的方法如下：

1. 创建工程文件

   ＃$QTOPIADIR/bin/qtopiamake -project

2. 创建桌面文件

   除了工程文件外，还需要为应用程序创建一个桌面文件，现以 QTOIADIR\apps\Applications\textedit.desktop 文件的内容为例进行说明：

   ```
   [Translation]
   File=QtopiaApplications                 //文件类型,是应用程序还是设置还是游戏，如果是应用程序，设为 QtopiaApplications；如果是设置，设为 QtopiaSettings；如果是游戏，设为 QtopiaGames
   Context=Notes                           //上下文
   [Desktop Entry]
   Comment[]=A Text Editing Program        //备注
   Exec=textedit                           //可执行文件
   Icon=textedit/TextEditor                //图标
   Type=Application                        //文件类型,一般设为 Application，即可执行文件
   MimeType=text/*                         //Mimi 类型
   Name[]=Notes                            //文件名
   ```

3. 添加工程

   为了能够使新建工程在进行系统整体编译时能够被编译，需要将新建工程的工程路径添加到 QTOIADIR\src\general.pri 文件中，如 Camera 的工程路径“applications/camera \”，这样在系统进行整体编译读取相应的工程文件信息时，就可以找到相应的工程所在的路径。

   接下来就可以按照常规的方法进行编译了。

## 5.3 配置文件

在 qt 中为了编译等的方便，提供了 pro 文件等几种配置文件共研发者使用。

### 5.3.1 Pro文件

工程（pro）文件包含了 qmake 编译应用（泛指应用程序、库、插件）所需要的全部信息，对于每个应用而言，都有一个 pro 文件。简单的工程文件仅定义了标准变量来包含源文件和头文件信息，复杂的工程文件会采用控制流来调节编译过程。Qmake 在创建 Makefile.target 的时候会读取 pro 文件。

下面介绍下重要的语法信息并以实例说明：

1. 语法要点

   ➢ 变量

Qmake 在写 Makefile 文件时会在工程文件中查找变量信息，然后根据信息的内容来写 Makefile 文件。

- HEADERS

  工程中的头文件信息，qmake 一般会自动提取头文件信息，不需要研发人员手工添加。

- SOURCES

  工程中的源文件信息，qmake 一般会自动提取头文件信息，不需要研发人员手工添加。

- CONFIG

  工程配置信息，工程文件中最重要的一个变量，

| release | 生成的应用为 release 模式 |
|---|---|
| debug | 生成的应用为 debug 模式 |
| debug_and_release | 同时生成 debug 模式和 release 模式的应用 |
| build_all | 如果设了 debug_and_release，默认编译同时生成 debug 模式和 release 模式的应用 |
| ordered | 当设置了 subdirs 的 TEMPLATE，说明所列目录应按给定的顺序处理 |
| warn_on | 输出尽可能多的警告信息 |
| Warn_off | 输出尽可能少的警告信息 |

表7 CONFIG

- DESTDIR
  可执行文件或二进制文件的安装信息
- FORMS
  ui 文件信息，如果 ui 文件存在，则会调用 uic 来进行编译该文件。
- RESOURCES
  资源文件信息
- TEMPLATE
  说明编译过程输出的文件类型是应用程序、库、插件，qmake 会根据该变量生成适当的 Makefile 文件。

| TEMPLATE | 说明 |
|---|---|
| app(默认) | 创建一个应用程序 |
| lib | 创建一个库 |
| subdirs | 说明该目录存在子文件夹，Makefile 要搜索子文件夹的信息 |

表8 TEMPLATE

➢ 常用函数

- CONFIG(config)

通过设置范围条件，可以检查某些模块是否可用，如判断 OpenGL 模块是否可用：

```
CONFIG(opengl)
{
    message(Building with OpenGL suport);
}
else
{
    message(OpenGL support is not available);
}
```

- contains(variablename,value)

检查 variablename 中是否包含 value，如果包含，则返回 true，否则返回 false。如检查驱动中是否包含了网络驱动：

```
contains( drivers, network )
{
      message( "Configuring for network build..." )
      HEADERS += network.h
      SOURCES += network.cpp
}
```

- exists(filename)

测试文件 filename 是否存在。

- include(filename)

测试文件 filename 是否包含在当前工程中。如果没有包含，将该文件包含。

- for(iterate, list)

一个提供循环测试能力的函数。

- system(command)

在辅 Shell 中执行 command 命令。

2. 实例说明

下面为 QTOPIADIR 目录下的 project.pro 文件：

```
CONFIG+=ordered                                          //目录类型
SUBDIRS=src
build_qtopia:!enable_singleexec
{
    defineTest(append_examples)
    {
        qtopia_all.commands+=$$esc(\n\t)@$$MAKE -C examples
        export(qtopia_all.commands)
    }
    runlast(append_examples())
}
build_qtopiadesktop:SUBDIRS+=src/qtopiadesktop
!isEmpty(EXTRA_SUBDIRS):SUBDIRS+=$$EXTRA_SUBDIRS
enable_singleexec:SUBDIRS+=src/server
!win {                                                   //如果不是 Windows 平台
    clean_qtopia.commands=$$COMMAND_HEADER\  //设置清空脚本
      env QTOPIADIR=$$QTOPIADIR $$QTOPIA_DEPOT_PATH/scripts/clean_qtopia
    QMAKE_EXTRA_TARGETS+=clean_qtopia
    qtopia_distclean.depends+=clean_qtopia
}
win32:RMRF=rmdir /s /q                                   //设置 windows 下删除目录命令
else:RMRF=rm –rf                                         //设置 Linux 下的删除目录命令
cleanimage.commands=$$COMMAND_HEADER
PREFIXES=$(IMAGE) $(DIMAGE)
for(prefix,PREFIXES)                                     //循环测试
{
    win32:cleanimage.commands+=if exist $$prefix
    cleanimage.commands+=$$RMRF $$prefix $$LINE_SEP
}
cleanimage.commands+=$$DUMMY_COMMAND
QMAKE_EXTRA_TARGETS+=cleanimage
qtopia_install.depends+=cleanimage
runlast(append_install.commands=\$$qtopia_install.commands)
QMAKE_EXTRA_TARGETS+=append_install
!equals(QTOPIA_SDKROOT,$$QTOPIADIR) {                    //判断是否相等
    sdk_inst.commands=$(MAKE) sdk
    QMAKE_EXTRA_TARGETS+=sdk_inst
    qtopia_install.depends+=sdk_inst
    cleansdk.commands=$$COMMAND_HEADER
    win32:cleansdk.commands+=if exist $$prefix
    cleansdk.commands+=$$RMRF $(SDKROOT)
    QMAKE_EXTRA_TARGETS+=cleansdk
    check_sdk.depends+=cleansdk
}
CONFIG+=no_cleaninstall
cleaninstall.commands=
cleaninstall.depends=qtopia_install
QMAKE_EXTRA_TARGETS+=cleaninstall
cinstall.commands=
cinstall.depends=qtopia_install
QMAKE_EXTRA_TARGETS+=cinstall
```

下面是地址薄的 pro 文件，配置了与地址薄应用程序相关的信息如目标文件名、头文

件、源文件、编译所依赖的库、图片路径、桌面图标信息：

```
qtopia_project(qtopia app)                          //应用程序
TARGET=addressbook                                  //目标文件
CONFIG+=qtopia_main //提醒用户模块使用了 QTOPIA_ADD_APPLICATION、QTOPIA_MAIN 宏
HEADERS+=\                                          //头文件
    abeditor.h\
    imagesourcedialog.h\
    ablabel.h\
    contactsource.h\
    addressbook.h
SOURCES+=\                                          //源文件
    abeditor.cpp\
    imagesourcedialog.cpp\
    ablabel.cpp\
    addressbook.cpp\
    contactsource.cpp\
    main.cpp
phone {                                             //是否为手机版本
    !free_package|free_plus_binaries:depends(libraries/qtopiaphone)
    SOURCES += emaildialogphone.cpp
    HEADERS += emaildialogphone.h
} else {
    FORMS += emaildlg.ui
    HEADERS += emaildlgimpl.h
    SOURCES += emaildlgimpl.cpp
}
enable_cell {
    !enable_singleexec {
        SOURCES += ../../settings/ringprofile/ringtoneeditor.cpp
        HEADERS += ../../settings/ringprofile/ringtoneeditor.h
    }
}
TRANSLATABLES += emaildialogphone.cpp \   //本地化包含文件
                 emaildialogphone.h \
                 emaildlg.ui \
                 emaildlgimpl.h \
                 emaildlgimpl.cpp \
                 ../../settings/ringprofile/ringtoneeditor.cpp \
                 ../../settings/ringprofile/ringtoneeditor.h
depends(libraries/qtopiapim)                        //所依赖的库
service.files=$$QTOPIA_DEPOT_PATH/services/Contacts/addressbook   //配置文件路径
service.path=/services/Contacts                                   //配置文件安装路径
receiveservice.files=$$QTOPIA_DEPOT_PATH/services/Receive/text/x-vcard/addressbook
receiveservice.path=/services/Receive/text/x-vcard/
desktop.files=$$QTOPIA_DEPOT_PATH/apps/Applications/addressbook.desktop//桌面图标
desktop.path=/apps/Applications
desktop.hint=desktop
help.source=$$QTOPIA_DEPOT_PATH/help
help.files=addressbook*
help.hint=help
pics.files=$$QTOPIA_DEPOT_PATH/pics/addressbook/*                 //图标文件
pics.path=/pics/addressbook
pics.hint=pics
im.files=named_addressbook-*.conf
im.path=/etc/im/pkim
phoneservice.files=$$QTOPIA_DEPOT_PATH/services/ContactsPhone/addressbook
```

```
phoneservice.path=/services/ContactsPhone
qdlservice.files=$$QTOPIA_DEPOT_PATH/services/QDL/addressbook
qdlservice.path=/services/QDL
qdsservice.files=$$QTOPIA_DEPOT_PATH/etc/qds/Contacts
qdsservice.path=/etc/qds
qdsphoneservice.files=$$QTOPIA_DEPOT_PATH/etc/qds/ContactsPhone
qdsphoneservice.path=/etc/qds
INSTALLS+=service receiveservice desktop help pics im qdlservice qdsservice
enable_cell {
    INSTALLS+=phoneservice qdsphoneservice
}
pkg.desc=Contacts for Qtopia.
pkg.domain=pim,window,qdl,qds,beaming,phonecomm,pictures,msg,docapi,cardreader,camera,
pictures,mediarecorder
```

其中，TARGET 变量表示该应用程序的文件名，如果在 CONFIG 中没有提供编译方式选项，Qt 会按照默认的 quicklauncher 方式编译，生成的应用程序名为 libaddressbook.so，如果用户不希望采用 quicklauncher 方式编译，可以在 CONFIG 选项中添加 CONFIG -=quicklauncher。在工程文件中，CONFIG 表示配置信息选项，当选项为 qtopia_main 时，指用户模块使用了 QTOPIA_ADD_APPLICATION 和 QTOPIA_MAIN 宏。

在研发人员研发应用模块时，常需要提供一定的配置文件，如应用程序的 desktop 图标，某些应用程序在启动时播放的音乐等，Qt 在工程文件中为这类需求同样提供了支持。如在 desktop 配置信息中，工程文件指明了 desktop 文件为 QTOPIA_DEPOT_PATH 路径下的 apps/Applications/addressbook.desktop。安装路径为/apps/Applications 文件夹。需要说明的是，类似的语法规则为配置文件的安装提供了重要的参考，研发人员在为应用模块提供配置文件时，必须在相应的工程文件中指明欲安装的文件和安装路径，同时需要在 INSTALLS 选项中加入欲安装的配置信息如 service、receiveservice、desktop 等等，如果存在多个配置信息，应该用空格将其隔开。

需要说明的是，利用编译工具如调用 make install 来安装应用程序和库对类 Unix 操作系统而言是种普遍的操作方式，基于这个因素，qmake 提供了安装集的概念，在工程文件中 INSTALLS 选项就表示了该应用程序的安装集。如果需要在安装过程中进行更多的控制，就需要对该安装对象的 extra 选项进行定义，如：

```
unix:documentation.extra = mv master.doc toc.doc
```

表示在安装 documentation 对象时，要首先执行重命名命令，其中 unix 表示命令执行域为 Unix 平台。

另外，当应用程序依赖于特定的库文件时，需要在工程文件中给出声明，如地址薄依赖于 libraries/qtopiapim 库文件。就可以这样声明：depends(libraries/qtopiapim)。

### 5.3.2 Pri文件

pri 文件也是一类重要的配置文件，general.pri 配置了 Qtopia 包含的不同类型模块的模块，如果希望新添加的模块能够被 Qtopia 识别，需要将模块信息添加到该文件中，如果因为某种原因，不希望某一模块在完全编译时被编译，则在该文件中将该模块注释掉即可。下面为 QTOPIADIR/ src 下的 general.pri 文件：

```
QTE_PROJECTS=\
    tools/qtopiacore/moc\
    tools/qtopiacore/uic\
    tools/qtopiacore/rcc\
    libraries/qtopiacore/corelib\
    libraries/qtopiacore/gui\
    libraries/qtopiacore/network\
    #libraries/qtopiacore/opengl\
    libraries/qtopiacore/sql\
    libraries/qtopiacore/xml\
```

```
        plugins/qtopiacore
!equals(QTE_MINOR_VERSION,1):QTE_PROJECTS+=libraries/qtopiacore/svg
PROJECTS*=$$QTE_PROJECTS
PROJECTS*=\
        qt\
        server\
        libraries/qtopia\
        libraries/qtopiail\
        libraries/qtopiacomm\
        libraries/qtopiaaudio\
PROJECTS*=\
        libraries/qtopiacomm/bluetooth\
        libraries/qtopiacomm/ir\
        libraries/qtopiacomm/network\
        libraries/qtopiacomm/obex\
        libraries/qtopiacomm/serial\
        libraries/qtopiacomm/vpn
PROJECTS*=\
        3rdparty/libraries/zlib\                    //第三方模块
        3rdparty/libraries/alsa\
        3rdparty/libraries/md5\
        3rdparty/libraries/tar\
        #obex
        3rdparty/libraries/openobex\
        3rdparty/libraries/inputmatch\
        3rdparty/libraries/sqlite\
        3rdparty/applications/sqlite
enable_ssl:PROJECTS*=\
        3rdparty/libraries/openssl/crypto\
        3rdparty/libraries/openssl/ssl
PROJECTS*=\
        plugins/inputmethods/keyboard\              //插件模块
        plugins/inputmethods/dockedkeyboard
!media:PROJECTS*=\
        libraries/qtopiapim
enable_qtopiabase {
        PROJECTS*=libraries/qtopiabase
        PROJECTS-=libraries/qtopiail
}
qtopiatest:PROJECTS*=\
        libraries/qtopiatest/qtesttools/host \
        libraries/qtopiatest/qtesttools/target \
        tools/validator \
        tools/validator/3rdparty/qscintilla
qtopiatest:!enable_singleexec:PROJECTS*=\
        libraries/qtopiatest/qunittest
qtopiatest:PROJECTS*=\
        libraries/qtopiatest/qsystemtest \
        libraries/qtopiatest/qtestslave \
        libraries/qtopiatest/qsystemtestslave \
        libraries/qtopiatest/qtopiasystemtestslave \
        libraries/qtopiatest/qtopiaservertestslave \
        libraries/qtopiatest/overrides
!platform:!media {
        PROJECTS*=\
        applications/addressbook \                  //应用程序模块
        applications/datebook \
```

```
        applications/todo\
        applications/calculator \
        applications/camera \
        applications/clock \
        applications/mediarecorder \
        applications/photoedit \
        applications/textedit \
        games/qasteroids\
        games/fifteen\
            games/snake\
            games/minesweep\
            plugins/content/id3 \
            plugins/content/exif
        build_helix {
            PROJECTS*=\
                3rdparty/libraries/helix \
                3rdparty/libraries/libtimidity \
                3rdparty/plugins/codecs/libtimidity \
                plugins/mediadevices/builtin \
                libraries/qtopiamedia \
                tools/mediaserver \
                applications/mediaplayer
        }
}
PROJECTS*=\
        settings/appearance\                            //设置模块
        settings/language \
        settings/logging \
        settings/network \
        settings/systemtime \
        settings/worldtime \
        settings/light-and-power \
        settings/packagemanager\
        applications/helpbrowser \
        applications/sysinfo \
        tools/qcop \
        tools/vsexplorer \
        tools/symlinker \
        tools/qdawggen \
        tools/dbmigrate
!build_helix:PROJECTS*=tools/qss
!media:PROJECTS*=\
        settings/security
!no_quicklaunch|enable_singleexec:PROJECTS*=tools/quicklauncher
PROJECTS*=tools/content_installer
build_libamr:PROJECTS*=\
        3rdparty/libraries/amr\
        3rdparty/plugins/codecs/libamr
PROJECTS*=\
        plugins/network/lan \
        plugins/network/dialing \
        plugins/qtopiacore/iconengines/qtopiaiconengine \
        plugins/qtopiacore/iconengines/qtopiasvgiconengine
media {
        PROJECTS*=\
        applications/camera \
        applications/clock \
```

```
        applications/mediarecorder \
        applications/photoedit \
            plugins/content/id3
        build_helix {
            PROJECTS*=\
                3rdparty/libraries/helix \
                libraries/qtopiamedia \
                applications/mediaplayer
        }
}
PROJECTS*=\
        libraries/qtopiaprinting \
        tools/printserver
enable_bluetooth:PROJECTS*=\
        plugins/qtopiaprinting/bluetooth
enable_infrared:PROJECTS*=\
        settings/beaming
enable_bluetooth:PROJECTS*=\
        settings/btsettings \
        plugins/network/bluetooth
enable_dbus {
        PROJECTS*=\
        3rdparty/libraries/dbus \
        3rdparty/libraries/qtdbus
}
enable_dbusipc {
        PROJECTS*=\
        3rdparty/applications/dbus \
        tools/qtopia-dbus-launcher
}
enable_singleexec:PROJECTS*=\
        plugins/qtopiacore/imageformats/jpeg\
        plugins/qtopiacore/imageformats/mng\
        plugins/qtopiacore/imageformats/svg
PROJECTS*=\
        libraries/handwriting\
        settings/handwriting\
        3rdparty/plugins/inputmethods/pkim
THEMES *= smart
phone {
        PROJECTS*=\
            3rdparty/libraries/gsm\
            libraries/qtopiasmil\
            settings/ringprofile\
            settings/speeddial\
            plugins/codecs/wavrecord\
        enable_infrared:PROJECTS*=\
            settings/beaming
        THEMES*=\
            crisp\
            qtopia\
            portal
}
!platform {
        PROJECTS*=settings/words
        !media:PROJECTS*=\
            libraries/qtopiamail\
```

```
        applications/qtmail
}
enable_samples:PROJECTS+=settings/serverwidgets
```

### 5.3.3 Desktop文件

当系统启动时，在配置好 Theme 等文件后，会首先查看 apps 文件夹，搜集 Desktop 文件所包含的信息。Desktop 文件保含了应用程序执行的重要信息，配置了模块的类型、模块名称、桌面图标、可执行文件名等，为系统进一步搜索并关联相关文件提供了支持。下面为 QTOPIADIR/apps/Applications 下的 addressbook.desktop 文件：

```
[Translation]
File=QtopiaApplications
Context=Contacts                        //模块桌面名称
Comment[Desktop Entry/Name]=Use soft hyphen (char U00AD) to indicate hyphenation
[Desktop Entry]
Comment[]=An Address Book Program
Exec=addressbook                        //可执行文件名
Icon=addressbook/AddressBook            //桌面图标
Type=Application                        //文件类型
MimeType=text/x-vcard                   //Mime 类型
Name[]=Contacts
```

addressbook.desktop

### 5.3.4 Conf文件

下面为 QTOPIADIR/ etc 目录下的 defaultbuttons-phone.conf 文件：

```
[Translation]
File=QtopiaDefaults
Context=Buttons
[Menu]
Rows=4
Columns=3
Map=123456789*0#                        //定义映射
Default=5
1=Applications/camera.desktop
2=Applications/datebook.desktop
3=Applications
4=Applications/qtmail.desktop
5=Applications/addressbook.desktop
6=Games
7=Settings/Beaming.desktop
8=Applications/simapp.desktop{@/Telephony/Status/SimToolkitAvailable},Applications/calcul
ator.desktop
9=Settings
*=Applications/mediarecorder.desktop
0=Applications/todolist.desktop
#=Documents
[SoftKeys]                  //软键
Count=3
Key0=Context1
Key1=Select
Key2=Back
```

```
[SystemButtons]
Count=5
Key0=Context1
Key1=Select
Key2=Back
Key3=Call
Key4=Hangup
[TextButtons]
Buttons=0123456789*#
Hold0='0
Hold1='1
Hold2='2
Hold3='3
Hold4='4
Hold5='5
Hold6='6
Hold7='7
Hold8='8
Hold9='9
Hold*=symbol
Hold#=mode
Tap0=space
Tap1="\".,'?!-@:1"
Tap2="\"a\xe4\xe5\xe6\xe0\xe1\xe2\x62\x63\xe7\x32"
Tap3="\"de\xe8\xe9\xea\x66\x33"
Tap4="\"ghi\xec\xed\xee\x34"
Tap5="\"jkl5"
Tap6="\"mn\xf1o\xf6\xf8\xf2\xf3\x36"
Tap7="\"pqrs\xdf\x37"
Tap8="\"tu\xfc\xf9\xfav8"
Tap9="\"wxyz9"
Tap*=modify
Tap#=shift
[LocaleTextButtons]
Buttons=23456789
Tap2[]='abc
Tap3[]='def
Tap4[]='ghi
Tap5[]='jkl
Tap6[]='mno
Tap7[]='pqrs
Tap8[]='tuv
Tap9[]='wxyz
[PhoneTextButtons]
Buttons=*#
Tap*='*+pw
Hold*=+
Tap#='#
Hold#=mode
[Device]
PrimaryInput=Keypad                    //如果是触摸屏，设为 Touchscreen
```

## 5.4 模拟器配置

模拟器是一组应用程序，它运行于一个 OS 之上（linux 或 windows），提供对目标板功能的模拟，包括中断系统，RAM，FLASH 系统，以及目标 OS 到宿主 OS 任务的映射。可以有

效的加快研发进度。

通过 Qtopia 模拟器方式模拟目标机，在启动时有如下的几个步骤：

- 设置环境变量
- 选择一个合适的皮肤
- 检查声卡
- 检查 HOME 路径
- 运行 QVFB
- 运行手机模拟器（可选）
- 运行 Qtopia

所谓皮肤即 QVFB 的观感，为一个 xml 文件集，这依赖于进行 Qtopia 安装时候的安装包，当然你也可以自己定义自己想要的皮肤。

## 5.5 双屏显示

在当前的移动终端中，除了直板手机和滑盖手机外，还有翻盖手机的存在，在翻盖手机中，常常为了用户的操作方便，进行了双屏设计。在 Qtopia 中，提供了两种方式来对双屏进行支持：

1. 将辅屏作为一个帧缓冲，由 Qtopia 直接控制

   在这种方式中，主屏和辅屏都作为一个独立的帧缓冲来实现，其分辨率可以通过 QWS_DISPLAY 在 defaultbuttons.conf 设置。

   举例如下：

   单屏设备：QWS_DISPLAY= LinuxFb:mmWidth34:mmHeight44:0

   双屏设备：QWS_DISPLAY =multi: LinuxFb:/dev/fb0 LinuxFb:/dev/fb1:offset=0,320:1 :0

   在实际的实现中，主屏和辅屏是按照虚拟布局的方式进行的。主屏的原点默认位于（0,0）；辅屏则位于主屏之下，如果主屏按照 240*320 显示，则辅屏的原点位置在（0,320）。

   在 Qtopia 中，系统利用 themes 来设置辅屏的标题和主界面，themes 文件位于 QTOPIADIR/etc/themes/下，分别为 secondarytitle.xml、secondaryhome.xml。

2. 辅屏不被 Qtopia 控制

   如果不准备使用 Qtopia 多屏支持。可以利用 QPhoneStatus 类来获取手机的状态信息供辅屏显示状态信息。QPhoneStatus 类能俘捉道系统的来电、短信、电量等变化信息。

# 第 6 章　启动过程

千里之行，始于足下。

《老子·六十四章》

## 6.1 C/S模型

X Window 系统中经常被混淆的一个关键概念是服务器与客户之间的区别。在计算机网络中，服务器指的是向其他机器传输文件的机器，然而X Window系统中的服务器起的是完全不同的作用。X Window系统中，服务器是从用户处接收输入并向用户显示输出的硬件（或）软件。客户是指连接到服务器的应用程序。

在X Window系统中，服务器与客户可以存在于同一个工作站或者计算机上，使用进程间通信（IPC）机制，如UNIX 管道与套接字，在它们之间传递信息。

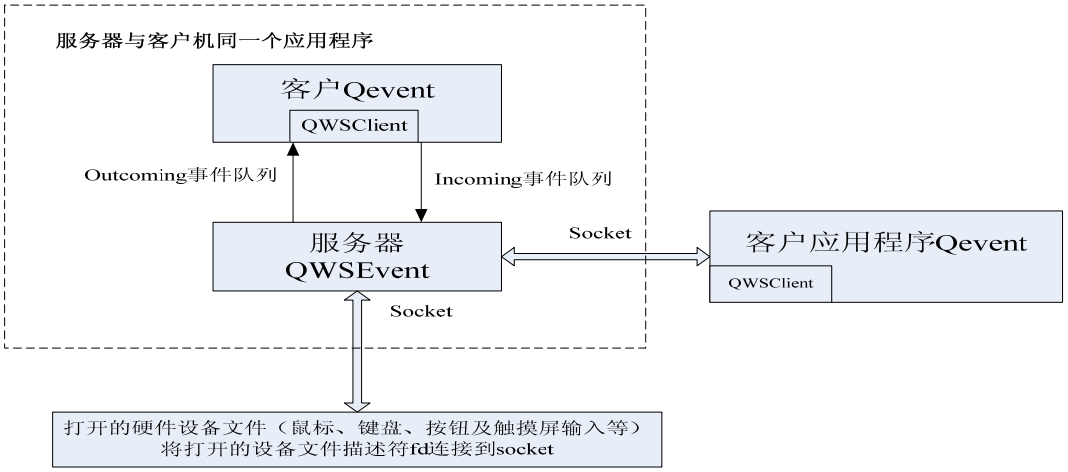


图38.　Qt 的 C/S 架构

由于Qt采用了C/S架构，当一个Qtopia Core应用程序运行时，如果它自己不是一个服务器端程序，必须有一个服务器端程序在运行，它会作为一个客户端程序连接到服务器端应用程序中。服务器端程序和客户端应用程序负有不同的责任，服务器端程序负责指针处理、字符输入和显示输出。除此之外，服务器端程序还负责控制着光标的观感，而客户端程序则执行特定的操作。

在实际的Qt实现中，服务器端程序是作为一个QWSServer的实例来出现的，而客户端程序则被实现为一个QWSClient的实例。

当有事件发生时，客户端程序会将事件传递给服务器端程序进行处理，然后服务器端程序会将处理结果返回给相应的客户端程序。由客户端程序在屏幕上进行渲染（render）。

1. 服务器端

对于多进程应用来说，服务器端程序将来自鼠标及键盘等硬件输入的数据通过socket以事件形式发送给客户端程序，另外还把服务器程序对窗口的全局管理方面的事件发送给客户端程序，同时，服务器端程序还通过socket接收来自客户端程序的消息，并激活相关操作函数。对于单进程应用而言，因为在同一个进程空间中，所以只需要通过一个队列结构即可传递事件。

2. 客户端

对于多进程应用来说，客户端程序将一些需要服务器端程序处理的窗口管理及鼠标键盘命令数据通过socket以事件的形式发送给服务器程序，同时，客户端程序还通过socket接收

来自服务器端程序的事件，分发给客户端程序相关的窗口进行处理，对于单进程应用而言，因为在同一个进程空间中，所以只需要通过一个队列结构即可传递事件。

Qtopia的C/S与一般的C/S模型不完全一致，其特点是客户端程序和服务器端程序都是QT应用程序，服务器端程序并不是独立运行，而是附加在客户端应用程序上运行的，在代码中，客户端和服务器的代码在上层都基本上在一个函数中处理，用一些变量来标识是否是服务器用的代码。

Qtopia的C/S模型被封装在Qtopia Core中，通过QApplication建立模型的各种关系，用Qtopia Core类库完成各种通信及操作。

在Qtopia的C/S模型中，第一个启动的应用程序会启动服务器端程序，后面启动的应用程序仅包括客户端程序。

QWSServer类通过发送或接收QWS协议事件，并调用QWSDisplay来完成诸如窗口区域分配的工作。对于QT/Embedded服务器进程而言，QWSServer类被QApplication类实例化。研发者不需要自己构造实例。

## 6.2 服务器端的启动

Qtopia 服务器是应用程序的管理器，作为一个设置屏幕上的主窗口，它提供服务器窗口，负责启动应用程序，协调管理各个应用程序的运行，打开文档，并显示应用程序或硬件状态等。

Qtopia 服务器窗口由 Launcher、LauncherTab、LauncherTabBar、LauncherView、LauncherTabWidget、TaskBar 等类组成。LauncherTabWidget 窗口分为 LauncherTabBar 和 LauncherView，

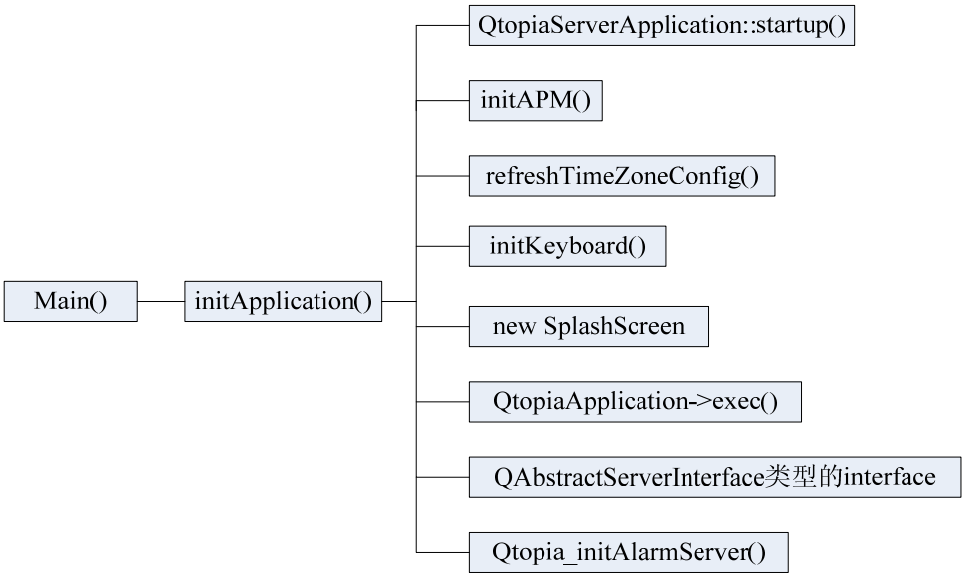


图39. Qtopia 服务器 main 函数调用层次图

Qtopia 服务器由多个独立的各司其职的任务（Task）组成，QtopiaServerApplication 和 QAbstractServerInterface 是其中最主要的任务。在 Qtopia 服务器中，QtopiaServerApplication 作为 QtopiaApplication 的一个实例出现，负责启动和关闭 Qtopia 服务器，对整个服务器应用程序的事件、信道及运行进行全局性管理，在整个系统中扮演一个核心控制器的角色；QAbstractServerInterface 负责与用户的直接交互。

跟服务器相关的类有：

QWSServer：管理鼠标和键盘设备，初始化显示设备、激活或关闭绘制屏幕、获得客户窗口的链表，并且设置窗口的方框大小等。

QWSWindow：保存 QWSClient 类指针及窗口的名字、标题、区域及可视化等属性的参数，可设置或得到窗口的区域及可视属性。

QWSClient：代表一个客户端，它创建 socket，并通过 socket（多线程服务器）发送事件，或通过全局的 QWSEvent 对象（单线程服务器）发送事件，它还发送窗口属性改变相关的事件（如焦点、窗口最大化、选中窗口或区域等）发送事件的一个基本函数是 sendEvent(QWSEvent* event)。

QWSEvent：派生于 QWSProtocolItem 类，描述了服务器事件的种类：NoEvent,Connected,Mouse,Focus,Key,Region,Creation,PropertyNotify,PropertyReply,SelectionClear,SelectionRequest,SelectionNotify,MaxWindowRect,QCopMessage,WindowOperation,IMEvent,IMQuery,IMInit,Embed,NEvent。QWSProtocolItem 类保存事件数据的原始流，并提供成员函数 write 将原始流数据写入 socket，提供成员函数 read 从 socket 读出原始流数据。

各种具体服务器事件从 QWSEvent 派生。

### 6.2.1 服务器端的初始化

QWSServer 通过调用 QWSServerPrivate::initServer(int flags)来完成服务器的初始化过程。

```
void QWSServerPrivate::initServer(int flags)
{
    Q_Q(QWSServer);
    Q_ASSERT(!qwsServer);
    qwsServer = q;
    qwsServerPrivate = this;                          //将此实例赋给全局变量
    disablePainting = false;
#ifndef QT_NO_QWS_MULTIPROCESS                        //如果是多线程
    //利用 qws_qtePipeFilename()获得用于 C/S 通信的管道的文件名，然后创建本地 Socket，//绑定到文件 qws_qtePipeFilename()上并监听 socket
    ssocket = new QWSServerSocket(qws_qtePipeFilename(), q);
    QObject::connect(ssocket, SIGNAL(newConnection()), q, SLOT(_q_newConnection()));
    if ( !ssocket->isListening())
    {
        perror("QWSServerPrivate::initServer: server socket not listening");
        qFatal("Failed to bind to %s", qws_qtePipeFilename().toLatin1().constData());
    }
    struct linger tmp;
    tmp.l_onoff=1;
    tmp.l_linger=0;
    /*
    C函数，设置socket选项，socket()表示一个打开的套接口描述子，选项可能存在多个协议级别上，SOL_SOCKET 表示处理 socket 级别上的选项，选项名为 SO_LINGER，值为tmp
    */
    setsockopt(ssocket->socketDescriptor(),SOL_SOCKET,SO_LINGER,
    (char*)&tmp,sizeof(tmp);
    signal(SIGPIPE, ignoreSignal);                    //安装处理函数到信号 SIGPIPE
#endif
    focusw = 0;
    mouseGrabber = 0;
    mouseGrabbing = false;
    keyboardGrabber = 0;
    keyboardGrabbing = false;
#ifndef QT_NO_QWS_CURSOR
    haveviscurs = false;
    cursor = 0;
    nextCursor = 0;
#endif
#ifndef QT_NO_QWS_MULTIPROCESS
```

```
        if (!geteuid())
        {
            #if !defined(Q_OS_FREEBSD) &&
            !defined(Q_OS_SOLARIS) && !defined(Q_OS_DARWIN)
            if(mount(0,"/var/shm", "shm", 0, 0))     //挂载共享文件内存系统到/var/shm
            {
            }
    #endif
        }
    #endif
        // no selection yet
        selectionOwner.windowid = -1;
        selectionOwner.time.set(-1, -1, -1, -1);
        //打开一个到帧缓冲服务器的连接，即建立显示相关驱动，framebuffer 等
        openDisplay();
        screensavertimer = new QTimer(q);            //建立屏幕保护定时器
        screensavertimer->setSingleShot(true);
        QObject::connect(screensavertimer,SIGNAL(timeout()),q,
        SLOT(_q_screenSaverTimeout()));
        _q_screenSaverWake();                        //唤醒屏幕保护
        /*
            创建客户端 QWSClient，参数-1 表示服务器，全局变量 CSocket 为 0，由于已建
            了 QWSServerSocket，所以不需要再建立 Socket，QMap<int,QWSClient*>
            ClientMap
        */
        clientMap[-1] = new QWSClient(q, 0, 0);
        if (!bgBrush)
            bgBrush = new QBrush(QColor(0x20, 0xb0, 0x50));
        initializeCursor();                          //设置鼠标箭头形状及在屏幕中间位置
        // input devices
        if (!(flags&QWSServer::DisableMouse))
        {
        /*
            打开鼠标，通过 QSocketNotifier 类，利用信号与槽机制将打开的设备文件描述
            符 fd 与读取处理函数连接起来，通过信号 activated（int）来激活这个处理函数。
        */
            q->openMouse();
        }
    #ifndef QT_NO_QWS_KEYBOARD
        if (!(flags&QWSServer::DisableKeyboard))
        {
            q->openKeyboard();                       //打开键盘
        }
    #endif
    #if    !defined(QT_NO_SOUND)    &&    !defined(QT_EXTERNAL_SOUND_SERVER)
    && !defined(Q_OS_DARWIN)
        soundserver = new QWSSoundServer(q);     //创建音频服务器
    #endif
    }
```

在 QWSServer 初始化的过程中，会首先利用 qws_qtePipeFilename()获取 Qtopia 用于 C/S 通信的管道名，通常管道文件位于/tmp 下，然后利用该管道创建套接字 ssocket 并监听来自该套接字的信息；然后打开显示设备，并设置设备的宽、高等信息；接着创建服务器并设置背景刷，然后初始化光标位置在屏幕的正中央；然后根据环境变量 QWS_MOUSE_PROTO 打开鼠标设备；然后根据环境变量 QWS_KEYBOARD 打开键盘设备；创建音频服务器。

### 6.2.2 服务器端的启动

```
int initApplication( int argc, char ** argv )
{
    if(!QtopiaServerApplication::startup(argc, argv))//启动服务器 task
        qFatal("Unable to initialize task subsystem.  Please check '%s' exists and its content
        is valid.", QtopiaServerApplication::taskConfigFile().toLatin1().constData());
#ifdef QPE_OWNAPM
    initAPM();                                  //初始化电源管理
#endif
    refreshTimeZoneConfig();                    //刷新时区配置
    qLog(Performance) << "QtopiaServer : " << "Refresh time zone information : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    initKeyboard();                             //初始化键盘
    qLog(Performance) << "QtopiaServer : " << "Keyboard initialisation : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
#if defined(QTOPIA_ANIMATED_SPLASH)
    SplashScreen *splash = new SplashScreen;
    splash->splash(QString(":image/splash"));//启动 splash 屏幕
    qLog(Performance) << "QtopiaServer : " << "Splash screen creation : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
#endif
    // Load and show UI
    QAbstractServerInterface *interface =
        qtopiaWidget<QAbstractServerInterface>(0, Qt::FramelessWindowHint);
#if defined(QTOPIA_ANIMATED_SPLASH)
    splash->setReplacement(interface);
#else
    if(interface)
        interface->show();                      //导出并显示 UI
#endif
    qLog(Performance) << "QtopiaServer : " << "Display the server : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    //初始化 alarm 服务器，从文件中读出定时器时间数据，并设置定时器.
    Qtopia_initAlarmServer();
        qLog(Performance) << "QtopiaServer : " << "AlarmServer startup : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    qLog(Performance) << "QtopiaServer : " << "Entering event loop : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
//      int rv =    a.exec();
    //Can't do this either, exec is static int rv =    qApp->exec();
    int rv = static_cast<QtopiaApplication *>(qApp)->exec();
    qLog(QtopiaServer) << "exiting...";
    qLog(Performance) << "QtopiaServer : " << "Event loop exited : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    delete interface;
    qLog(Performance) << "QtopiaServer : " << "Leaving initApplication : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    return rv;
}
```

在初始化应用的过程中，会首先启动服务器，然后进行电源管理初始化、时区配置刷新、键盘初始化，接着就会显示启动界面，显示空闲 UI 等。完成系统的初始化。下面是服务器的启动过程。

```
bool QtopiaServerApplication::startup(int &argc, char **argv)
{
```

```
    QtopiaServerTasksPrivate *qst = qtopiaServerTasks();
    Q_ASSERT(qst);
    qLog(QtopiaServer) << "Starting tasks...";
    qLog(Performance) << "QtopiaServer : " << "Starting tasks... : "
                      << qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    qst->argc = &argc;
    qst->argv = argv;
    QList<QtopiaServerTasksPrivate::Task *> startupOrder;
    if(!qst->determineStartupOrder(startupOrder))          // 确定启动顺序
        return false;
    for(int ii = 0; ii < startupOrder.count(); ++ii)
    {
        qLog(QtopiaServer) << "Starting task" << startupOrder.at(ii)->name.toByteArray();
        qLog(Performance) << "QtopiaServer : " << "Starting task" <<
        startupOrder.at(ii)->name.toByteArray()
                          << " : " <<
    qPrintable( QTime::currentTime().toString( "h:mm:ss.zzz" ) );
    qst->startTask(startupOrder.at(ii), false);          //按顺序启动 tasks
    }
    return true;
}
```

## 6.3 应用程序的启动

在 Qtopia 中，应用程序的启动主要由 Launcher 系统来完成，Launcher 系统主要由 LauncherTabWidget 、LauncherView、Launcher 和 QContent 等组成，其中 LauncherTabWidget 是一个 TabWidget，有若干个 LauncherView 窗口组成，单个 LauncherView 则包含了同种类型的应用程序的所有启动图标，当发生触发事件时，系统将会调用 Launcher 的相应接口依赖 QContent 处理这些事件。下面是 Launcher 系统的类图。

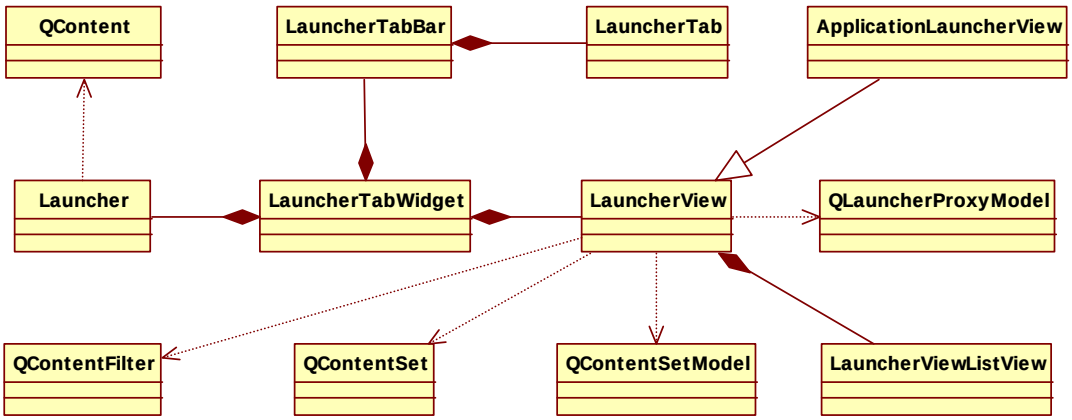


图40. Launcher 系统

在 LauncherView 视图窗口中，当用户点击图标时会产生 clicked(const QModelIndex &)信号，这个信号与 LauncherView 类中的槽函数 temClicked(const QModelIndex & index)连接起来，函数 temClicked(const QModelIndex & index)确认触发按键是否为鼠标左键，发送 clicked(model->content(bpModel->mapToSource(index)))信号。

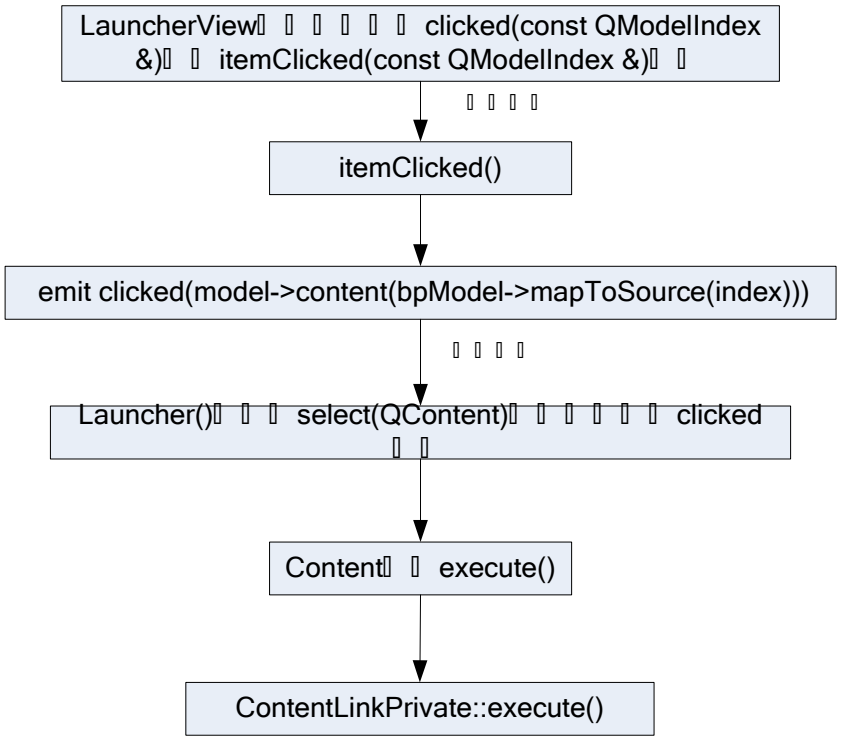


图41. 应用程序启动过程

下面是 LauncherView::LauncherView()的处理过程：

```
LauncherView::LauncherView( QWidget* parent, Qt::WFlags fl )
: QWidget(parent, fl), icons(NULL), model(NULL), nColumns(1),
busyTimer(0), bpModel(NULL)
{
    QVBoxLayout *vbl = new QVBoxLayout(this);
    vbl->setMargin(0);
    vbl->setSpacing(0);
    icons = new LauncherViewListView(this);                    //设置视图
    icons->setItemDelegate(new LauncherViewDelegate(icons));   //设置 Delegate
    vbl->addWidget(icons);
    setFocusProxy(icons);                                      //设置 FocusProxy
    QtopiaApplication::setStylusOperation(icons->viewport(),
    QtopiaApplication::RightOnHold );
    icons->setFrameStyle( QFrame::NoFrame );
    icons->setResizeMode( QListView::Fixed );
    icons->setSelectionMode( QAbstractItemView::SingleSelection );
    icons->setSelectionBehavior( QAbstractItemView::SelectItems );
    //icons->setUniformItemSizes( true );
    icons->setLayoutMode(QListView::Batched);
//      icons->setWordWrap( true );
    contentSet = new QContentSet(this);
    contentSet->setSortOrder(QStringList() << "name");         //设置序号
    bpModel = new QLauncherProxyModel(this);
    model = new QContentSetModel(contentSet, bpModel);
    bpModel->setSourceModel(model);
//  setViewMode(QListView::IconMode);
    setViewMode(QListView::ListMode);
    connect( icons, SIGNAL(clicked(const QModelIndex &)),
                SLOT(itemClicked(const QModelIndex &)));       //处理图标单击事件
```

```
        connect( icons, SIGNAL(activated(const QModelIndex &)),      //处理图标激活事件
                SLOT(returnPressed(const QModelIndex &)) );
        connect( icons, SIGNAL(pressed(const QModelIndex &)),        //处理图标按下事件
                SLOT(itemPressed(const QModelIndex &)));
    #ifndef QTOPIA_PHONE
        setBackgroundType( Ruled, QString() );
    #endif
        icons->setModel(bpModel);
    }
```

下面是 LauncherView::itemClicked()的处理过程：

```
    void LauncherView::itemClicked(const QModelIndex & index)
    {
        if(QApplication::mouseButtons () == Qt::LeftButton)              //左键事件
        {
            icons->setCurrentIndex( index );
            /*
                发射信号，model 为 QcontentSetModel 类型，content()函数返回对应于
                QModelIndex 的 content。
            */
            emit clicked(model->content(bpModel->mapToSource(index)));
        }
    }
```

在 LauncherView 发出 clicked(QContent)信号后，会被 LauncherTabWidget 接收，然后 LauncherTabWidget 会再次发出 clicked(QContent)信号，在 Launcher::select()中进行处理。Launcher 的初始化过程如下：

```
    Launcher::Launcher(): QWidget(0, Qt::FramelessWindowHint), categories(0)
    {
        setAttribute(Qt::WA_GroupLeader);
        tabs = 0;
        tb = 0;
        delayedAppLnk = 0;
        tid_today = startTimer(3600*2*1000);
        last_today_show = QDate::currentDate();
        setWindowTitle( tr("Launcher") );
        // we have a pretty good idea how big we'll be
        setGeometry( 0, 0, qApp->desktop()->width(), qApp->desktop()->height() );
        tb = new TaskBar;                                    //创建任务栏
        QVBoxLayout *vb = new QVBoxLayout(this);
        vb->setMargin(0);
        vb->setSpacing(0);
        tabs = new LauncherTabWidget( this );                //创建 TabBar
        vb->addWidget(tabs);
        WindowManagement::instance()->dockWindow( tb, WindowManagement::Bottom );
        qApp->installEventFilter( this );                    //安装事件过滤器
        connect( qApp, SIGNAL(authenticate(bool)), this, SLOT(askForPin(bool)) );//检查密码
        connect( tb, SIGNAL(tabSelected(const QString&)),
            this, SLOT(showTab(const QString&)) );     //显示选择的 Tab
        connect( tabs, SIGNAL(selected(const QString&)),
            this, SLOT(viewSelected(const QString&)) );//显示标题
        connect( tabs, SIGNAL(clicked(QContent)),
            this, SLOT(select(QContent)));                   //clicked(QContent)信号在这里被连接
        connect( tabs, SIGNAL(rightPressed(QContent)),
            this, SLOT(properties(QContent)));      //显示视图中选中图标对应的属性对话框
        QtopiaChannel* sysChannel = new QtopiaChannel( "QPE/System", this );
        connect( sysChannel, SIGNAL(received(const QString&,const QByteArray&)),
```

```
                this, SLOT(systemMessage(const QString&,const QByteArray&)) );
        // 文档视图 all documents
        QImage img( QImage( ":image/qpe/DocsIcon" ) );
        int smallIconSize = qApp->style()->pixelMetric(QStyle::PM_SmallIconSize);
        QPixmap pm = QPixmap::fromImage(img.scaled( smallIconSize, smallIconSize ));
        // It could add this itself if it handles docs
        // 创建文档视图 akennedy
        tabs->newView("Documents", pm, tr("Documents") )->setToolsEnabled( true );
        QTimer::singleShot( 0, tabs, SLOT(initLayout()) );
        // Setup all types
        categories = new QCategoryManager("Applications", this);
        QObject::connect(categories, SIGNAL(categoriesChanged()),
                         this, SLOT(categoriesChanged()));
        categoriesChanged();

    }
```

Launcher::select()对接收到的 clicked(QContent)信号的处理过程如下：

```
    void Launcher::select( const QContent *appLnk )
    {
        if ( appLnk->type() == "Folder" ) { // No tr
            // Not supported: flat is simpler for the user
        } else {
           if ( appLnk->executableName().isNull() ) {
                if ( ! delayedAppLnk ) {
                    delayedAppLnk = new QContent(*appLnk);
                    QTimer::singleShot( 0, this, SLOT(delayedSelect()) );
                }
            } else {
                tabs->setBusy(true);
                emit executing( appLnk );          //发射 executing( appLnk )
                appLnk->execute();                 //执行应用程序
            }
        }
    }
```

当 QContent 的 execute()函数被调用后，它会再调用其私有类的 execute()进行处理，ContentLinkPrivate::execute()的处理过程如下：

```
    void ContentLinkPrivate::execute(const QStringList& args) const
    {
    #ifdef Q_WS_QWS
        if (um == QContent::Application)           //判断是否为应用程序
        {
            qLog(DocAPI) << "ContentLinkPrivate::invoke" << cPath << args;
            Qtopia::execute( cPath, args.count() ? args[0] : QString::null );
        }
        else
        {
            QMimeType mt(type());
            QContent app = mt.application();
            if ( app.isValid() ) {
                QStringList a = args;
                if ( !linkFile().isNull() && QFile::exists( linkFile() ) )
                    a.append(linkFile());
                else
                    a.append(file());
                app.execute(a);
            }
```

```
        }
    #else
        Q_UNUSED(args)
    #endif
    }
```

需要说明的是，Qtopia 利用 Qcontent 来表述最终用户能看到或者管理的资源如：流、文件、DRM 控制文件、临时文件、容器中的条目等，并控制对资源中元数据的接入。

# 第 7 章　风格与主题

**行百里者，半于九十。**

《战国策·秦策五》

在移动终端的研发过程中，很多时候为了更快的推出新的产品，大多数缺乏核心技术的中小终端厂商都会基于采用的平台厂商提供的参考设计进行研发，但这样的研发存在的问题是，不同厂商的产品之间出现了越来越多的同质现象。对有志于进一步从竞争激烈的市场中脱颖而出的厂商而言，这是个严重的问题，为了更好地进行差异化竞争，同时也为了推出更多的产品来迎合市场的需求，对产品的风格和主题进行定制则成为了一个非常重要的课题。

## 7.1 风格

风格（Style）也即 GUI 的观感， Qt 基于不同平台提供了一个风格集（QWindowsStyle、QmacStyle、QmotifStyle、QcommonStyle、QCleanlooksStyle 等），这些风格被内置于 QtGUI 库中，风格可以按照插件的方式使用。Qt 内置的部件（widgets）使用 QStyle 来实行近乎所有的着色操作。以确保其观感于本地（native）部件相近。

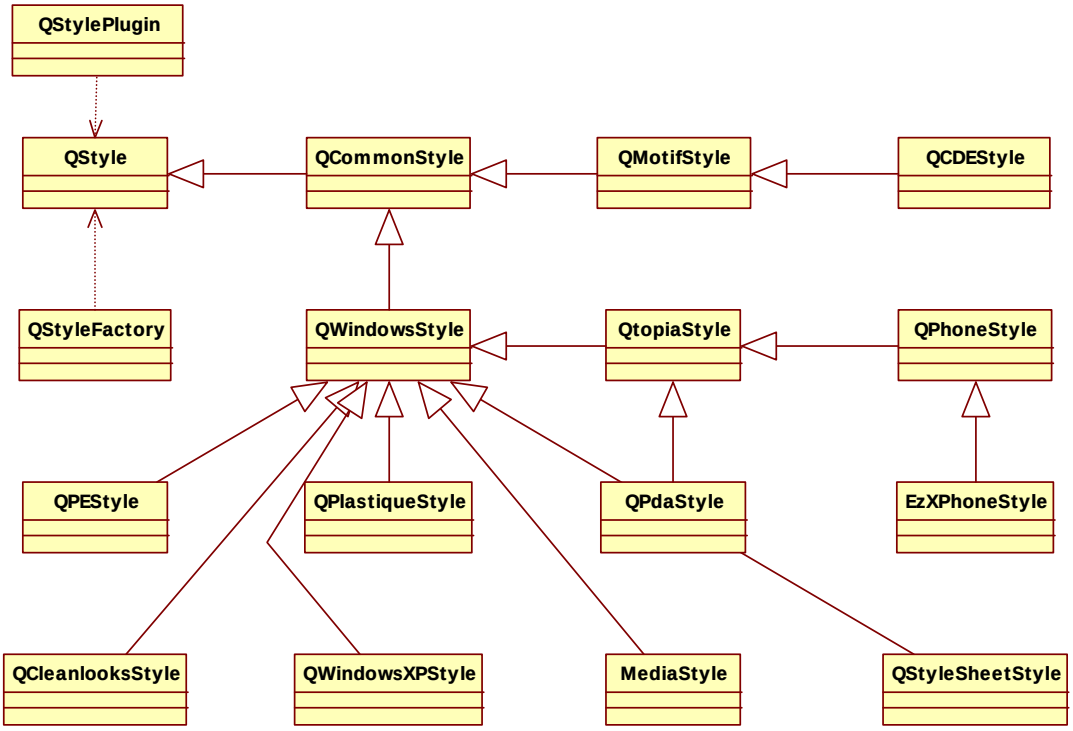


图42.　风格集类图

风格系统主要有以下几个类构成：

QStyleFactory：一个构造 Qstyle 的工厂类，调用其 create()函数就可以构造出 Qstyle ，另外调用 QstylePlugin 也可以构建或者动态导出 Qstyle。

QStyle：在 Qt 中，Qstyle 是所有风格类的基类，对 GUI 的观感进行了封装。

QStyleOption：包含了 Qstyle 用来渲染图像元素所需要的所有信息。

下图为风格系统的类图：

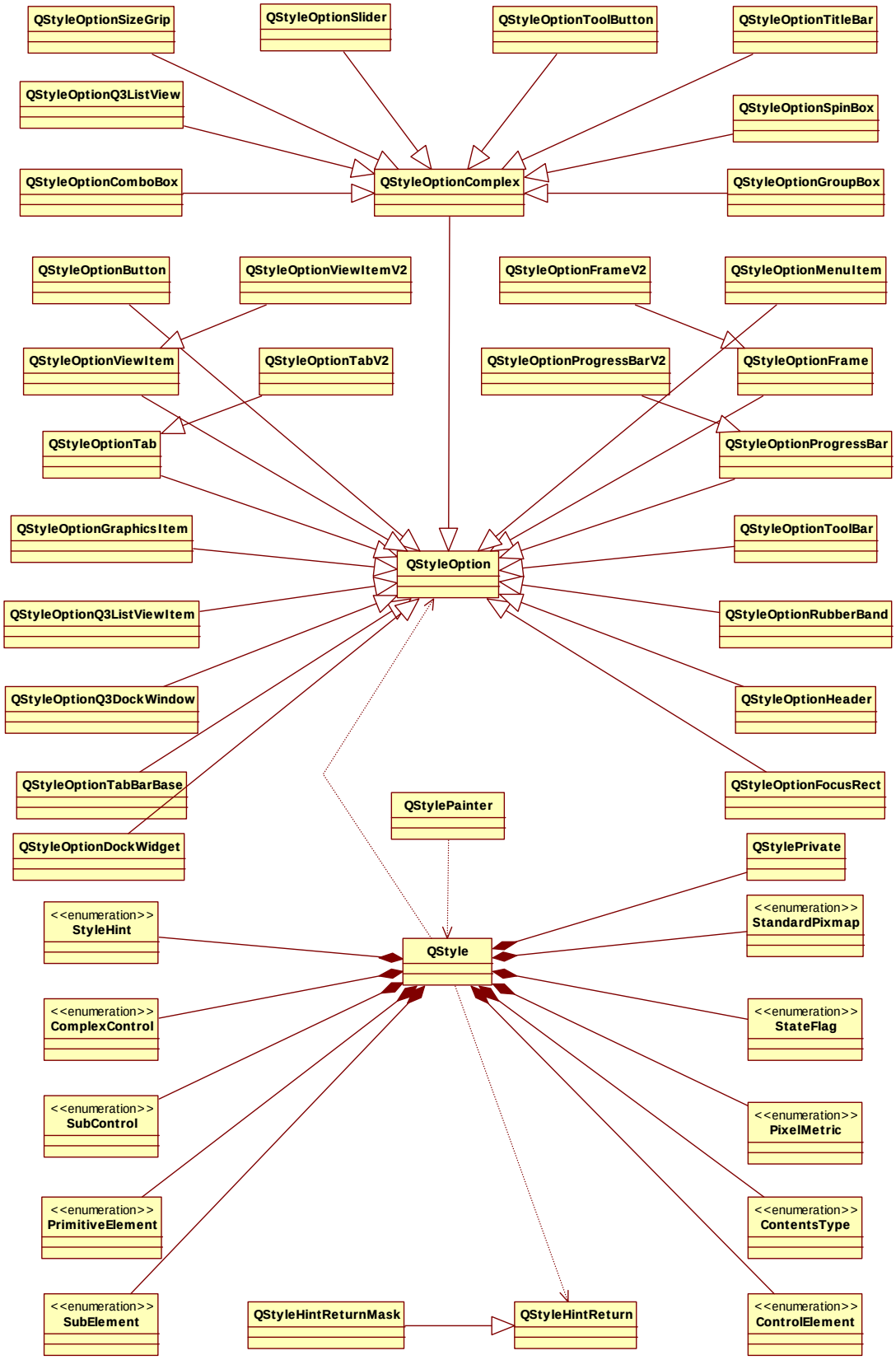


图43. 风格类图

如果需要自定义整个应用程序的风格，可以通过 QApplication::setStyle()的方式进行，也可以在启动应用程序时利用命令行的方式进行：

＃./applicationName –style motif

如果需要为单个部件选择风格，也可以利用 QWidget::setStyle()的方式来进行。如果没有为系统选择风格，系统会自动为用户平台或桌面环境选择最适当的风格。

如果要开发自定义的部件并且希望该部件能够在任何平台上都表现出色，可以考虑利用 QStyle 来为部件渲染。QStyle 提供的着色函数有：drawItemText()、drawItemPixmap()、drawPrimitive()、drawControl()和 drawComplexControl()等。

在进行渲染时，QStyle 一般需要四个参数：

- 一个对应于预渲染的图像元素的枚举类型的值
- 一个说明如何渲染和在那里渲染的 QstyleOption
- 一个用来渲染的 Qpainter
- 一个被渲染的部件

下面是一个说明如何渲染的例子：

```
void MyWidget::paintEvent(QPaintEvent * /* event */)
 {
      QPainter painter(this);
      QStyleOptionFocusRect option;
      option.initFrom(this);
      option.backgroundColor = palette().color(QPalette::Background);
      style()->drawPrimitive(QStyle::PE_FrameFocusRect, &option, &painter, this);
 }
```

为了方便期间，Qt 提供了 QstylePainter 类，它联合了 Qstyle、Qpainter、Qwidget，当需要利用风格进行渲染时，可以如下处理：

```
void MyWidget::paintEvent(QPaintEvent * /* event */)
 {
      QStylePainter painter(this);
      QStyleOptionFocusRect option;
      option.initFrom(this);
      option.backgroundColor = palette().color(QPalette::Background);
      painter ->drawPrimitive(QStyle::PE_FrameFocusRect, option);
 }
```

下面以 QCommonStyle 为例来说明如何自定义风格。

```
class Q_GUI_EXPORT QCommonStyle: public QStyle
{
    Q_OBJECT
public:
    QCommonStyle();
    ~QCommonStyle();
    void drawPrimitive(PrimitiveElement pe, const QStyleOption *opt, QPainter *p,
                       const QWidget *w = 0) const;
    void drawControl(ControlElement element, const QStyleOption *opt, QPainter *p,
                     const QWidget *w = 0) const;
     QRect subElementRect(SubElement r, const QStyleOption *opt, const QWidget *widget
    = 0) const;
    void drawComplexControl(ComplexControl cc, const QStyleOptionComplex *opt,
    QPainter *p,const QWidget *w = 0) const;
    SubControl hitTestComplexControl(ComplexControl cc, const QStyleOptionComplex
    *opt,const QPoint &pt, const QWidget *w = 0) const;
    QRect subControlRect(ComplexControl cc, const QStyleOptionComplex *opt,
    SubControl sc,const QWidget *w = 0) const;
    QSize sizeFromContents(ContentsType ct, const QStyleOption *opt,
    const QSize &contentsSize, const QWidget *widget = 0) const;
    int pixelMetric(PixelMetric m, const QStyleOption *opt = 0, const QWidget *widget = 0)
```

```
        const;
        int styleHint(StyleHint sh, const QStyleOption *opt = 0, const QWidget *w = 0,
                      QStyleHintReturn *shret = 0) const;
        QPixmap standardPixmap(StandardPixmap sp, const QStyleOption *opt = 0,
                               const QWidget *widget = 0) const;
        QPixmap generatedIconPixmap(QIcon::Mode iconMode, const QPixmap &pixmap,
                                    const QStyleOption *opt) const;
    protected Q_SLOTS:
        QIcon standardIconImplementation(StandardPixmap standardIcon, const QStyleOption
        *opt = 0,const QWidget *widget = 0) const;
    protected:
        QCommonStyle(QCommonStylePrivate &dd);
    private:
        Q_DECLARE_PRIVATE(QCommonStyle)
        Q_DISABLE_COPY(QCommonStyle)
    };
```

根据自定义风格的需要，你必须实现或者重载 drawItemText()、drawItemPixmap()、drawPrimitive() 、 drawControl() 、 subElementRect() 、 drawComplexControl() 、 subControlRect()、styleHint()、standardPixmap()等函数。

## 7.2 主题

Qt 通过主题（themes）模块来定义 qt 的主题风格，如标题、拨号界面、主界面、窗口的装饰等等，都是利用 themes 模块来控制的。

Theme 系统主要有以下几个类构成：

ThemeControl：负责 ThemedView 类的注册。ThemeControl 是个单子模式的类，通过 ThemeControl::instance()可以获得 ThemeControl 的单子对象。利用 registerThemedView()可以对对 ThemedView 进行注册。当 ThemedView 发生变化时，ThemeControl 会发出 themeChanged()信号。

下面时 ThemedView 的注册过程：

```
void ThemeControl::registerThemedView(ThemedView *view,
                                      const QString &name)
{
    m_themes.append(qMakePair(view, name));          //向 QList 中追加
    doTheme(view, name);
}
void ThemeControl::doTheme(ThemedView *view, const QString &name)
{
    QString path = m_themeFiles[name + "Config"];    //获取 XML 文件名
    if(!path.isEmpty())
    {
        QString file = m_themeDir + path;
        view->setThemeName(m_themeName);             //设置 Theme 名
        view->loadSource(m_themeDir + path);         //导出 XML 文件
    }
    else
        {
            qWarning("Invalid %s theme.", name.toAscii().constData());
        }
}
```

ThemedView：负责构建、管理和显示 Theme 视图。Qtopia 的 Theme 由多个 Theme 视图构成。本质上 Theme 视图是一个自定义得用户接口组件。它的核心功能是向用户显示其包含得数据， Theme 视图由相应的 XML 文件描述，当需要构建 Them 视图时，利用 ThemedView ::loadSource()来导出 XML 文件，ThemedView 会分析 XML 文件，构建出 Theme

的条目树（item tree）并在窗口上进行渲染。研发人员可以利用 ThemedView::itemPressed()、ThemedView::itemClicked()和 ThemedView::itemReleased() 信号来监控 Theme 与用户之间的交互行为，除此之外，Theme 条目还能够通过向服务或信道发送 IPC 消息来对系统加以影响。当条目的可视性发生变化时，ThemedView 会发出 visChanged()信号。

下面是 Dialer 类为条目单击事件建立的信号与槽的连接：

```
connect( this, SIGNAL(itemClicked(ThemeItem*)), this,
                                                SLOT(dialerItemClicked(ThemeItem*)) );
```

下面是 Dialer 类为对条目单击事件的处理过程：

```
void Dialer::dialerItemClicked( ThemeItem *item )
{
    if( !item || !item->isInteractive() )
        return;

    QString n = item->itemName();
    if( n.isNull() )
        return;
    if( n == "star" )
        character();
}
```

Theme 的条目树是对 Theme 视图的实时描述，条目树的每个节点都由 ThemeItem 获得并实现了特定的功能。其中 ThemeTextItem 用来显示文本信息；ThemeRectItem 用来渲染矩形，ThemeLayoutItem 用来为条目（Item）布局。利用 ThemedView::findItem()，研发人员可以接入 Theme 视图条目树中的条目。

ThemeItem：是 ThemedView 中所有条目的基类，封装了条目的公共功能，管理着条目的公共属性如：区域、激活状态、透明状态、可视状态、布局、用户交互等。常用的 ThemeItem 有 ThemeTextItem、ThemePixmapItem、ThemeWidgetItem 等。

下图为 Theme 的类图：

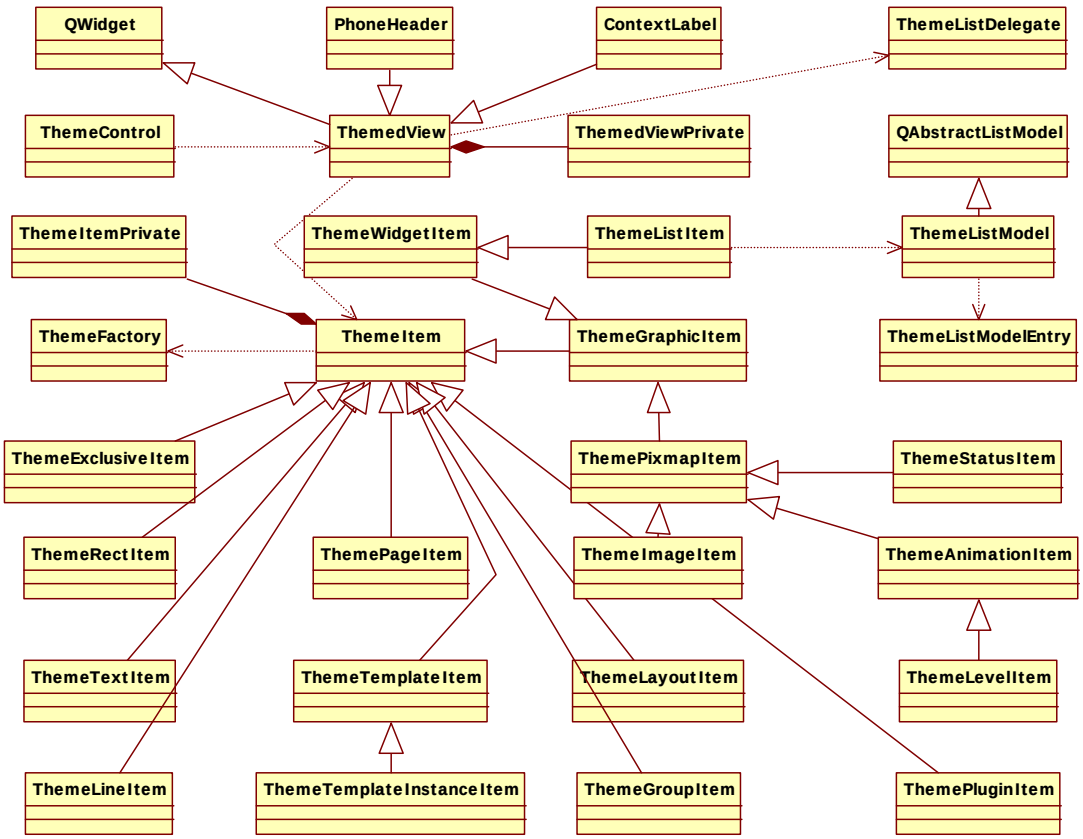

图44. Theme 类图

Qt 关于 themes 的配置文件位于 QTOPIADIR/etc/themes/目录下。Themes 主要由工程文件和配置文件组成。

下面是 smart 主题的工程文件 smart.pro：

1. 工程文件

```
qtopia_project(theme)
smartconf.files=$$QTOPIA_DEPOT_PATH/etc/themes/smart.conf
smartconf.path=/etc/themes                          //安装路径
smartconf.trtarget=Smart
smartconf.hint=themecfg
smartconf.outdir=$$PWD
smartdata.files=$$QTOPIA_DEPOT_PATH/etc/themes/smart/*.xml
$$QTOPIA_DEPOT_PATH/etc/themes/smart/*rc
smartdata.path=/etc/themes/smart
smartpics.files=$$QTOPIA_DEPOT_PATH/pics/themes/smart/*
smartpics.path=/pics/themes/smart
smartpics.hint=pics
INSTALLS+=smartconf smartdata smartpics          //安装模块
smartbgimage.files=$$QTOPIA_DEPOT_PATH/pics/themes/smart/background.png
smartbgimage.path=/pics/themes/smart
smartbgimage.hint=background
# let this install first so we can overwrite the image
smartbgimage.depends=install_smartpics
INSTALLS+=smartbgimage
pkg.name=qpe-theme-smart
pkg.desc=Smart theme
pkg.domain=theme
```

2. 配置文件

下面是 smart 主题的配置文件 smart.conf：

```
[Theme]
Name[] = Smart                                  //主题名
Style = Qtopia                                  //风格名
TitleConfig = smart/title.xml                   //标题配置文件所在路径
HomeConfig = smart/home.xml                     //主界面配置文件所在路径
ContextConfig = smart/context.xml               //上下文配置文件所在路径
DialerConfig = smart/dialer.xml                 //触摸屏拨号配置文件所在路径
CallScreenConfig = smart/callscreen.xml         //呼叫界面配置文件所在路径
DecorationConfig = smart/decorationrc           //窗口装饰配置文件所在路径
SecondaryTitleConfig = smart/secondarytitle.xml //辅标题配置文件所在路径
SecondaryHomeConfig = smart/secondaryhome.xml   //辅主界面配置文件所在路径
BootChargerConfig = smart/bootcharger.xml       //引导配置文件所在路径
[Translation]
File=QtopiaThemes
Context=Themes
```

由themes的配置文件可以看出，themes的信息实际上是利用xml语言描述的，通过查看xml文件可以发现，Qt利用xml文件描述了相关信息的布局、源文件、颜色等以及其他信息，关于XML的更多知识，参考参考文献[32]。

下面以 dialer.xml 为例说明 Qtopia 的 Theme 的设计。

```
<page name="dialer" base="themes/smart" keypad="no">
```

```
// 该ThemedView的名字为"dialer",base路径为"themes/smart"
    <image name="background" rect="0,0,0,0" src="background" color="Highlight"
tile="yes"/>
    <widget name="dialernumber" rect="0,0,0x24" bold="yes"
colorGroup="Background=#EEEEEE;Base=#DDDDDD" bgcolor="Text" size="18"/>
    <layout name="vertical" rect="0,24,0,0" orientation="vertical" spacing="-1"
align="vcenter">
        <layout name="horizontal" rect="0,0,0x33" orientation="horizontal" spacing="-1"
align="hcenter">
            <group rect="0,0,44x33" name="one" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),1">
//点击后发送Qt::Key_1 的按键事件
                <image rect="0,0,0,0" name="one_bg" src="r1" onclick="src=r1p"
scale="yes"/>
                <image rect="5,6,-5,-5" name="one_fg" src="1" scale="yes"/>
            </group>
            <group rect="0,0,44x33" name="two" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),2">
//点击后发送Qt::Key_2 的按键事件
                <image rect="0,0,0,0" name="two_bg" src="r1" onclick="src=r1p"
scale="yes"/>
                <image rect="5,6,-5,-5" name="two_fg" src="2" scale="yes"/>
            </group>
            <group rect="0,0,44x33" name="three" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),3">
//点击后发送Qt::Key_3 的按键事件
                <image rect="0,0,0,0" name="three_bg" src="r1" onclick="src=r1p"
scale="yes"/>
                <image rect="5,6,-5,-5" name="three_fg" src="3" scale="yes"/>
            </group>
            <group rect="0,0,44x33" name="selectcontact" interactive="yes"
onclick="message=Qtopia/Phone/TouchscreenDialer,selectContact()">
//点击后向Qtopia/Phone/TouchscreenDialer信道发送selectContact()消息
                <image rect="0,0,0,0" name="contact_bg" src="r1" onclick="src=r1p"
scale="yes"/>
                <image rect="11,7,-11,-6" name="contact_fg" src="contacts" scale="yes"/>
            </group>
        </layout>
        <layout rect="0,0,0x33" orientation="horizontal" spacing="-1" align="hcenter">
            <group rect="0,0,44x33" name="four" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),4">
//点击后发送Qt::Key_4 的按键事件
                <image rect="0,0,0,0" name="four_bg" src="r2" onclick="src=r2p"
scale="yes"/>
                <image rect="5,6,-5,-5" name="four_fg" src="4" scale="yes"/>
            </group>
            <group rect="0,0,44x33" name="five" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),5">
//点击后发送Qt::Key_5 的按键事件
                <image rect="0,0,0,0" name="five_bg" src="r2" onclick="src=r2p"
scale="yes"/>
                <image rect="5,6,-5,-5" name="five_fg" src="5" scale="yes"/>
            </group>
            <group rect="0,0,44x33" name="six" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),6">
//点击后发送Qt::Key_6 的按键事件
```

```
                    <image rect="0,0,0,0" name="six_bg" src="r2" onclick="src=r2p"
scale="yes"/>
                    <image rect="5,6,-5,-5" name="six_fg" src="6" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="messages" interactive="yes"
onclick="message=Qtopia/Phone/TouchscreenDialer,sms()">
//点击后向Qtopia/Phone/TouchscreenDialer信道发送sms ()消息
                    <image rect="0,0,0,0" name="messages_bg" src="r2" onclick="src=r2p"
scale="yes"/>
                    <image rect="11,10,-11,-9" name="messages_fg" src="messages"
scale="yes"/>
                </group>
            </layout>
            <layout rect="0,0,0x33" orientation="horizontal" spacing="-1" align="hcenter">
                <group rect="0,0,44x33" name="seven" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),7">
//点击后发送Qt::Key_7 的按键事件
                    <image rect="0,0,0,0" name="seven_bg" src="r3" onclick="src=r3p"
scale="yes"/>
                    <image rect="5,6,-5,-5" name="seven_fg" src="7" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="eight" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),8">
//点击后发送Qt::Key_8 的按键事件
                    <image rect="0,0,0,0" name="eight_bg" src="r3" onclick="src=r3p"
scale="yes"/>
                    <image rect="5,6,-5,-5" name="eight_fg" src="8" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="nine" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),9">
//点击后发送Qt::Key_8 的按键事件
                    <image rect="0,0,0,0" name="nine_bg" src="r3" onclick="src=r3p"
scale="yes"/>
                    <image rect="5,6,-5,-5" name="nine_fg" src="9" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="callhistory" interactive="yes"
onclick="message=Qtopia/Phone/TouchscreenDialer,showCallHistory()">
//点击后向Qtopia/Phone/TouchscreenDialer信道发送showCallHistory ()消息
                    <image rect="0,0,0,0" name="calls_bg" src="r3" onclick="src=r3p"
scale="yes"/>
                    <image rect="11,7,-11,-6" name="calls_fg" src="callhistory" scale="yes"/>
                </group>
            </layout>
            <layout rect="0,0,0x33" orientation="horizontal" spacing="-1" align="hcenter">
                <group rect="0,0,44x33" name="star" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),*">
//点击后发送”*”的按键事件
                    <image rect="0,0,0,0" name="star_bg" src="r4" onclick="src=r4p"
scale="yes"/>
                    <image rect="5,6,-5,-5" name="star_fg" src="star" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="zero" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),0">
//点击后发送Qt::Key_0 的按键事件
                    <image rect="0,0,0,0" name="zero_bg" src="r4" onclick="src=r4p"
scale="yes"/>
```

```
                    <image rect="5,6,-5,-5" name="zero_fg" src="0" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="hash" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),#">
//点击后发送”＃”的按键事件
                    <image rect="0,0,0,0" name="hash_bg" src="r4" onclick="src=r4p"
scale="yes"/>
                    <image rect="5,6,-5,-5" name="hash_fg" src="hash" scale="yes"/>
                </group>
                <group rect="0,0,44x33" name="hangup" interactive="yes"
onclick="message=Qtopia/Phone/TouchscreenDialer,hangup()">
//点击后向Qtopia/Phone/TouchscreenDialer信道发送hangup ()消息
                    <image rect="0,0,0,0" name="hangup_bg" src="r4" onclick="src=r4p"
scale="yes"/>
                    <image rect="11,7,-11,-6" name="hangup_fg" src="hangup" scale="yes"/>
                </group>
            </layout>
        </layout>
</page>
```

下面是 Dialer 类对按键事件的处理过程：

```
void Dialer::keyPressEvent( QKeyEvent* e )
{
    if( characterMenu()->isVisible() )
    {
        characterMenu()->setVisible(false);
        return;
    }
    switch( e->key() )
    {
    case Qt::Key_0:
    case Qt::Key_1:
    case Qt::Key_2:
    case Qt::Key_3:
    case Qt::Key_4:
    case Qt::Key_5:
    case Qt::Key_6:
    case Qt::Key_7:
    case Qt::Key_8:
    case Qt::Key_9:
    // handled manually by dialerItemClicked
    //case Qt::Key_Asterisk:
    case Qt::Key_NumberSign:
        e->accept();
        appendDigits( e->text() );
    break;
    default:
        QWidget::keyPressEvent( e );
    break;
    }
}
```

下面是 Dialer 类对“Qtopia/Phone/TouchscreenDialer”信道消息的处理过程：

```
void Dialer::msgReceived(const QString& str, const QByteArray&)
{
    if( str == "selectContact()" )
    {
        selectContact();                                    //显示电话薄界面
```

```
        }
        else if( str == "showCallHistory()" )
        {
        selectCallHistory();                    //显示呼叫历史界面
        }
        else if( str == "sms()" )
        {
            sms();                              //显示短信界面
        }
        else if( str == "hangup()" )
        {
            emit closeMe();                     //挂机
        }
        else if( str == "saveToContact()" )
        {
            saveToContact();                    //保存到电话簿
        }
        else if( str == "dial()" )
        {
            numberSelected();                   //显示拨号界面
        }
    }
```

从 dialer.xml 可以看出，Qt 为了使 Theme 能够与用户交互，采用了许多元素，所谓元素即 Theme 中定义组件结构和布局的要素。如"page"元素是一个定义了视图组件的大小和外形的顶级元素。为了更好的定义组件，每个元素又提供了一个或多个的属性，常用的元素有："page"、"rect"、"line"、"text "、"image "、"anim "、"status"、 "level "、"plugin"、"layout "、"exclusive "、"group"、"widget "、"list"、"template"。

下面详细介绍下"template"元素。

模版元素允许在 Theme 的 XML 文件中描述抽象的外观并在需要的时候进行显示。下面是一个按钮外观的 XML 文件设计：

```
<page vspath="/UIComponents">
    <template name="buttonnormal">
        <image rect="0,0,0,0" src="buttonnormalbg"/>
        <text rect="0,0,0,0" align="hcenter,vcenter">expr:@NormalText</text>
    </template>
    <template name="buttonclicked">
        <image rect="0,0,0,0" src="buttonclickedbg"/>
        <text rect="0,0,0,0" align="hcenter,vcenter">expr:@ ClickedText</text>
    </template>
</page>
```

在上面的 XML 文件中，采用了模版元素，为按钮定义了两种种外观：标准、点击。下面是在程序设计中显示外观的方法：

```
class ThemedButton : public QPushButton
{
private:
    ThemedView* view; // contains the loaded theme
    bool buttonPressed;
protected:
    void paintEvent(QPaintEvent* e)
    {
        // find our template
        QString templateName = buttonPressed ? "buttonnormal" : "buttonclicked";
```

```
        ThemeTemplateItem* template =
        (ThemeTemplateItem*)view->findItem(templateName,
        ThemedView::Template);
        Q_ASSERT(template != 0);
        // create an instance of the template
        ThemeTemplateInstanceItem* instance =
        template->createInstance(QString::number( (int)this )); // pass a uid
        to createInstace, used as value for vspath
        // paint the template instance to this widget
        QPainter p(this);
        view->paint(&p, e->rect(), instance);
        // cleaup
        delete instance;
    }
};
```

除了公共属性如"name"、"active"、"vspath"、"rect"等以外，各元素还有其特殊的属性，下面分析下 Theme 提供的几种重要的属性类型：

1. 名称属性

名称属性对所有的元素都是可用的，但需要说明的是，由于 Qt 的 Theme 与用户交互时利用属性的名称进行，所以在每个类型的条目中该名称必须是唯一的。下面是系统已经使用的名称属性。

- Title Bar, Home Screen, Context Bar

| 名称 | 类型 | 描述 |
|---|---|---|
| time | text | 当前时间 |
| date | text | 当前日期 |
| location | text | 小区位置 |
| caption | text | 标题 |
| roaming | status | 漫游 |
| messages | text, status | 未读消息 |
| call_active | status | 呼叫激活状态 |
| signal | level | 信号强度 |
| battery | level | 电量变化 |
| inputmethod | rect | 输入法 |
| calls | text, status | 未接来电 |
| alarm | status | 闹钟状态 |
| lock | status | 键盘锁状态 |
| calldivert | status | 呼叫转移状态 |
| profile | text | 当前 profile |
| infobox | text, image | 显示信息 |
| button0 | text, image | Context button 0 |
| button1 | text, image | Context button 1 |
| button2 | text, image | Context button 2 |

表9 主屏等的名称属性

| 名称 | 类型 | 描述 |
|---|---|---|
| dialernumber | input | 数字输入域 |
| zero | Any type - interactive | 数字 0 |
| one | Any type - interactive | 数字 1 |
| two | Any type - interactive | 数字 2 |
| three | Any type - interactive | 数字 3 |
| four | Any type - interactive | 数字 4 |
| five | Any type - interactive | 数字 5 |
| six | Any type - interactive | 数字 6 |
| seven | Any type - interactive | 数字 7 |
| eight | Any type - interactive | 数字 8 |
| nine | Any type - interactive | 数字 9 |

表10 Dialer 的名称属性

| 名称 | 类型 | 描述 |
|---|---|---|
| callscreen | input | 当前来电列表 |
| callscreennumber | input | 数字域 |
| active | listitem | 激活呼叫 |
| incoming | listitem | 来电 |
| outgoing | listitem | 去电 |
| onhold | listitem | 阻塞呼叫 |
| dropped | listitem | 挂断电话 |
| selected | listitem | 选择来电 |
| conference | status | 多方通话 |
| identifier | text | 对方姓名 |
| status | text | 通话状态 |
| contact | image | 对方号码 |

表11 Call Screen 的名称属性

2. 位置属性

rect 属性对所有元素来说都是可用的，它为所属元素提供了相对于父元素区域的相对位置。在实际的实现中有两种方式来表述（x,y,wxh）和(x1,y1,x2,y2)。

```
<image rect="5,6,-5,-5" name="zero_fg" src="0" scale="yes"/>
```

3. 显示/隐藏属性

显示/因此属性对所有的元素都是可用的，transient 和 active 属性控制了元素的可视性， transient 属性的默认值为"no",active 属性的默认值为"yes"。只要在 transient 属性的值为"yes"的时候，active 属性才会有效，除了"yes"和"no"外，active 属性的值还可以被设为布尔表达式，这就使 Theme 元素的可视性能够通过值空间来控制。

```
〈rect name="newmessages" transient="yes" active="expr:@/Communications/Messages/NewMessages" 〉
```

4. 排列属性

排列属性 align 有以下几种取值："Left"、"hcenter"、"right"、"top"、"vcenter"、"bottom"。orientation 有以下几种取值："horizontal"、"vertical"。

5. 图像伸缩属性

图像伸缩属性 scale 具有两种取值："yes"和"no"。默认值为"yes"。

6. 交互属性

当元素被点击时，为了允许元素能有所反馈，interactive 属性的值应为"yes"。

```
<layout rect="0,0,0x33" orientation="horizontal" spacing="-1" align="hcenter">
                    <group rect="0,0,44x33" name="seven" interactive="yes"
onclick="message=VirtualKeyboard,keyPress(int),7"
```

# 第 8 章 安装与集成

**靡不有初, 鲜克有终。**

《诗经·大雅·荡》

在主机端完成应用程序的交叉编译后，需要将引导程序和生成的文件映像下载到嵌入式目标版裸机上，然后通过引导程序启动映像，目标板才可以运行。

嵌入式系统与通用 PC 机不同，一般没有硬盘这样的存储设备而是使用 Flash 闪存芯片、小型闪存卡等专为嵌入式系统设计的存储装置，本章分析了嵌入式系统中常用的存储设备及其管理机制，介绍了常用的几种基于 Flash 文件系统类型和映像文件的下载方式以及服务的配置。

## 8.1 文件系统

构建适用于嵌入式系统的 Linux 文件系统，必然会涉及到两个关键点，一是文件系统类型的选择，它关系到文件系统的读写性能、尺寸大小;另一个就是根文件系统内容的选择，它关系到根文件系统所能提供的功能及尺寸大小。

嵌入式设备中使用的存储器是像 Flash 闪存芯片、小型闪存卡等专为嵌入式系统设计的存储装置。Flash 是目前嵌入式系统中广泛采用的主流存储器，它的主要特点是按整体/扇区擦除和按字节编程，具有低功耗、高密度、小体积等优点。目前，Flash 分为 NOR, NAND 两种类型。

NOR 型闪存可以直接读取芯片内储存的数据，因而速度比较快，但是价格较高。NOR 型芯片，地址线与数据线分开，所以 NOR 型芯片可以像 SRAM 一样连在数据线上，对 NOR 芯片可以“字”为基本单位操作，因此传输效率很高，应用程序可以直接在 Flash 内运行，不必再把代码读到系统 RAM 中运行。它与 SRAM 的最大不同在于写操作需要经过擦除和写入两个过程。

NAND 型闪存芯片共用地址线与数据线，内部数据以块为单位进行存储，直接将 NAND 芯片做启动芯片比较难。NAND 闪存是连续存储介质，适合放大文件。擦除 NOR 器件时是以 64-128KB 的块进行的，执行一个写入/擦除操作的时间为 5s；擦除 NAND 器件是以 8-32KB 的块进行的，执行相同的操作最多只需要 4ms。NAND Rash 的单元尺寸几乎是 NOR 器件的一半，由于生产过程更为简单，NAND 结构可以在给定的模具尺寸内提供更高的容量，也就相应地降低了价格。NOR flash 占据了容量为 1—16MB 闪存市场的大部分，而 NAND flash 只是用在 8—128MB 的产品当中，这也说明 NOR 主要应用在代码存储介质中，NAND 适合于数据存储。寿命(耐用性)，在 NAND 闪存中每个块的最大擦写次数是一百万次，而 NOR 的擦写次数是十万次。NAND 存储器除了具有 10 比 1 的块擦除周期优势，典型的 NAND 块尺寸要比 NOR 器件小 8 倍，每个 NAND 存储器块在给定的时间内的删除次数要少一些。

所有嵌入式系统的启动都至少需要使用某种形式的永久性存储设备，它们需要合适的驱动程序，当前在嵌入式 Linux 中有三种常用的块驱动程序可以选择。

1. Blkmem 驱动层

Blkmem 驱动是为 uclinux 专门设计的，也是最早的一种块驱动程序之一，现在仍然有很多嵌入式 Linux 操作系统选用它作为块驱动程，尤其是在 uClinux 中。它相对来说是最简单的，而且只支持建立在 NOR 型 Flash 和 RAM 中的根文件系统。使用 Blkmem 驱动，建立 Flash 分区配置比较困难，这种驱动程序为 Flash 提供了一些基本擦除/写操作。

2. RAMdisk 驱动层

RAMdisk 驱动层通常应用在标准 Linux 中无盘工作站的启动，对 Flash 存储器并不提供任何的直接支持， RAM disk 就是在开机时，把一部分的内存虚拟成块设备，并且把之前所准备好的档案系统映像解压缩到该 RAM disk 环境中。当在 Flash 中放置一个压缩的文件系

统，可以将文件系统解压到 RAM，使用 RAM disk 驱动层支持一个保持在 RAM 中的文件系统。

3. MTD 驱动层

为了尽可能避免针对不同的技术使用不同的工具，以及为不同的的技术提供共同的能力，Linux 内核纳入了 MTD 子系统(memory Technology Device)。它提供了一致且统一的接口，让底层的 MTD 芯片驱动程序无缝地与较高层接口组合在一起。JFFS2, Cramfs, YAFFS 等文件系统都可以被安装成 MTD 块设备。MTD 驱动也可以为那些支持 CFI 接口的 NOR 型 Flash 提供支持。虽然 MTD 可以建立在 RAM 上，但它是专为基于 Flash 的设备而设计的。MTD 包含特定 Flash 芯片的驱动程序，研发者要选择适合自己系统的 Flash 芯片驱动。Flash 芯片驱动向上层提供读、写、擦除等基本的操作，MTD 对这些操作进行封装后向用户层提供 MTD char 和 MTD block 类型的设备。MTD char 类型的设备包括/dev/mtd0, /dev/mtdl 等，它们提供对 Flash 原始字符的访问。MTD block 类型的设备包括/dev/mtdblock0,/dev/mtdblock1 等，MTD block 设备是将 Flash 模拟成块设备，这样可以在这些模拟的块设备上创建像 Cramfs, JFFS2 等格式的文件系统。

MTD 驱动层也支持在一块 Flash 上建立多个 Flash 分区，每一个分区作为了一个 MTD block 设备，可以把系统软件和数据等分配到不同的分区上，同时可以在不同的分区采用不用的文件系统格式。这一点非常重要，正是由于这一点才为嵌入式系统多文件系统的建立提供了灵活性。

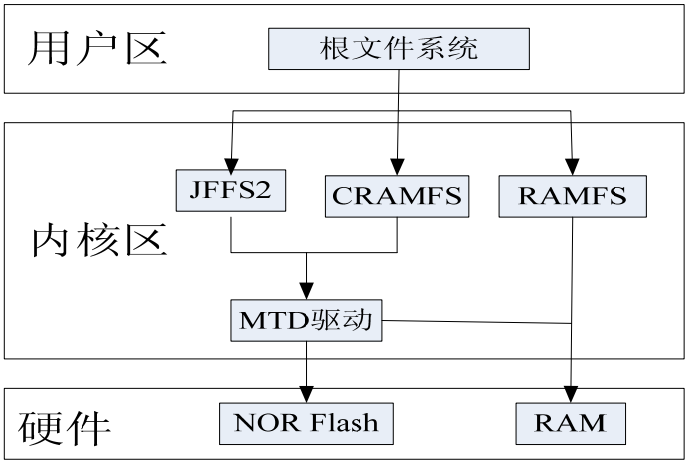


图45. 文件系统体系结构

鉴于 Flash 存储介质的读写特点，传统的 Linux 文件系统已经不适合应用在嵌入式系统中，像 Ext2fs 文件系统是为像 IDE 那样的块设备设计的，这些设备的逻辑块是 512 字节、1024 字节等大小，没有提供很好的扇区擦写支持，不支持损耗平衡，没有掉电保护，也没有特别完美的扇区管理，这不太适合于扇区大小因设备类型而划分的闪存设备。基于这样的原因，产生了很多专为 Flash 设备而设计的文件系统。

### 8.1.1 Romfs

传统型的 Romfs 文件系统是最常使用的一种文件系统，它是一种简单的、紧凑的、只读的文件系统，不支持动态擦写保存;它按顺序存放所有的文件数据，所以这种文件系统格式支持应用程序以 XIP 方式运行，在系统运行时，可以获得可观的 RAM 节省空间。uClinux 系统通常采用 Romfs 文件系统。

### 8.1.2 Ramfs/Tmpfs

Ramfs 也是 Linus Torvalds 研发的，Ramfs 文件系统把所有的文件都放在 RAM 里运行，

通常是 Flash 系统用来存储一些临时性或经常要修改的数据，相对于 ramdisk 来说，Ramfs 的大小可以随着所含文件内容大小变化，不像 ramdisk 的大小是固定的。Tmpfs 是基于内存的文件系统，因为 tmpfs 驻留在 RAM 中，所以写/读操作发生在 RAM 中。tmpfs 文件系统大小可随所含文件内容大小变化，使得能够最理想地使用内存;tmpfs 驻留在 RAM，所以读和写几乎都是瞬时的。tmpfs 的一个缺点是当系统重新引导时会丢失所有数据。

### 8.1.3 JFSS2

JFFS2 是 RedHat 公司基于 JFFS 研发的闪存文件系统，最初是针对 RedHat 公司的嵌入式产品 eCos 研发的嵌入式文件系统，所以 JFFS2 也可以用在 Linux,uCLinux 中。JFFS 文件系统最早是由瑞典 Axis Communications 公司基于 Linux2.0 的内核为嵌入式系统研发的文件系统。JFFS2 是一个可读写的、压缩的、日志型文件系统，并提供了崩溃/掉电安全保护，克服了 JFFS 的一些缺点:使用了基于哈希表的日志节点结构，大大加快了对节点的操作速度;支持数据压缩；提供了“写平衡”支持;支持多种节点类型；提高了对闪存的利用率，降低了内存的消耗。这些特点使 JFFS2 文件系统成为目前 Flash 设备上最流行的文件系统格式，它的缺点就是当文件系统已满或接近满时，JFFS2 运行会变慢，这主要是因为碎片收集的问题。

瑞典的 Axis Communications 研发了最初的 JFFS，Red Hat 的 David Woodhouse 对它进行了改进。 第二个版本，JFFS2，作为用于微型嵌入式设备的原始闪存芯片的实际文件系统而出现。JFFS2 文件系统是日志结构化的，这意味着它基本上是一长列节点。每个节点包含有关文件的部分信息 — 可能是文件的名称、也许是一些数据。相对于 Ext2fs，JFFS2 因为有以下这些优点而在无盘嵌入式设备中越来越受欢迎：

- JFFS2 在扇区级别上执行闪存擦除 / 写 / 读操作要比 Ext2 文件系统好。
- JFFS2 提供了比 Ext2fs 更好的崩溃 / 掉电安全保护。当需要更改少量数据时，Ext2 文件系统将整个扇区复制到内存（DRAM）中，在内存中合并新数据，并写回整个扇区。这意味着为了更改单个字，必须对整个扇区（64 KB）执行读 / 擦除 / 写例程 — 这样做的效率非常低。要是运气差，当正在 DRAM 中合并数据时，发生了电源故障或其它事故，那么将丢失整个数据集合，因为在将数据读入 DRAM 后就擦除了闪存扇区。JFFS2 附加文件而不是重写整个扇区，并且具有崩溃 / 掉电安全保护这一功能。
- 这可能是最重要的一点：JFFS2 是专门为象闪存芯片那样的嵌入式设备创建的，所以它的整个设计提供了更好的闪存管理。

JFFS2 的缺点是由于垃圾收集的问题，当文件系统已满或接近满时，JFFS2 会大大放慢运行速度。

创建 JFFS2 文件系统的方法如下：

＃mkdir jffsfile

#cd jffsfile

然后将创建文件系统所需的文件目录放置在当前目录下。然后利用 mkfs.jffs2 工具创建 JFFS2 文件系统，命令如下：

```
＃./mkfs.jffs2 -e 0x40000 -p -o ../jffs.image
```

其中，-e 选项确定闪存的擦除扇区大小（通常是 64 千字节），-p 选项用来在映像的剩余空间用零填充，-o 选项用于输出文件，通常是 JFFS2 文件系统映像，在本例中即 jffs.image。一旦创建了 JFFS2 文件系统，它就被装入闪存中适当的位置（引导装载程序告知内核查找文件系统的地址）以便内核能挂装它。

### 8.1.4 CRAMFS

Cramfs 是 Linux 的创始人 Linus Torvalds 研发的一种可压缩只读文件系统在 Cramfs 文件系统中，每一页被单独压缩，可以随机页访问，其压缩比高达 2:1,为嵌入式系统节省大量的 Flash 存储空间。Cramfs 文件系统以压缩方式存储，在运行时解压缩，所以不支持应用程序以 XIP 方式运行，所有的应用程序要求被拷到 RAM 里去运行，但这并不代表比 Ramfs 需求

的RAM 空间要大一点，因为Cramfs是采用分页压缩的方式存放档案，在读取档案时，不会一下子就耗用过多的内存空间，只针对目前实际读取的部分分配内存，尚没有读取的部分不分配内存空间，当我们读取的档案不在内存时， Cramfs 文件系统自动计算压缩后的资料所存的位置，再即时解压缩到 RAM 中。另外，它的速度快，效率高，其只读的特点有利于保护文件系统免受破坏，提高了系统的可靠性;但是它的只读属性同时又是它的一大缺陷，使得用户无法对其内容对进扩充。Cramfs 映像通常是放在 Flash 中，但是也能放在别的文件系统里，使用 loopback 设备可以把它安装别的文件系统里。使用 mkcramfs 工具可以创建 Cramfs 映像。

1. Cramfs 的特性：

cramfs 文件系统具有一下特性：

- 采用实时解压缩方式，但解压缩的时侯有延迟。
- cramfs 的数据都是经过处理、打包的，对其进先写操作有一定困难。所以 cramfs 不支持写操作，这个特性刚好适合嵌入式应用中使用 Flash 存储文件系统的场合。
- 在 cramfs 中，文件最大不能超过 16MB。
- 支持组标识(gid)，但是 mkcramfs 只将 gid 的低 8 位保存下来，因此只有这 8 位是有效的。
- 支持硬链接。但是 cramfs 并没有完全处理好，硬链接的文件属性中，链接数仍然为 1.
- cramfs 的目录中，没有“.”和“..”这两项。因此，cramfs 中的目录的链接数通常也仅有一个。
- cramfs 中，不会保存文件的时间戳(timestamps)信息。当然，正在使用的文件由于 inode 保存在内存中，因此其时间可以暂时地变更为最新时间，但是不会保存到 cramfs 文件系统中去。

当前版本的 cramfs 只支持 PAGE_CACHE_SIZE 为 4096 的内核。因此，如果发现 cramfs 不能正常读写的时侯，可以检查一下内核的参数设置。

2. 使用 cramfs

可以从 http://sourceforge.net/projects/cramfs/下载 cramfs-1.1.tar.gz。然后执行：

＃tar zxvf cramfs-1.1.tar.gz

进入解包之后生成 cramfs-1.1 目录，执行编译命令：

＃make

编译完成之后，会生成 mkcramfs 和 cramfsck 两个工具，其中 cramfsck 工具是用来创建 cramfs 文件系统的，而 mkcramfs 工具则用来进行 cramfs 文件系统的释放以及检查。

下面是 mkcramfs 的命令格式：

mkcramfs [-h] [-e edition] [-i file] [-n name] dirname outfile

mkcramfs 的各个选项解释如下：

-h：显示帮助信息

-e edition：设置生成的文件系统中的版本号

-i file：将一个文件映像插入这个文件系统之中(只能在 Linux2.4.0 以后的内核版本中使用)

-n name：设定 cramfs 文件系统的名字

dirname：指明需要被压缩的整个目录树

outfile：最终输出的文件

cramfsck 的命令格式：

cramfsck [-hv] [-x dir] file

cramfsck 的各个参数解释如下：

-h：显示帮助信息

-x dir：释放文件到 dir 所指出的目录中

-v：输出信息更加详细

file：希望测试的目标文件

### 8.1.5 YAFFS

YAFFS/YAFFS2 是一种和 JFFSx 类似的闪存文件系统，它是专为嵌入式系统使用 NAND 型闪存而设计的一种日志型文件系统。和 JFFS2 相比它减少了一些功能，所以速度更快，而且对内存的占用比较小。此外，YAFFS 自带 NAND 芯片的驱动，并且为嵌入式系统提供了直接访问文件系统的 API，用户可以不使用 Linux 中的 MTD 与 VFS，直接对文件系统操作。YAFFS2 支持大页面的 NAND 设备，并且对大页面的 NAND 设备做了优化。JFFS2 在 NAND 闪存上表现并不稳定，更适合于 NOR 闪存，所以相对大容量的 NAND 闪存，YAFFS 是更好的选择。

在具体的嵌入式系统设计中可根据不同目录存放的内容不同以及存放的文件属性，确定使用何种文件系统。

## 8.2 下载与安装

主机端编译的 Linux 内核映像必须至少有一种方式下载到目标版上执行。通常是目标板的引导程序负责把主机端的映像文件下载到内存中。根据不同的连接方式，可以有多种文件传输方式，每一种方式都需要相应的传输文件和协议。

### 8.2.1 HOST-TARGET通信接口

1. USB 接口

通用串行总线（USB,Universal Serial Bus）接口，带有 5V 电压，支持热插拔，具有即插即用的优点，最多可连接 127 台 USB 设备，由康柏、IBM、Intel 和 Microsoft 于 1994 年共同推出，旨在统一外设如打印机、外置 Modem、扫描仪、鼠标等的接口，便于外设的安装使用。但直到 1996 年才最终颁布 USB1.0 标准，但由于标准并不成熟，且当时的主流操作系统 windows 95 并不支持 USB 1.0，直到 1998 年 USB 1.1 版本发布和 Windows 98 提供了对 USB 1.1 的支持。USB 接口开始迅速流行。

目前市场上主流的 USB 规范有两个，即 USB 1.1 和 USB 2.0。其中 1998 年发布的 USB 其高速方式的传输速率为 12Mps，低速方式的传输速度为 1.5Mbps。

而 USB 2.0 规范是最新的 USB 规范，2004 年发布，它的传输速率可以达到 480Mbps。足以满足大多数外设的速率要求。USB 2.0 中的“增强主机控制器接口”定义了一个与 USB 1.1 相兼容的架构。所有支持 USB 1.1 的设备都可以直接在 USB 2.0 的接口上使用而不必担心兼容性问题，而且像 USB 线、插头等附件也都可以直接使用。

USB 口通信的缺点是 USB 设备区分主从端，两端分别要有不同的驱动程序支持，主机端识别出从机端后，才可以开始通信。

2. 串行通讯接口

串行通讯接口常用 9 针串口（DB9）和 25 针串口（DB25）,通信距离较近（<12m），可以用电缆线直接连接标准 RS232C 端口；如果距离较远，可以采用 RS422 或者 RS485 接口，需附加调制解调器（Modem）。其中最常用的是三线制接法。即地、接收数据和发送数据三脚相连，直接用 RS232C 相连，PC 机上一般带有 2 个 9 针串口。

通过串口可以向目标板发送，命令，显示信息、传送文件、调试内核及程序。串口的设备驱动程序实现比较简单。

主机端利用 kermit、mincom 或者 windows 超级终端等工具都可以通过串口通信。当然通信之前需要配置好数据传输率和传输协议，目标板也要做好接收准备。通常波特率可以配置成 115200bit/s，8 位数据位，不带校验位。

串口通信的缺点是通讯速率慢，不适合大数据量的传输。

3. 以太网接口

以太网以其高度灵活、相对简单、易于实现的特点，成为当今最重要的一种局域网建网技术，虽然其他网络技术也曾经被认为可以取代以太网的地位，但是绝大多数的网络管理人员仍然把以太网作为首选的网络解决方案。

以太网 IEEE 802.3 通常使用专门的网络接口卡或通过系统主电路板上的电路实现。以太网使用收发器与网络媒体进行连接。收发器可以完成多种物理层功能，其中包括对网络碰撞进行检测。收发器可以作为独立的设备通过电缆与终端站连接，也可以直接被集成到终端站的网卡当中。

以太网采用广播机制，所有与网络连接的工作站都可以看到网络上传递的数据。通过查看包含在帧中的目标地址，确定是否进行接收或放弃。如果证明数据确实是发送给自己的，工作站将会接收数据并传递给高层协议进行处理。

以太网采用 CSMA/CD 媒体访问机制，任何工作站都可以在任何时间访问网络。在发送数据之前，工作站首先需要侦听网络是否空闲，如果网络上没有任何数据传送，工作站就会把所要发送的信息投放到网络中，否则，工作站只能等待网络下一次出现空闲的时候再进行数据的传送。

作为一种基于竞争机制的网络环境，以太网允许任何一台网络设备在网络空闲时发送信息，因为没有任何集中式的管理措施，所以非常有可能出现多台工作站同时检测到网络处于空闲状态，进而同时向网络发送数据的情况，这时，发出的信息会相互碰撞而导致损坏。工作站必须等待一段时间之后，重新发送数据。补偿算法用来决定发生碰撞后，工作站应当在何时重新发送数据帧。

网络接口一般采用 RJ-45 标准插头，PC 机上一般都配置 10M/100M 以太网卡，实现局域网连接。通过以太网连接和网络协议，可以实现快速的数据通讯和文件传输。

网络传输方式一般采用 TFTP(Trivial FileTransport Protocol)协议。TFTP 协议是一种简单的网络传输协议，基于 UDP 实现，没有传输控制，所以对于大文件的传输是不可靠的。不过正好适合目标板的引导程序，因为协议简单，功能容易实现。当然，使用 TFTP 传输之前，需要驱动目标版以太网接口并且配置 IP 地址。

4. JTAG 等接口

JTAG 技术是一种嵌入式调试技术，它在芯片内部封装了专门的测试电路测试接口（TAP,Test Access Port），通过 JTAG 测试工具对芯片的核进行测试。它是联合测试行动小组（JTAG,Joint Test Action Group）定义的一种国际标准测试协议，主要用于芯片内部测试及对系统进行仿真和测试数据输出。

JTAG 接口的时钟一般在 1MHz～16MHz 之间，所以传输速率可以很快。但是实际的数据传输速度要取决于仿真器与主机端的通讯速度和传输软件。

另外还有 EJTAG（Extended JTAG）和 BDM(Background Debug Mode)接口定义，分别在 MIPS 芯片和 PowerPC 5xx/8xx 芯片上设计应用。这些接口的电气性能不同，但是功能大体上是相似的。

JTAG 仿真器跟主机之间的连接通常是串口、并口、以太网接口或者 USB 接口。传输速率会受到主机连接方式的限制，这取决于仿真器硬件的接口配置。

采用并口连接方式的仿真器最简单，也叫作 JTAG 电缆（CABLE）,价格也最便宜。性能好的仿真器一般会采用以太网接口或者 USB 接口通信。

### 8.2.2 服务配置

1. TFTP 服务的配置

TFTP 协议是用来下载或上传文件的最简单网络协议，目前由 H. Peter 维护。其最新版本为 2003 年 2 月发布的 0.42。TFTP 它基于 UDP 协议而实现。嵌入式 linux 的 tftp 研发环境包括两个方面：一是 linux 服务器端的 tftp-server 支持，二是嵌入式目标系统的 tftp-client 支持。因为 u-boot 本身内置支持 tftp-client，所以嵌入式系统目标就不用配置了。

用法：tftp [-v][-m mode][host [port]] [-c command]

选项：

| | |
|---|---|
| -c command | //如果已经启动 tftp,执行相应命令 |
| -m mode | //设置模式 |
| -v | //设为冗余模式 |
| -V | //向标准输出打印版本号和配置信息 |

主要命令：

```
ascii                                   //设为 ascii 模式
binary                                  //设为 binary 模式
connect host[port]                      //设置传输的主机和端口号
get file                                //下载文件
mode      transfer-mode                 //设置传输模式
put file                                //上传文件
quit                                    //退出
```

在 SUSE Linux 10.1 上，可以通过 YAST2 控制中心查看 TFTP 服务器的配置情况，为了启用 TFTP 服务，应启动 TFTP 服务，并设置好映像目录/tftpboot。需要说明的是由于 TFTP 协议本身没有考虑鉴权或安全事宜，因此如果远程服务器设置了接入限制或防火墙，很可能会造成客户端无法连接，因此应关闭服务器上的防火墙。

2. NFS 服务的配置

NFS 服务的主要任务是把本地的一个目录通过网络输出，其他计算机可以通过远程地挂载这个目录进行访问文件。由 Rick Sladkey 创建。

NFS 服务有自己的协议和端口号，但是为了文件传输或者其他相关信息传递的方便，NFS 使用远程调用（RPC,Remote Procedure Call）协议。

RPC 负责管理端口号的对应与服务相关的工作。NFS 本身的服务并没有提供文件传输的协议，它通过 RPC 实现。因此，还需要系统启动 protmap 服务。

NFS 服务通过一系列工具来配置文件输出，配置文件是/etc/exports。配置文件的语法格式如下：

#共享目录 主机名称 1 或 IP1(参数 1，参数 2) 主机名称 2 或 IP2（参数 3，参数 4）

其中“共享目录”是主机上要向外输出的一个目录；“主机名称或者 IP”则是允许按照指定权限访问这个共享目录的远程主机；“参数”则定义了各种访问权限。

exports 配置文件参数说明如下：

| 参数 | 含义 |
|---|---|
| rw | 具有可擦写的权限 |
| ro | 具有只读的权限 |
| no_root_squash | 如果登录共享目录的使用者是 root 的话，那么他对于这个目录具有 root 的权限 |
| root_squash | 如果登录共享目录的使用者是 root 的话，那么他的权限将被限制为匿名使用者，通常他的 UID 与 GID 都会被 nobody |
| all_squash | 不论登录共享目录的使用者是什么身份，他的权限将被限制为匿名使用者 |
| anonuid | 前面关于*_squash 提到的匿名使用者的 UID 设定值，通常为 nobody。这里可以设定为 UID 值，并且 UID 也必须/etc/passwd 中设置 |
| anongid | 与上面的 anonuid 类似，只有 GID 变成 group ID |
| sync | 文件同步写入到内存和硬盘当中 |
| async | 文件会先暂存在内存中，而不是直接写入磁盘 |

表12 exports 配置文件参数说明

编辑修改好/etc/exports 这个文件，就可以启动 portmap 服务和 nfs 服务了。常用系统启动脚本来启动服务。

```
# /etc/rc.d/init.d/portmap start
# /etc/rc.d/init.d/nfs start
```

也可以通过 service 命令来启动 portmap 服务和 nfs 服务：

```
# service nfs start
# service portmap start
```

启动完成后，可以查看/var/log/messages，确认是否正确激活服务。

如果只修改了/etc/exports 这个文件，并不总是需要重启 NFS 服务。可以使用 exportfs

工具重新读取/etc/exports，就可以加载输出的目录。

exportfs 工具的使用语法如下：

#exportfs [aruv]

选项：

-a:全部挂载（或卸载）/etc/exports 的设置

-r:重新挂载/etc/exports 的设置，更新/etc/exports 和/var/lib/nfs/xtab 里面的内容。

-u:卸载某一个目录

-v:在输出的时候，把共享目录显示出来

在 NFS 已经启动的情况下，如果又修改了/etc/exports 文件，可以执行命令：

#/etc/exports –rs

系统日志文件/var/lib/nfs/xtab 中可以查看共享目录访问权限，不过只有已经被挂载的目录才会出现在日志文件中。

远程计算机作为 NFS 客户端，可以简单通过 mount 命令挂载这个目录使用。例如：

# mount -t nfs 192.168.1.1 :/home/test /mnt

这条命令就是把 192.168.1.1 主机上的/home/test 目录作为 NFS 文件系统挂载到/mnt 目录下，如果系统每次启动的时候都要挂载，可以在 fstab 中添加相应一行配置。

如果希望 NFS 服务在每次系统引导时都要启动，可以通过 chkconfig 打开这个实现。

# /sbin/chkconfig nfs on

3. 串口软件 C-Kermit 的配置

Kermit 是一个流行的文件传输和管理协议，由哥伦比亚大学的 Kermit 项目组成员 Frank da Cruz, Christine M. Gianone 研发和维护。其中 Kermit 项目组成立于 1981 年，目前 Kermit 在 Linux 平台上的最新版本为 2004 年 4 月发布的 8.0.211 版本。

C-Kermit 协议独立于平台，可以在 Windows、Unix、Linux 等多个平台上运行，能够支持串口和网络链接方式，能基于链接进行终端会话、传输文本或二进制文件。

用法：kermit [filename][-x arg [-x arg]…[-YYY]..][{=,--,+}text]]

或者：kermit URL

其中，-X 表示选项要求有参数；-Y 表示选项不要求带参数。

使用 C-Kermit 的交互 kermit 命令可支持多种连接方式，如 Telnet、Rolgin、SSH、Modem、串口等。

- 创建 Telnet 连接

  在 C-Kermit 命令行状态，输入如下：

  #telnet foo.bar.com //列出期望的主机名或地址。

  #telnet xyzcorp.com 3000 //包含端口号

  如果连接成功，Kermit 就自动进入连接状态。

- 创建 Rlogin 连接

  与 Telnet 方式类似，不同的是用户应以 root 方式登录，因为 Rlogin（Remote Login）使用特许的 TCP 端口。

  #rlogin foo.bar.com //列出期望的主机名或地址

- 创建 SSH 连接

  与 Telnet 和 Rlogin 不同，SSH 并没有内建在 C-Kermit 中，但可以通过在一个伪终端上运行 SSH 客户端建立，使用 C-Kermit 来控制 SSH 客户端可以使你获得所有的 Kermit 特性。

  #ssh foo.bar.com //列出期望的主机名或地址

- 串口直连

  用一个电缆直接将两台计算机连接。

  #set modem type none //没有调制解调器

  #set line /dev/ttyS0 //设置设备名

  #set carrier-watch off //关闭载波监控器

  #set speed 57600 //设置速度

  #set flow rts/cts //如果 RTS 和 CTS 被交叉连接

  #set flow xon/xoff //如果无法使用 RTS/CTS

#connect //进入连接状态

- 下载文件

利用 Kermit 下载文件的方法如下：

#kermit –x oofa.txt //oofa.txt 为文件名

- 上传文件

利用 Kermit 上传文件的方法如下：

#kermit –g oofa.txt //oofa.txt 为文件名

如果 Kermit 的第一个命令行选项为 Telnet、Ftp、Iksd 或者 HTTP URL，Kermit 会自动创建相应的连接。

#kermit telnet:xyzcorp.com //打开一个 Telnet 会话

#kermit ftp://olga@xyzcorp.com/public/oofa.zip //下载文件

#kermit http://www.columbia.edu/kermit/index.html//抓取网页

更详细的内容请参考官方网页为：http://www.columbia.edu/kermit/index.html。

# 第 9 章　设计之道

**学而不思则罔，思而不学则殆。**

《论语·为政》

软件研发发展至今，经历了曲曲折折的过程，已经逐渐摆脱了小作坊式的生产方式，其工程化的轮廓已经逐渐显现，软件研发不再是一个人和少数人的即兴之作，而变得渐渐有迹可循。

软件开发成功的例子很多，失败的例子也不胜枚举，究其原因，各种各样。有管理的原因，也有技术或者资金的原因，但从设计的角度而言，凡是糟糕的设计一般都出现了下面的情况：

1. 过分强调技术

技术上的领先固然能够提升产品的竞争力，但技术只是产品成功的重要因素之一，在产品的研发上，就技术方面应该注重实效、适可而止，只要用户希望或者可以达到期望的效果，哪怕是陈旧的技术也应该加以保留或者采用。在这方面，SMS 就是一个成熟技术成功的典型。

2. 缺乏用户研究

用户研究对一个产品而言某种意义上是觉得产品成败的一个重要因素，从大的方面来讲，什么样的产品才是用户需要的；从小的方面而言，如何从细微处来改善用户体验，这些都是用户研究所关注的课题。

什么样的产品才是用户需要的？从 nokia 大多数手机朴实无华的外表和 thinkpad 笔记本近乎永恒的外观但两者都能一直站在所处领域的巅峰就会明白，真正能够长久吸引消费者眼光的是产品的质量和良好的用户体验。只有真正从用户角度大力改善用户体验和具有过硬质量的产品才会在激烈的商场竞争中立于不败之地。

2007 年下半年，在国内的手机行业疯狂流行电视营销，对产品的性能进行近乎没有边际的放大，一定程度上迷惑了部分不明真相的购买者，但这样饮鸩止渴的销售办法显得目光过于短浅，最终在业绩短暂的狂欢过后，带来的是整个企业的沉沦。这种缺乏战略眼光的举措是对消费者心理的最大误解。

2007 年 10 月 13 日，联想推出迄今为止联想最高端的家用 PC 锋行 King，价格接近 4 万元。联想特意请到了魔兽的世界前三甲的选手 moon 和 sky 来参与鼠标和键盘设计，以获得更好的游戏感受。抛弃价格不谈，以用户参与的方式最大程度的把握用户的需求是产品能够打动人心的关键。

下面从人机交互和软件设计两方面来分析设计需要遵从的原则。

## 9.1 人机交互

设计是一项复杂的工作，涉及到很多学科，工程师设计出桥梁、大坝，也设计出电路和新型材料。“设计”一词被用在服装、建筑、室内装修和园艺等各个领域。设计人员大多是艺术家，他们强调产品的美感，而有时候则更关心成本。设计实际是一个对表面上相互冲突的各种要求进行协调的过程，总之，大部分产品的研发与众多学科有关。

要想设计出以人为中心、方便适用的产品，设计人员从一开始就要把各种因素考虑进去，协调与设计相关的各类学科。设计的目的是让产品为人所用，因此，用户的需求应当贯穿在整个设计过程之中。

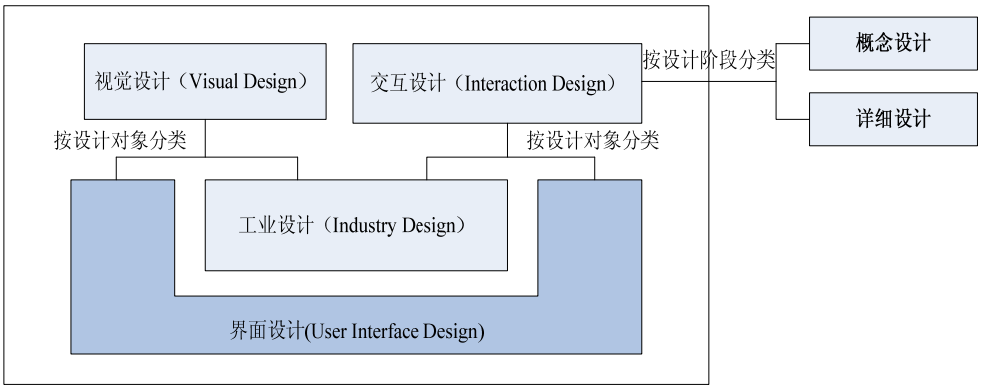


图46. 交互设计

所谓交互设计，就是通过研究人和产品（包括软硬件）的互动关系，定义产品的工作方式，它如何把其功能传达给用户，让用户有理想的使用体验。所谓视觉设计包含数字产品（通常是软件、网站）的外观设计，包括图形设计、颜色设计、字体设计，也扩展到动画设计、声效设计，通过塑造产品的美感（包括颜色、形态、构图、纹理、材质等等）来使用户获得理想体验。通常说的GUI设计是交互设计和视觉设计的统称，为给使用者带来的来自操作逻辑和美感两方面的感受的综合。需要说明的是交互设计与接口设计不同，接口设计主要集中在产品的观感的细节，而交互设计则包含了更丰富的内容，说明的是产品应该具有什么样的功能[36]。

在软件研发的过程中，设计也同样是个很重要的问题，如何在艺术美、可靠性、安全性、易用性、成本何功能之间寻找到关键的平衡点，如何更好的设计出满足用户需要、让用户操作方便同时又能对用户产生较强的吸引力的产品是每个 GUI 设计人员都必须考虑的课题。

为了更容易的表述概念，在参考文献[45]中，将设计人员所使用的概念模型成为设计模型，将用户在与协调交互作用的过程中形成的概念模型称为用户模型，而系统表象基于系统的物理结构(包括用户使用手册和各种标识)。设计人员希望用户模型与设计模型完全一样，但问题是，设计人员常常无法与用户直接交流，必须通过系统表象这一渠道。如果系统表象不能清晰、准确的反映出设计模型，用户就会在使用过程中、建立错误的概念模型。

对于用户而言，当用户面对一台新购的手机枉然不知所措，不知道该如何才能使用时，这时候最大的可能是手机的 GUI 设计出现了问题。

产品设计是设计人员的设计模型在产品上的体现，进一步说，是设计人员在用户研究的直接和间接成果在产品上的体验。设计人员必须明白，在产品的操作和相应的结果直接实际存在着一种自然的对应关系。只有能深刻把握住用户的使用习惯，使设计模型和用户模型实现无缝对接，才能产生一个优秀的设计。

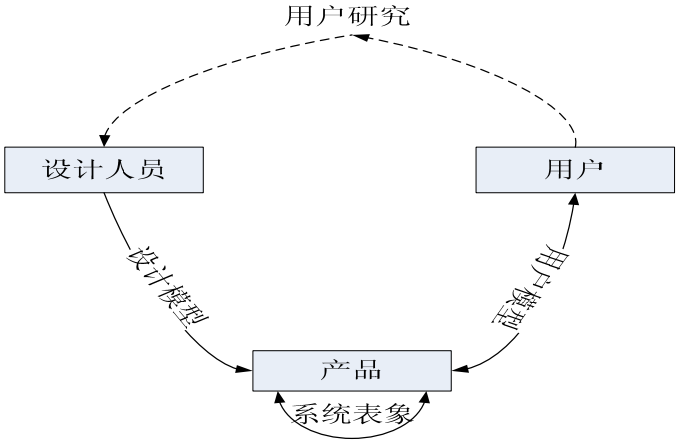


图47. 模型间的依赖关系

要想设计出一款优秀的、让用户很容易的就能看得懂、知道怎么使用的产品。设计人员从一开始就要把各种因素考虑进去，并对表面上相互冲突的各种要求进行协调。找到一个各种要素的平衡点，需要说明的是产品的易用性只是设计中需要考虑的一个要素，它并不凌驾于其他要素之上，任何伟大的阿设计都是再艺术美、可靠性、安全性、易用性、成本和功能之间寻求平衡与和谐。

### 9.1.1 用户第一

人的大脑是一个绝妙的器官，它有它自己的理解方式，对于设计人员而言，必须理解大脑在面对一个新事物的时候的理解方法，如果所进行的设计违背了其中的规律，那么在用户使用所设计的产品时，往往会觉得千头万绪、毫无规律。一旦使用出错，抱怨就会迭起，呵呵，如果恰好公司的服务热线不错，相信公司的领导层会很快知道自己公司推出了一个糟糕的产品，一定会感叹：哦，天啊，这可是我们公司花了××年的辛苦啊，花掉了××的研发费用。结果……。

笔者曾经使用过一款手机的闹钟功能，花了很久时间都没搞懂该如何去定时，界面上没有任何提示的信息，后来又与同事一起研究，仍然不得其解，如此设计不能不说过于糟糕，设计的目的不是为了炫耀自己的技术，而在于有效的对功能进行整合，为用户提供一个更好的体验。

笔者自己的一款某著名企业生产的手机，在短信这样常用的功能的设计上也有着糟糕的表现，一次笔者想单独清空收件箱时，发现竟然手机没有提供这样的功能，要么完全清空收件箱、发件箱、草稿箱，要么只能单独清空发件箱。这让笔者大跌眼镜。无法相信这样的设计竟然是出自著名公司之手。

另外，需要提及的是，在 GUI 设计人员和软件研发人员沟通时，往往出于某种需要，软件研发人员不乐意主动去给 GUI 设计人员提与设计相关的非技术性建议，认为自己需要关注的只是技术上的问题，至于如何设计、设计的好和坏则是 GUI 设计人员的事情，而忽略了软件设计是一个有机的整体，每一个设计上的瑕疵都可能招致最终用户的不满，打爆公司的服务热线。没有提供收件箱单独清空这一功能就体验了这样的问题，当然也有另外的情况，那就是在研发的过程中，有关人员已经提出了类似的缺陷，但却得不到产品定义人员的积极响应，以至于最终没有将建议体现在产品上。

### 9.1.2 概念模型

一个好的概念模型使我们能够预测操作行为的效果，在进行 GUI 设计时，如果没有一个好的概念模型，而是凭着自己天马行空式的灵感进行设计，往往会招致意想不到的结果，尤其在设计一个成熟的用品时，毕竟在人们的心中已经存在了一个操作惯式。这在设计学中称为心理模型（mental models），心理模型是指人们通过经验、训练和教导，对自己、他人、环境以及接触到的事物形成的模型。因此在进行物品的设计时，不必过分了解物品的内部实现机制，所侧重的应该是操作何操作结果之间的关系。

### 9.1.3 可视性

一个物品的心理模型大多产生于人们对该物品可感知到的功能和可视结构进行解释的过程中。在现代产品的设计上，由于产品承载的功能越来越多、越来越负责，这对产品的可视性而言越来越构成一个挑战。

对于一个产品而言，通过提供给用户足够的可视信息，让用户能够建立起一个完善的心理模型，让产品变得更加易用，这对产品的用户研究而言，是个很重要的课题。

产品缺乏可视性，这在很多著名公司的产品中也是一个问题，笔者的同事曾经使用过一部一家市场占有率在世界绝对领先的公司出品的最顶级的手机。当他想进行拍照时，问题就

出现了，尽管笔者同事从事手机的研发工作多年，按说在手机的使用上也是专业人士，但遗憾的是，在找拍照操作的程序入口问题上，笔者同事花费了很长的时间。笔者自己也有类似的经历，在使用不同公司的相似产品时，发现也许是出于个性化的考虑，类似功能的操作入口，相似产品与主流产品放置的位置往往不同。这样在用户使用产品时，往往就会浪费大量的时间查找。因为潜意识上。当产品缺乏可视性，无法为用户建立起一个新的心理模式时，用户的操作会按照用户的以往使用类似产品的概念模型来进行。

### 9.1.4 匹配原则

匹配是指两种事物之间的关系，在本书中特指操作与其产生的结果之间的关系。以手机上的按键为例，通话键一般为绿色，开关机键则为红色，这和人们的心理习惯保持了一致，因为在人们的心目中，红绿灯的概念已经深入人心，“绿”表示通行，“红”表示停止。即使对于一个从未接触过手机的用户而言，潜意识的上红绿按键也表示着类似的功能，如果有一天，某家公司推出了表示相反含义的红绿键，相信该公司的服务电话会被打爆。

匹配设计是指利用物理环境类比和文化标准理念设计出让用户一看就明白如何使用的产品。

一个良好的设计必须考虑到设计是否和人们的心理模型是否匹配。

### 9.1.5 反馈原则

对操作的结果进行适当的反馈，这也是设计的一个重要方面。如果没有反馈，用户就无法知道操作是否获得了成功，就会心存顾虑。是没有正确操作？按键按的轻了？触笔用力太小？特别是机器在执行常时间操作时，比如正在下载网页，因为速度比较慢，页面可能一段时间内就是空白，如果没有进度条或者文字之类的提示，用户可能就会很困惑，它在干什么？正在处理还是死机了？如果时等待时间很长，用户就会很自然的倾向于“它死机了！天哪，我没干什么啊”，直接的结果就是关机重启，造成不必要的损失。

## 9.2 软件设计

设计的目的不在于华丽的外观，在于易用、实用，短信的出现就有力的佐证了这一点，从技术上而言，短信并不包含特别的另外的技术，但其操作的方便性和实用性导致了短信的极大发展。成为了一个成功的应用。

### 9.2.1 研究目的

为什么要研究软件设计那？从某种角度而言，软件设计和其他行业的设计一样，存在着从实践中总结规律上升为理论，再用理论指导实践的过程。

总结前人的经验，主要有以下几个方面的考虑：

- 复用解决方案
- 采用通用术语沟通和交流
- 更高层次的视角

1. 复用解决方案

古语云“它山之石，可以攻玉”，牛顿也曾曰“站在巨人的肩膀上可以让我们看的更远”。从前人的经验教训中获取自己需要的知识，从而快速提供我们的个人技能，在现代社会，这是每个研发人员都必须具备的基础意识。在软件行业，也同样如此。

2. 通用术语

只会汉语和只会英语的人显然无法在语言上有效沟通，在软件已经超越个人范畴的今天，确立一套所有人都可以理解的通用规则，这是一个基本的课题，无规矩不成方圆，没有标准也成不了产业。当多个相关人员在一起进行交流时，没有人希望看到：每次主讲人提到一个

名词时，下面一片茫然，近乎鸡对鸭讲的场面。

由于软件行业发展所处时期的限制，大力提升软件素养，采用通用术语来描述所进行的设计是当前软件设计领域一个十分重要的课题。

3. 更高层次的视角

人的思考能力是有限的，过于停留在细节，就无法在更大的场景中思考问题。通常情况下，当你描述电视时只会用“电视”一个名词来概括，而不会花费半天功夫从高度、宽度、按钮数量等等细节去描述所知的实物。

在软件设计中也是一样，在描述设计时，用“组合模式”来描述你在采用组合模式进行设计时采用的一个设计思想远比你口干舌燥的比划和用感性语言表述半天更有效，让对方理解的更加准确。利用通用术语，可以让谈论的内容保持在设计层次，不会被牵涉进对象和类等这些琐碎的内容上面。

### 9.2.2 命名原则

现代的软件研发早已经不是一个人的问题，常常需要多个人甚至成百上千人一起协作才能完成。同事间常常要相互阅读他人代码。因此采用一种大家都认同的、从语意上也可以让他人快速理解的命名规则显得十分重要。

变量、类的命名在软件设计中本来是个很小的问题，但遗憾的是，众多的软件研发人员由于所处环境的影响，往往只重视程序设计的语法问题，做出的常常是糟糕的命名，无法清晰表达标识符自身含义甚至极易造成他人误解的命名常常充斥着整个工程，如果没有注释，常让需要阅读代码的同事或者他人往往汗流浃背才搞明白其中的含义，极大的影响了软件研发的效率！

软件研发如同构造精美的艺术，好的代码往往能让人伤心悦目，好的命名规则往往能让人最快速度明白代码编写者的意图，Qt在这方面完全可以树立为一个标杆，如QStringListModel，一看就知道是QString的list的模型类，采用的是MV模式。

良好的设计不仅包括标识符的命名规则、类等实体的设计，也包括体系结构的设计，设计绝对不是一般软件研发人员认为的那样是一门花拳绣腿式的门面功夫，而是软件研发的真谛，优雅的设计不仅能够让人容易理解、使程序运行高效、健壮性强，也具有着良好的可扩展性。

比较著名的命名规则当推 Microsoft 公司的“匈牙利”命名法，该命名规则的主要思想是“在变量和函数名中加入前缀以增进人们对程序的理解”。例如所有的字符变量均以 ch 为前缀，若是指针变量则追加前缀 p。如果一个变量由 ppch 开头，则表明它是指向字符指针的指针。但“匈牙利”命名法的最大缺点是烦琐。

根据前人的经验，在标识符命名时，应该遵循以下几个方面的原则：

- 标识符应当直观且可以拼读，可望文知意，不必进行“解码”。
- 标识符的长度应当符合“min-length && max-information”原则。
- 命名规则尽量与所采用的操作系统或开发工具的风格保持一致。
- 程序中不要出现仅靠大小写区分的相似的标识符。
- 程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解。
- 变量的名字应当使用“名词”或者“形容词＋名词”，类的成员函数应当只使用“动词”，被省略掉的名词就是对象本身
- 用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。
- 尽量避免名字中出现数字编号

更多的信息请参考参考文献[19]和参考文献[21]。

### 9.2.3 面向对象

在参考文献[25]中，作者根据他们多年的经验和对面向对象技术的领悟，为我们进一步揭开了面向对象的面纱，用流畅的文笔为我们指明了在面向对象设计中必须注重的几个要点：

- 优先使用对象组合而不是类继承
- 发现并封装变化点
- 按照场景来进行设计

下面来简要介绍以上几个要点的思想。

1. 优先使用对象组合而不是类继承

在实际的研发和维护中，需求可能是经常变化的，举个最常见的例子，在游戏研发中，里面的角色可能有不同的道具，同样的工具武器——剑可能有多把，短剑，长剑，带勾的剑？使用剑的行为也可能有多种变化，刺、砍、劈？也许还有我们在最初的设计中没有考虑到的方式，比如在维护过程中，对刺的这个使用剑的行为如果需要细分，直刺、挑刺？为了在最初的研发中能够尽可能的去适应以后的变化，系统必须要具有足够的弹性。尽管我们真的无法准确预测将来的需求为发生怎样的变化或者扩展！呵呵，这是个相当严重的课题。通过优先使用对象组合而不是类继承何以最大限度的解决这类问题。在实际的设计中，应该考虑遵循该策略，如果遵循该策略不会使设计和实现的开销严重增加，就要考虑采用该策略，它能够为你带来长期的利益。当然短期的开销却是不可避免的。

2. 发现并封装变化点

找出应用中可能需要变化之处，把他们独立出来并加以封装，不要让他们和那些不需要变化的代码混在一起。这样在以后根据需要对该部分代码进行改动或者加以扩充时，就不对对其他部分造成影响。系统就会变得更加有弹性。

事实上，面向对象的每个类的设计都体现了这个原则，研发人员在使用按钮时，不必去考虑菜单的风格如何，windows 风格还是 mac 风格！不必去考虑显示的文本的字体如何！不必去考虑点击后其外观的行为变化！这些对于研发人员而言都是变化点，都不是应用层需要考虑的内容。通过 QPushButton 类对这些外观的行为变化进行封装，可以较大限度的减少系统设计的复杂性，更加能够适应需求的变化。

3. 按照场景来进行设计

正如人的行为必与人的性格相符合，战术服从与战略一样，设计也必须服从于场景，如何设计必须符合实际的场景，成功的设计必然是来自于对场景的充分理解，没有对场景的充分理解，就没有成功的设计。

在进行设计以前，一定要充分的其理解场景，把握好本质的需求，尤其是需要充分了解需求在短期和长期中可能存在的变化。关于如何充分的理解场景，把握需求的本质，在软件开发过程中称为需求分析，这是一个十分重要的问题，能否真实把握客户的需求，直接关系到软件产品的成功或失败。关于需求分析的更多内容请参考参考文献[30]。

在进行细节的设计以前，一定要充分的把握整体的视图，然后再去做功能分解的工作，窥一斑而欲适应全貌总是一件很危险的事情。

在进行细节的设计时，通过对共同点和变化点的分析和细节所处的场景，可以看到当前必须处理的情况，和将来可能出现的变化，然后设计人员才有可能根据不同策略的代价的大小来决定采用何种策略，这通常比采用其他方式来进行设计更容易获得可扩展性的效果，缺点则是会增加一些细微的开销。

对共同点和变化点的分析是细节设计的一个重要环境，需要注意的是一定要放在整个场景下去分析，通过和其他细节的关联来分析可能存在的变化。在进行设计决策时，以下因素是衡量设计优缺的重要参考：

- 场景未来是否会存在变化，选择的设计能否适应变化。
- 软件设计目的，是否应该去适应可能的变化。
- 采用的编程语言
- 数据库或配置文件的可用性

需要强调的是，在进行方案选择时，方案本身并没有绝对的优劣，选择的依据是那个方案更能适应当前的场景。更多的知识请参考参考文献[25]。

# 第 10 章 Qtopia设计

他山之石，可以攻玉。

《诗经・小雅・鹤鸣》

架构是一个软件系统中的核心元素，是系统中最难改变的部分，也是构建软件系统中其他部分所依赖的基础，因此系统架构的好坏会从根本上决定基于这个架构所构建的软件系统的质量。架构（Architecture）是一个系统的基本组织，它蕴含于系统的组件中、组件之间的相互关系中、组件与环境的相互关系中、以及呈现于其设计和演进的原则中。

好的架构具有和谐、弹性、可靠等特性。

## 10.1 设计模式的运用

在软件领域，我们常说的设计模式指的是GoF模式[24]，更准确的讲是“面向对象的设计模式”，是一套人们在面向对象设计过程中被反复使用、经过分类编目的、代码设计经验的总结。但实际上，模式是针对特定背景下的特定问题的可重复、可表达的解决方案。它不局限于面向对象，也不局限于设计阶段，甚至不局限于软件研发领域，事实上，设计模式的开山之作《The Timeless Way of Building》讲述的就是建筑领域的模式，使用设计模式是为了可重用代码，让代码更容易被他人理解、保证代码的可靠性。使用模式的最好方法为：“把模式印到你的脑子里，然后在你的设计和已有的应用中，寻找何处可以使用他们”[26]。

Gamma 、 Helm 、 Johnson 、 Vlissides 的 《 Design Patterns - Elements of Reusable Object-Oriented Software》在软件领域第一次将设计模式提升到理论高度，并将之提炼、总结出了 23 种基本的设计模式，由此该书也成为了设计模式领域的圣经，人们亲切的将Gamma、Helm、Johnson、Vlissides称为四人团。

按照《The Timeless Way of Building》中Christopher Alexander的定义,模式就是“在某一个场景下的问题的解决方案”，在参考文献[25]中，Alan Shalloway和James R. Trott曾多次提到《The Timeless Way of Building》，这让笔者无比神往，是什么书籍让两大高手如此敬佩那？遗憾的是本人一直挖空心思的在网上寻找这本书，无奈在将百度、google等搜索工具发挥到极致后仍然两手空空，也就无缘在本书中为读书简单介绍这么模式的经典之作了。如果那位读者手头刚好有这本书的电子版，不妨拷贝给笔者一份，十分感谢哟！

### 10.1.1 Composite 模式

Composite模式的意图是“将对象组合成树形结构以表示‘部分-整体’的层次结构”[24]。基本对象可以被组合成组合对象，组合对象也可以被组合，可以不断的递归下去。Composite模式使得用户对单个对象和组合对象的使用具有一致性，这样就有效的简化了用户代码。下图为QObject类的设计。

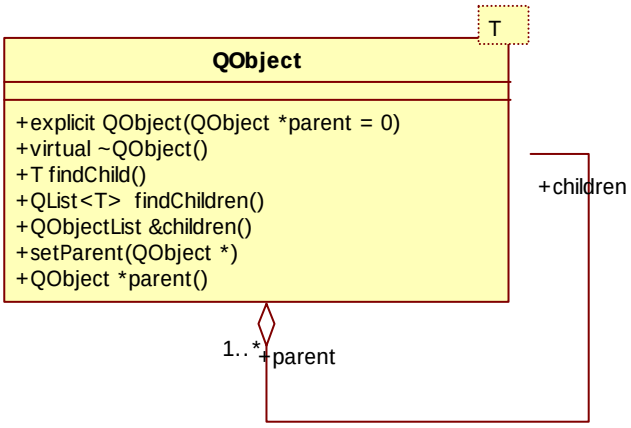


图48. QObject 的设计

在 Qt 中，TrollTech 的研发人员将 Composite 模式的威力发挥的淋漓尽致，通过对单根 QObject 的实现，将 Composite 模式变成了整个 Qt 的基石。

在 Qt 的 QObject 设计中，采用了一个 QObjectData 作为 QObject 对象组合的基础。通过 QObjectData-> parent 设置其上一级 QObject 对象，通过 QObjectData-> children 来存储其下一级 QObject 对象的列表。

```
class QObjectData {
    ……
    QObject *parent;
    QObjectList children;
    ……
};
```

当一个 QObject 对象被创建时，QObject 对象会自动的将传过来的父对象作为自身的上一级 QObject 对象也即父类。如果父对象为空，则该 QObject 对象即为最高以及的 QObject 对象，自然需要显式销毁了。

```
QObject::QObject(QObject *parent)
    : d_ptr(new QObjectPrivate)
{
    Q_D(QObject);
    ……
    setParent(parent);
}
```

当 QObject 对象被销毁时，QObject 对象会自动检查其 children 列表，逐个进行销毁。下面是 QObject 对象被销毁时的处理过程。

```
QObject::~QObject()
{
    Q_D(QObject);
    ……
    if (!d->children.isEmpty())
        d->deleteChildren();
    ……
}
void QObjectPrivate::deleteChildren()
{
    const bool reallyWasDeleted = wasDeleted;
    wasDeleted = true;
    for (int i = 0; i < children.count(); ++i) {
        currentChildBeingDeleted = children.at(i);
```

```
            children[i] = 0;
            delete currentChildBeingDeleted;
        }
        children.clear();
        currentChildBeingDeleted = 0;
        wasDeleted = reallyWasDeleted;
    }
```

从以上分析可以看出，这就是 Qt 中父子化机制的本质。为了最大限度的减少内存泄漏，Qt 利用组合模式完美的实现了这一目的。

下面是布局管理器的设计：

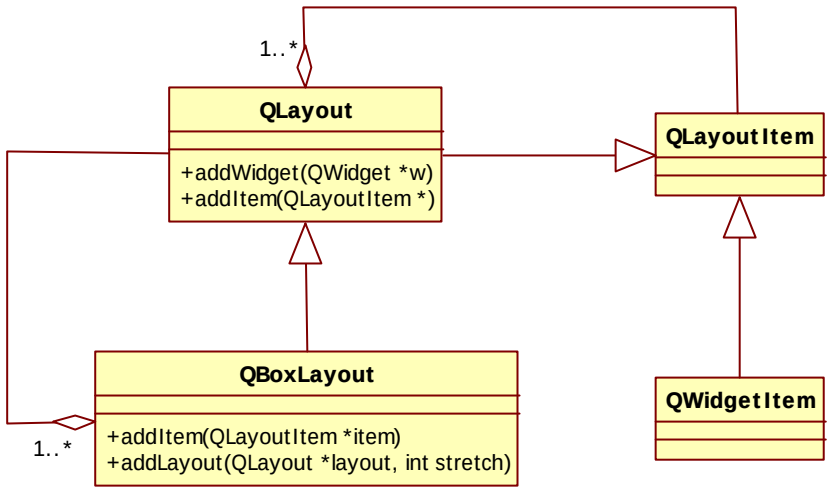


图49. 布局管理器的设计

对于布局管理器而言，基于 Composite 模式可以很简单的对各种简单布局 QHBoxLayout、QVBoxLayout、QgridLayout 等进行组合，从而实现较理想的布局。

需要说明的是，无论是 QObject 还是布局管理器，其存储结构都是基于 QList 的，但实际上 Composite 模式还可以基于其他存储结构如数组、哈希表、树等，在具体设计时需要综合考虑实现难度和查找速度。

### 10.1.2 Singleton 模式

Singleton模式的意图是“保证一个类仅有一个实例，并提供一个访问它的全局访问点”[24]。通过拥有一个特定的方法来实例化需要的对象，当这个方法被调用时，它会检查这个对象是否已经被实例化，如果发现该对象已经被实例化，则该方法仅返回该对象的一个引用，否则才实例化该对象并返回一个引用。

QCoreApplication 包含了主事件循环，来自操作系统和其他事件源的所有事件如定时器事件、网络事件等都要经过 QCoreApplication 来进行分发。同时 QCoreApplication 还对应用程序的构造和销毁进行管理，这就要求 QCoreApplication 在一个进程中应该是唯一的，下面是 QCoreApplication 利用单子模式的实现过程。

```
void QCoreApplication::init()
{
    ……
    QCoreApplication::self = this;
    ……
}
```

下面是 QCoreApplication 的类定义：

```
class Q_CORE_EXPORT QCoreApplication : public QObject
```

```
{
    Q_OBJECT
    public:
        ……
        static QCoreApplication *instance() { return self; }
        ……
    private:
        ……
        static QCoreApplication *self;
        ……
}
```

需要说明的是，在每个进程中，作为事件分发器的 QAbstractEventDispatcher 也是唯一存在的。

当需要维护的单子对象为一个组合时，实现方法同样类似，下面是 Feature 类利用单子模式实现的过程：

```
Feature* Feature::getInstance(const QString &key)
{
    QString ukey = key.toUpper();
    if (!instances.contains(ukey))
        instances[ukey] = new Feature(ukey);
    return instances[ukey];
}
```

下面是 Feature 类中关于单子模式的部分。

```
class Feature : public QObject
{
    Q_OBJECT
public:
    static Feature* getInstance(const QString &key);
    ……
signals:
    void changed();

private:
    Feature(const QString &key);
    ……
    static QMap<QString, Feature*> instances;
    FeaturePrivate *d;
};
```

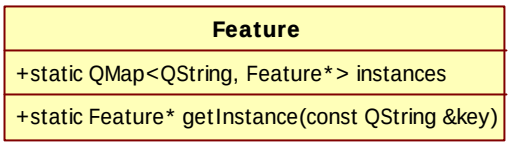


图50. Feature 的设计

在 Qt 中运用 singleton 模式的地方除了 Feature 以外，还有 QTransportAuth、ContentServer、ResultIndicator、QTimeStringData、QPackageRegistry 、GAREnforcer 等类。最重要的是在 QCoreApplication 类的设计上，Qt 也采用了 singleton 模式,保证了一个进程只有一个事件循环。

```
class Q_CORE_EXPORT QCoreApplication : public QObject
{
    ……
```

```
    static QCoreApplication *instance() { return self; }
    ……
    static QCoreApplication *self;
}
```

Singleton 模式是个非常重要的应用，特别是在系统性能低下时，因为系统反应缓慢，常会引起用户的疑惑，我是不是操作没成功？从而因继续操作而给系统带来更大的开销。进而引发一系列问题，通过 Singleton 模式可以有效的减少此类的消耗。避免异常的发生！

### 10.1.3 Observer 模式

Observer模式的意图是“定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新”[24]。

下面是 Qt 中 Observer 和 Subject 类的设计：

Observer 类的声明：

```
class Observer
{
public:
    virtual ~Observer() { };
    virtual void update( Subject* subject ) = 0;
};
```

Subject 类的声明：

```
class Subject
{
public:
    virtual ~Subject() { };
    virtual void attach( Observer* observer );
    virtual void detach( Observer* observer );
    virtual void notify();
protected:
    Subject() { };
private:
    QList<Observer*> m_observers;
};
```

Subject::attach()函数的定义：

```
inline void Subject::attach( Observer* observer )
{
    m_observers.append( observer );
}
```

Subject::detach()函数的定义：

```
inline void Subject::detach( Observer* observer )
{
    m_observers.removeAll( observer );
}
```

Subject::notify()函数的定义：

```
inline void Subject::notify()
{
    QList<Observer*>::iterator it = m_observers.begin();
    for (;it != m_observers.end(); ++it)
    {
        (*it)->update( this );
    }
}
```

下图为 Qt 在设计 HelixPlayer 时运用 Observer 模式的情况，当播放的进度和音量等信息发生了改变时，Subject 对象会调用 notify 函数通知 HelixSession 和 HelixPlayer。

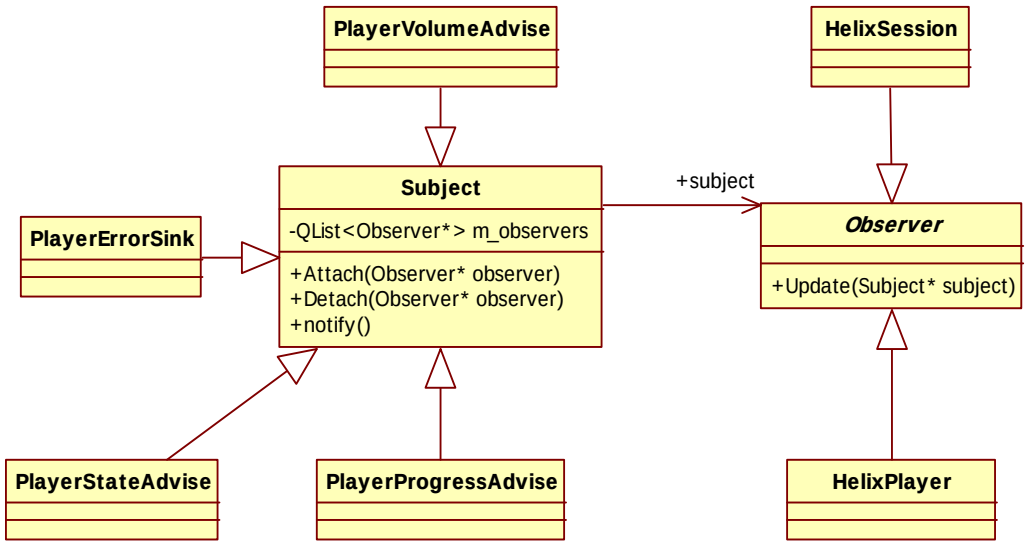


图51. HelixPlayer 的设计

下面是 HelixPlayer 的初始化过程：

```
HelixPlayer::HelixPlayer(IHXClientEngine* engine):
    m_stopTimerId(0),
    m_engine(engine)
{
    if (HXR_OK == m_engine->CreatePlayer(m_player))
    {
        m_progressadvise = new PlayerProgressAdvise;
        HX_ADDREF( m_progressadvise );
        m_progressadvise->attach( this );  // 将 HelixPlayer 注册到 PlayerProgressAdvise
        m_player->AddAdviseSink( m_progressadvise );
        m_stateadvise = new PlayerStateAdvise(this);
        HX_ADDREF( m_stateadvise );
        m_stateadvise->attach( this );      // 将 HelixPlayer 注册到 PlayerStateAdvise
        m_player->AddAdviseSink( m_stateadvise );
        IHXAudioPlayer *audioplayer;
        if( m_player->QueryInterface( IID_IHXAudioPlayer, (void**)&audioplayer ) ==
        HXR_OK )
        {
            m_volume = audioplayer->GetAudioVolume();
            HX_RELEASE( audioplayer );
        }
        else
        {
                    REPORT_ERROR( ERR_HELIX );
                }
                m_volumeadvise = new PlayerVolumeAdvise;
                HX_ADDREF( m_volumeadvise );
                m_volumeadvise->attach( this ); // 将 HelixPlayer 注册到
    PlayerVolumeAdvise
                m_volume->AddAdviseSink( m_volumeadvise );
                m_sitesupplier = new HelixSiteSupplier( m_player );
                HX_ADDREF( m_sitesupplier );
                m_sitesupplier->attach( this );
                m_errorsink = new PlayerErrorSink;
```

```
                HX_ADDREF( m_errorsink );
            m_errorsink->attach( this ); // 将 HelixPlayer 注册到 PlayerErrorSink
            if( m_player->QueryInterface( IID_IHXErrorSinkControl, (void**)&m_errorcontrol )
            == HXR_OK )
            {
                m_errorcontrol->AddErrorSink( m_errorsink, HXLOG_EMERG,
                HXLOG_INFO );
            }
            else
            {
                        REPORT_ERROR( ERR_HELIX );
            }
            m_context = new GenericContext( QList<IUnknown*>()
                << m_sitesupplier
                << m_errorsink );
            HX_ADDREF( m_context );
            m_player->SetClientContext( m_sitesupplier );
        }
    }
```

这样当播放的进度、状态、音量等发生变化，或者发生错误时，Subject 类可以调用 notify ()函数通知 HelixPlayer 有关信息的变化，但 Observer 仅仅能被动的接收信息有时是不够的，Observer 还需要能够主动的获得信息。下面是 HelixPlayer 的 update()函数的处理过程：

```
    void HelixPlayer::update( Subject* subject )
    {
        if( subject == m_progressadvise ) {
            PlaybackStatus::notify();
        } else if( subject == m_stateadvise ) {
            PlayerState::notify();
        } else if( subject == m_volumeadvise ) {
            VolumeControl::notify();
        } else if( subject == m_sitesupplier ) {
            VideoRender::notify();
        } else if( subject == m_errorsink ) {
            ErrorReport::notify();
        }
        else {
            REPORT_ERROR( ERR_UNEXPECTED );
        }
    }
```

但是并不是说每当对象之间存在依赖关系时，就可以使用 Observer 模式，并且能取得良好的效果，当依赖关系固定时，加入一个 Observer 模式可能只是增加了复杂性，比如在设计手机上的 Camera 时，其图像设置的变化只有一个特定的观察对象即摄像头。这时候就没必要使用 Observer 模式了。

Observer 模式为多个对象间的数据同步提供了一个可靠的方案。尤其在观察对象比较多的情况下就更显其威力。

在 Qt 中，内置的信号槽机制就是 Observer 模式的一个典型例子。

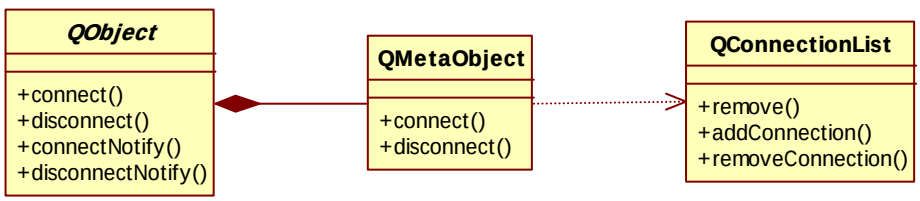

图52. Observer 模式在 QObject 中的运用

在 QObject 中，由于采用了 Composite 模式，QObject 得以即可以作为 Subject 类存在，也同时作为 Observer 类存在！也正是得意于对 Observer 模式的巧妙运用，Qt 才得以拥有威力强大的信号与槽机制。这是整个 Qt 架构设计的灵魂所在。

### 10.1.4 Factory 模式

Factory模式的意图是“定义一个用于创建对象的接口，让子类决定实例化哪一个类，使一个类的实例延迟到其子类”[24]。

当有一群需要实例化的具体类时，究竟实例化那个类，要在运行时才能根据外界条件决定，这是就要考虑工厂模式，下面是 QWSEvent 利用工厂模式的实现过程：

```
QWSEvent *QWSEvent::factory(int type)
{
    QWSEvent *event = 0;
    switch (type) {
    case QWSEvent::Connected:
        event = new QWSConnectedEvent;
        break;
    case QWSEvent::MaxWindowRect:
        event = new QWSMaxWindowRectEvent;
        break;
    case QWSEvent::Mouse:
        event = new QWSMouseEvent;
        break;
    case QWSEvent::Focus:
        event = new QWSFocusEvent;
        break;
    case QWSEvent::Key:
        event = new QWSKeyEvent;
        break;
    case QWSEvent::Region:
        event = new QWSRegionEvent;
        break;
    case QWSEvent::Creation:
        event = new QWSCreationEvent;
        break;
#ifndef QT_NO_QWS_PROPERTIES
    case QWSEvent::PropertyNotify:
        event = new QWSPropertyNotifyEvent;
        break;
    case QWSEvent::PropertyReply:
        event = new QWSPropertyReplyEvent;
        break;
#endif // QT_NO_QWS_PROPERTIES
    case QWSEvent::SelectionClear:
        event = new QWSSelectionClearEvent;
        break;
    case QWSEvent::SelectionRequest:
        event = new QWSSelectionRequestEvent;
        break;
    case QWSEvent::SelectionNotify:
        event = new QWSSelectionNotifyEvent;
        break;
#ifndef QT_NO_COP
```

```
        case QWSEvent::QCopMessage:
            event = new QWSQCopMessageEvent;
            break;
#endif
        case QWSEvent::WindowOperation:
            event = new QWSWindowOperationEvent;
            break;

#ifndef QT_NO_QWS_INPUTMETHODS
        case QWSEvent::IMEvent:
            event = new QWSIMEvent;
            break;
        case QWSEvent::IMQuery:
            event = new QWSIMQueryEvent;
            break;
        case QWSEvent::IMInit:
            event = new QWSIMInitEvent;
            break;
#endif
#ifndef QT_NO_QWSEMBEDWIDGET
        case QWSEvent::Embed:
            event = new QWSEmbedEvent;
            break;
#endif
        default:
            qCritical("QWSEvent::factory() : Unknown event type %08x!", type);
        }
        return event;
}
```

将具体的实现放在子类中进行，这样就预先封装了变化点，使得子类的变化不会对程序的其他部分构成影响。

除了前面介绍的 Factory、文件管理采用了 Factory 模式的设计外，QFormBuilder 在设计上也采用了类似的策略。作为 QAbstractFormBuilder 的子类，将自己的具体实现进行了封装。确保了自身的变化不会影响到其他的部分。

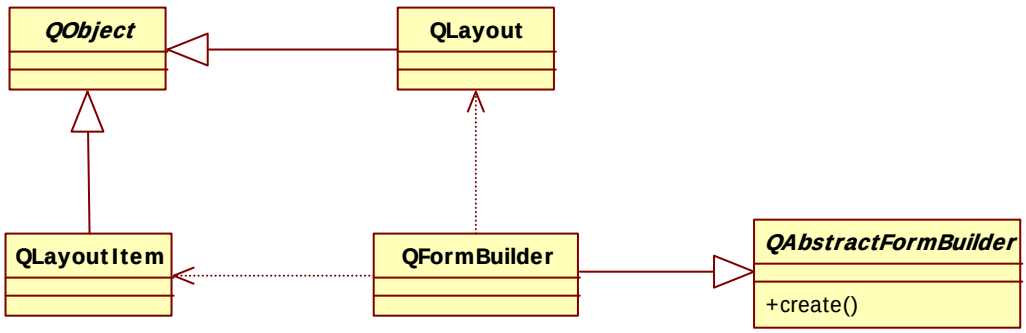


图53. QFormBuilder 的设计

当一个类不知道它将要创建的对象或者，它希望由它的子类来指定需要创建的对象时，可以考虑采用工厂模式的设计。

### 10.1.7 Strategy 模式

Strategy模式的意图是“让你可以使用不同的业务规则或算法——取决于它们出现的场景”[24]。

Strategy模式基于以下原则建立：

- 对象拥有责任
- 这些责任的不同的特定实现通过使用多态来表现
- 需要将几个不同的实现按照概念上相同的算法来管理
- 一个好的设计经验：将问题领域中发生的行为彼此分离——也就是说，使他们解耦，这让我可以改变对某一行为负责的类，而不会对其他行为产生不好的影响。

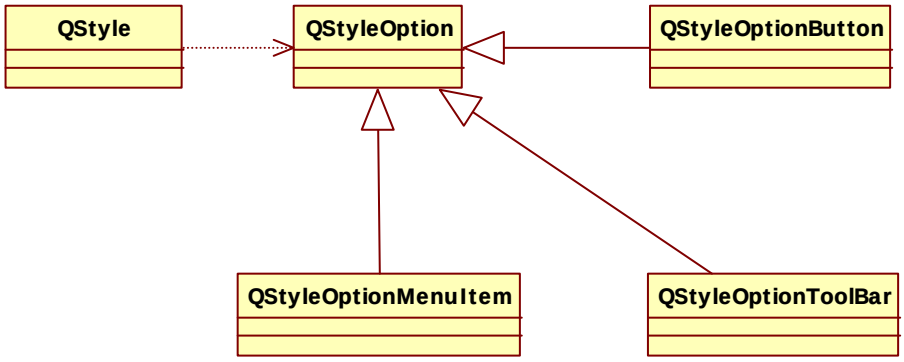


图54. QStyleOption 的设计

### 10.1.8 Qt 模式

Qt 模式？笔者自命名的模式，如果熟悉 Qt 源代码，与通常的通过设置访问权限将数据封装不同，你会发现在 Qt 的设计中采用了接口函数与数据放在不同类中实现的原则。笔者姑且将其成为“qt 模式”。根据笔者的理解，qt 模式的意图是“通过将具体类的数据放在其私有类中实现，可以封装可能存在的变化并并数据设置访问权限保证了数据安全”。

下面首先介绍下 Qt 模式需要用到的几个宏：

```
#define Q_DECLARE_PRIVATE(Class) \
inline Class##Private* d_func() { return reinterpret_cast<Class##Private *>(d_ptr); } \
    inline const Class##Private* d_func() const { return reinterpret_cast<const Class##Private
*>(d_ptr); }
friend class Class##Private;
```

从上面的代码可以看出，在一个类的定义中如果加入了宏 Q_DECLARE_PRIVATE(Class)，表示声明了一个 Class##Private 作为该类的友元类，同时为该类添加了一个成员函数 d_func()，通过该成员函数，可以获得一个指向类 Class##Private 的指针。

```
#define Q_DECLARE_PUBLIC(Class) \
        inline Class* q_func() { return static_cast<Class *>(q_ptr); } \
        inline const Class* q_func() const { return static_cast<const Class *>(q_ptr); } \
        friend class Class;
friend class Class;
```

从上面的代码可以看出，在一个类的定义中如果加入了宏 Q_DECLARE_ PUBLIC (Class)，表示声明了一个 Class 类作为该类的友元类，同时为该类添加了一个成员函数 q_func()，通过该成员函数，可以获得一个指向类 Class 的指针。

```
#define Q_D(Class) Class##Private * const d = d_func()
```

通过 Q_D 宏，声明了一个指向 Class##Private 类的指针 d。

```
#define Q_Q(Class) Class * const q = q_func()
```

通过 Q_Q 宏，声明了一个指向 Class 类的指针 q。

下面以 QObject 为例来说明 Qtopia 模式的实现。

下面是 QObject 的定义：

```
class Q_CORE_EXPORT QObject
```

```
{
    Q_OBJECT
    ……
    Q_DECLARE_PRIVATE(QObject)          //声明私有类。
    ……
protected:
    QObjectData *d_ptr;                 //声明 d_ptr 指针
    ……
};
```

下面是 QObjectData 的定义：

```
class QObjectData {
public:
    QObject *q_ptr;                     //声明 q_ptr 指针
    ……
};
```

下面是 QObjectPrivate 的定义：

```
class Q_CORE_EXPORT QObjectPrivate : public QObjectData
{
    Q_DECLARE_PUBLIC(QObject)           //声明具体类
    ……
};
```

下面是 QObject 类的构造函数。

```
QObject::QObject(QObject *parent, const char *name)
    : d_ptr(new QObjectPrivate)
{
    Q_D(QObject);
    ……
}
```

这样就可以利用 d 指针接入 QObjectPrivate 进行操作了。

当需要在私有类中调用具体类的公有成员函数进行操作时，可以如下实现：

```
QObjectList QObjectPrivate::receiverList(const char *signal) const
{
    Q_Q(const QObject);
    ……
}
```

这样就可以利用 q 函数接入 QObject 进行操作了。

需要说明的是，私有类并非只包含数据，也可以拥有成员函数。私有类在定义时其数据和成员函数都是具有共有访问权限的。在作为具体类的私有类。其数据和成员函数对于具体类而言都是属于私有性质，这和普通的 C＋＋语法中将私有信息利用 Private 访问权限的方法进行限制是一样的，但 qt 模式的目的在于简化单个类的大小，降低了实现的复杂度，同时又能够将变化点进行封装，增强了具体类的不变性，达到了简化设计的目的。

## 10.2 架构设计

GOF 模式等通常适用于类的设计，在较小的场景中应用 GOF 模式可以获得意向不到的效果，但对软件的整体设计而言，只具有战术上的意义，为了更好的进行设计，必须对软件整体具有洞察力，这就要求在更广阔的场景中总结、提炼成功的经验。下面简要介绍以下 Qtopia 在架构上的几点设计原则。

### 10.2.1 Qtopia Core

Qtopia Core 是一个针对嵌入式设备的 GUI 和应用程序的 C++框架，能够在多种处理器上运行，它提供了标准的 Qt API 为嵌入式设备提供了轻量级的窗口系统。

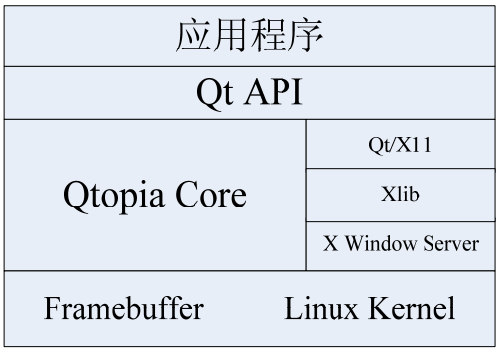


图55. Qtopia Core 的位置

Qtopia Core 基于 C/S 架构，当系统启动后，必须有一个应用程序充当服务器端，任何 Qtopia Core 应用程序都可以作为一个服务器端，当有多个应用程序运行时，随后的应用程序会主动作为一个客户端连接到当前的服务器端上。

作为服务器端而言，应用程序需要管理鼠标（触笔）、字符处理和屏幕输出等，而客户端应用程序主要负责执行特定的操作。

在实现上，服务器端作为一个 QWSServer 类的实例出现，而客户端则作为一个 QWSClient 类的实例出现，所有的客户端产生的系统事件如按键事件和鼠标事件(在嵌入式中为触摸屏事件）都讲被传递给服务器端进行处理，客户端只执行操作。

当进行渲染（render）时，默认的方式为客户端将其窗口部件放入内存，而由服务器端负责将内存中的内容渲染到屏幕上，但对于嵌入式而言，Qtopia Core 还支持由客户端直接操作将窗口部件渲染到显示屏上的方式。

1. C/S 通信

当有窗口部件被添加或移去时，服务器端通过 QWSWindow 对象来维护相关的顶层（top-level）窗口信息。当服务器端收到一个事件时，它会首先查询其维护顶层窗口信息的栈，找到与相应窗口对应的客户端的 ID，然后服务器端通过 QWSEvent 来封装事件，并将事件发送给相应的客户端。

服务器端和客户端和客户端之间的通信是基于 UNIX 的域套接字（domain socket）进行的，通过重载 QApplication 的 qwsEventFilter()函数，可以截获从服务器端发送给客户端的所有事件。

在实现中，Qt 通过 QCopChannel 类对 UNIX 的域套接字进行了封装，每个 QCopChannel 信道都有唯一的名字标识，如果需要接收该信道的消息，则需要调用 QCopChannel::receive()函数。

2. 鼠标（触笔）处理

当服务器端启动时，服务器端应用程序会导入鼠标（触笔）驱动程序，当发生鼠标（触笔）点击事件时，鼠标（触笔）驱动程序会通过 QWSEvent 将该事件进行封装，然后再将事件发送给服务器端。

### 10.2.2 Model/View

MVC（Model View Controller）是一个起源于Smalltalk的设计模式，常常被用于构建用户接口，在参考文献[24]中，对MVC模式是这样描述的：

MVC 模式有三个对象组成，模型（Model）是应用对象，视图（View）是它在屏幕上的表示，控制器（Controller）定义了用户界面对用户输入的响应方式，不使用 MVC 模式，用户界面设计往往将这些对象混在一起，而 MVC 模式则将他们分离开来提供了灵活性和复用性。

从中我们可以体会到，视图和控制器实现了经典的策略模式，视图是一个对象，可以被调整使用不同的策略，而控制器则提供了策略，视图只关心系统中可视的部分，对于任何界面行为，都委托给控制器处理。使用策略模式也可以让视图和模型之间的关系解藕，因为控

制器负责和模型交互来传递用户的请求，对于工作是如何完成的，视图则一无所知。

在 qt 实现中，视图和控制器对象被合并，这样仍然做到了数据存储和显示的分离，但基于同样的原则却提供了一个简化的框架（framework），为了更灵活的处理用户的输入，在 Qt 中引入了代理（delegate）的概念，将代理引入框架的好处在于它允许通过自定义的方式渲染和编辑数据。

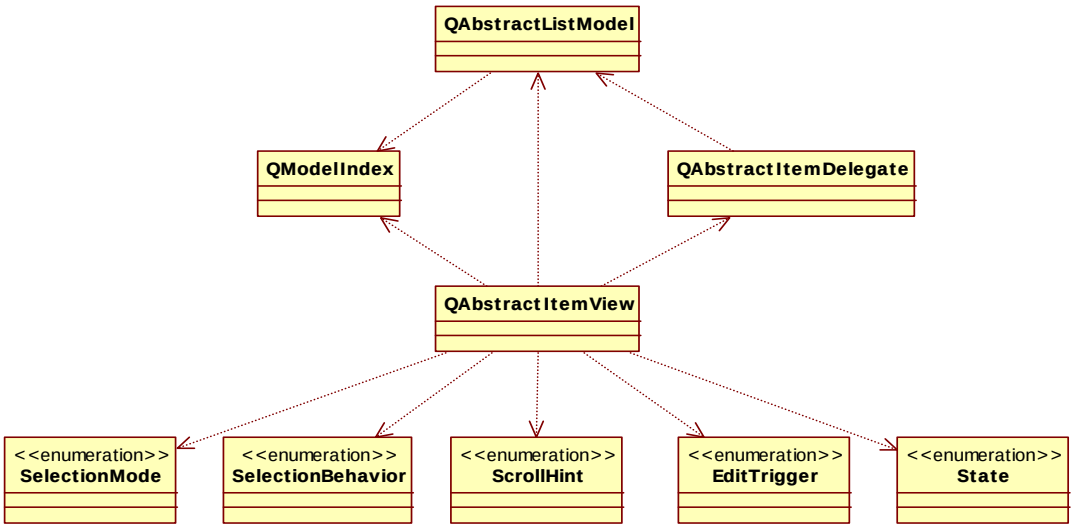


图56. 模型视图架构

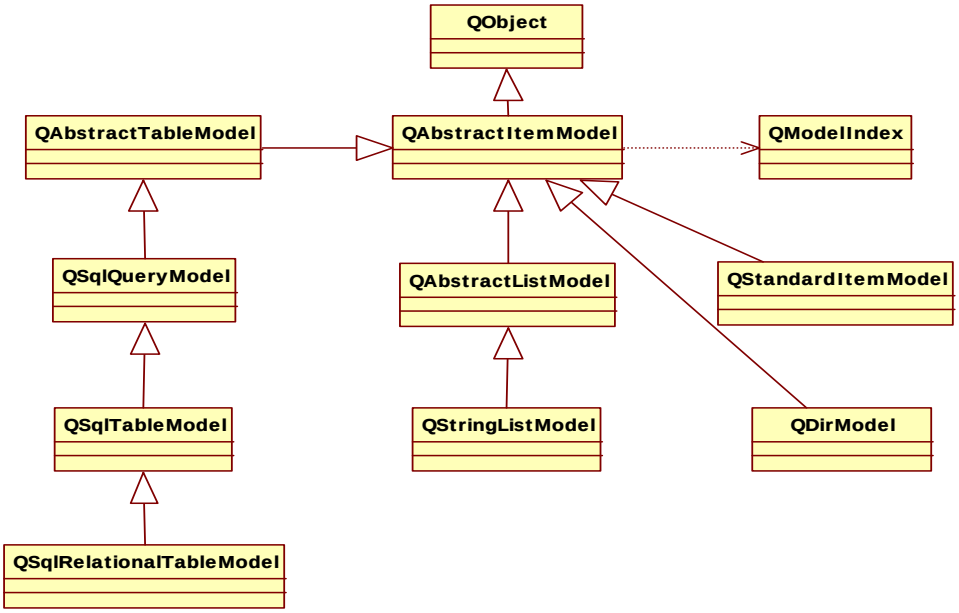


图57. Model 类图

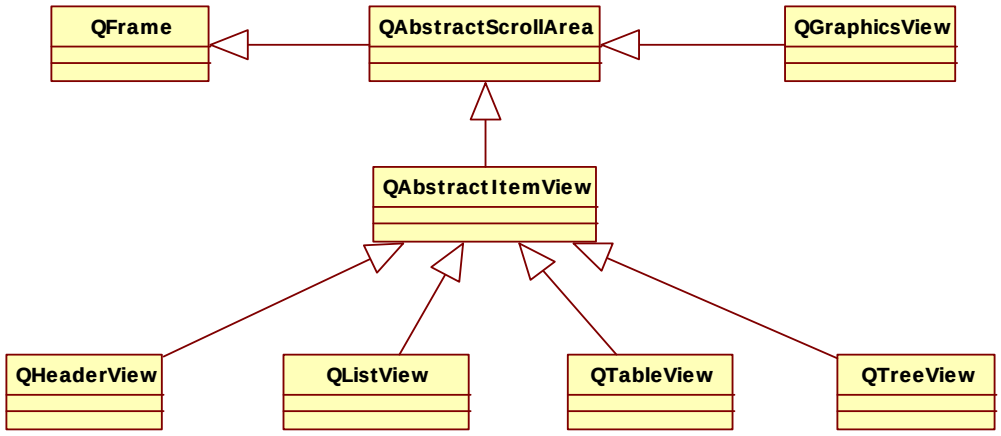


图58. View 类图

模型为架构中的其他组件提供了间接与数据源通信的接口，其与数据源通信的方式依赖于数据源类型和模型的实现方式。

视图通过模型索引来检索数据源条目。

在标准视图中，由代理来渲染数据源条目，当数据源条目被编辑时代理通过模型索引来直接于模型通信。

在 Qt 实现中，一般而言，模型、视图、代理被定义为抽象类来提供公共接口，通过继承来实现特定功能的需要，模型、视图、代理利用信号与槽的方式通信。当发生点击事件时，视图会发出 clicked(const QModelIndex &index)信号，当发生双击事件时，视图会发出 doubleClicked(const QModelIndex &index)信号，当数据发生变化时，模型会发出 dataChanged(const QModelIndex &topLeft, const QModelIndex &bottomRight)信号。

# 第 11 章 书籍简介

**尽信书，不如无书。**

《孟子·尽心下》

## 11.1 Qt

1. C++ GUI Programming with Qt 4

作者：Jasmin Blanchette, Mark Summerfield

介绍 Qt 的书籍本来就少，精品就更是寥寥了，幸运的是除了 Trolltech 提供的帮助文档外，我们还有这本书可以参详，Jasmin Blanchette, Mark Summerfield 出于自己对 Qt 的深刻把握，更是喊出了“Qt 4.1 编程唯一最佳实践向导”的口号，本书的功力也就可见一斑了。

备注：

- 英文版：Jasmin Blanchette, Mark Summerfield, C++ GUI Programming with Qt 4, Prentice Hall, June 2006

2. An Introduction to Design Patterns in C++ with Qt 4

作者：Alan Ezust, Paul Ezust

本书是唯一以 C++、设计模式和 Qt 4 跨平台研发结合为写作目标的书。很遗憾，对内容的深度似乎挖掘的并不够。不过有总比没有强啦，所以笔者还是推荐读一读。

备注：

- 英文版：Alan Ezust, Paul Ezust,An Introduction to Design Patterns in C++ with Qt 4 ,Prentice Hall,August 2006
- 中文版：李仁见 战晓明 译，《C++设计模式--基于Qt4 开源跨平台研发框架》，清华大学出版社，2007 年 8 月

3. Qt 及 Linux 操作系统窗口设计

作者：倪继利

本书不仅阐述了 KDE 及 X Window 的机制，分析了 Qt 及 Qt/Embedded 研发工具的核心技术，而且还详细介绍了如何在嵌入式设备上建立 Qtopia 窗口系统。本书号称“凝聚研发经验，具备专业深度”，笔者本人曾经买过一本，深感受益匪浅，值得收藏。

备注：

中文版：倪继利 著，《Qt 及 Linux 操作系统窗口设计》，电子工业出版社，2006 年 4 月

## 11.2 C/C++

C++是一门广泛用于工业软件研发的大型语言。它自身的复杂性和解决现实问题的能力，使其极具学术研究价值和工业价值。和 C 语言一样，C++已经在许多重要的领域大获成功。

然而，一个不可否认的现实是，在低阶程序设计领域，C++挤压着 C 同时也在承受着 C 的强烈反弹，而在高阶程序设计领域，Java 和 C#正在不断蚕食着 C++的地盘。也许 C++与 C 合为一体永远都是一个梦想，也许 Java 和 C#的狂潮终将迫使 C++回归本位 — 回到它有着根本性优势的研发领域：低级系统程序设计、高级大规模高性能应用设计、嵌入式程序设计以及数值科学计算等。

C++吸引如此之多的智力投入，以至于这个领域的优秀作品，包括重量级的软件产品、程序库以及书籍等，数不胜数。

1. The C++ Programming Language

作者：Bjarne Stroustrup

迄今为止，本书是除了 C++标准文献之外最权威的 C++参考手册。本书对 C++语言的描述轮廓鲜明、直截了当。它从 C++语言创建者的角度来观察 C++，这是任何别的作者和书籍做不到的，或者说没有任何人比 Bjarne 自己更清楚该怎么来使用 C++。

这也是一本严肃的著作，以中、高级 C++研发人员为目标读者。如果你是一名有经验的 C++程序员，需要了解更加本质的 C++知识，本书正是为你而写。它不是那种让你看了会不断窃喜的小书，需要用心体会，反复咀嚼。

作者小传：

Bjarne Stroustrup：C++之父，现任 AT＆T 实验室的大型程序设计研究部的主管。1990 年，Bjarne 荣获《财富》杂志评选的"美国 12 位最年轻的科学家"称号。1993 年，由于在 C++领域的重大贡献，Bjarne 获得了 ACM 该年度的 Grace Murray Hopper 大奖并成为 ACM 院士（成立于 1947 年的 ACM 协会是历史最悠久、目前世界上最大的教育和科学计算协会，成为 ACM 院士是个人成就的里程碑）。1995 年，BYTE 杂志颁予他"近 20 年来计算机工业最具影响力的 20 人"的称号。

备注：

- 英文版：Bjarne Stroustrup,The C++ Programming Language (Special 3rd Edition),Addison Wesley, February 2000
- 影印版：《C++程序设计语言（特别版）》，高等教育出版社，2001 年 8 月
- 中文版：裘宗燕 译,《C++程序设计语言（特别版）》，机械工业出版社，2002 年 7 月

2. C++ Primer

作者：Stanley B. Lippman, Josée Lajoie, Barbare E.Moo

本书是久负盛名的 C++经典教程，其内容是 C++大师 Stanley B. Lippman 丰富的实践经验和 C++标准委员会原负责人 Josée Lajoie 对 C++标准深入理解的完美结合。

这本书的名字多少有点让人误解。尽管作者声称这本书是为 C++新手而写，但无论是它的厚度还是讲解的深度都暴露了似乎并非如此。也许说它是一本"从入门到精通"的 C++教程会更合适一些。我个人认为它并不适合完全不懂 C++的初学者 — 在阅读这本书之前，你至少应该先有那么一点 C 或 C++的背景知识，或者至少要具有一些其他语言的编程经验。

尽管这本书省略了一些高级 C++特性的讨论，但仍然可以称得上是迄今为止最全面的 C++学习教程。事实上，如果一名 C++初学者能够扎扎实实地读完本书并对照《C++ Primer Answer Book》完成全部习题的话，他的水平肯定可以进入职业 C++程序员的行列。我个人认为，即使你已经拥有了 TCPL，这本书依然有拥有的价值，因为在许多方面它比 TCPL 来得更详细、更易懂。

作者小传：

Stanley B.Lippman 的职业是提供关于 C++和面向对象的训练、咨询、设计和指导。他在成为一名独立咨询顾问之前，曾经是迪士尼动化公司的首席软件设计师。当他在 AT&T Bell 实验室的时候，领导了 cfront 3.0 版本和 2.1 版本的编译器研发组。他也是 Bjarne Stroustrup 领导的 Bell 实验室 Foundation 项目的成员之一，负责 C++程序设计环境中的对象模型部分。他还撰写了许多关于 C++的文章。目前他已受雇于微软公司，负责 Visual C++项目。遍及全球，深受广大 C++学者的喜欢。

备注：

- 英文版：Stanley B. Lippman, Josée Lajoie, Barbara E. Moo,C++ Primer,Fourth Edition，Addison Wesley，Feb 2005
- 影印版：《C++ Primer（第 4 版）》，人民邮电出版社，2006 年 11 月
- 中文版：李师贤 蒋爱军 梅晓勇 林瑛等译,《C++ Primer（第 4 版）》，人民邮电出版社,2006 年 3 月

3. Thinking C++

作者：Bruce Eckel

《C++编程思想》（第 1 版）荣获 1996 年度《软件研发》杂志的图书震撼大奖（Jolt Award），成为该年度最佳图书。这同样是一本享有崇高声誉的经典之作。

作者小传：

Bruce Eckel：MindView 公司（www.MindView.net）总裁，向客户提供软件咨询和培训。

他是 C++标准委员会拥有表决权的成员之一。从 1986 年至今，已经发表了超过 150 篇计算机技术文章，出版了 6 本书（其中 4 本是关于 C++的），并且在全世界做了数百次演讲。他是《Thinking in Java》、《Thinking in C++》、《C++ Inside & Out》《Using C++》和《Thinking in Patterns》的作者，同时还是《Black Belt C++》文集的编辑。他的《Thinking in C++》一本书在 1995 年被评为“最佳软件研发图书”，《Thinking in Java》被评为 1999 年 Java World“最爱读者欢迎图书”，并且赢得了编辑首选图书奖。

备注：

- 英文版：Bruce Eckel ,Thinking in C++, Volume 1: Introduction to Standard C++, Second Edition, Prentice Hall,April 2000
- 英文版：Bruce Eckel ,Thinking in C++, Vol. 2: Practical Programming, Second Edition, Prentice Hall,November 2003
- 影印版：《C++编程思想（第 2 版）》，机械工业出版社, 2002 年 1 月
- 中文版：刘宗田 袁兆山 潘秋菱等译，《C++编程思想（第 2 版） 第 1 卷：标准 C++导引》，机械工业出版社 ，2002 年 9 月
- 中文版：刁成嘉 等译,《C＋＋编程思想－第 2 卷 实用编程技术》，机械工业出版社,2006 年 1 月

4. Ruminations on C++: A Decade of Programming Insight and Experience

作者：Andrew Koenig, Barbara E. Moo

Andrew 是世界上屈指可数的 C++专家。这是一本关于 C++编程思想和程序设计技术而非语言细节的著作。如果你已经具有一定的基础，这本书将教你在进行 C++编程时应该怎样思考，应该如何表达解决方案。整本书技术表达透彻，文字通俗易懂。Bjarne 这样评价这本书：本书遍布“C++是什么、C++能够做什么”的真知灼见。

作者小传：

Andrew Koenig ： AT&T 大规模程序研发部（前贝尔实验室）成员。他从 1986 年开始从事 C 语言的研究，1977 年加入贝尔实验室。他编写了一些早期的类库，并在 1988 年组织召开了第一个完全意义上的 C++会议。在 ISO/ANSI C++委员会成立的 1989 年，他就加入了该委员会，并一直担任项目编辑。他已经发表了 C++方面的 100 多篇论文，在 Addsion-Wesley 出版了 C Trap and Pitfalls,还应邀到世界各地演讲。

Barbara Moo 现任 AT&T 网络体系结构部门负责人。在 1983 年加入贝尔实验室不久，她开始从事 Fortran 77 编译器的研究工作，这时第一个用 C++编写的商业产品，她负责 AT&T 的 C++编译器项目直到 AT&T 转让出软件研发业务，她还负责指导 SIGS 会议，Lund 技术学院和 Stanford 大学。

Andrew Koenig 和 Barbara Moo 堪称 C++研究领域的“第一神仙眷侣”，他们不光有着多年的 C++研发、研究和教学经验，而且还亲身参与了 C++的演化和变革，是对 C++的变化和发展起到重要影响的人。

备注：

- 英文版： Andrew Koenig, Barbara E. Moo, Ruminations on C++:A Decade of Programming Insight and Experience, Addison Wesley Professional,August 1996
- 中文版：黄晓春 译，《C++沉思录》，人民邮电出版社，2002 年 11 月

5. The Design and Evolution of C++

作者：Bjarne Stroustrup

本书是一本关于 C++语言设计原理、设计决策和设计哲学的专著。它清晰地回答了 C++为什么会成为今天这个样子而没有变成另外一种语言。作为 C++语言的创建者，Bjarne 淋漓尽致地展示了他独到而深刻的见解。除了广受赞誉的语言特性外，Bjarne 没有回避那些引起争议的甚至被拒绝的 C++特性，他一一给出了逻辑严密、令人信服的解释。内容涵盖 C++的史前时代、带类的 C、C++的设计规则、标准化、库、内存管理、多重继承、模板等，对包括异常机制、运行时类型信息和名字空间在内的重要的新特性都分别进行了深入探讨。每一名 C++程序员都应该可以从 Bjarne 的阐释中加深对手中这门语言的认识。

备注：

- 英文版：Bjarne Stroustrup, The Design and Evolution of C++, Addison Wesley, March 1994

- 影印版：《C++语言的设计和演化》，机械工业出版社，2002 年 1 月
- 中文版：裘宗燕 译,《C++语言的设计和演化》，机械工业出版社，2002 年 1 月

6. Inside the C++ Object Model

作者：Stanley B. Lippman

这本书让你知道：一旦你能够了解底层实现模型，你的程序代码将获得多么大的效率。Lippman 澄清了那些关于 C++额外负荷与复杂度的各种错误信息和迷思，但也指出其中某些成本和利益交换确实存在。他阐述了各式各样的实现模型，指出它们的进化之道及其本质因素。本书涵盖了 C++对象模型的语意暗示，并指出这个模型是如何影响你的程序的。

C++成山似海的书籍堆中，这一本不是婴幼儿奶粉，也不是较大婴儿奶粉，它是成人专用的低脂高钙特殊奶粉。 对于C++底层机制感兴趣的读者，这本书会给你"漫卷诗书喜欲狂"的感觉。 了解 C++ Object Model，是学习 Component Object Model 的最短路线。 如果你是一位 C++程序员，渴望对于底层知识获得一个完整的了解，那么 Inside The C++ Object Model 正适合你。

备注：

- 英文版：Stanley B. Lippman, Inside the C++ Object Model, Addison Wesley, May 1996
- 影印版：《深度探索 C++对象模型》，中国电力出版社，2003 年 8 月
- 中文版：侯捷 译,《深度探索 C＋＋对象模型》,华中科技大学出版社,2001 年 5 月

7. C++ Strategies and Tactics

作者：Robert B.Murray

本书在一开始就向我们讲解了如何为我们的设计选择正确的抽象，提示我们注意抽象和实现之间的区别—然后，我们就将学到如何将已得到的抽象转化成一个(或多个)C++中的类，期间进行的讨论所涵盖的范围上至高层的设计策略，下至底层的接口和实现细节。
对于 C++中新增的模板特性，通过从基础开始到逐步地接触实际应用中的示例，Rob Murray 向我们展示了其空前的洞察力。作者同时也向我们展示了多种特定的技巧，以使我们的程序更快、重用性更高，并且更健壮。异常是 C++中另外一个新增的特性，对于何时该使用它，何时不该使用它，Murray 也向我们给出了他的建议-在本书的最后，我们还可以学到如何将一个项目从 C 移植到 C++之上，书中对该过程的讨论不但包括了其中可能出现的技术问题，也包括了使用技术的"人"的问题。

作者小传：

Robert B．Murray：量子数据系统公司(Quantitative Data Systems)中负责软件工程的副总裁，该公司的业务包括向财富 500 强(Fodune 500)中的公司提供面向对象的软件解决方案在此之前，他曾经在 AT&T 的 Bell 实验室工作，在那里他参与了 C++语言、编译器以及库的研发。他同时也是《The C++ Report》杂志的创立编辑(founding editor)，主持 Obfuscated C++专栏。从 1987 年起，他就开始在学术会议和专业会议上向人教授 C++语言。目前 Murray 在 FNIS 公司进行着软件研发的管理，他的兴趣主要包括：程序语言的研发和工具，以及轻量级的研发过程(如极限编程)。

备注：

- 英文版：Robert B．Murray, C++ Strategies and Tactics, Addison Wesley, 1993
- 中文版：王昕 译，《C＋＋编程惯用法－高级程序员常用方法和技巧》，中国电力出版社，2004 年 2 月

8. Effective C++&&More Effective C++

作者：Scott Meyers

本书是世界顶级 C++大师 Scott Meyers 成名之作的第二版。其第一版诞生于 1991 年。在国际上，本书所引起的反响之大，波及整个计算机技术出版领域，余音至今未绝。几乎在所有 C++书籍的推荐名单上，本书都会位于前三名。作者高超的技术把握力，独特的视角、诙谐轻松的写作风格、独具匠心的内容组织，都受到极大的推崇和仿效。甚至连本书简洁明快的命名风格，也有着一种特殊的号召力。

如果说《Effective C++》主要讨论 C++中一些相对基础的概念和技巧的话，那么《More Effective C++》则着重探讨了包括异常处理在内的一系列高级技术。与前者相比，后者具有两大主要区别：其一，它包含很多时新的标准 C++的内容；第二，它讨论的主题倾向于"战略化"而非"战术化"，并且讨论得更深入、更彻底。尤其是对虚析构函数、智能指针、引用计

数以及代理类（proxy classe）等技术和模式论述的深入程度，让人很难想象是出现于这样的一本小书之中。

游刃有余的技术，高超的写作技巧，Scott 无疑是世界上最优秀的 C++技术作家之一。在简洁、清晰、易读等方面，这两本书都卓尔不群。总之，Scott 提供的这 85 个可以改善编程技术和设计思维的方法，都是中、高级 C++程序员必备的技能。我强烈推荐这两本书（实际上还有一本，稍后就会看到）。

作者小传：

Scott Meyers：世界顶级的 C++软件研发技术权威之一。他是两本畅销书 Effective C++和 More Effective C++的作者，以前曾经是 C++ Report 的专栏作家。他经常为 C/C++ Users Journal 和 Dr. Dobb's Journal 撰稿，也为全球范围内的客户做咨询活动。他也是 Advisory Boards for NumeriX LLC 和 InfoCruiser 公司的成员。他拥有 Brown University 的计算机科学博士学位。

备注：

- 英文版: Scott Meyers,Effective C++ Third Edition 55 Specific Ways to Improve Your Programs and Designs, Addison Wesley Professional, May 2005
- 英文版：Scott Meyers,More Effective C++: 35 New Ways to Improve Your Programs and Designs, Addison Wesley, Professional 1996
- 影印版：《Effective C++（第二版）》，中国电力出版社，2003 年 5 月
- 影印版：《More Effective C++》，机械工业出版社，2006 年 4 月
- 中文版：《Effective C++：改善程序技术与设计思维的 55 个有效做法（第三版）》，电子工业出版社，2006 年 6 月
- 中文版：刘晓伟 译，《More Effective C++中文版 35 个改善编程与设计的有效方法》，机械工业出版社，2007 年 4 月

9. Exceptional C++&& More Exceptional C++

作者：Herb Sutter

你自认为是一名 C++语言专家吗？读一读 ISO C++标准委员会秘书长的这两本书再回答。在这两本书中，Herb 采用了“问答”的方式指导你学习 C++语言特性。对于每一个专题，Herb 首先合理地设想出你的疑问和困惑，接着又猜测出你十有八九是错误的解答，然后给你以指点并提出最佳解决方案，最后还归纳出解决类似问题的普适性原则。

这两本书是典型的深究 C++语言细节的著作，很薄，但内容密集，远远超过 Scott 的那两本书，读起来很费脑筋 — 我个人认为它们要比 Scott 的书难懂得多。若要研习这薄薄的两本书所包含的知识，至少需要花费数月的时间！（在 Scott 的荐序中，他坦陈不止一次陷入 GotW 问题的陷阱，你应该知道这意味着什么）对于语言细节的深究有什么好处呢？尽管在大多数情况下，我们不必关心 C++代码幕后的动作，然而当我们不得不关心时，这两本书可以为我们提供很好的线索，因为它们揭示了 C++语言中微妙而又至关重要的东西。

作者小传：

Herb Sutter，ISO C++标准委员会主席，C++ Users Journal 杂志特邀编辑和专栏作家。他目前在微软公司领导.NET 环境下 C++ 语言扩展的设计工作。除本书外，他还撰写了三本广受赞誉的图书：《C++编程规范》 (中、英文版都由人民邮电出版社出版) 、Exceptional C++和 More Exceptional C++。

备注：

- 英文版：Herb Sutter, Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions, Addison Wesley, November 1999
- 英文版：Herb Sutter, Exceptional C++ Style 40 New Engineering Puzzles, Programming Problems, and Solutions, Addison Wesley, August 2004
- 中文版：聂雪军 译，《Exceptional C++：47 个C++难题、编程问题和解决方案》，机械工业出版社，2007 年 1 月
- 中文版：刘未鹏 译，《Exceptional C++ Style》，人民邮电出版社，2006 年 1 月
- 中文版：於春景 译，《More Exceptional C++》，华中科技大学出版社，2002 年 9 月

10. C++ Gotchas

作者：Stephen C. Dewhurst

Stephen 的理论素养和实践经验注定这是一本值得一读的好书。Stephen 曾经是贝尔实验室中第一批 C++使用者。他已经使用 C++成功解决了包括编译器、证券交易、电子商务以及嵌入式系统等领域中的问题。本书汇集了作者来自研发一线的 99 条编程真知灼见，洞悉它们，你可以避免几乎所有常见的 C++设计和编程问题。

我甚至认为，对于 C++编程菜鸟而言，阅读这本书会比阅读 Scott 和 Herb 的书更能轻松而立竿见影地获得更大的提高。我个人很喜欢这本书的写作风格 — Stephen 的许多观点看似极端却无可辩驳。当然了，这种自信（以及冷幽默）来自于作者深厚的技术素养，而非自大的偏执。

除了上面推荐的书籍外，Dov Bulka 和 David Mayhew 合著的《Efficient C++: Performance Programming Techniques》(《提高 C++性能的编程技术》，清华大学出版社）也值得一看。这本超薄小书聚焦于高性能 C++应用程序研发。两位作者都是 IBM 软件专家，都工作于对性能要求极高的系统构建领域，本书是他们的经验之谈。也有人不喜欢这本书，因为它花了不少的篇幅讲述和 C++无关的东西，我却恰恰因为这一点而对这本书产生好感，正是这些东西让我开阔了眼界。

备注：

- 英文版：Stephen C. Dewhurst,C++ Gotchas: Avoiding Common Problems in Coding and Design,Addison Wesley,Nov 2002
- 影印版：《C++ Gotchas》，中国电力出版社，2003 年 6 月
- 中文版：陈君等译，《C++程序设计陷阱》，中国青年出版社，2003 年 5 月

11. Accelerated C++: Practical Programming by Example

作者：Andrew Koenig,Barbara E.Moo

和市面上大多数 C++教程不同，本书不是从“C++中的 C”开始讲解，而是始于地道的 C++特性。从一开始就使用标准库来写程序，随着讲述的逐渐深入，又一一解释这些标准库组件所依赖的基础概念。另外，和其他 C++教材不同的是，这本书以实例拉动语言和标准库的讲解，对后两者的讲解是为了给实例程序提供支持，而不是像绝大多数 C++教材那样，例子只是用作演示语言特性和标准库用法的辅助工具。

作者在 C++领域的编程实践、教育培训以及技术写作方面都是世界一流水准。我喜欢这种大量使用标准库和 C++语言原生特性的清新的写作风格。在这本教材面前，几乎迄今为止的所有 C++教材都黯然失色或显得过时。尽管这本教材也许对于国内的高校教育来说有些前卫，不过我仍然极力向我的同行们推荐。顺带一提，Bjarne 曾经这样评价本书：对于有经验的程序员学习 C++而言，这本书可能是世界上最好的一本。

备注：

- 英文版：Andrew Koenig,Barbara E.Moo, Accelerated C++: Practical Programming by Example, Addison Wesley, January 2001
- 影印版：《Accelerated C++》,机械工业出版社,2006 年 4 月
- 中文版：覃剑锋 柯晓江 蓝图译，《Accelerated C++》，中国电力出版社，2003 年 12 月

12. C++ Templates

作者：David Vandevoorde Nicolai M.Josuttis

模板和基于模板的泛型编程无疑是当今发展最活跃的 C++程序设计技术。模板的第一个革命性的应用是 STL，它将模板技术在泛型容器和算法领域的运用展现得淋漓尽致，而 Boost、Loki 等现代程序库则将模板技术的潜能不断发挥到极致。在模板和泛型编程领域，我推荐以下两本重量级著作：

有一种老套的赞美一本书的手法，大致是“没有看过这本书，你就怎么怎么地”，这里面往往夸张的成分居多。不过，倘若说“没有看过《C++ Templates: The Complete Guide》，你就不可能精通 C++模板编程”，那么这个论断对于世界上绝大多数 C++程序员来说是成立的。

这本书填补了 C++模板书籍领域由来已久的空白。此前，上有《Modern C++ Design》这样的专注于模板高级编程技术和泛型模式的著作，下有《The C++ Standard Library》这样的针对特定模板框架和组件的使用指南。然而，假如对模板机制缺乏深入的理解，你就很难

“上下”自如。鉴于此，我向每一位渴望透彻理解 C++模板技术的朋友推荐这本书。

备注：

- 英文版： David Vandevoorde, Nicolai M. Josuttis,C++ Templates: The Complete Guide,Addison Wesley,November 2002
- 中文版:陈伟柱 译,《C++ Templates》，人民邮电出版社，2004 年 1 月

13. Modern C++ Design

作者：Andrei Alexandrescu

你自认为是 C++模板编程高手吗？请看过这本书再回答。这是一本出自天才之手令人敬畏的杰作。泛型模式，无限延伸你的视野，足以挑战任何一名 C++程序员的思维极限。

这本书共分为两大部分，第一部分讨论了 Loki 程序库采用的基础技术以及一些高级语言特性，包括基于策略的类设计、模板局部特化、编译期断言、Typelist 以及小型对象分配技术等。第二部分则着重介绍了 Loki 中的重要组件和泛型模式技术，包括泛化仿函数（Generalization Functor）、单件（Singleton）、智能指针、对象工厂（Object Factory）、抽象工厂（Abstract Factory）、访问者（Visitor）以及多方法（Multimethods）等。每一种技术都让人大开眼界，叹为观止。

在 C++的学习方面，过犹不及往往成了不求甚解的借口。然而，面向对象并非 C++的全部，模板和泛型编程亦占半壁江山。对于“严肃”的 C++程序员而言，及时跟进这项早经例证的成功技术，不失为明智之举。

备注：

- 英文版：Andrei Alexandrescu, Modern C++ Design: Generic Programming and Design Patterns Applied, Addison Wesley, Feb 2001
- 影印版：《C++设计新思维》，中国电力出版社，2003 年 5 月
- 中文版：侯捷 於春景 译，《C++设计新思维：泛型编程与设计模式之应用》，华中科技大学出版社，2003 年 3 月

14. Multi-Paradigm Design for C++

作者：James O,Coplien

C++是一种支持多种范型的编程语言：类、重载函数、模板、模块以及过程编程，等等。除了该语言的灵活性和丰富性以外，此前创建一种设计模式以支持在单个应用中使用多种范型的努力还很欠缺。本书介绍了使用多范型设计的一连串框架，提供了形成正式多范型设计方法的基础的一系列超前的设计实践。James O．Coplien 是对象范型和 C++方面的主要专家和作者，自从进入 AT&T 以后，他就一直致力于对 C++语言的研究。现在他是朗讯贝尔实验室的成员。

备注：

- 英文版： James O,Coplien, Multi-Paradigm Design for C++, Addison-Wesley Professional, 1998
- 中文版：鄢爱兰 周辉等译，《C＋＋多范型设计》，中国电力出版社，2004 年 2 月

15. C++ Standard Library: A Tutorial and Reference

作者：Nicolai M. Josuttis

这是一本百科全书式的 C++标准库著作，是一本需要一再查阅的参考大全。它在完备性、细致性以及精确性方面都是无与伦比的。本书详细介绍了每一标准库组件的规格和用法，内

容涵盖包括流和本地化在内的整个标准库而不仅仅是 STL。正如本书副标题所示，它首先适合作为教程阅读，尔后又可用作参考手册。 浅显易懂的写作风格使得这本书非常易读。如果你希望学习标准库的用法并尽可能地发挥其潜能，那你必须拥有这本书。正如网络上所言，这本书不仅仅应该摆在你的书橱中，更应该放到你的电脑桌上。我向每一位职业 C++程序员强烈推荐。

备注：

- 英文版：Nicolai M. Josuttis, C++ Standard Library: A Tutorial and Reference, Addison Wesley, August 1999
- 中文版：《C++标准程序库：自修教程与参考手册》，华中科技大学出版社 ，2002 年 9 月

16. Standard C++ IOStreams and Locales: Advanced Programmer's Guide and Reference

作者：Angelika Langer, Klaus Kreft

C++标准库由 STL、流和本地化三部分构成。关于 STL 的书市面上已经有不少，但罕见流和本地化方面的专著。本书是这两个领域中最优秀的一本，迄今为止没有任何一本书比这一本更全面详尽地讨论了流和本地化。如果你不满足于停留在“会用”流库的层面，千万不要错过它。

备注：

- 英文版：Angelika Langer, Klaus Kreft ,Standard C++ IOStreams and Locales, Addison Wesley, 2000
- 中文版：何渝 孙悦红 刘宏志 武剑 译,《标准C++输入输出流与本地化》，人民邮电出版社，2001 年 4 月

17. The C++ Standard Template Library

作者：P.J.Plauger, Alexander A. Stepanov, Meng Lee, David R.Musser

P.J.Plauger 的大作《The Standard C Library》等书都早已成为经典著作，并且目前使用最多的 MS VC++中 C++编译器中自带的 STL 就是出自他手，其实力可见一斑；Alexander A. Stepanov 被人们成为 GP 之父，STL 就是出自他天才的思想;Meng Lee 则实现了 STL 的第一个实现（HP STL）;David R.Musser,作为 Stepanov 的最初也是时间最长的合作者，对 GP 理论的形成也称得上功不可没。如此 4 人来亲身合作讲述 STL 实现，则本书的意义就自然非凡了。

备注：

- 英文版：P. J. Plauger, Alexander A. Stepanov, Meng Lee, David R. Musser ,C++ Standard Template Library, Prentice Hall PTR, December 2000
- 中文版：王昕 译,《C++ STL》，中国电力出版社，2002 年 5 月

18. Beyond the C++ Standard Library: An Introduction to Boost

作者：Björn Karlsson

Björn Karlsson 为中级至高级的 C++研发者描述了所有 58 个 Boost 库的轮廓，并完整叙述了 12 个可能最有用的库。Karlsson 的主题范围从智能指针和类型转换，到容器和数据库结构，解释了如何正确地使用每一个库来改进你的代码。他详细论述了可以让你写出更简明、清晰、易读的代码的高级函数对象。他还带你到 Boost 的"幕后"，看看那些对你创建自己的泛型库有益的工具和技术。

Boost 库已被证明了是非常有用的，它们中的大多数已准备列入下一个版本的 C++标准库。现在就开始，Beyond the C++ Standard Library.

备注：

- 英文版：Björn Karlsson, Beyond the C++ Standard Library: An Introduction to Boost, Addison Wesley, August 2005
- 中文版：张杰良 译，《超越 C++标准库:Boost 库导论》，清华大学出版社 ，2007 年 05 月

## 11.2 Linux

1. Modern Operating Systems

作者：Andrew S. Tanenbaum

Andrew S. Tanenbaum 是 Minux 的设计者，而 Linux 则是从 Minux 演化而来的，当时的芬兰小伙子 Linus Torvalds 也是学着 Minux 长大的，大师的经典巨著，怎一个“好”字了得。笔者案头就有一本。

作者小传：

Andrew S. Tanenbaum：ACM 和 IEEE 的资深会员，荷兰皇家艺术和科学学院院士，获得过 1997 年度 ACM/SIGCSE 计算机科学教育杰出贡献奖。当前，他的主要研究方向是设计规模达十亿级用户的广域分布式系统。在进行这些研究项目的基础上，他在各种学术杂志及会议上发表了 70 多篇论文，并出版了多本计算机专著。他还入选了《世界名人录》。

备注：

- 英文版：Andrew S. Tanenbaum, Modern Operating Systems 2nd Edition, Prentice Hall PTR, February 2001

- 影印版：《现代操作系统（第 2 版）》，机械工业出版社，2002 年 1 月
- 中文版：陈向群 马洪兵 等译，《现代操作系统》，机械工业出版社，2005 年 9 月

2. Operating Systems Design and Implementation

作者：Andrew S. Tanenbaum

Tanenbaum 大师的另一本巨著，同样堪称经典之作。本书详细探讨了操作系统的基本原理，包括进程、进程间通信、信号量、管程、消息传递、调度算法、输入/输出、死锁、设备驱动程序、存储管理、调页算法、文件系统设计、安全和保护机制等；此外，还详细讨论了一个特殊的操作系统 MINIX 3（一个与 UNIX 兼容的操作系统），并提供了该系统的源代码，以便于读者仔细研究。这种安排不仅可让读者了解操作系统的基本原理，而且可让读者了解到这些基本原理是如何应用到真实的操作系统中去的。

备注：

- 英文版：Andrew S. Tanenbaum, Operating Systems Design and Implementation, Third Edition, Prentice Hall, January 2006
- 中文版：陈渝，谌卫军 译，《操作系统设计与实现(第三版)》，电子工业出版社，2007 年 3 月

3. A Practical Guide to Linux® Commands, Editors, and Shell Programming

作者：Mark G. Sobell

要想真正高效地使用 Linux，就必须全面掌握 shell 和命令行。知名 Linux 专家 Mark Sobell 以深入浅出的笔触向 Linux 用户、系统管理员、软件研发者、质量控制工程师等以 Linux 操作系统为工作平台的用户描绘了常用的核心命令，对 Linux 初学者而言，是本难得的好书。笔者本人也曾深深获益，因此推荐给大家。

备注：

- 英文版：Mark G. Sobell，A Practical Guide to Linux® Commands, Editors, and Shell Programming，Prentice Hall PTR，July 01, 2005
- 中文版：杨明军 王凤芹 译，《LINUX 命令、编辑器与 SHELL 编程》，清华大学出版社，2007 年 3 月

4. How Linux Works: What Every Super-User Should Know

作者：Brian Ward

作者以初级程度的读者为对象介绍了 Linux 的文件系统、boot 过程和网络、防火墙、研发工具、设备管理、脚本语言和 Samba 服务等更高级的主题，笔者认为即使已经是一位 Linux 的大虾，本书也值得读者收藏，爱书之心，研发者皆而有之，本书确属一本难得的入门读物。

备注：

- 英文版：Brian Ward, How Linux Works: What Every Super-User Should Know, No Starch Press, 2004

5. Beginning SUSE Linux

作者：Keir Thomas

想熟练掌握 SUSE Linux 10.1 吗？那么请看看这本书，本书号称“SUSE Linux 的完全向导，包含了你所熟练使用 SUSE Linux10.1 所需要的一切！”虽然作者有夸大之嫌，但笔者依然推荐读者能够读一读，间接经验往往能够让你更快速的上手！

备注：

- 英文版：Keir Thomas,Beginning SUSE Linux,Second Edition,Apress,Nov 2006

6. Fedora Linux

作者：Chris Tyler

Fedora Linuxs 是关于 Red Hat 的社区发行版的完全向导，通过本书你可以了解到 Fedora core 是如何工作的。

备注：

- 英文版：Chris Tyle, Fedora Linux, O'Reilly, October 2006

7. Managing Projects with GNU make

作者：Robert Mecklenburg

自 1970 年问世以来，make 至今仍旧是大多数程序研发项目的核心工具，它甚至被用来编译 Linux 内核。阅读本书，读者将可以了解，尽管出现了许多新兴的竞争者为何 make 仍旧是

研发项目中编译软件的首选工具。.GNU make 已经成为行业标准，这本书是关于 GNU make 的权威著作，如果想成为 Linux 下的编程更好，本书是一本必读书。

备注：

- 英文版：Robert Mecklenburg, Managing Projects with GNU make, 3rd Edition, O'Reilly, November 2004
- 影印版：《GNU Make 项目管理 第三版》，东南大学出版社，2005 年 7 月
- 中文版：O'Reilly Taiwan 公司 译，《GNU Make 项目管理（第三版完全修订版》，东南大学出版社，2006 年 7 月

8. Learning the bash Shell

作者：Cameron Newham

bash 是自由软件基金会发布的“Bourne Again Shell”的缩写。它是流行的 UNIX Bourne shell 的免费可用替代产品，供全球 Linux 用户选用。《学习 bash》正是 bash 的权威指南。本书对 bash shell 中的要点如关键字绑定、命令行编辑和处理、信号处理等等，都做了详细的介绍，并提供了大量的实例来补充说明。

备注：

- 英文版：Cameron Newham , Learning the bash Shell, 3rd Edition, O'Reilly, March 2005
- 中文版：徐炎 查石祥等，《学习 bash （第二版)》，机械工业出版社，2003 年 1 月

9. Hacking Vim

作者：Kim Schulz

在 Linux 下你最常用的是什么文本编辑器？90％的人会回答：Vi。但是能够真正把 Vi 的潜能发挥出来的人仍是少数，如果你想成为一个 Linux 高手，这本书则是必须要读的书。

备注：

- 英文版：Kim Schulz, Hacking Vim,Packt Publishing,May 2007

10. Linux Application Development

作者：Michael K. Johnson, Erik W. Troan

本书适合所有层次的读者，Linux 世界的顶尖编程高手 Michael K. Johnson, Erik W. Troan 为读者带来的是 Linux 编程的华丽篇章，更难得的是本书已经完全更新至 Linux 2.6 内核、GNU C 2.3、最新的 POSIX 标准，对 Linux 扩展和特征都做了深度的覆盖。为读者最大限度的发挥 Linux 的性能提供了难得的助力。

备注：

- 英文版：Michael K. Johnson, Erik W. Troan ,Linux Application Development, Second Edition, Addison Wesley, November 2004
- 中文版：武延军 郭松柳 译，《Linux 应用程序研发（第 2 版)》，电子工业出版社，2005 年 11 月

11. Understanding the Linux Kernel

作者：Daniel P. Bovet, Marco Cesati

为了彻底理解是什么使得 Linux 能正常运行以及其为何能在各种不同的系统中运行良好，你需要深入研究内核最本质的部分。内核处理 CPU 与外界间的所有交互，并且决定哪些程序将以什么顺序共享处理器时间。它如此有效地管理有限的内存，以至成百上千的进程能高效地共享系统。它熟练地统筹数据传输，这样 CPU 不用为等待速度相对较慢的硬盘而消耗比正常耗时更长的时间。

本书指导你对内核中使用的最重要的数据结构、算法和程序设计诀窍进行一次遍历。通过对表面特性的探究，作者给那些想知道自己机器工作原理的人提供了颇有价值的见解。书中讨论了 Intel 特有的重要性质。相关的代码片段被逐行剖析。然而，本书涵盖的不仅仅是代码的功能，它解释了 Linux 以自己的方式工作的理论基础。本书是关于 Linux 内核的经典和权威著作，是每一个 Linux 平台研发人员的案头常备书，笔者在这里不再妄加评论。

备注：

- 英文版：Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, 3rd Edition, O'Reilly, November 2005
- 影印版：深入理解 Linux 内核（第三版），东南大学出版社，2006 年 3 月
- 中文版：陈莉君 张琼声张宏伟 译，《深入理解 Linux 内核（第三版)》，中国电力

出版社，2007 年 9 月

12. Linux Kernel Development Second Edition

作者: Robert Love

来自 Novell 的 Linux 高手为你讲解 Linux 2.6 内核的研发信息，使读者有机会同时从理论和应用的视角来窥视 Linux 2.6 内核的信息，如算法、系统调用接口、页置换策略、内核同步等。

作者小传：

Robed Love 是开源社区的名人，很早就开始使用 Linux。他活跃于 Linux 内核和 GNOME 两个社区。最近，他受聘于 Novell 公司，作为高级内核工程师在 Ximian 桌面组工作。他的内核项目包括抢占式内核、进程调度程序、内核事件层，VM 增强以及多任务处理性能优化。他创建和维护的另外两个开源项目是 schedutils 和 GNOME 桌面管理器。此外，他还是 Lmux Journal 杂志的特邀编辑。.

备注:

- 英文版：Robert Love, Linux Kernel Development Second Edition, Sams Publishing, January 2005
- 中文版：陈莉君 康华 张波 译，《Linux 内核设计与实现（第 2 版)》，机械工业出版社，2006 年 1 月

13. Linux Device Drivers

作者:Jonathan Corbet, Greg Kroah-Hartman, Alessandro Rubini

顾名思义，本书是讲述如何编写 Linux 设备驱动程序的。面对层出不穷的新硬件产品，必须有人不断编写新的驱动程序以便让这些设备能够在 Linux 下正常工作。从这个意义上讲，讲述驱动程序的编写本身就是一件非常有意义的工作。但本书也涉及到 Linux 内核的工作原理，同时还将讲述如何根据自己的需要和兴趣来定制 Linux 内核，这是关于 Linux 设备驱动的经典著作，笔者不再妄加评论。

备注:

- 英文版：Jonathan Corbet, Greg Kroah-Hartman, Alessandro Rubini, Linux Device Drivers, 3rd Edition,O'Reilly, February 2005
- 中文版：魏永明 骆刚 姜君 译，《LINUX 设备驱动程序（第二版)》，中国电力出版社，2006 年 1 月

## 11.4 设计模式

1. Design Patterns--Elements of Reusable Object-Oriented Software

作者：Erich Gamma: Richard Helm: Ralph Johnson; John Vlissides.

设计可复用的面向对象的软件，你需要掌握设计模式。本书并非专为 C++程序员而写，但它采用了 C++（以及 Smalltalk）作为主要示例语言，C++程序员尤其易于从中受益。四位作者都是国际公认的面向对象软件领域专家，他们将面向对象软件的设计经验作为设计模式详细记录下来。这本书影响是如此深远，以至于四位作者以及本书都被昵称为 GoF（Gang of Four）。本书学院气息浓厚，行文风格严谨简洁，虽然它不如某些讲解模式的书籍易读，但真正要精准地理解设计模式，本书是终极权威。学习设计模式，这本书需要一而再、再而三的咀嚼。顺带一句：请将设计模式化作开拓思维的钥匙，切莫成为封闭思维的枷锁。

好好看看这本经典巨著吧！这可是设计模式的圣经！

作者小传：

Erich Gamma 在苏黎世大学获得计算机科学博士学位，曾供职于瑞士联邦银行、Taligent、OTI 公司。现在是 Eclipse 项目的主要技术负责人之一。

Richard Helm 在墨尔本大学获得计算机科学博士学位，曾在 IBM T．J．Watson 担任研究员。现在 IBM 咨询集团供职。

John Vlissides 在斯坦福大学获得计算机科学博士学位，目前是 IBM T．J．Watson 研究中心的研究员。除本书外，他还是 Addison-Wesley“软件模式”丛书的顾问。

Ralph Johhson 在康奈尔大学获得计算机科学博士学位，目前是伊利诺伊大学教授．在模

式、重构等领域均有很高造诣。

备注：

- 英文版：Erich Gamma: Richard Helm: Ralph Johnson; John Vlissides, Design Patterns--Elements of Reusable Object-Oriented Software, Addison Wesley，January 1995
- 影印版：《设计模式：可复用面向对象软件的基础》，机械工业出版社，2004 年 9 月
- 中文版：李英军 马晓星 蔡敏 刘建中 译，《设计模式：可复用面向对象软件的基础》，机械工业出版社，2004 年 9 月

2. Design Patterns Explained

作者：Alan Shalloway James R.Trott

“庄周梦蝶，孰蝶是我，我是孰蝶，一梦至今，蝶我已难分！”不知那位读者曾这样评价这本书，但从此可以窥见该书奇妙之一斑吧!

作者小传：

Alan Shalloway 美国 Net Objectives 咨询 / 培训公司的创始人、CEO 和资深顾问。他是麻省理工学院的计算机科学硕士，具有 20 多年面向对象咨询和软件研发的经验，并经常受邀在重要的软件研发会议(包括 SDcExpro、JavacOne、OOP 和 OOPSLA)上演讲。

James R. Trott 是位于美国西北太平洋地区一家大型金融机构的资深顾问。他是应用数学科学硕士、MBA 和跨文化研究艺术硕士。在其 20 年的职业生涯中，他一直将面向对象和基于模式的分析技术运用在知识管理、知识工程等方面，是运用认知设计模式与 KADS 方法学的专家。

备注：

- 英文版：Alan Shalloway James R.Trott, Design Patterns Explained: A New Perspective on Object-Oriented Design (2nd Edition) ,Addison-Wesley Professional, October 2004
- 影印版：《设计模式解析（影印版）》，中国电力出版社，2003 年 7 月
- 中文版：徐言声 译，《设计模式解析（第二版）》，人民邮电出版社，2006 年 10 月

3. Head First Design Patterns

作者：Elisabeth Freeman,Eric Freeman,Bert Bates,Kathy Sierra

你想在学习设计模式的过程中，不感觉到昏昏欲睡吗？如果你曾经读过任何一本深入浅出（Head First）系列书籍，你就会知道能够从本书中得到的是：透过丰富的视觉效果让你的大脑充分地运作。本书的编写运用许多最新的研究，包括神经生物学、认知 科学以及学习理论，这使得这本书能够将这些设计模式深深地烙印在你的脑海中，不容易被遗忘。你将会更擅长于解决软件设计中的问题，并能够和你的团队成员用 模式的语言来更好地沟通。本书荣获 2005 年第十五届 Jolt 通用类图书震撼大奖。 本书趋近完美，因为它在提供专业知识的同时，仍然具有相当高的可读性。叙述权威、文笔优美。

备注：

- 英文版：Elisabeth Freeman,Eric Freeman,Bert Bates,Kathy Sierra, Head First Design Patterns, O'Reilly, October 2004
- 影印版：《深入浅出设计模式》， 东南大学出版社，2005 年 11 月
- 中文版：O'Reilly Taiwan 公司 译,《Head First 设计模式》, 中国电力出版社, 2007 年 9 月

4. Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems

作者：Bruce Powel Douglass

创建实时和嵌入式系统时，不应给错误留有余地。最终产品的性质要求系统强大、高效和高可靠。处理器和内存资源的限制使这种挑战更为严峻。老练的研发者依靠设计模式——已被证明、可回应设计挑战的解决方案——来建造有安全保障的实时和嵌入式系统。研发者如欲探求使用这种强有力的技术，那么《实时设计模式》是一本一流的参考书。

作者小传：

Douglass Jensen 是知名度很高的实时计算系统的领头人之一，尤其是动态分布式实时计算系统。他因领导世界上第一个可部署的、分布式实时计算机控制系统产品的研究而获此殊荣。他做军用和民用工业的实时计算已有 30 多年，在硬件、软件、系统研究和技术方面有丰

富的经验，并在卡内基·梅隆大学计算机科学系任教 8 年，现在是 MITRE 公司资源的技术领导，正在为国家战略需要的实时计算系统项目进行研究和技术培训。

备注：

- 英文版：Bruce Powel Douglass, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems, Addison Wesley Professional, September 2002
- 中文版：麦中凡 陶伟 译，《实时设计模式——实时系统的强壮的、可扩展的体系结构》，北京航空航天大学出版社，2004 年 5 月

## 11.5 软件工程

1. Are your lights on?

作者：Donald C. Gause, Gerald M. Wenberg

上网简单的搜索一下，您会发现关于这本书的溢美之词实在是太多，据说它会教会你一种分析问题的全新思路,让你轻轻松松解决问题。走出问题的乌托邦。该书 20 年畅销不衰！1997 年，本书作者温伯格因其在软件领域的杰出贡献，被美国计算机博物馆的计算机名人堂选为首批 5 位成员之一。这个名人堂至今只有 20 名成员哟！

备注：

- 英文版：Donald C. Gause, Gerald M. Weinberg ，Are Your Lights On?: How to Figure Out What the Problem Really Is，Dorset House Publishing Company, Incorporated，March 1990
- 中文版：章柏幸 刘敏 译,《你的灯亮着吗？—发现问题的真正所在》，清华大学出版社,2003 年 9 月

2. Code Reading: The Open Source Perspective

作者：Diomidis Spinellis

阅读代码是程序员的基本技能，同时也是软件研发、维护、演进、审查和重用过程中不可或缺的组成部分。本书首次将阅读代码作为一项独立课题，系统性地加以论述。本书引用的代码均取材于开放源码项目--所有程序员都应该珍视的宝库。本书围绕代码阅读，详细论述了相关的知识与技能。"他山之石、可以攻玉"，通过仔细阅读并学习本书，可以快速地提高读者代码阅读的技能与技巧，进而从现有的优秀代码、算法、构架、设计中汲取营养，提高自身的研发与设计能力。本书荣获美国(2004 年度)第 14 届 Jolt 大奖。

备注：

- 英文版：Diomidis Spinellis, Code Reading: The Open Source Perspective, Addison Wesley, May 2003
- 中文版：赵学良 译，《代码阅读与实践》，清华大学出版社，2004 年 3 月

3. The Pragmatic Programmer From Journeyman to Master

作者：Andrew Hunt, David Thomas

《程序员修炼之道》由一系列独立的部分组成，涵盖的主题从个人责任、职业发展，直以用于使代码保持灵活、并且易于改编和复用的各种架构技术，利用许多富有娱乐性的奇闻轶事、有思想性的例子以及有趣的类比，全面阐释了软件研发的许多不同方面的最佳实践和重大陷阱。无论你是初学者，是有经验的程序员，还是软件项目经理，本书都适合你阅读。

备注：

- 英文版：Andrew Hunt, David Thomas, The Pragmatic Programmer From Journeyman to Master,
- 影印版：《程序员修炼之道——从小工到专家》，中国电力出版社，2003 年 8 月
- 中文版：马维达 译，《程序员修炼之道——从小工到专家》，电子工业出版社，2004 年 3 月

4. The Science of Debugging

作者：MattTelles, Yuan Hsich

本书将调试作为一门专业的学科进行研究和分析，提供的代码实例和总是描述，对调节器试的各个方面进行细致而深入的阐述和讨论。

备注：

- 英文版：MattTelles, Yuan Hsich,The Science of Debugging,Coriolis Group Books1st edition ,May 2001
- 中文版：邓劲生 等译,《程序调试思想与实践》，中国水利水电出版社,2002 年 3 月

5. Refactoring: Improving the Design of Existing Code

作者：Martin Fowler

软件工程领域的超级经典巨著，与另一巨著《设计模式》并称"软工双雄"，全美销量超过 100000 册，亚马逊书店五星书。在本书中，作者 Martin Fowler 充分展示了何处可能需要重构，以及如何将不好的设计改造为良好的设计。

备注：

- 英文版：Martin Fowler,Refactoring: Improving the Design of Existing Code, Addison Wesley, June 1999
- 中文版：侯捷 熊节译，《重构：改善既有代码的设计》，中国电力出版社，2003 年 8 月

6. Mastering the Requirements Process

作者：Suzanne Robertson, James Robertson

本书是为那些希望得到正确需求的人而写的。《掌握需求过程》一书用一个接一个的步骤、一个接一个的模板、一个接一个的例子，向我们展示了一个经过业界检验的需求收集和验证过程。它为精确地发现顾客所需所想提供了技巧和深刻见解。

作者小传：

Suzanne Robertson 与 James Robertson 多年来已帮助了数百家公司改进需求技术，进入系统研发的快车道。他们关于需求、分析和设计的课程和讲座采用了创新的方式，受到了广泛的赞誉。. Robertson 夫妇是 Atlantic Systems Guild 公司的主要成员，该公司是知名的顾问公司，擅长处理复杂系统构建中人员方面的问题。

备注：

- 英文版：Suzanne Robertson, James Robertson, Mastering the Requirements Process Second Edition, Addison Wesley, March 2006
- 中文版：王海鹏 译，《掌握需求过程》，人民邮电出版社，2003 年 2 月

7. The Unified Modeling Language User Guide

作者：Grady Booch, James Rumbaugh, Ivar Jacobson

本书是 UML 方面最权威的一本著作，三位作者是面向对象方法最早的倡导者，是 UML 的创始人。大师的巨著，笔者就不妄加评论了！

作者小传：

Grady Booch、James Rumbaugh 和 Ivar Jacobson 是 UML 的创始人，均为软件工程界的权威，除了著有多部软件工程方面的著作之外，在对象技术发展上也有诸多杰出贡献，其中包括 Booch 方法、对象建模技术（OMT）和 Objectory（OOSE）过程。目前 Booch 和 Rumbaugh 在 IBM 工作，Jacobson 是独立咨询师。

备注：

- 英文版：Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, second Edition, Addison Wesley, May 19, 2005
- 中文版：邵维忠 麻志毅 马浩海 刘辉 等译，《UML用户指南（第 2 版）》，人民邮电出版社，2006 年 6 月

8. Learning UML 2.0

作者：Kim Hamilton, Russell Miles

介绍 UML2.0 写的最详细的一本好书，学习 UML 的人一定要看。笔者自己曾经受益匪浅！但据说其中文版翻译的质量有待商榷，建议读原版。

备注：

- 英文版：Kim Hamilton, Russell Miles, Learning UML 2.0, O'Reilly, April 2006
- 中文版：汪青青 译，《UML2.0 学习指南》，清华大学出版社，2007 年 2 月

9. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development

作者：Craig Larman

本书介绍了“对象思想”，并在实际的面向对象分析和设计中应用了这一思想，即如何以对象进行思考和设计，以及如何创建精致、健壮和可维护的系统。

作者小传：

Craig Larman 是 Valtech 公司首席科学家，专长于 OOA/D 与设计模式、敏捷/迭代方法、统一过程的敏捷途径和UML 建模。这本由闻名于国际软件界的专家和导师所著的经典之作自面世以来就享誉世界。是世界上最广为使用的 UML 建模实例教材。

备注：

- 英文版：Craig Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, Addison Wesley, October 2004
- 影印版：《UML 和模式应用(原书第 3 版)》，机械工业出版社，2006 年 1 月
- 中文版：方梁 译,《UML和模式应用(原书第 2 版)》，机械工业出版社，2005 年 6 月

## 11.6 通用程序设计

1. Memory Management: Algorithms and Implementation in C/C++

作者：Bill Blunden

本书是为数不多的介绍垃圾收集算法和内存管理算法的书籍之一，对于想成为大虾级的读者而言，这是一本难得的好书。

备注：

- 英文版：Bill Blunden , Memory Management: Algorithms and Implementation in C/C++, Wordware Publishing,2003

2. Memory as a Programming Concept in C and C++

作者：Frantisek Franek

本书以 C/C++为工具从操作系统、计算机体系结构、编译器、编程原则等多个层次对内存这个概念进行了深层次的分析，实是一本难得的好书。

备注：

- 英文版：Frantisek Franek,Memory as a Programming Concept in C and C++, Cambridge University Press,2004

3. Compilers: Principles, Techniques, and Tools

作者：Alfrred V.Aho, Ravi Sethi, Jeffrey D.UHman

知道编译器领域大名鼎鼎的龙书吗？据说国内所有的编译原理教材都是抄的它的，而且只是抄了最简单的前端的一些内容。这是和 Steven S.Muchnick 的鲸书 《Advanced Compiler Design and Implementation》和 Andrew W.Appel,with Jens Palsberg 的虎书《Modern Compiler Implementation in Java/C++/ML,Second Edition》齐名的经典巨著。号称编译技术的三本圣经。

作者小传：

Alfrred V.Aho 博士：哥伦比亚大学计算机科学系主管本科生教学的副主任，tEEE Fellow，美国科学与艺术学院及国家工程学院院士，曾获得 IEEE 的冯·诺伊曼奖。他是《编译原理》(Compiler：Principles，Techniques，andTools)的第一作者。他目前的研究方向为量子计算、程式设计语言．编译器和算法等。

Jeffrey D. Ullman 是斯坦福大学的 Stanford W. Ascherman 计算机科学教授。美国国家工程学院院士，1996 年 Sigmod 贡献奖和 1998 年 Karl V. Karstrom 杰出教育家奖获得者。他作为作者或合作者出版了 15 本著作，发表了 170 篇技术论文，其中包括《A First Course in Database Systems》（Prentice Hall 出版社，1997）和《Elements of ML Programming》（Prentice Hall 出版社，1998）。他的研究兴趣包括数据库理论、数据库集成、数据挖掘和利用信息基础设施进行教育。他获得了 Guggenheim Fellowship 等多种奖励。

备注：

- 英文版：Alfrred V.Aho, Ravi Sethi, Jeffrey D.UHman ,Compilers: Principles, Techniques, and Tools, Addison Wesley January 1986
- 影印版：《编译原理 技术与工具》，人民邮电出版社,2002 年 2 月

## 11.7 嵌入式

1. Programming Embedded Systems in C and C++

   作者：Michael Barr

   专为熟悉 C&C++的嵌入式研发初学者和从事嵌入式研发但刚刚转向 C&C++的读者而写的书，值得一读哟！

   备注：

   - 英文版：Michael Barr, Programming Embedded Systems in C and C++, O'Reilly, January 1999
   - 中文版：于志宏 译，《C/C++嵌入式系统编程 》，中国电力出版社，2001 年 3 月

2. Building Embedded Linux Systems

   作者：Karim Yaghmour

   这本书将会为您讲述如何使用 Linux 设计和搭建嵌入式设备上。研发工具、内核、文件系统、Bootloader、网络服务，从事嵌入式 Linux 研发所需要的种种技术，作者都会为您娓娓到来。

   备注：

   - 英文版：Karim Yaghmour, Building Embedded Linux Systems, O'Reilly, April 2003
   - 中文版：O'Reilly Taiwan 公司 译，《构建嵌入式 Linux 系统》，中国电力出版社，2004 年 12 月

3. Mobile Interaction Design

   作者：Matt Jones Gary Marsden

   讲移动终端设计的书本就不多，讲移动终端 GUI 设计的书就更少了，如果你想真正跨入移动终端研发的一流高手行列，那么就请挤点时间看看这本书吧！

   备注：

   - 英文版：Matt Jones Gary Marsden, Mobile Interaction Design,John Wiley and Sons, 2006

4. Mobile Usability: How Nokia Changed the Face of the Mobile Phone

   作者：Christian Lindholm, Turkka

   知道诺基亚手机背后的设计哲学，看看这本书吧！它会告诉你一切！

   备注：

   - 英文版：Christian Lindholm, Turkka, Mobile Usability: How Nokia Changed the Face of the Mobile Phone, McGraw-Hill, 2003

5. Professional Microsoft Smartphone Programming

   作者：Baijian Yang, Pei ZhengandLionel M. Ni

   俗话说“知己知彼，百战不殆”，作为操作系统领域的老冤家，windows 和 Linux 的战场已经开辟到了移动通信领域，Microsoft 员工亲身为您讲述如何在 windows 智能手机平台上编程的大作。不可不读。

   备注：

   - 英文版：Baijian Yang, Pei ZhengandLionel M. Ni, Professional Microsoft Smartphone Programming, Wrox Press , 2007

## 11.8 无线通信

1. Wireless Foresight: Scenarios of the Mobile World in 2015

   作者：Bo Karlson

   2015 年的移动终端是个什么样子？在这个信息技术飞快发展的今天，让任何人都不敢妄加推测，要知道 2000 年的时候怎么会想到手机会飞入寻常百姓家那！你有好奇心吗？想在未来的移动终端领域抢占先机吗？那么看看这本书吧，也许会给你带来启示。

   备注：

   - 英文版：Bo Karlson, Wireless Foresight: Scenarios of the Mobile World in 2015, John Wiley & Sons, 2003

2. UMTS Networks: Architecture, Mobility and Services

作者：Heikki Kaaranen, Ari Ahtianen, Lauri Latinen, Siamak Naghian

知道移动终端领域目前谁是其中的王者吗？诺基亚，这个曾是是以生产橡胶鞋和木材产品为业务的小公司如今已经是移动终端领域绝对的翘楚。现在来自芬兰的专家亲身为您讲述 UMTS 网络的体系结构、移动性和服务，这一切都让本书值得一读。

备注：

- 英文版：Heikki Kaaranen, Ari Ahtianen, Lauri Latinen, Siamak Naghian, UMTS Networks: Architecture, Mobility and Services, John Wiley & Sons, 2005
- 中文版：彭木根等编译，《3G 技术和 UMTS 网络》，中国铁道出版社，2004 年 4 月

3. Mobile Messaging Technologies and Services SMS, EMS and MMS

作者：Gwenael Le Bodic

不知道曾几何时，SMS 已经成为国人的最爱，大拇指的不断按动下，让中移动公司的老总们多么眉开颜笑啊，不过做技术的，更关心它如何实现，如果你有兴趣刨根问底，就看看这本书吧，来自 Vodafone 的架构师会为您讲述一切！

备注：

- 英文版：Gwenael Le Bodic, Mobile Messaging Technologies and Services SMS, EMS and MMS, Second Edition, John Wiley and Sons,2005

## 11.9 计算机网络

1. Computer Networks

作者：Andrew S. Tanenbaum

Tanenbaum 大师的又一部经典巨著,笔者也是看着这本书长大的。泰山北斗的大作，用再多的溢美之辞赞扬都不过分。

备注：

- 英文版：Andrew S. Tanenbaum, Computer Networks, Fourth Edition, Prentice Hall, March 2003
- 中文版：潘爱民 译，《计算机网络（第四版）》，清华大学出版社，2004 年 8 月

2. Internetworking with TCP/IP

作者：Douglas E.Comer

Comer 大师的经典巨著，一共三卷，详细任何从事与网络技术相关工作的研发人员都是很熟悉的。笔者就不再妄加评论了。

备注：

- 英文版：Douglas E.Comer, Internetworking with TCP/IP Vol I: Principles, Protocols,and Architectures ,4th Edition
- 中文版：林瑶 蒋慧 杜蔚轩等译《用 TCP/IP 进行网际互联第一卷：原理、协议与结构（第四版）》，电子工业出版社,2001 年 5 月
- 中文版：张娟等译《用TCP/IP进行网际互联第二卷：设计、实现与内核（第三版）》，电子工业出版社, 2001 年 4 月
- 中文版：赵刚 译，《用 TCP/IP 进行网际互联第三卷：客户-服务器编程与应用（Linux/POSIX 套接字版）》电子工业出版社,2001 年 4 月

## 11.10 其他

1. The Design of Everyday Things

作者：Donald A.Norman

优秀的设计是设计人员与用户之间的一种交流，用户的需求应当贯穿在整个设计过程之中，但这并不是说产品的易用性可以凌驾于其他因素之上，所以伟大的设计，都是在艺术美、可靠性、安全行、易用性、成本和性能之间寻求平衡与和谐。软件设计同样如此，懂得设计是一个优秀的软件研发人员的必备修养。因此笔者强烈为读者推荐本书。本书一定为带

给你不一样的感觉。

备注：

- 英文版：Donald A.Norman,The Design of Everyday Things, Basic Books ,September 2002
- 中文版：梅琼 译,《设计心理学》，中信出版社,2003 年 10 月

2. 资治通鉴（柏杨白话版）

作者：司马光

唐太宗李世民曰:“以铜为镜,可以正衣冠;以古为镜,可以知兴替”，司马光也曰：“鉴于往事,有资于治道”。常读读历史，也许你能学到意想不到的智慧！

备注：

中文版：柏杨 译,《资治通鉴（柏杨白话版）》，北岳文艺出版社，2006 年 7 月

# 第 12 章　参考文献


[1]. Jasmin Blanchette, Mark Summerfield,C++ GUI Programming with Qt 4, Prentice Hall, June 2006
[2]. Alan Ezust, Paul Ezust, An Introduction to Design Patterns in C++ with Qt 4, Prentice Hall, August 2006
[3]. Keir Thomas, Beginning SUSE Linux, Second Edition , Apress,November 2006
[4]. Chris Tyler, Fedora Linux, O'Reilly, October 2006
[5]. Bill McCarty,Learning Debian GNU/Linux 1st Edition,O'Reilly,1999
[6]. Rickford Grant,Ubuntu Linux for Non-geeks,No Statch Press,2006
[7]. Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, 3rd Edition, O'Reilly, November 2005
[8]. Mark G. Sobell, A Practical Guide to Linux Commands, Editors, and Shell Programming, Prentice Hall, July 2005
[9]. Brian Ward ,How Linux Works: What Every Super-User Should Know, No Starch,2004
[10]. Cameron Newham, Learning the bash Shell, 3rd Edition, O'Reilly, March 2005
[11]. Robert Mecklenburg, Managing Projects with GNU make, 3rd Edition, O'Reilly, November 2004
[12]. Richard Stallman,Roland Pesch,Stan Shebs,et al,Debugging with GDB,9th Edition,GNU Press，2002
[13]. Kim Schulz,Hacking Vim,Packat,May 2007
[14]. Debra Cameron, James Elliott, Marc Loy, Learning GNU Emacs, 3rd Edition, O'Reilly, 2004
[15]. Robert Eckstein, Jay Ts, Gerald Carter, Using Samba, 3rd Edition, O'Reilly, January 2007
[16]. Ben Forta, Sams Teach Yourself Regular Expressions in 10 Minutes, Sams Publishing, February 2004
[17]. Stanley B. Lippman, Josée Lajoie, Barbara E. Moo, C++ Primer, Fourth Edition, Addison Wesley, February 2005
[18]. Stanley B. Lippman, Inside the C++ Object Model, Addison Wesley, May 1996
[19]. Scott Meyers, Effective C++ :55 Specific Ways to Improve Your Programs and Designs, 3rd Edition, Addison Wesley, May 2005
[20]. Matthew Wilson, Imperfect C++ Practical Solutions for Real-Life Programming, Addison Wesley,October 2004
[21]. Stephen C. Dewhurst, C++ Gotchas: Avoiding Common Problems in Coding and Design, Addison Wesley, November 2002
[22]. Matt Weisfeld, Object-Oriented Thought Process, The, 2nd Edition, Sams Publishing, December 10, 2003
[23]. Jim Keogh and Mario Giannini , OOP Demystified: A Self-Teaching Guide, McGraw-Hill/Osborne,2004
[24]. Gamma, Helm, Johnson, and Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software,Addison Wesley,1995
[25]. Alan Shalloway, James R. Trott, Design Patterns Explained A New Perspective on Object-Oriented Design ,2nd Edition, Addison Wesley, October 2004
[26]. Elisabeth Freeman,Eric Freeman,Bert Bates,Kathy Sierra, Head First Design Patterns, O'Reilly, October 2004
[27]. Craig Larman,Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, Addison Wesley, October 2004
[28]. Russell Miles,Learning UML 2.0, Kim Hamilton, O'Reilly, April 2006
[29]. Terry Quatrani,Visual Modeling with Rational Rose 2000 and UML, 2nd Edition , Addison Wesley,October 1999
[30]. Suzanne Robertson, James Robertson, Mastering the Requirements Process,2nd Edition, Addison Wesley, March 2006
[31]. Diomidis Spinellis,Code Reading: The Open Source Perspective, Addison Wesley ,May 2003
[32]. Erik T. Ray, Learning XML, 2nd Edition, O'Reilly, September 2003
[33]. Jennifer Vesperman,Essential CVS, O'Reilly, June 2003
[34]. Joe Marasco, The Software Development Edge: Essays on Managing Successful Projects,

Addison Wesley, April 2005
[35]. Matthew B. Doar, Practical Development Environments, O'Reilly, September 2005
[36]. Matt Jones,Gary Marsden,Mobile Interaction Design, Addison Wesley,2006
[37]. Andrew S. Tanenbaum, Modern Operating Systems 2nd Edition, Prentice Hall PTR, February 2001
[38]. Trolltech,Qtopia Open Source Edition 4.2.4 help documentation, Jul.2007
[39]. Steve Holzner, Eclipse.A.Java.Developers.Guide, O'Reilly,2004
[40]. 孙纪坤 张小全 编著,嵌入式 Linux 系统研发技术详解－基于 ARM,人民邮电出版社,2006.8
[41]. Frederick P.Brooks Jr 著,汪颖 译,人月神话,清华大学出版社,2002.11
[42]. Matt Telles,Yuan Hsieh 著,邓劲生 等译,程序调试思想与实践,中国水利水电出版社,2002
[43]. 周爱民 著,大道至简——软件工程实践者的思想, 2005.10
[44]. 梁肇新 编著,编程高手箴言,电子工业出版社,2003.10
[45]. 唐纳德.诺曼 著,梅琼 译,设计心理学,中信出版社,2003.10