中图分类号：　　学校代码：　10055
UDC：　　密级：　公开

南開大學

硕 士 专 业 学 位 论 文

Android手机本地信息搜索系统的设计与实现

Design and Implementation of Local Information Search System on Android

论文作者　袁拓　　指导教师　贾春福　教授

申请学位　工程硕士　　培养单位　计算机与控制工程学院

学科专业　　研究方向

答辩委员会主席　张建忠教授　　评 阅 人　张红光 赵宏 杨敬钰

南开大学研究生院

二〇一三年十一月

摘要

随着科技的进步，智能手机的综合处理能力，存储空间已经有了极大的提高，手机内存储的信息越来越多，如何更好的查找手机中的数据，已经成为一个迫切的需求。

本文设计并实现了一个基于 Android 手机的本地信息搜索系统，用户输入查询关键词，即可快速、准确、方便地查找 Android 手机上的短信息的号码、内容、联系人号码、姓名以及本地文件，并使用简洁易用的界面展现出来，进一步根据搜索的结果可进行下一步的动作。在软件开发过程中本文进行了黑盒测试和性能测试，并进行了进一步的优化，以保证软件的可靠性和良好的用户体验。

本文从 Android 手机系统数据的存储形式，软件的需求分析与设计，界面的设计与实现等几个方面全面阐述了实现本地信息搜索的过程。对需求进行了详尽的分析，使用了设计模式，软件工程的设计思想。在实现技术上，采用了手机上的微型数据库 SQLite 查询，多线程异步搜索模式，以及图像 Cache 等技术。使用了 Android 的动态界面布局方法，使得软件界面可以自适应不同型号，不同分辨率的 Android 手机。

测试和实际使用表明本系统可以较好的完成对短信息、联系人和文件的搜索。

关键词：本地信息搜索 Android 多线程 缓存 SQLite

Abstract

With the advancement of technology, the processor power and storage capacity of smart phones have been greatly improved, the information stored in the phone grows too. How to find the data in the phone more precisely and more efficiently, has become an urgent demand.

In this paper, the author designs and implements an Android-based local text message retrieval system, which could search for the number, content of the message, contact's phone number and name, and the local file on the phone in a quick, precise and convenient way. The retrieval results could be shown on a simple and clear interface, on which the user could choose the next action based on the results. Black Box Testing and performance testing are applied in the development process and the software is further optimized, which ensures the reliability and better user experience of the software.

From the aspects of storage format of Android system data, software requirement analysis and design, and interface design and implementation, etc., the local message retrieval implementation process is demonstrated.

The author does the detailed analysis of demands. With design patterns and software engineering design principles. The queries of micro database SQLite in phone, multi-threaded asynchronous search mode. And image Cache technologies are used in the implementation of this system. With the Android dynamic interface layout, the software interface fits different models, different resolutions of Android phones.

The testing and actual use of the system shows that the query of short message, contacts and file system is well performed.

Key Words: Local Information Searching, Android, Multithread, Cache, SQLite

目录

第一章 绪论

第一节 研究背景及目的

1.1.1 研究背景

当前，随着技术的不断进步，智能化的手持设备，尤其是手机，取得了翻天覆地的变化，智能手机的份额，在近几年中取得了爆炸性的进展。2012 年，全球智能手机用户超过 11 亿，2013 年中旬，中国的智能手机用户已达 3.54 亿，超过美国成为全球第一[1]。手机从以前的打电话，变成了智能终端，拥有更快的处理器，更大的屏幕，更多的存储空间。当前的智能手机系统，是苹果公司的 IOS 系统和 Google 的 Android 系统独占绝大部分份额，市场分析机构 IDC 发布了 2013 年第二季度智能手机研究报告，报告显示，第二季度全球智能手机出货量达到 2.36 亿，其中 Android 智能机占到近 80%的份额；而 iPhone 的份额为 13%。人们的生活越来越离不开手机，使用手机的时间甚至比使用计算机的时间还多。由于手机的处理能力和存储空间的加强，很多信息逐渐的在手机上存储下来。现在智能手机的综合处理能力已经超越了十年前的笔记本计算机，如何更好的管理手机中的数据，已经成为一个迫切的需求。

1.1.2 目的

本文将要描述的手机信息搜索软件（以下简称 AnySearch），其设计目的就是实现友好易用的界面，能够快速，准确，方便的搜索到手机内的信息，特别是针对短信，联系人，手机内各种类型的文件进行精确/模糊的搜索。

由于智能手机有着比以前手机更强的处理能力，更大的存储，手机内的信息随着使用时间会越来越多，4-5 百人的联系人，1-2 千的短信，10000 以上的文件，是很常见的事情。如何能快速搜索这些信息，正是开发 AnySearch 的目的。

Android 是一个开放系统，在 Google 公司推出的 Android 系统的基础上，各个厂商会根据原生的 Android 加以修改、定制，设计出符合自家硬件的 Android 手机。各家厂商推出的 Android 手机，其处理器，屏幕、存储空间等均不相同，屏幕尺寸的不同，各种辅助硬件的不同和五花八门的定制 ROM 给 Android 的开发带来了麻烦。一个应用程序将适配那么多不同的设备，这需要花费很大的精力，这就是著名的 Android 碎片化问题[2]。随着时间的累积，Android 的快速升级导致各个时期的版本出于并存状态，也慢慢变得碎片化。新的版本不断推出，但是旧的版本没有立刻被淘汰，有长期共存的趋势。AnySearch 的设计和实现必须要考虑这些问题。

第二节　　手机信息搜索的现状

Android 是由 Google 公司推出的智能手机系统。Google 公司是以其强大的搜索引擎而闻名于世的，但是针对手机本机的信息搜索，并未有很好的实现，其搜索优势主要是在互联网领域。在 Android 4.1 版本里，Google 才推出了 Google now，扩充了 Google 搜索手机应用程序的功能，Google Now 通过一系列的 Web 服务来进行回答问题、提供建议。除了在线搜索，Google Now 也提供了对于本地信息的全局搜索功能，但这需要有 Android 4.1 以上的版本。对于以前的 Android 手机，就无能为力了。并且，由于某种原因，Google Now 并不会出现在中国的 Android 手机内。

Android 的应用市场里，有着成千上万的应用，但是对于手机本机信息的较全面搜索的软件，寥寥无几，更多的是针对某一部分数据，如特殊文件，如特定应用信息等进行的单一的搜索。

AnySearch 实现了对手机本机上的典型数据，如联系人、短信息、本地文件和已安装应用等，结合各种软件开发技术，实现了快速准确方便的信息搜索。

本项目特点：

(1) 信息搜索：针对用户保存在手机上的短信进行全文搜索，可根据手机号码（部分）、短信内容进行搜索；针对用户的通讯录进行搜索，可根据手机号码（部分）、姓名（部分）进行搜索；针对手机上的各种文件进行名称搜索和类型搜索。

(2) 快速：快速实现搜索，快速显示搜索结果，减少用户等待时间。
准确 / 方便：搜索分为精确、模糊搜索，可以按照用户的不同要求，得到不同的结果。

(3) 人性化的用户界面：主界面简洁明了，操作使用友好直观。默认启用对图片、音乐、视频文件的缩略图。对短信搜索结果显示短信正文的部分内容，对搜索关键字进行高亮。

(4) 灵活的结果处理：针对搜索结果的不同，AnySearch 提供了不同的结果处理方案，如针对图片文件进行查看，针对 APK 文件进行安装，针对短信进行回复等等。

第三节　　功能及技术特色

1.3.1　功能

用户提供搜索关键字，AnySearch 根据用户的输入，自动判断搜索方法，搜索结果以列表形式呈现，并提供对搜索结果的进一步的处理方法。

技术特色：

(1) 智能判断关键字：判断用户输入关键字，选择不同的搜索方法。例如，用户输入的是数字，那么搜索联系人，就可以根据其电话号码进行搜索。

(2) 模糊搜索和精确搜索：按照关键字，可分别使用模糊搜索和精确搜索。其中，模糊搜索：搜索结果只需要包含任意一个关键字。精确搜索：搜索结果必须全部包含所有的关键字。

(3) 多页面显示：短信息、联系人、本地文件的搜索结果按照多页面显示在一屏中，可任意切换，查看相应的搜索结果。

(4) 后台多线程搜索：不必等待搜索完成即可看到部分搜索结果。

(5) 特色搜索：幸运搜，在用户未提供关键字的情况下，点击幸运搜按钮，将进入幸运搜结果页面，应用将按照短信、联系人、特殊类型文件进行分类搜索：

a. 短信息幸运搜：将会按照预定关键字、联系人发送的最近时期短信、非联系人发送的最近时期短信进行搜索。

b. 联系人幸运搜：将会按照特定关键字、最近时期联系最多次数的联系人、联系次数最少的联系人进行搜索。

c. 文件幸运搜：将会按照影音文件 mp3/avi/mpeg/mpg/rm/rmvb/wma/wmv)，图像文件(jpg/jpeg/bmp/png/gif)，文本文件(txt/doc/docx/pdf)，压缩文件(rar/zip)，Android 应用安装文件(apk)等类型进行搜索。

用户可以点按右下角的切换按钮，完成短信息、联系人、本地文件等的幸运搜的功能切换。

1.3.2 技术特色

1.3.2.1 多线程的应用

Android 手机系统中，为了保持良好的用户体验，当一个应用长时间没有反应，或正在进行一项非常耗时的工作，如长时间的搜索，下载一个很大的文件等等无法及时的响应用户的事件，为了避免用户认为该程序已经死掉，Android 系统设置了一个 5 秒的超时时间，一旦用户的事件因为在主线程中阻塞而超过 5 秒钟没有响应时，Android 会弹出一个应用程序没有响应的系统对话框，要求用户终止应用或继续等待。为了避免出现系统的提示对话框，AnySearch 将所有的搜索任务均放在后台进行。主线程负责显示搜索结果的 UI，子线程进行搜索工作，并发送消息到主线程来更新 UI。 AnySearch 使用 Android 系统的 Message Queue(消息队列)，并结合 Handler 和 Looper 组件，来完成主线程和子线程之间的通信。

1.3.2.2 动态页面布局

Android 手机系统，由于存在着多机型，多分辨率的“碎片化”的问题，因此应用不能像通常的应用程序那样在开发时刻，固定每个页面的大小，而必须使用自适应的页面布局技术，使得应用可以在不同的手机屏幕上正确显示。

1.3.2.3 图片 Cache 技术

在文件搜索中，AnySearch 将对搜索结果中的文件，在列表中显示器缩略图。针对不同的文件类型，产生的缩略图的方法是不同的。对于图片的缩略图，

AnySearch 使用了基于 Java 的 Soft Reference 的 Cache 技术, 来维护文件列表中的图片文件的缩略图，避免在上下列表滚动时重复生成缩略图。

第四节　　本文组织结构

在这一小节，将对本文结构做一个概要性质的描述。

首先在第一章中介绍了项目背景，本项目的实际意义与目的。

第二章对本项目采用的具体技术发展现状做了简要的分析，说明所采用的技术的优势与选择理由。

第三章对项目需求分析做了比较详细的描述。给出了经过反复论证的需求内容。

第四章详细描述了基于需求分析做出的整个项目各个部分的设计方案。

第五章描述了基于设计方案的系统功能实现、系统的测试方法以及整个应用系统的部署情况。

第六章对本文做出小结，总结了本系统从分析、设计到实现、测试的开发过程，并分析讨论本系统的现有的不足和对后期的需求和实现上的展望。

第二章 相关关键技术分析

第一节 Android 系统简介

Android 智能手机操作系统，是 Google 公司于 2007 年 11 月 05 日宣布的基于 Linux 平台的开源手机操作系统的名称。该平台由操作系统、中间件、用户界面和应用软件组成。它采用软件堆层（Software Stack，又名软件叠层）的架构，主要分为三部分：底层以 Linux 内核工作为 基础，由 C 语言开发，只提供基本的系统功能，如内存管理，进程管理，IO 管理，网络，驱动模块等；中间层包括函数库 Library 和虚拟机 Dalvik，由 C++开发，Library 提供了大部分的 Java 语言核心库的功能；最上层是各种应用软件，包括系统通话程序，短信程序等，应用软件则由各公司开发者自行开发，以 Java 作为编写程序的主要语言[3]。

由于 Android 系统的开源策略，打破了以往手机操作系统的授权模式，降低了各大厂商的开发成本，并且给与开发者更大的自由发挥的空间，而且 Android 的底层系统是 Linux, 这是 Android 比其他 Mobile 操作系统扩展性更强的重要原因。另外，由于 Linux 的良好的可移植性，使得 Android 可以方便的适应不同的设备，包括手机，平板甚至是家用电器。IDC 公布报告称 2013 年第二季度，Android 系统已经占据智能手机市场份额的 79.3%，在中国的智能手机市场，Android 有望在 2013 年内突破 90%。

第二节 SQLite 数据库

SQLite 是 D.Richard Hipp 在 2005 年实现的一个开源嵌入式数据库引擎[4]。它是由 C 语言编写的。它实现了一个小型的关系型数据库，支持大多数的 SQL92 标准，并且可以在几乎所有主要的操作系统上运行。

不像一般的数据库的客户—服务机制，使用 SQLite 数据库，不需要运行一个独立的数据库管理系统进程。其主要的通信协议是在应用中直接进行 API 的调用，这非常有利于在资源消耗总量，延迟时间，简单性上的提升。其整个数

据库（包括表，索引，数据等）均存放在一个单一的宿主机的文件中，单个数据库大小可达 2TB。

SQLite 由以下几个部分组成：SQL 编译器、内核、后端以及附件。SQLite 通过利用虚拟机和虚拟数据库引擎(VDBE)，使得调试、修改和扩展 SQLite 的内核变得更加方便。所有 SQL 语句都被编译成易读的、可以在 SQLite 虚拟机中执行的程序集。SQLite 的整体结构图[5] 如图 2.1。

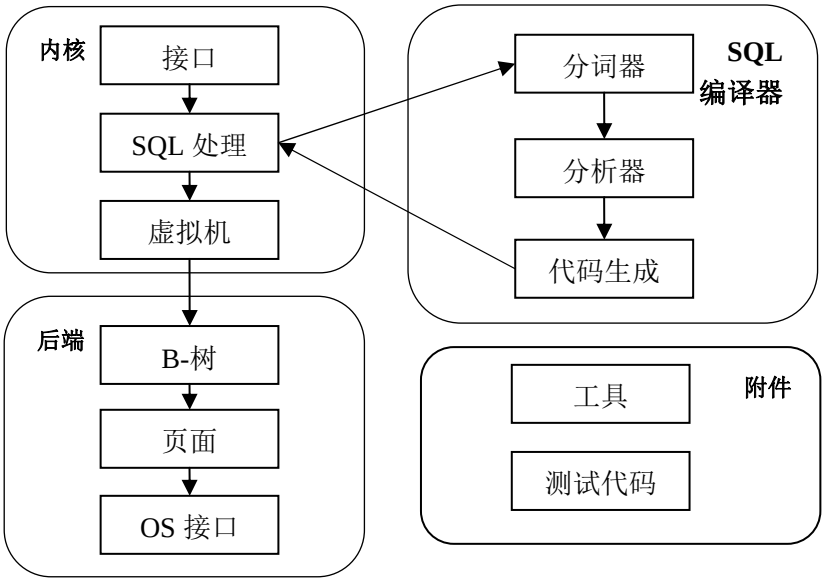


图 2.1 SQLite 结构体系

相对于传统的关系型数据库，SQLite 有着如下的特性：

(1) 支持 ACID 事务。

(2) 无需进行服务器的安装和配置，完全的零配置，非服务式的。

(3) 数据库是储存在单一磁盘文件中的一个完整的文件，没有重新部署的问题。

(4) 数据库文件可以在不同字节顺序的机器间自由共享。

(5) 支持单一数据库大小可达 2TB。

(6) 系统足够小，少于 3 万行 C 代码, 250K，运行速度快。

(7) 字符和 BLOB 数据，仅仅依赖可用内存的限制。

(8) 简单易用的 API。

(9) 独立: 没有额外依赖。

(10)完全的开放源码。

(11)支持多种开发语言，C、 PHP、Perl、Java、ASP.NET、Python 等。

SQLite 也有一些缺陷：

(1) 并发访问的锁机制：SQLite 在并发（包括多进程和多线程）读写方面的性能一直不太理想。数据库可能会被写操作独占，从而导致其它读写操作阻塞或出错。

(2) 某些 SQL92 规范不支持，如外键约束，GRANT 权限管理等。

(3) 由于 SQLite 的数据是以文件形式存放的，在 SQLite 数据库位于 NFS 等网络文件系统之上是，在并发文件读写时可能会出现问题。

(4) 只支持 LEFT OUTER JOIN。

由于 SQLite 的快速、占用资源小、非服务式等特性，它广泛的被主流手机操作系统使用，如在 IOS，Android 智能手机操作系统中[6]，SQLite 常被用作存储一些复杂的数据。Android 在运行时（Run-Time）集成了 SQLite，任何一个 Android 应用程序都可以使用 SQLite 数据库，其数据库文件存放在【/data/data/应用的包名/databases/数据库名】文件中[7]。特别的，其他应用是不可以访问该数据库的。要想访问该数据库，应用必须实现一个 Content Provider，这也是 Android 系统提供给开发者访问系统数据的标准方法。对于熟悉 SQL 的开发者来说，在 Android 应用开发中使用 SQLite 是相当容易的。Android 的主要开发语言是 Java，而 SQLite 也提供了 Java 语言的 API，特别的，Android 还提供了更高级的 SQLite API，来访问 Android 系统的一些特定的数据。

第三节 Content Provider 机制

Android 系统是运行在 Linux 内核上，是一个权限分离的系统。Android 与 Linux 分别有着自己的严格的安全与权限机制[8]。一个 Android 的应用程序，运行在 Android 虚拟机上，它不仅受到 Android 的权限机制的约束，也同时受 Linux 权限的约束。在两者的权限管理下，每个不同的 Application 之间的私有数据和互相访问，是受到严格限制的。如果某个 APP 的私有数据需要公开，则必须使用 MODE_WORLD_READABLE / MODE_WORLD_WRITEABLE 标记创建文件，这也意味着安全性的降低[9]。对于一般应用，是可以这么做。但对于系统

的数据而言，包括短信，联系人，邮件，视频等是不能直接被应用访问的。在 Android 系统中，并不存在一个公共的内存区域，供多个应用共享存储数据。为使得应用可以在不同的应用程序之间（包括与系统之间）共享数据，Android 提出了 Content Provider 机制[10]，通过对底层数据源的抽象，Content Provider 大大减低了应用程序层与数据层间的耦合，使得应用程序不需要关心相应的数据来源的问题。

Android 的 Content Provider 机制为应用的数据存储和读取提供了统一的接口，提供了应用之间的数据共享的方法。在 Android 系统内，已经为一些常见的数据如短信息，联系人，邮件，多媒体等，均提供了相应的 Content Provider。

Content Provider 提供了数据共享的统一的接口，它是通过一个简单的 Uri 机制来进行访问。共享的数据通过可以查询语句获取，同时也支持写入和删除。因此，一个拥有合适权限的应用程序可以添加、删除和更新其它任一应用程序的数据——包括一些系统的 Android 数据库。

许多 Android 系统的数据库对第三方应用程序来说，通过 Content Provider 方式是可见的甚至是可以修改的。也就是说，一旦应用程序被授予了相应的权限，就能访问手机中的联系人、媒体播放器和其它本地的数据库。

除了读取系统的数据库，开发者还可以通过以 Content Provider 方式公布自己的数据，使得这些数据可供你（和其他开发者）在新的应用程序中使用和扩展。

Content Provider 主要有如下几个特点：

(1) Content Provider 使用表的形式来组织数据。无论数据的来源是什么，Content Provider 都会认为它是一种表，然后把数据组织成表格。

(2) Content Provider 提供了统一的接口方法，见表 2.1。

表 2.1 Content Provider 接口

接口	含义
query	查询，调用者通过 Uri 进行查询，返回一个 Cursor。
insert	插入，可以将将一组数据插入到 Uri 指定的地方。
update	更新，可以更新 Uri 指定位置的数据。
delete	删除，删除指定 Uri 并且符合一定条件的数据。

续表 2.1

接口	含义
getType	得到数据类型
onCreate	创建数据时调用的回调函数

(3) 每个 Content Provider 都有一个公共的 Uri，这个 Uri 用于表示这个 Content Provider 所提供的数据源。Uri 主要包含了两部分信息：

a. 需要操作的 ContentProvider。.

b. 对 ContentProvider 中的什么数据进行操作。

一个 Uri 由三部分组成：Scheme，主机名或 Authority 和路径。

(1) scheme：Content Provider 的 scheme 已经由 Android 规定为：content://

(2) 主机名（Authority）：这是用来唯一标识该 Content Provider 的，调用者可以根据这个标识来找到它。

(3) 路径（path）：它可以用来表示要操作的数据。

在 AnySearch 应用中，必须利用 Android 系统提供的 Content Provider 才能进行短信息，联系人的搜索。

第四节 Android 中的多线程技术

Android 是一个多任务，多线程的手机操作系统。在 Android 应用中，Service，Activity，及 Broadcast 均是一到多个线程处理。其中，有一个线程是在程序运行时创建的，是一个进程当中的主线程，这里我们可以理解为 UI 线程，它主要是完成某些处理，相应传感器及用户输入，并更新 UI。对一般应用来讲，在主线程里完成各项功能是可以的，但是在有需要一些耗时操作时，比如在进行大文件读写，数据库操作以及网络下载需要很长时间的时候，在主线程中完成这任务，会导致用户界面的阻塞，前文提到，当应用主线程无反应超过 5 秒钟时，Android 系统会弹出 ANR（Application Not Responding）的响应提示窗口，提示用户关闭进程。为了避免用户界面阻塞，系统应该使用多线程来完成耗时的任务[11]。

Android 中使用线程的问题：

在 Android 平台上，所有的 UI 均未设计成线程安全的，因此不可以在子线程中对主线程的 UI 元素进行状态的更新[12]。如果在子线程中更新了 UI，Android 系统会抛出异常 android.view.ViewRoot$CalledFromWrongThreadException。要避免在子线程中进行 UI 的刷新，必须使用使用某种异步的机制进行 UI 的刷新。以下方式均可完成在子线程中通知 UI 线程更新[13]。

(1) Activity.runOnUiThread(Runnable)

(2) View.post(Runnable)

(3) View.postDelayed(Runnable, long)

(4) Handler

View.post 和 View.postDelayed 方法，只要是继承 View 的组件，都可以使用。这种方式非常适合仅仅是更新 UI 而不涉及到多线程的场合，如 ProgressBar 的进度显示。

Activity.runOnUiThread 方法适合新开线程处理数据，比如 IO 操作和循环。

当子线程和主线程之间需要传递状态，数据以及做相关处理时，就需要使用 Handler + Thread[14]。

AnySearch 就是使用 Handler + Thread 来实现搜索和结果的显示。

Handler+Thread 方法本质上是在一个线程的 run 方法中调用 handler 对象的 postMessage 或 sendMessage 方法来实现的。

其基本使用方法是：

主线程中创建 Handler 对象，在子线程中调用 Handler 的 sendMessage()方法，就会把消息放入主线程的消息队列，并且将会在 Handler 主线程中调用该 handler 的 handleMessage 方法来处理消息，配合主线程进行更新 UI[15]。

这里涉及了 Android 里的几个概念：

(1) Handler：接受子线程发送的数据，并用此数据配合主线程更新 UI。handler 可以分发 Message 对象和 Runnable 对象到主线程中，每个 Handler 实例，都会绑定到创建他的线程中(一般是位于主线程)。

(2) Message Queue：消息队列，存放消息的地方。每一个线程最多只可以拥有一个 Message Queue 数据结构。创建一个线程的时候，并不会自动创建其 Message Queue。通常使用一个 Looper 对象对该线程的 Message Queue 进行管理。主线程创建时，会创建一个默认的 Looper 对象，而 Looper 对象的创建，

将自动创建一个 Message Queue。其他非主线程，不会自动创建 Looper，要需要的时候，通过调用 prepare 函数来实现。

(3) Message：消息对象，Message Queue 中的存放的对象。一个 Message Queue 中包含多个 Message。通过 Bundle 对象可以封装 String、Integer 以及 Blob 二进制对象，用 Handler 对象的 sendMessage 进行消息传递[16]。

(4) Looper：是 Message Queue 的管理者，它帮助 Thread 维护一个消息队列。

除了 Handler＋Thread 之外，Android 在 1.5 之后的版本里提供了 AsyncTask，一个用于实现异步操作的一个类。AsyncTask 是 Android 平台自己的异步工具，融入了 Android 平台的特性，让异步操作更加的安全，方便和实用。实质上它也是对 Java SE 库中 Thread 的一个封装，加上了平台相关的特性。AsyncTask 与主线程交互，仍然是通过 Handler 来进行的。这和使用 thread 没有什么区别。

AsyncTask 通过创建一个进程作用域的线程池来管理要运行的任务，也就就是说当调用了 AsyncTask#execute()后，AsyncTask 会把任务交给线程池，由线程池来管理创建 Thread 和运行 Thread。

AsyncTask 的问题：

线程池的限制： Android2.3 以前的版本，也即在 SDK/API 10 和以前的版本中，内部的线程池的提供的最大线程数限制是 5 个，也就是说同时只能有 5 个线程运行，超过 5 个的线程只能等待，等待前面的某个线程执行完了才被调度和运行，这样会照成多个异步任务不能按预期的设想运行。Android 3.0 开始对 AsyncTask 的 API 做出了一些调整，新增了接口#executeOnExecutor()。

这个接口允许开发者提供自定义的线程池来运行和调度 Thread。如果想让所有的任务都可以同时并发运行，可以通过创建一个没有数量限制的线程池来实现。

第五节 Cache 技术

开发手机上的应用，用户体验是非常重要的，尤其是在操作的响应时间上。手机的硬件资源，相对于同时期的桌面 PC 而言，无论处理器的能力还是内存大小还是远远不如。因此，需要使用相应的技术来降低内存、CPU 的使用。AnySearch 是搜索应用，其搜索结果可能会产生大量的数据。如何在手机上显示大量的数据列表，如何快速相应用户的操作，如大列表的滚动操作，大量图

像文件的缩略图的生成/显示等等，将是 AnySearch 要解决的问题。AnySearch 使用了 Adapter 适配器模式和优化技术，通过使用基于 Java 的 Soft Reference 技术[17]，实现了图像 cache 技术[18]，较好的解决了这个问题。

第六节 汉字首字母转换

在 AnySearch 中，希望输入姓名的首字母，来搜索联系人，比如：输入 ZS 来搜索“张三”，这样我们就需要将联系人的姓名转换为首字母进行匹配。为此 AnySearch 实现了将 UNICODE 字符转换为拼音首字母[19][20]。

第七节 信息在手机中的存储

AnySearch 应用，针对的是用户保留在本机上的数据进行搜索，其中，文件，短信，联系人是应用的处理对象，了解这些数据是如何在 Android 手机中存储，是必须的第一步。

2.7.1 文件对象的存储

由于 Android 和 Linux 的安全模型机制，应用程序(apk)在安装时将会分配一个 userid，当该应用要去访问其他资源比如文件的时候，就需要有正确的 userid 匹配。默认情况下，任何应用创建的文件，数据库，shared preferences 都是私有的（位于/data/data/your_app_project/files/下），其它的程序是无法访问的。所以，一般情况下，应用是无法访问其他应用的数据文件，其他非私有数据如 SDcard 卡上的数据是可以通过 Java API，来访问文件系统的。所以，AnySearch 的文件搜索功能，即是针对对 SDCard 卡上的数据进行搜索。

2.7.2 短信息的存储

在 Android 系统中，sms 短信是存储在/data/data/com.android. providers. telephony/databases/mmssms.db 中[21]，其中有 13 张表，使用 Android 提供的 Content Provider，AnySearch 可以根据不同的 Uri 在不同的表中进行查询[22]，相应的要查询的表见表 2.2。

表 2.2 Android SMS 表

Uri	**表的内容**
content://sms/	所有短信
content://sms/inbox	收件箱
content://sms/sent	已发送
content://sms/draft	草稿
content://sms/outbox	发件箱
content://sms/failed	发送失败
content://sms/queued	待发送列表

在短信息的各个表中，有一些主要字段是通用的。下表 2.3 是一些主要字段的结构和含义。AnySearch 对于短信息的搜索，即是对相关字段的搜索[23]。

表 2.3 Android SMS 表的常用字段

字段名	**字段类型**	**备注**
_id	INTEGER	记录标识
thread_id	INTEGER	对话的序号（conversation）
address	TEXT	发件人地址，手机号
person	INTEGER	发件人，返回一个数字就是联系人列表里的序号，陌生人为 null
date	INTEGER	发送短信的时间
read	INTEGER	是否已经阅读
status	INTEGER	状态（和运营商相关的短信状态）
type	INTEGER	类型 1 是接收到的，2 是发出的
body	TEXT	短信息正文
service_center	TEXT	短信中心号码

2.7.3 联系人的存储

在 Android 系统中联系人的存储也是存放在 SQLite 数据库中，位于 /data/data/com.android.providers.contacts/databases/contacts2.db。

在这个库里，基本的联系人信息放在联系人表里，而其他的详细信息则被储存在独立的数据表中，如图 2.2 所示。

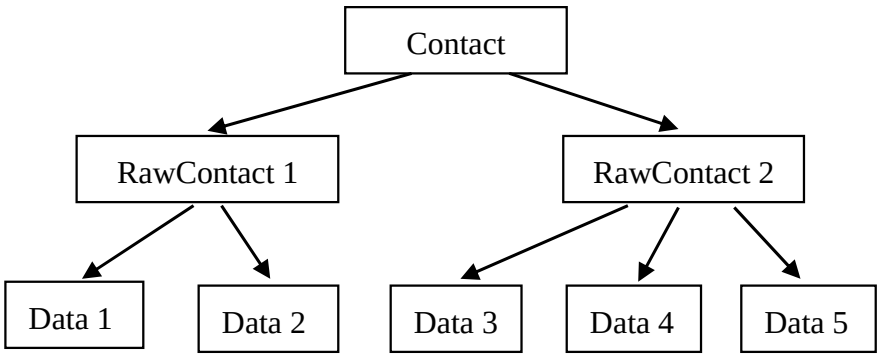


图 2.2 Android 手机联系人存储结构示意图

这里 rawContact 和 data 表构成了主从表的关系。每个联系人在 rawContact 中有一条记录，具体的信息则在 data 表中，这样设计的好处是增加了可扩展性。

Data 表存储了联系人的详细信息，表中的每一行存储一个特定类型的信息，比如地址，名称，Email、电话等。每一行通过一个特殊的 mimetype_id 的字段来表示该行存储的是什么类型的数据，该字段引用了 mimetyps 表，在此表中存储了常用的数据类型。

RawContact 表中的一行存储 Data 表中一些数据行的集合及一些其他的信息，表示一个联系人某一特定帐户的信息，比如 Facebook 的一个联系人。

Contact 表中的一行表示一个联系人，它是 RawContact 表中的一行或多行的数据的组合，这些 RawContact 表中的行表示同一个人的不同的帐户信息。Contact 中的数据由系统组合 RawContact 表中的数据自动生成。

通常在 Android 系统里是不可以直接操作这些表的，Android 提供了一系列的 Uri，可以通过 Content Provider 机制来引用。

这些相关的 URI 是：

ContactsContract.Contacts.CONTENT_URI 基本的联系人表 其字段描述见表 2.4。

表 2.4 Android 基本联系人表字段描述

字段名称	说明
_id	索引 Key,自增长
name_raw_contact_id	当前联系人的 id
photo_id	联系人的图片 id
custom_ringtone	为该联系人自定义铃声
send_to_voicemain	直接将来电转到语音信箱：0 为不设置；1 为设置
times_contacted	通话次数
last_time_contacted	最后通话的日期（以 1970-01-01 00:00:00）计算
starred	加星标的:0 为不加星；1 为加星
in_visible_group	联系人在 UI 中是否可见:0 为不可见；1 为可见
has_phone_number	联系人是否至少有一个电话号码

ContactsContract.Data.CONTENT_URI 通讯录子表。其常用字段如表 2.5 描述。

表 2.5 data 子表的字段描述

字段名称	说明
_id	索引 Key,自增长
mimetype_id	当前行保存数据的类型：1 为邮箱；2 聊天账号；3 住址；4 图片；5 电话号码；6 姓名；7 公司+职位；8 昵称；9 所属组；10 备注；11 网址
raw_contact_id	与 contacts 表中的 name_raw_contact_id 相同，该数据所属联系人
is_primary	0；1；(是否为主要)一直都为 0
is_super_primary	0
data_version	数据版本（更改次数）
data1	基本数据类型

续表 2.5

字段名称	说明
data2,data3	存放 data2 名字，data3 姓氏
data4	存入电话号码；职位；街道；
data5	若是聊天账号该行则有数据
data7	若是住址该行则有数据
data8	若是住址该行则有省数据
data9	若是住址该行则有邮编数据
data10,data11	若是姓名该行则有数据
data15	若是照片该行则有数据

在 Data 表中有一个 mimetype_id 字段，通过这个字段关联 mimetypes 表表示该行代表的信息类型。因为 Data 表中的每一行可以表示如电话号码或地址等不同类型的信息，所以对于不同类型的信息，data1~data15 这 15 列表示不同的含义。为了使用方便，Android 提供了 CommonDataKinds 类，每个类都对应了常量，方便特定项目的查找：

(1) ContactsContract.CommonDataKinds.Phone.CONTENT_URI：对应联系人的手机号码，可以有多个。

(2) ContactsContract.CommonDataKinds.Email.CONTENT_URI：对应电子邮件地址邮箱地址，可以为多个。

第八节 其他技术

设计模式的使用

设计模式（Design Pattern）是由 Erich Gamma 等人在 1990 年代，从建筑设计领域，引入到计算机科学中的。是针对软件设计中普遍存在的各种问题，所提出的解决方案，是一种代码设计经验的总结[24]。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，提高代码的可读性，可靠性[25]。

在 AnySearch 项目中，使用了设计模式中的单例（Singleton）模式[26]和适配器（Adapter）模式[27]，在整个开发过程中，AnySearch 使用了 MVC（模型—视图—控制器）模式来设计整个软件[28]。

单例模式：单例模式是指在整个系统运行期间，单例对象的类必须保证只有一个实例存在，一般用于提供一个统一的管理入口，和全局对象。见图 2.3。

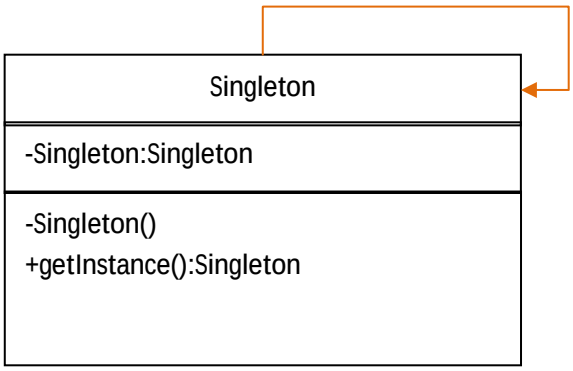


图 2.3　单例模式 UML 图

单例模式的优点：

(1) 由于单例模式在内存中只有一个实例，减少了内存开销。

(2) 单例模式可以避免对资源的多重占用。

(3) 单例模式可以在系统设置全局的访问点，优化和共享资源访问。

(4) 通过 Application 的单例化，还可以保证应用的正常退出。

适配器模式： 其主要功能是将一个类的接口转换为希望的另一个类的接口，适配器模式可以使得原本不兼容的类可以一起协同工作。其原理图见图 2.4。

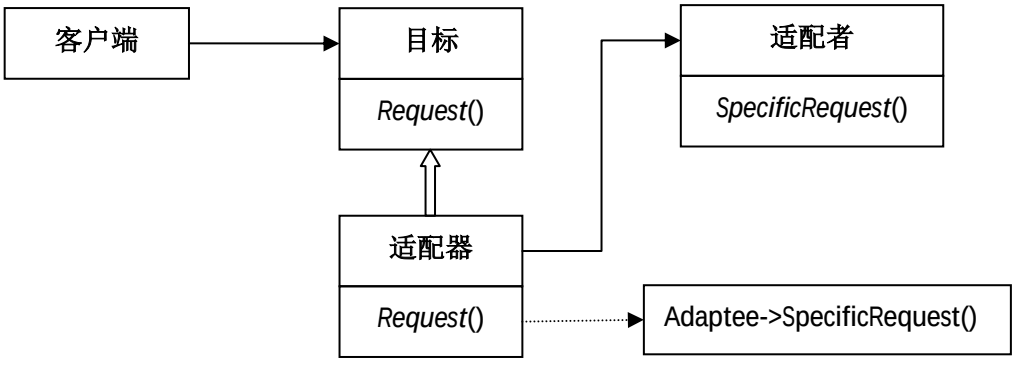


图 2.4　适配器模式 UML 图

其中：

(1) Target 定义 Client 使用的与特定领域相关的接口。

(2) Client 与符合 Target 接口的对象协同。

(3) Adaptee 定义一个已经存在的接口，这个接口需要适配。

(4) Adapter 对 Adaptee 的接口与 Target 接口进行适配

MVC 模式： MVC 架构是"Model-View-Controller"的缩写。MVC 模式是一种复合模式，它将多个设计模式结合起来，以便解决设计问题。它将应用分为三个核心部件：

(1) “数据模型（Model）”：可以封装与应用程序的业务逻辑相关的数据及对数据的处理方法。Model 独立于视图和控制器。

(2) “视图（View）”：能够实现数据有目的的显示。视图通常从模型中得到她需要显示的数据和状态，对于相同的数据，可以有多个不同的显示形式或视图。

(3) “控制器（Controller）”：位于视图和模型之间，负责输入和处理，能够在不同层面间起到组织作用，对应用程序的流程进行控制。

MVC 模式将三个关键部件分离，提高了系统的灵活性和可复用性。独立的模型是可以复用的，开发界面时，程序员只需要考虑界面的布局，改变数据处理，并不会影响模型的数据显示，大大提高了开发效率，提高了代码的可维护性。图 2.5 显示了三者之间的关系。

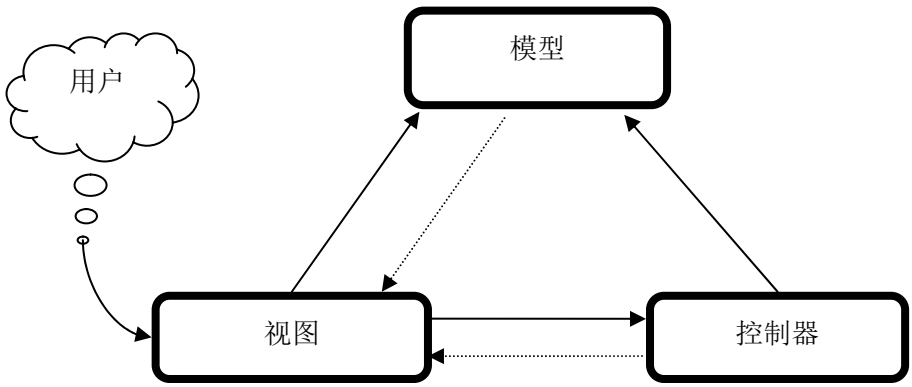


图 2.5 MVC 模式示意图

第三章　　系统需求分析

第一节　　总体功能需求概述

本文的目标是针对 Android 手机上的短信息、联系人、存储卡上的文件进行搜索。用户输入关键字，应用根据关键字，进行上述三项的搜索，并对搜索结果，根据上述三项进行相应的结果展示，并提供进一步的后期处理。

具体过程是：打开应用，用户使用键盘或语音输入关键字（一到多个，以空格分隔）选择精确搜索或模糊搜索模式，进入搜索结果页。在该页面，可查看短消息，联系人，本地文件的搜索结果。采用滚动列表方式，可使用手指进行滑动滚动，对某一搜索结果，使用单指长按（2-4 秒），显示针对该搜索结果的进一步处理的选择菜单，进行后期动作。

该流程的示意图如图 3.1 所示。

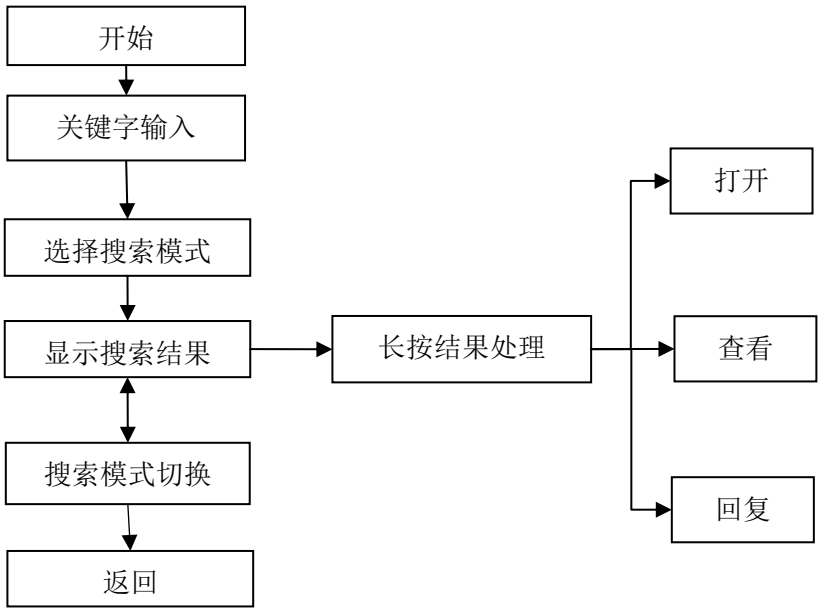


图 3.1　总体流程示意图

基于上述功能需求，下面分别从输入、搜索、结果展示、后续处理等几个方面描述 AnySearch 的详细需求。

第二节 关键词输入的需求

用户的搜索需求是多种多样的，如需要对某一个单词进行输入，对若干单词进行搜索，对电话号码进行搜索，对姓名的首字母进行搜索，甚至仅仅需要搜索出包含其中某一个词的信息即可，为此，本文总结出如下的关键词需求：

(1) 允许用户输入一到多个关键字，用空格分隔。

(2) 允许用户输入数字，文字（包括中英文）。

(3) 允许用户输入姓名首字母进行搜索。

(4) 允许用户选择模糊搜索（即只满足某个关键字）/ 精确搜索（全部关键字均满足）。

(5) 允许用户通过语音输入文字。

第三节 搜索方法的需求

(1) 精确搜索: 搜索结果包含全部关键字。

(2) 模糊搜索： 搜索结果包含任一关键字。

(3) 幸运搜：是特色搜索，无需输入关键字，按照短信（最近时期短信，特定关键字短信，随机短信），联系人（最近时期联系人，特定关键字联系人，随机联系人），特殊类型文件（影音，图像，文本，压缩，安装）进行分类循环搜索。

3.3.1 各部分的搜索需求

系统对短信息的搜索需求：

可以对短信息的手机号码，短信息的正文内容进行搜索。

系统对联系人的搜索需求：

可以对联系人的姓名，手机号码（多个）进行搜索，也可以对联系人的姓名的首字母进行搜索，得到联系人后，显示该联系人所有的电话号码。

系统对本地文件的搜索需求：

可以对存储卡上的文件根据文件名进行搜索。

可以对已安装的应用根据名字搜索。

3.3.2 搜索结果显示需求

搜索结果，要以醒目，直接的形式显示出来。针对短信息、联系人、文件等不同的搜索对象，要有不同的表现形式：

(1) 短信息：显示搜索结果的数目，对搜索结果以列表的方式显示。对搜索结果按照时间排序。显示收发信息的号码，短信息的部分内容，收发时间，如果可能，显示联系人的头像。并高亮关键字。

(2) 联系人：显示联系人搜索结果的数目。对搜索结果以列表的方式显示。显示联系人姓名、号码、头像。并高亮关键字。

(3) 文件：显示文件搜索结果的数目，对搜索结果以列表的方式显示。根据文件种类不同，显示相应的缩略图或图标。显示文件所在的路径，显示文件大小，并高亮关键字。

3.3.3 搜索结果的后续处理

针对不同的搜索结果，本文总结出不同的后续处理方法：

短信息：针对每条的短信息搜索结果，用户长按该条目，可弹出菜单，允许针对该短信息进行如下的操作。

(1) 转发：转发短信息。

(2) 回复：回复短信息。

(3) 拨号：针对短信息的号码进行拨号。

联系人：

(1) 拨号：和该联系人通话。

(2) 短信息：给该联系人发送信息。

(3) 打开联系人：打开联系人表，可进行联系人信息的编辑。

文件：

(1) 查看：查看文件内容。

(2) 跳转：跳转到该文件所在的文件夹。

(3) 编辑：对该文件进行编辑操作。

(4) 打开方式：可对该文件选择打开方式。

(5) 发送：将该文件进行分享，可选择蓝牙，WiFi 等多种方式。

(6) 删除：删除该文件。

已安装应用：可选择打开、运行该应用。

第四节 界面需求说明

关键字的输入，需要提供一个统一的输入界面。在该用户界面里，可以选择是使用键盘输入，或是进行语音输入；可以选择搜索的方法（模糊搜索/精确搜索/幸运搜）；无关键字时提示输入关键字，选择幸运搜时提示无需输入关键字。

搜索结果的展示：

短信息搜索结果、联系人搜索结果、本地文件搜索结果，按照分页的方式，显示在同一屏内，要求可以在不重新输入的情况下，进行模糊搜索和精确搜索的切换。

对于短信息搜索结果、联系人搜索结果、本地文件搜索结果，应当按照各自的需求进行每个结果的显示。并提供在长按该结果时弹出相应的选择菜单，以便后续功能的处理。

第五节 用户体验需求说明

由于手机的硬件功能的限制，在使用过程中要避免长时间的等待，要求搜索结果的即时显示，特别是对文件的内容缩略图的生成，要避免花费太长的时间。上下滚动列表时，文件缩略图应当避免重复生成，避免时间和资源的重复浪费。

其他需求：

(1) 手机屏幕自适应需求：由于 Android 手机的硬件“碎片化”问题，应用需要自动适应不同的手机的分辨率屏幕。

(2) 缩略图的生成：对于图片文件，将生成图片缩略图。APK 文件将提取该文件的应用图标作为缩略图。影音文件则采用固定图标。

(3) 简要的帮助手册。

第四章　　系统设计方案

第一节　　设计原则

系统要以统一的模式来完成各个部分的搜索功能，对多个关键字需要统一的处理方法，对后台的多线程处理，使用统一的消息机制，保证应用各部分的良好的一致性，针对未实现搜索的领域，从设计上要考虑可扩充性，实现搜索工具的可用性和扩展性。

构建一个灵活的功能良好并符合标准的应用，需要着眼全局，综合考虑需求功能与现有技术手段，合理运用成熟的软件工程建模方法，从技术、设计两方面双管齐下保证软件功能最终成功实现。

第二节　　总体框架设计

AnySearch 采用 Eclipse 开发工具[29]，基于 Android SDK，构建了一个多 Activity 的应用系统，采用了多种技术，包括多线程，HtmlString，Cache 技术，各个功能模块之间低耦合，模块内高内聚，确保各个部分的良好的软件工程特性，增强了代码的可移植性和可扩展性[30]。

按照 AnySearch 的功能需求，主要可以分作两个功能模块：关键字输入模块和搜索模块，两大模块通过关键字数据和搜索方法连接起来。 搜索模块又分为短信息搜索，联系人搜索，本地文件搜索。各个搜索模块之间低耦合，整个应用的体系结构如图 4.1 所示。

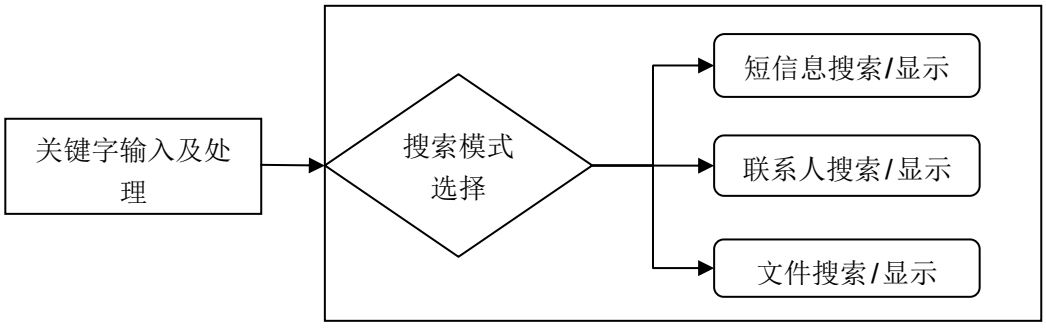


图 4.1　总体框架示意图

根据总体设计原则与功能需求，结合具体搜索方法，将 AnySearch 应用设计为一个基于 Android SDK 之上的符合 MVC 设计模式的体系结构。分别为 UI 表现层、数据层、控制层。 UI 表示层只处理相关数据的显示功能，由 Activity 的主线程完成，数据层负责传递各种数据，包括搜索关键字，搜索模式，搜索结果等。 控制层进行数据的处理和生成，包括关键字的处理，根据搜索模式选择相应的搜索方法，使用多线程进行搜索，并通过 Android 的消息机制将搜索结果提交给 UI 表现层。

为了更好的管理 AnySearch 的各个 Activity 的生存周期，AnySearch 采用了单例模式的 Activity 的管理办法：所有的 Activity 均在唯一的一个 APP（ 从 android.app.Application 派生）的控制之下，每个 Activity 在创建时，均在 APP 上注册，当某个 Activity 推出时，均在 APP 上注销。

并且，在 AnySearch 启动运行后，在 APP 中，立刻开始搜索工作的初始化工作，包括准备文件信息，收集已安装应用等耗时的任务，均在后台以线程方式运行。

第三节 UI 表现层设计

AnySearch 使用 Android 的布局文件，使用 XML 来描述界面[31]。使用 LinearLayout / RelativeLayout 等实现页面布局。使用 ImageView，TextView 等实现各个页面元素的显示。不采用绝对定位，均采用相对定位的方法，使得可以自适应多种不同分辨率、不同大小的屏幕[32]。

根据总体设计原则和功能需求，AnySearch 设计为两个页面，分别是搜索输入页面和搜索结果页面。

4.3.1 搜索输入页面设计

搜索输入页面采用了竖状页面模式，搜索关键字输入框在页面顶端，输入框右侧围语音输入按钮，可以通过该按钮进行语音输入。

在其下方是两种搜索模式的选择：精确搜索按钮/模糊搜索按钮。

中下方是特色功能【幸运搜】，该功能不需要用户输入关键字。

接下来是搜索 ICON 以及版权信息。最后是帮助与退出。

界面设计如图 4.2 所示。

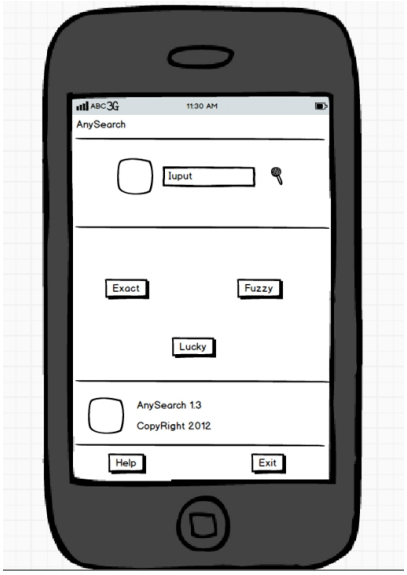


图 4.2　首页面设计示意图

界面的 XML 结构如图 4.3 所示。

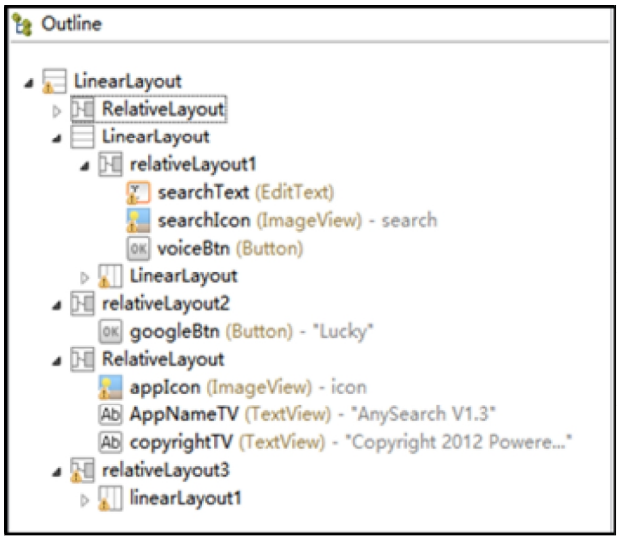


图 4.3　首页面布局结构图

4.3.2 搜索结果页面 UI 设计

为了用户方便查看搜索结果，对于短信息、联系人、本地文件的搜索结果，系统设计为在同一页中显示，采用分页显示的方法。鉴于搜索结果可能会有很多，AnySearch 使用 ListView 列表控件来显示结果，每个 ListView 的表项均为自定义的布局。以下是各部分 UI 的设计：

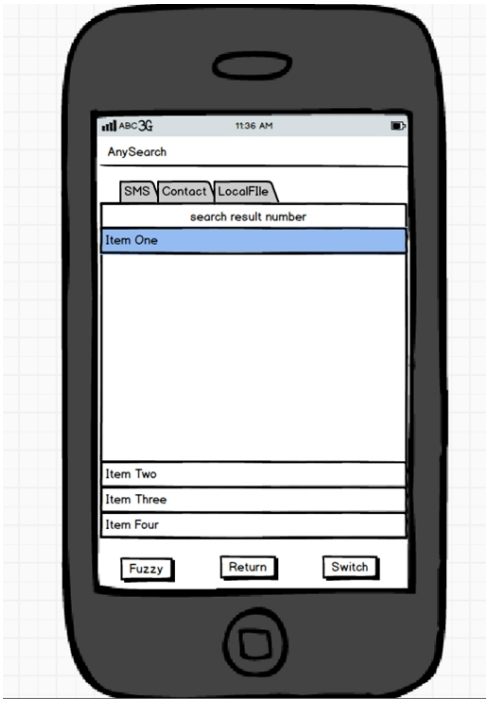
分页栏，分别是短信，联系人，本地文件，点击可切换到相应的搜索结果中，点击某分页面，该分页面标题会高亮。

标题，显示该页面的搜索结果名称以及搜索结果的数目。

接下来是列表，显示各部分搜索结果。这里，随着选择的分页面不同，列表项的内容布局也不同。

最后是三个按钮，分别是精确/模糊切换查询按钮，返回按钮，幸运搜切换按钮。

搜索页面 UI 设计如图 4.4 所示。



图四.4 搜索界面设计示意图

搜索结果页面 XML 布局如图 4.5 所示。

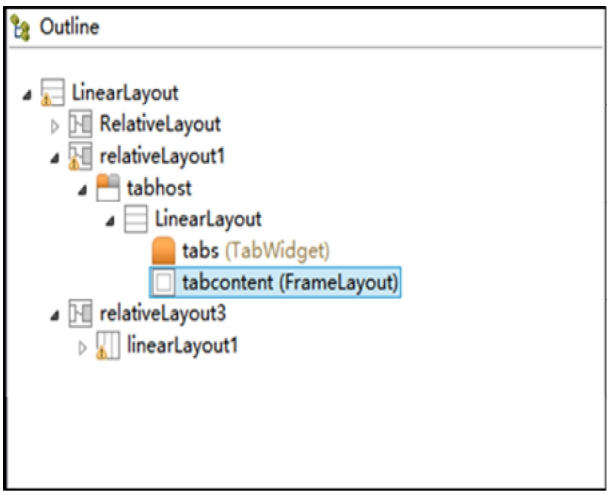


图 4.5　搜索界面布局结构图

短信息搜索结果列表布局，如图 4.6 所示。

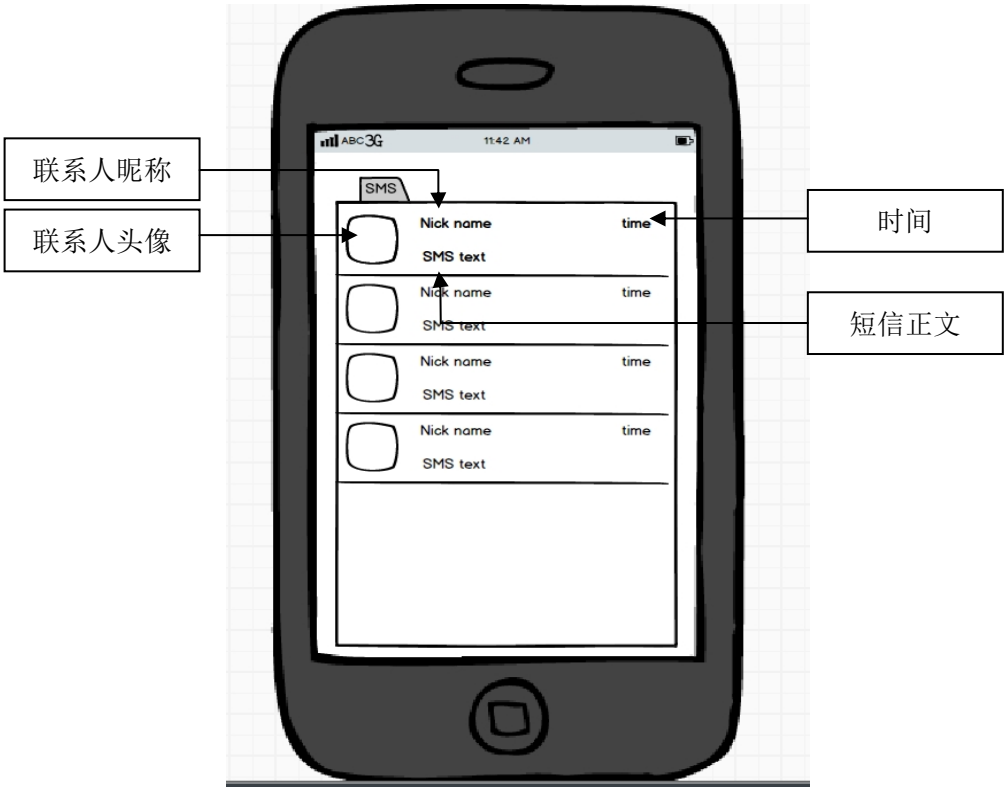


图 4.6　短信息搜索结果界面设计示意图

短信息搜索结果列表的 XML 布局结构，如图 4.7 所示。

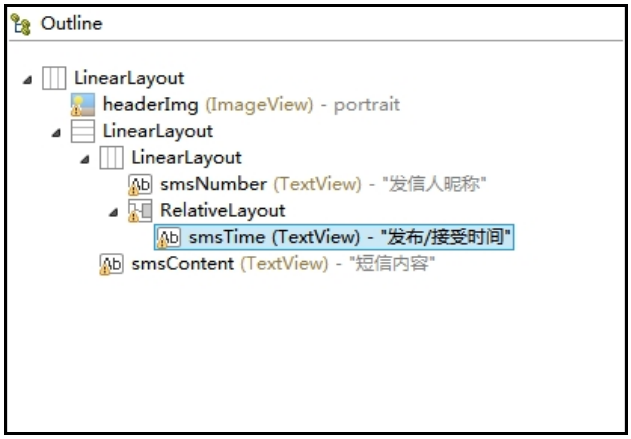


图 4.7　短信息搜索结果界面布局结构图

联系人搜索结果列表布局，如图 4.8 所示。

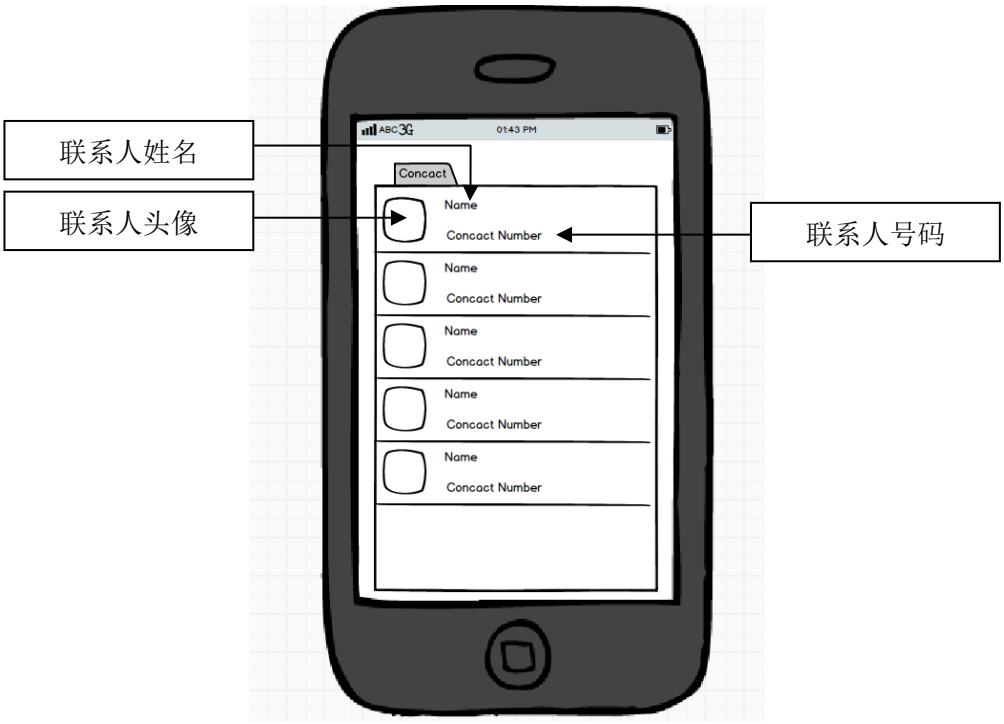


图 4.8　联系人搜索结果设计示意图

联系人搜索结果列表 XML 布局结构，如图 4.9 所示。

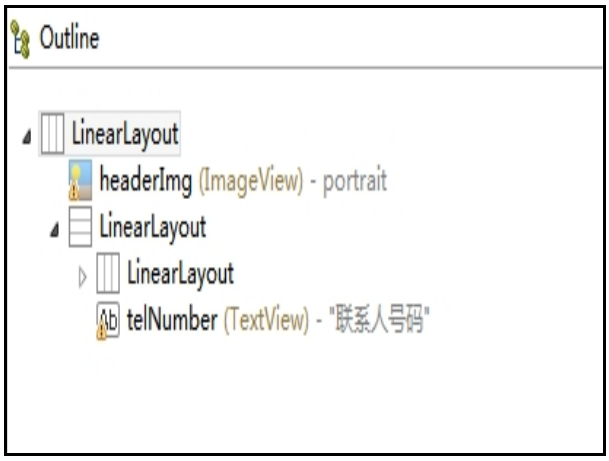


图 4.9　联系人搜索结果布局结构图

本地文件搜索结果列表布局如图 4.10 所示。

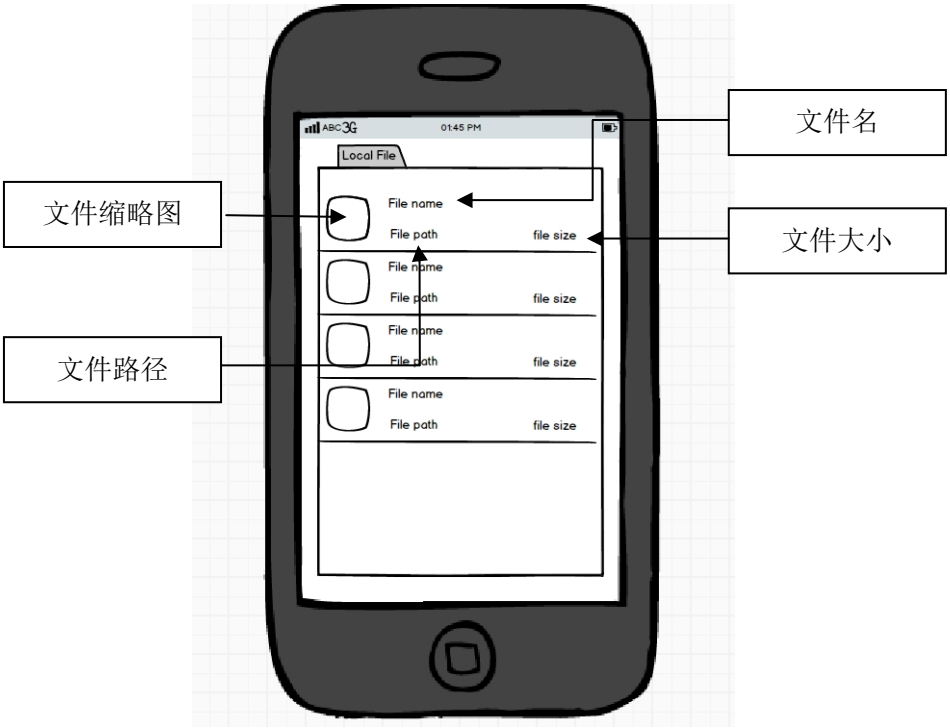


图 4.10　本地文件搜索结果设计示意图

本地文件搜索结果列表 XML 布局结构，如图 4.11 所示。

图 4.11 本地文件搜索结果布局结构图

第四节 系统功能模块设计方案

4.4.1 关键词输入模块设计

在 AnySearch 搜索关键字输入模块部分，用户使用键盘或语音进行关键字的输入，系统经过处理，转给下一个搜索模块进行搜索。

对输入的关键字进行分解：

由于精确搜索和模糊搜索的要求，需要将关键字进行分解，再者，针对短信搜索和联系人搜索，如果关键字均为数字，则需要以手机号码进行搜索，如均不是数字，则只需要在正文，姓名中搜索。

关键字分解流程如图 4.12 所示。

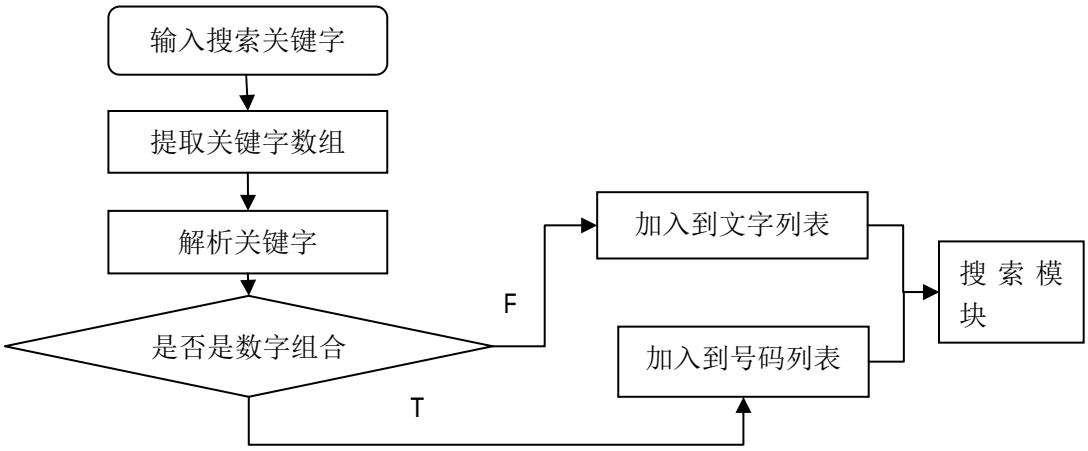


图 4.12 关键词分解流程图

最终生成一个文字关键字数组和一个纯数字号码数组。

4.4.2 短信搜索模块设计

短信搜索分为两个部分：

其一是针对短信号码的搜索。

其二是针对短信正文的搜索。

短信搜索，是通过 Android 系统提供的 ContentProvider 机制，搜索短信息 SQLite 数据库来完成的，分别需要对发件箱，收件箱，草稿箱进行搜索。

本模块的设计思路是：

针对关键字数组和纯号码数组，分别生成相应的 SQL 的查询条件

针对查询方法（精确/模糊），生成相应的 SQL 的查询语句，搜索数据库。转换短信发送/接收时间值为显示的时间格式，生成短信查询结果数据对象，设置相应的关键字为高亮格式，结果数据对象打包，通过消息发往主线程，进行结果显示，图 4.13 所示为搜索收件箱。

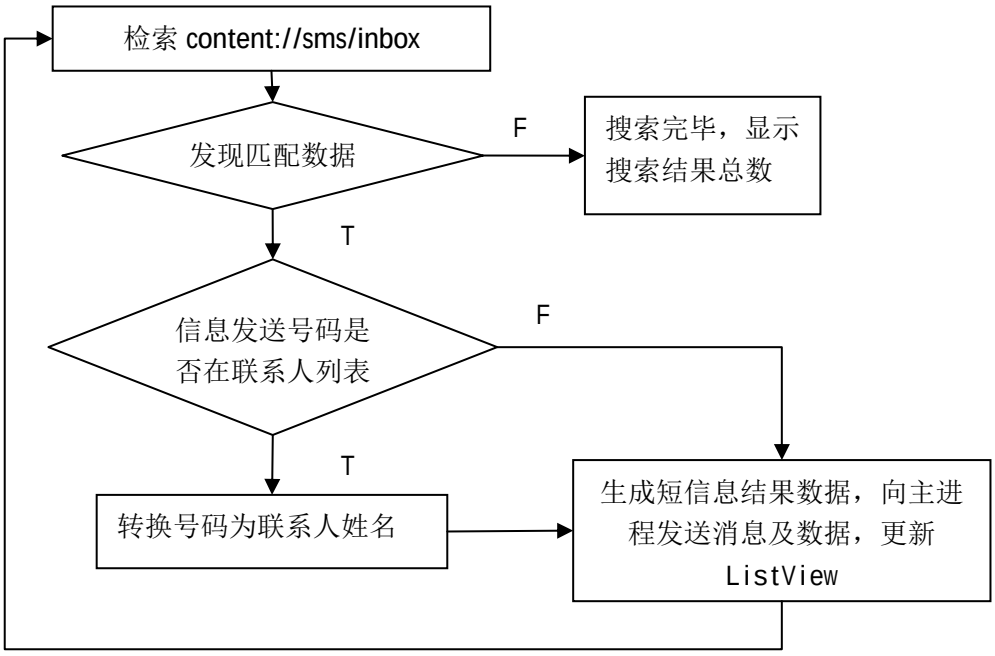


图 4.13 短信息搜索流程图

短信幸运搜流程图，见图 4.14（以搜索最近的联系人短信息为例）。

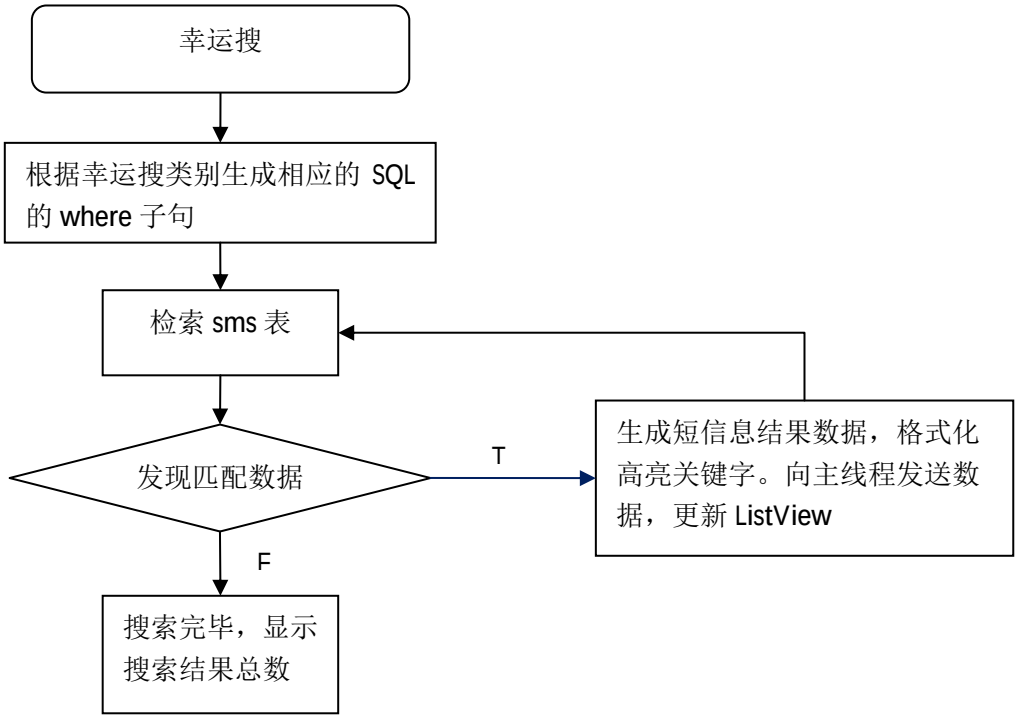


图 4.14　短信息幸运搜流程图

4.4.3　联系人搜索模块设计

联系人的搜索相对比较复杂。有时候一个联系人可能拥有多个电话号码。联系人搜索的需求是在搜索到联系人时，将显示其全部电话号码。因此，联系人搜索将分为若干步骤：通过关键字搜索出所有的联系人，创建一个 hash 表，再遍历 hash 表，根据联系人 ID 搜索，得到在 hash 表内的联系人的所有电话号码。

根据数字关键字搜索，创建以联系人名字为 key 的 hash 表。

根据非数字关键字，在联系人表内搜索匹配的条目。如该联系人不在 hash 表中，就添加到 hash 表内。

根据首字母匹配，得到匹配的联系人姓名，并加入到 hash 表中。

遍历 hash 表，根据联系人姓名，得到联系人 ID，再查询得到该 ID 的所有号码。

其流程图见图 4.15 、图 4.16。

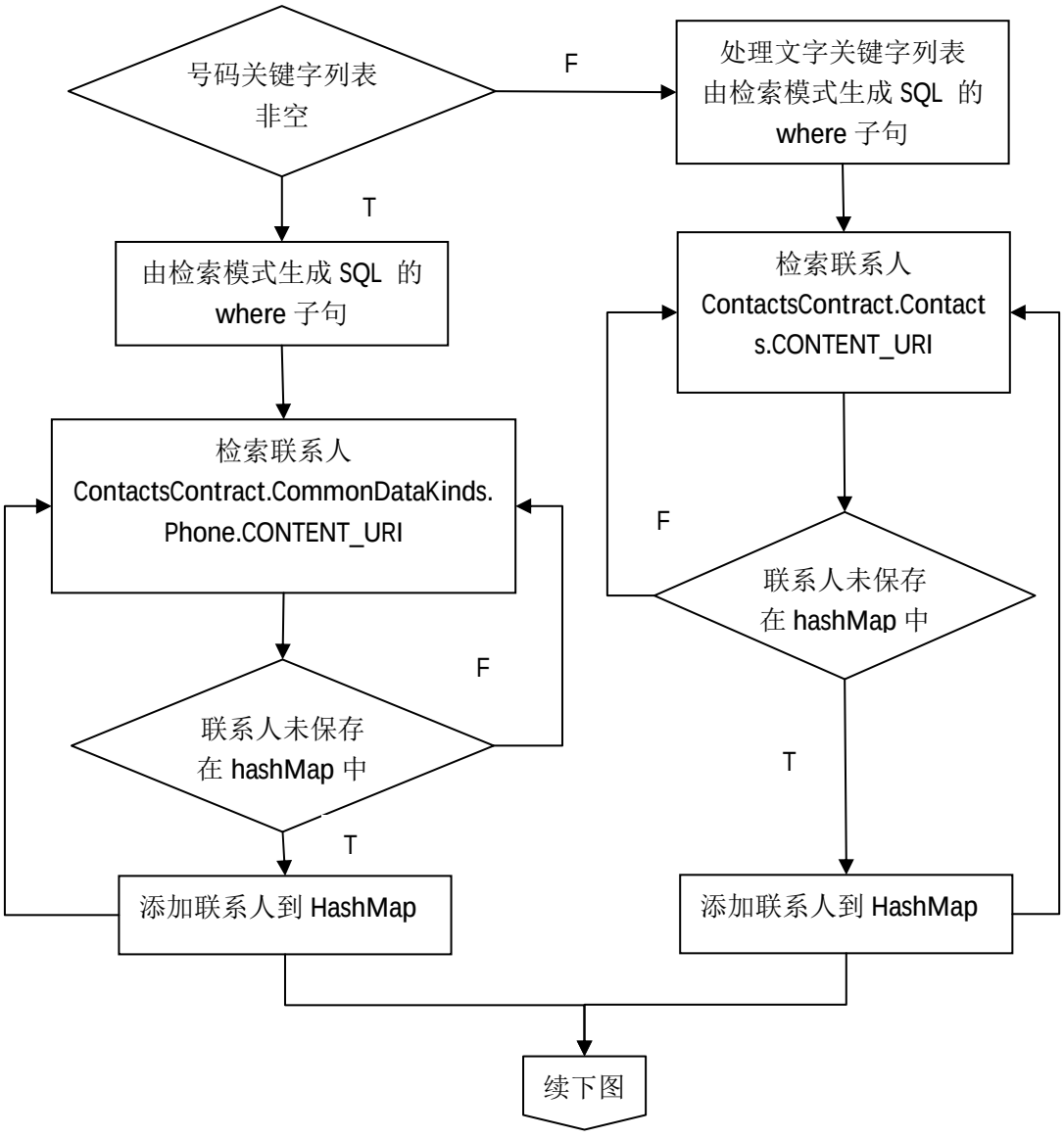


图 4.15　联系人搜索流程图

接图 4.15

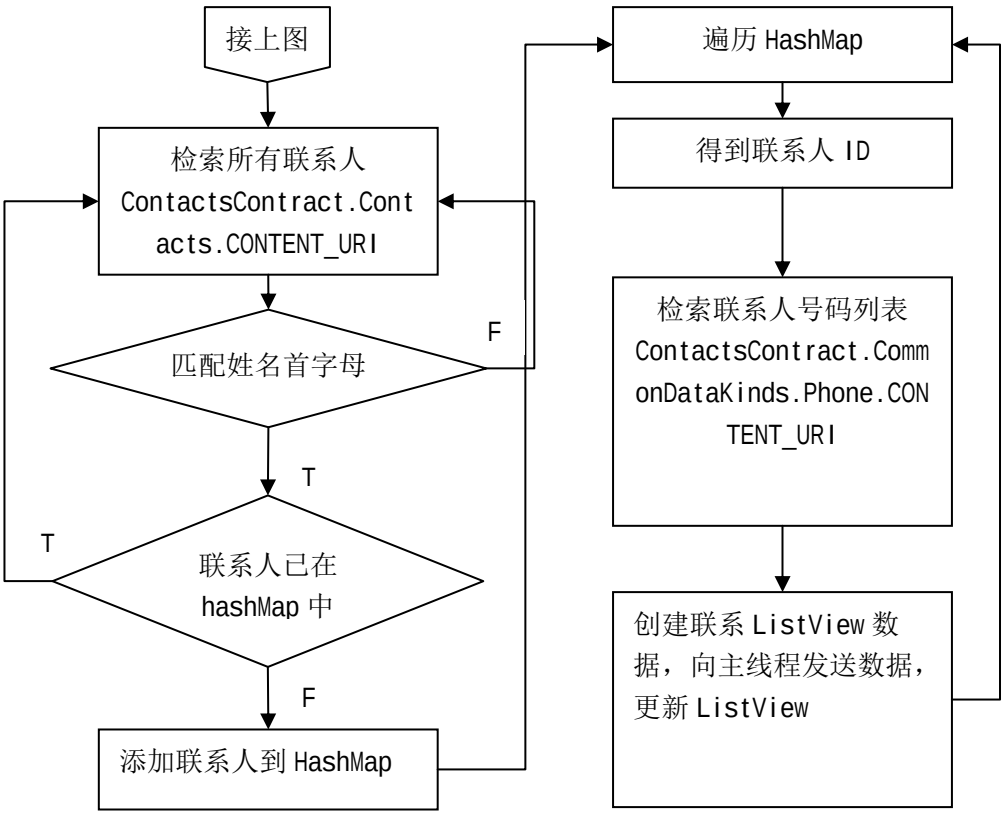


图 4.16 联系人搜索流程图（续）

4.4.4 文件搜索模块设计

在程序开始运行后，App 将创建一个线程，进行文件信息的收集，也就是遍历 SDCard 卡上的所有文件，并保存在列表中。在文件信息收集好后，就可以进行文件名的匹配搜索。见图 4.17 所示。

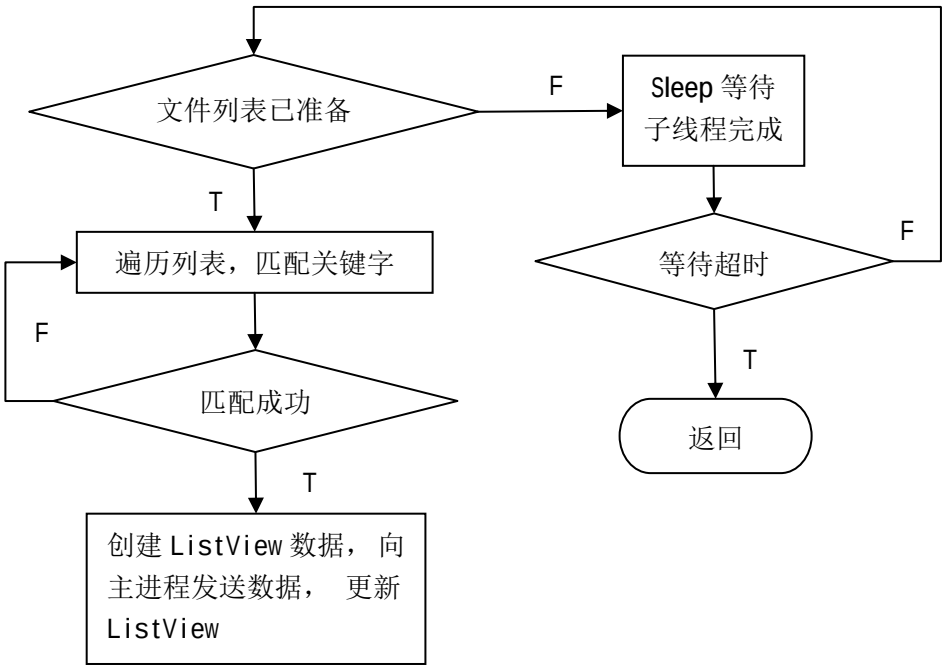


图 4.17 文件搜索流程图

根据扩展名的文件搜索：

根据扩展名的文件搜索，基本上和上述文件搜索算法一样。只是，匹配的仅仅是扩展名部分。速度上比文件名匹配要快。

4.4.5 已经安装的应用程序的搜索设计

在 APP 已经完成已安装应用的信息收集后，即可开始使用关键字匹配应用名称，并发送消息，显示结果。其流程和文件搜索类似。

4.4.6 搜索结果显示技术设计

AnySearch 产生的搜索结果，无论是短信息搜索，联系人搜索，文件搜索，已安装应用搜索，均是一系列的对象的集合，虽然内容，具体表现不同，但是

有同样的显示模式。AnySearch 使用 Adapter 模式，来处理不同类型的列表数据的显示。

Adapter 是设计模式中的一个概念，它是当一个类需要使用另一个类，而接口不同时，对两方的不同接口进行适配，使原本不匹配而无法在一起工作的两个类能够在一起工作。其要达到的目的是，在调用方，采用统一的接口进行调用，而不管被调用者是什么，而被调用方更不会知道自己将会被谁使用，所以无法事先为调用者定制其接口，因此就没有意义去考虑调用方采用什么样的接口，这些接口的适配工作就由 adapter 来完成。

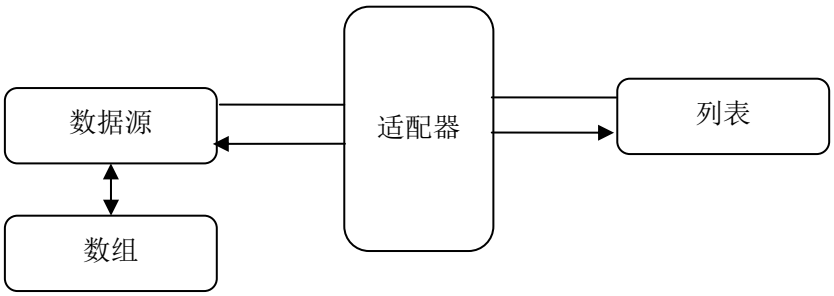


图 4.18 Android Listview Adapter 结构示意图

在 AnySearch 应用中，搜索结果是使用 Andorid SDK 提供的 ListViw 来表示的，Listview 是显示列表的 UI 组件，它只负责管理列表的行为，如列表的上下滑动，列表中某个 Item 的点击等，而每个 Item 显示成什么样的视图(View)，这些不是由 ListView 决定，它只需要知道每个 Item 的 View 是什么，然后将这些 View 罗列出来，并管理这些 View 的滑动效果，点击响应等。

另一方面，ListView 要显示的是一系列的对象，通常是以一个 array 数组或者 list 的形式来组织。比如，短信息的搜索结果 ListView，其每个显示的 item 应该包含有联系人头像 icon，短信息号码，短信息正文，发送/接受时间等。所有的 item 存放在一个 list 中，那么如何将 list 和 ListView 的每个 item 的 View 对应起来呢？这里就要用到 Adapter 设计模式。

Adpater 是连接数据和 ListView 视图的纽带。

其应用具体表现在三个方面：

(1) 当每条数据进入可见区域时，Adapter 的 getView()会被调用，返回代表具体数据的视图。

(2) 用户进行触摸滚动操作，频繁调用时。

(3) 当列表内有成百上千条数据时。

在 Android SDK 里，已经提供了一个很简单的 adapter： BaseAdapter，BAseAdapter 描述了 Adapter 的基本接口并简单的实现，ArrayAdapter 是在 BaseAdapter 上实现了以数组作为数据源的 Adpater。

Anyseach 在 ArrayAdapter 基础上继承并实现了自己的 Adapter。并实现了相应的接口函数：

```
public int getCount()
public T getItem(int position)
public View getView(int position, View convertView, ViewGroup parent)
```

AnySearch 针对短信，联系人，文件，分别实现了如下的 Adapter：见图 4.19，图 4.20，图 4.21。

```
public class SmsListItemAdapter extends ArrayAdapter<SmsItemData> {
    private ArrayList<SmsItemData> items;
    private Context mContext=null;
    private LayoutInflater mInflater;
    private TextView smsNumnerTextView;
    private TextView smsTimeTextView;
    private TextView smsContentTextView;
    public SmsListItemAdapter(Context context, int textViewResourceId, ArrayList<SmsItemData> items)

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

    @Override
    public int getCount() {

    @Override
    public SmsItemData getItem(int position) {

    @Override
    public int getPosition(SmsItemData item) {

    @Override
    public long getItemId(int position) {
}
```

图 4.19 SmsListItemAdapter

```
public class ContactListItemAdapter extends ArrayAdapter<ContactItemData> {
    private ArrayList<ContactItemData> items;
    private Context mContext=null;
    private LayoutInflater mInflater;
    private TextView telNumnerTextView;
    private TextView contactNameTextView;
    public ContactListItemAdapter(Context context, int textViewResourceId, ArrayList<ContactItemData> items)

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

    @Override
    public int getCount() {

    @Override
    public ContactItemData getItem(int position) {

    @Override
    public int getPosition(ContactItemData item) {

    @Override
    public long getItemId(int position) {
}
```

图 4.20 ContactListItemAdapter

```
public class FileListItemAdapter extends ArrayAdapter<FileItemData> {
    private ArrayList<FileItemData> items;
    private Context mContext=null;
    private LayoutInflater mInflater;
    private Handler mHandler;
    private ThumbnailUtil thumbnailUtil;
    public FileListItemAdapter(Context context, int textViewResourceId, ArrayList<FileItemData> items)
    public void setCallback(Handler handler){
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

    static class ViewHolder {


    @Override
    public int getCount() {

    @Override
    public FileItemData getItem(int position) {


    @Override
    public long getItemId(int position) {

}
```

图 4.21 FileListItemAdapter

在 FileListItemAdapter 中，实现了 setCallback(),其功能是创建文件缩略图 cache 机制，使得在显示文件缩略图时可以从 Cache 中得到相对应的缩略图。

Adapter 的优化：

AnySearch 的搜索结果以列表的形式呈现出来。使用 Android 的 ListView 来显示结果，使用派生于 ArrayAdapter 的 Adpater 来连接搜索结果和 ListView 的显示 item。

AnySearch 的搜索结果数目不定，也许有几十条，也许是成千上万条，而手机的内存，处理能力等是有限的，如何在大数据量的结果下能够保持良好的性能，这需要在实现 Adapter 的时候进行优化。

下面以文件搜索为例，说明文件搜索的 ListView 的 Adapter 是如何优化的：

FileListItemAdapter 派生于 ArrayAdapter。

最关键的 getView 接口的普通实现如下图 4.22。

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder=null;;
    View v ;
    v = mInflater.inflate(R.layout.fileitem, null);
    fileNameTextView= (TextView)v.findViewById(R.id.filename);
    filePathTextView= (TextView)v.findViewById(R.id.filepath);
    fileSizeTextView= (TextView)v.findViewById(R.id.filesize);
    fileIcon =(ImageView)v.findViewById(R.id.fileImg);

    FileItemData data=  items.get(position);
    fileNameTextView.setText(data.getNameHighLightByKeyWord());
    filePathTextView.setText(data.getPath());
    holder.fileSizeTextView.setText(data.getFilesize());
    String fname=data.getPath()+"/"+data.getName();

    if(data.getItemFlag() ==0){
        Bitmap bm=thumbnailUtil.get(fname);
        fileIcon.setImageBitmap(bm);
    }
    else if(data.getItemFlag() ==1){
        fileIcon.setImageDrawable (data.getAppIcon());
    }
    return v;
}
```

图 4.22　原始的 getView 接口实现

这种实现能够达到显示文件搜索结果的目的，但是，有很大的弊端：在搜索结果数据小的时候，没有问题，可以工作的很好。在大数据量时，由于每个

列表 Item 均要创建一个 View，这会消耗大量的 CPU 时间和内存。从而造成列表显示/滚动缓慢，甚至有可能出现 Out of Memory 的致命错误，导致程序崩溃。

在程序运行中，用户看到的 ListView 的内容，仅仅是整个搜索结果的一小部分，因此，ListView 并不需要显示所有数据，而只显示需要显示的数据。这样，Adapter 创建的 Item 的 view 数目就很少了，经过测试，更改后的显示速度有着数量级的提高，并且大大降低了内存的使用。图 4.23 是修改后的代码。

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder=null;;
    View v = convertView;
    if (v == null) {
        v = mInflater.inflate(R.layout.fileitem, null);
        fileNameTextView= (TextView)v.findViewById(R.id.filename);
        filePathTextView= (TextView)v.findViewById(R.id.filepath);
        fileSizeTextView= (TextView)v.findViewById(R.id.filesize);
        fileIcon =(ImageView)v.findViewById(R.id.fileImg);
    }
    FileItemData data=  items.get(position);
    fileNameTextView.setText(data.getNameHighLightByKeyWord());
    filePathTextView.setText(data.getPath());
    holder.fileSizeTextView.setText(data.getFilesize());
    String fname=data.getPath()+"/"+data.getName();

    if(data.getItemFlag() ==0){
        Bitmap bm=thumbnailUtil.get(fname);
        fileIcon.setImageBitmap(bm);
    }
    else if(data.getItemFlag() ==1){
        fileIcon.setImageDrawable (data.getAppIcon());
    }
    return v;
}
```

图 4.23 改进后的 getView 接口实现

在这里，convertView 的个数要比可显示的 View 的个数要多一些，这样，在上下滚动的时候，就可以利用多余的 convertView 来显示列表项，而不是每次都创建 convertView，大大减少了 View 的创建次数。convertView 在不用的时候会由系统回收，减少了内存的开销。

每个 ListView 的 item，一般都包含着若干子 view，详见上文所述的列表项的布局。上述代码中，每次创建 Item 的 convertView 的时候均创建了子 view，这页可以进一步优化，使得子 view 的创建次数也减少，如图 4.24 所示。

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder=null;;
    View v = convertView;
    if (v == null) {
        v = mInflater.inflate(R.layout.fileitem, null);
        holder= new  ViewHolder();
        holder.fileNameTextView= (TextView)v.findViewById(R.id.filename);
        holder.filePathTextView= (TextView)v.findViewById(R.id.filepath);
        holder.fileSizeTextView= (TextView)v.findViewById(R.id.filesize);
        holder.fileIcon =(ImageView)v.findViewById(R.id.fileImg);
        v.setTag(holder);
    }
    else
        holder = (ViewHolder)v.getTag();
    FileItemData data=  items.get(position);
    holder.fileNameTextView.setText(data.getNameHighLightByKeyWord());
    holder.filePathTextView.setText(data.getPath());
    holder.fileSizeTextView.setText(data.getFilesize());
    String fname=data.getPath()+"/"+data.getName();

    if(data.getItemFlag() ==0){
        Bitmap bm=thumbnailUtil.get(fname);
        holder.fileIcon.setImageBitmap(bm);
    }
    else if(data.getItemFlag() ==1){
        holder.fileIcon.setImageDrawable (data.getAppIcon());
    }
    return v;
}

static class ViewHolder {
    public TextView fileNameTextView;
    public TextView filePathTextView;
    public ImageView     fileIcon;
    public TextView fileSizeTextView;
}
```

图 4.24　最终的 getView 接口实现

这里使用了静态类 ViewHolder 来存储已经创建的各个子 View，使得子 view 可以重复使用，减少不必要的开销。经过测试，更改后的显示速度可以提高近 50%。

4.4.7 SoftCache 技术

在文件搜索结果列表里，AnySearch 将显示文件的缩略图。针对 txt，audio，video 等特定文件，AnySearch 使用固定的图标。对于图像文件, 如 jpeg，png，bmp 等，AnySearch 将读取文件内容，并作适当的缩放，在列表中显示其图像。如果图像文件很多，在来回滚动时势必会造成同一图像文件的重复读取，为了避免手机有限资源的浪费，AnySearch 针对图像文件的缩略图实现了 SoftCache 技术。

在 SoftCache 中，维护了一个 hash 表，其中 key 是文件名，value 是相应文件的 bitmap。SoftCache 还维护着一个队列，存放文件图像的请求，并使用线程来处理队列中的请求，完成图像文件的读取和请求队列的维护。 当请求被处理并完成后，SoftCache 将向主线程的 handler 发送完成消息，主线程更新相应的列表项。对应的流程图见图 4.25，图 4.26，图 4.27。

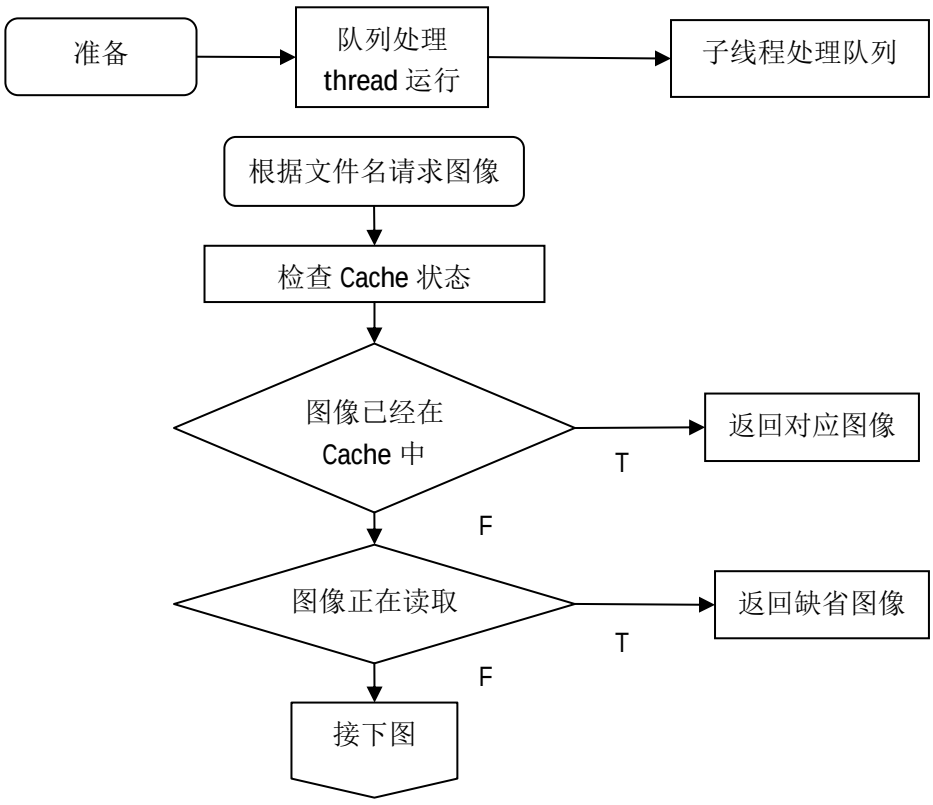


图 4.25 SoftCache 流程图

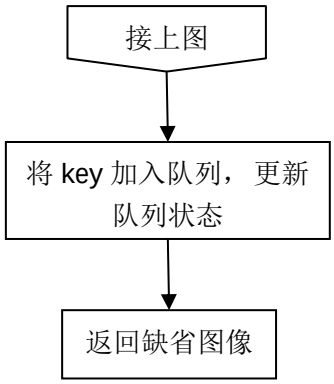


图 4.26 SoftCache 流程图（续）

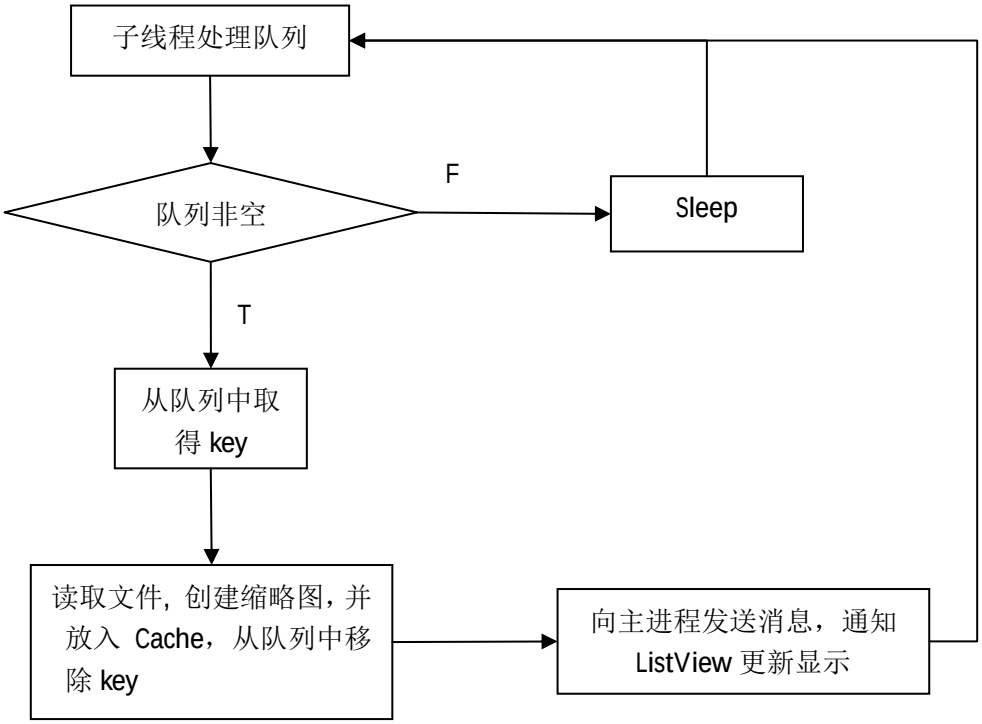


图 4.27 SoftCache 队列处理流程图

4.4.8 单例模式设计

AnySearch 是由多个 Activity 构成的。为了维护每个 activity 的生存周期，并使得每个 Activity 有一个公共的访问入口，AnySearch 使用了单例模式。

Android 使用 Google Dalvik VM，相对于传统 Java VM 而言有着很大的不同，在 Sun 的 Java 体系中入口点和标准 c 语言一样是 main()，而在每个 Android 程序都包含着一个 Application 实例，一个 Application 实例中有多个 Activity、Service、ContentProvider 或 Broadcast Receiver。所以，为了维护每个 Activity，AnySearch 将 Application 单例化，通过使用 private 的构造函数确保了在一个应用中产生一个实例，并且是自行实例化。

```
public class SmsearchApp extends Application {
    private static SmsearchApp instance;
public static SmsearchApp getInstance() {
    if(null == instance)  {
        instance = new SmsearchApp();
    }
    return instance;
}
```

SmsearchAPP 维护着 AnySearch 的各个 Activity 的生存周期，并提供完全的退出机制。

```
public void addActivity(Activity activity) {
    activityList.add(activity);
}
public void removeActivity(Activity activity)  {
    activityList.remove(activity);
}
public void exit()  {
    for(Activity activity:activityList)  {
        activity.finish();
    }
    System.exit(0);
}
```

4.4.9 数据结构设计

关键字数据结构设计：在对用户输入的关键字处理之后，AnySearch 使用两个 StringArray 来存放号码关键字列表和文字关键字列表，之后的信息搜索均是基于这两个列表来实现匹配的。

短信息数据类、联系人数据类、文件数据类：均是存放相应部分的搜索结果。在这几个数据类中，包含了必要的信息。如在 SMS 信息类中，有短信号码，发送/接受日期，短信来源(收件箱，发件箱等)，短信正文。并实现了在号码，短信正文中将搜索关键字高亮的方法。联系人数据类、文件数据类与短信息数据类类似。

图 4.28 是短信息数据对象的 UML 类图。

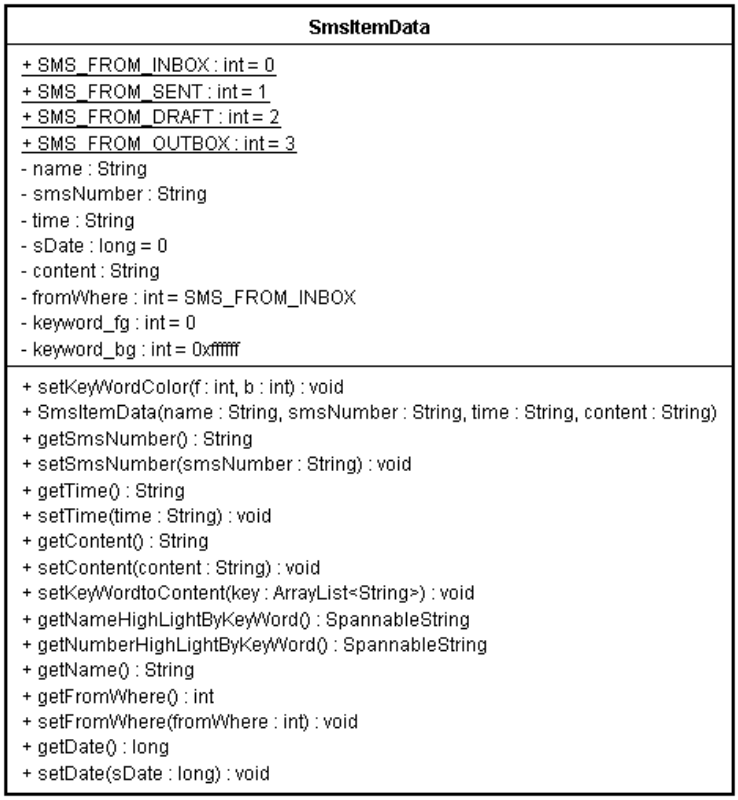


图 4.28 短信息对象 UML 类图

图 4.29 是联系人数据对象的 UML 类图。

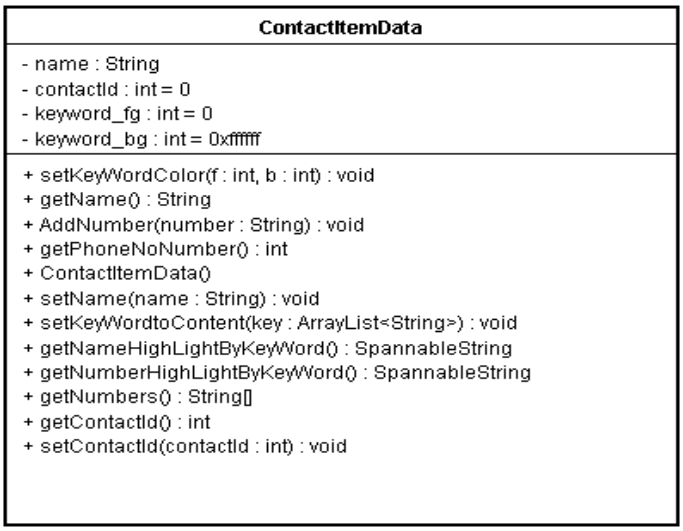


图 4.29　联系人对象 UML 类图

图 4.30 是文件对象的 UML 类图。

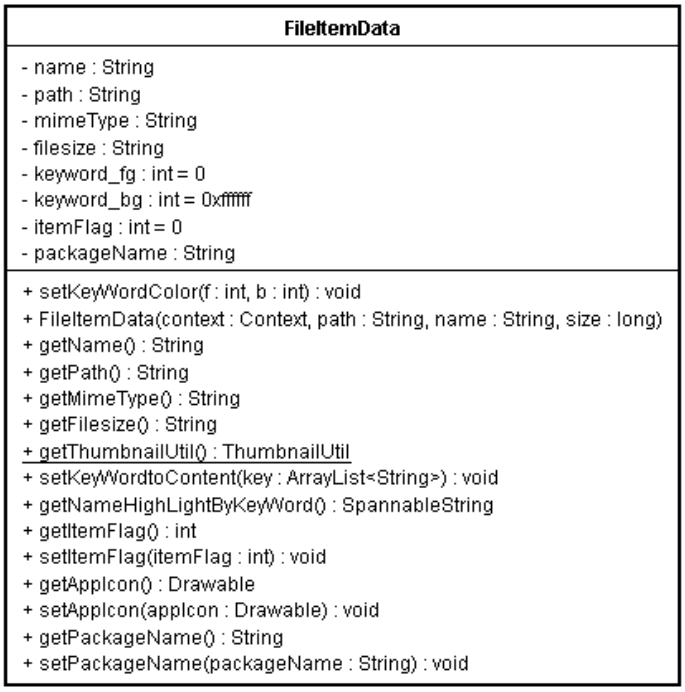


图 4.30　文件对象 UML 类图

第五章　　系统实现与部署

第一节　　系统功能实现综述

基于上一章中对整个 AnySearch 系统的详细需求分析，以及对各个部分的具体设计方案，Anysearch 最终实现了 MVC 设计模式，并基于 MVC 模式书写代码。

本系统的源代码结构如图 5.1 所示：

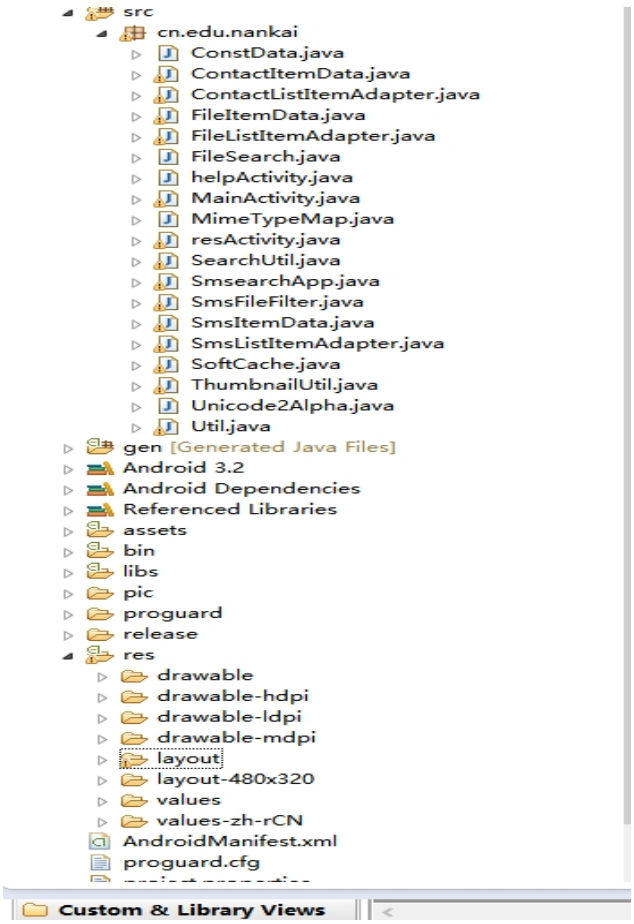


图 5.1　AnySearch 源代码结构图

整个系统使用 Eclipse 集成环境，基于 Android SDK，使用 Java 编程语言开发。在 Anysearch 系统里，使用了多线程技术，Cache 技术，Adapter 技术，单实例技术，实现了易用的用户界面和快速的用户响应。

Eclipse 本身是自带 Java 编程功能的，为了使得可以进行 Android 的应用程序的开发，需安装 ADT（Android Development Toolkit）。ADT 是 Google 公司推出的专门配合 Android SDK 开发的 Eclipse 的插件。到目前为止，Android SDK 提供了众多的版本，最高已经到 API 17，考虑到目前 Android 手机系统的各版本的占有率，AnySearch 选择了 API 10 这个版本，即 Android 2.3.3. 这个版本的 API 适应当前绝大多是的 Android 手机。

在程序开发完成，需要运行测试时，可以使用 Android SDK 提供的 Android 虚拟设备 AVD，AVD 是一个模拟真实世界的 Android 设备的模拟器。对于 AnySearch 来说，由于需要测试短信息，联系人等，这在 AVD 上是难以实现的。因此 AnySearch 使用了真机调试的方法。

第二节　　系统功能实现详解

按照本文前述的需求和设计方案，系统实现了 3 个 Activity。Activity 是 Android 开发中的特有概念，可以简单的理解为“页面”。本系统的 3 个 Activity 分别是：

mainActivity：主页面。实现了关键字的输入和搜索模式的选择。

resActivity：搜索结果页面。这个页面是显示各个部分的搜索结果的。分为 3 个子页面，分别针对短信息，联系人，文件搜索结果，以多 Tab 页面的形式显示出来。在这个页面中，用户可以切换显示不同部分的搜索结果，通过【模糊搜索】【精确搜索】切换搜索模式；可以浏览搜索内容；长按某搜索结果，进行进一步的处理。如果选择的是【幸运搜】模式，用户还可以通过右下方的【切换】按钮，进行各个部分的特色搜索的轮流显示。

helpActivity：一个简单的使用帮助页面，用户可以通过首页的帮助按钮进入。

smsearchApp 是整个系统最初的入口，它派生于 Android 的 Applicatioin 类。在这个类中，实现了应用的单例模式，保证应用有唯一的入口，并提供了各个 Activity 的生存管理。由于 smsearchApp 的唯一性， 一些公共函数也在其中实现。在应用开始运行时，smsearchApp 就创建文件信息搜集线程和已安装应用信息收集线程，为以后的文件搜索和应用搜索做好准备。

在 mainActivity 里，除了接受用户输入的关键字，提供搜索模式的选择外，还提供了语音输入，这是使用 Google 提供的语音输入功能来实现的，其前提是有互联网的可用连接，其语音输入功能可以将用户的语音转换为文字。

本系统采用了 MVC 的模式，针对相应的功能，按照前述的设计方案，均实现了相应的数据类，适配器类和显示布局文件。

例如，在短信息搜索中，实现了短信息数据（SmsItemData），短信息列表显示布局（smsitem.xml），短信息列表显示适配器（SmsListItemAdapter）。通过短信搜索模块生成结果数据（SmsItemData），通过适配器（SmsListItemAdapter）在列表中显示。

系统实现了一个工具类（SearchUtil），在其中，实现了关键字的数字和非数字的分解。根据相应的系统设计，分别实现了各个部分的搜索模块和一些常用的功能模块：

(1) 短信息的搜索功能：根据数字关键字和非数字关键字，分别生成了对应的 SQL 查询语句：例如，根据数字关键字，在 SQL 的 where 子句中，就需要加入对短信息表中的号码字段和正文字段的检索。而根据非数字关键字，则只需在 where 子句中加入对短信息表的正文字段的检索即可。

(2) 联系人的搜索功能：根据本文前述的联系人的搜索流程图来实现，重点是针对联系人的各个表的 SQL 检索语句的生成。同短信息的搜索功能类似，也是需要根据数字关键字和非数字关键字，分别对联系人号码和联系人姓名进行 SQL 检索。其中，需增加对输入关键字可能是姓名首字母的判断匹配。

(3) 文件搜索：根据关键字和搜索模式，完成文件名的匹配。特别的，针对幸运搜，是通过匹配扩展名来实现搜索匹配的。

(4) 日期数字的格式化：在短信息搜索，联系人搜索中，存储在相应的 SQLite 表中的时间信息，均为一个整数，系统需要将其转换为常规的年月日时的表示。在 SearchUtil 中实现了 formatTimeStampString( )方法。

特别的，对于搜索结果的关键字的高亮，是在相应的数据类中实现的，例如在 SmsItemData 类中，实现了 setKeyWordtoContent（）方法，通过 HtmlString 的技术，将结果中的关键字高亮。

在 resActivity 里，实现了 thread-handler 机制，使得搜索工作得以在线程中完成。在主线程中接收搜索结果的消息，实时更新相应的列表项。在文件搜索中，FileListItemAdapter 类里包含了一个 ThumbnailUtil 的实例，这是用来实现文件缩略图的 Cache 机制的。在 FileListItemAdapter 创建后，一个 ThumbnailUtil 实例被创建，由于 ThumbnailUtil 实现了 Cache 接口，因此，当搜索结果需要显示文件缩略图时，可调用 ThumbnailUtil.get()方法。该方法检查 Cache，如果发现了缩略图，即返回。如果没有发现，将请求放入 Cache 的对列中。一旦缩略图生成，Cache 将发 message 到主线程，申请更新相关的列表项。

SoftCache 是系统实现的一个 java 软引用的 Cache 抽象类，在之前的设计方案中已经详细描述了它的流程。它维护了一个可变大小的队列，并使用多线程来处理队列请求。通过消息机制通知请求已经完成。

在联系人搜索中，一个重要的功能既是对姓名首字母的搜索实现，在 Unicode2Alpha 类中，实现了姓名汉字和首字母的匹配检查，并实现了姓名到首字母的转换。

在 Util 类中，还实现了其他一些功能，如转换文件大小为 xxxM，xxxG 等易于阅读的格式；判断文件的 MIME 类型；高亮关键字等。

第三节 系统测试

尽管在上文提及的需求分析，系统设计过程中已经采用各种不同的技术来保证软件的质量，但在开发过程中仍然难免存在问题。因此，对设计完成的软件进行充分的测试是软件开发过程中必不可少的阶段[33]。软件的主要测试过程是根据软件开发各阶段的规则说明和程序内部结构，设计若干测试用例（输入数据和预期输出结果），使用这些测试用例运行系统，根据运行结果判断软件是否存在问题。测试阶段是查找错误阶段，是软件生命周期的重要环节，通过测试可以极大地提高系统的可靠性。

在软件测试阶段，开发者需要使用精心设计的测试用例，以用例数据集合运行于系统，进而发现系统运行时可能存在的问题。测试用例的选择是软件测

试的关键。根据用例方法的不同，软件测试分为白盒测试和黑盒测试[34]。在对AnySearch 系统的测试中，主要采用的是黑盒测试法。黑盒测试也称为功能测试，主要着眼于程序的外部特征和程序界面，而不考虑程序的内部结构。只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。

测试环境：

Huawei D1X Android 手机，Huawei 手机，参数见表 5.1。

表 5.1 华为手机基本配置

主屏分辨率	1280x720 像素
操作系统	Android OS 4.0
核心数	四核
CPU 型号	海思 K3V2 Hi3620
CPU 频率	1433MHz
RAM 容量	1GB

Huawei 手机内部数据情况，见表 5.2。

表 5.2 华为手机数据

SMS 信息	收件箱(537 条) 发件箱(565 条)
Contacts	486 人
SD 卡文件	5230

Sony Ericsson XperiA 手机，参数见表 5.3。

表 5.3 Sony Ericsson 手机基本配置

主屏分辨率	854X480 像素
操作系统	Android OS 2.3
核心数	单核
CPU 型号	高通 snapdragon MSM8255 1.4GHz
CPU 频率	1433MHz
RAM 容量	512MB

手机内部数据情况，见表 5.4。

表 5.4 Sony Ericsson 手机数据

SMS 信息	收件箱(532 条) 发件箱(361 条)
Contacts	223 人
SD 卡文件	3258

测试用例一 见表 5.5。

表 5.5 精确搜索测试用例

名称	精确搜索测试
目的	测试精确搜索结果，
测试方法	黑盒测试
操作步骤	输入关键字 按精确搜索按钮
测试数据	关键字银行 关键字【1380】
预期结果	关键字【银行】短信息搜索结果包含[银行],。联系人无。文件无 关键字【1380】短信息接受/发送者包含 1380，联系人号码包含 1380，文件名包含 1380
实际结果	关键字【银行】短信息搜索结果包含[银行],。联系人无。文件无 关键字【1380】短信息接受/发送者包含 1380，联系人号码包含 1380，文件名包含 1380
测试结论	测试结果正确

测试用例二，见表 5.6。

表 5.6 模糊搜索测试用例

名称	模糊搜索测试
目的	测试模糊搜索结果
测试方法	黑盒测试
操作步骤	输入关键字 按模糊搜索按钮
测试数据	关键字【高校】
预期结果	短信息搜索结果包含【高】或【校】。联系人高姓之人。文件无
实际结果	短信息搜索结果无，联系人无，文件无
测试结论	模糊搜索处理时，将 OR 关系误判为 AND 关系

测试用例三 本用例使用华为手机，来进行 AnySearch 的搜索速度测试。

表 5.7 搜索速度测试用例

名称	速度测试
目的	测试搜索速度
测试方法	黑盒
操作步骤	代码中使用 System.currentTimeMillis();来记录搜索开始和完全结束的时间。取其差为测试使用时间 1) 输入【138】，搜索短信，联系人 2) 输入【png】精确搜索图像文件 3) 使用幸运搜搜索文本文件，zip 文件，apk 文件，影音文件
结果	1) 短信搜索，联系人搜索，用时 0ms 2) 输入【png】，精确搜索，结果 290 个，用时 992ms 3) 幸运搜文本文件 66 个，用时 552ms Zip 文件 4 个，用时 137ms Apk 程序安装文件 47 个，用时 346ms 影音文件 336 个，用时 1066ms

这里列出的测试用例仅仅是众多测试中的几种，只是对 AnySearch 系统的测试方法和过程的说明。

第四节　　应用部署与安装

在程序编译完成后，AnySearch 将生成 Android 系统的 APK 文件。如果使用 Google Play 发布， 还需要对 apk 文件加上数字签名。安装 AnySearch 到 Android 手机有如下几种方法：

(1) 直接得到 apk 文件，copy 到手机上再安装。

(2) 通过豌豆荚类似的管理软件，安装 apk 到手机。

(3) 访问 Google Play 市场，从市场上下载安装。

下面是安装 AnySearch 应用后，程序运行时的实际截屏。

系统启动运行界面，如图 5.2。

图 5.2　启动页面

输入 【银行】 短信息的模糊搜索和精确搜索结果，见图 5.3。

图 5.3 短信息搜索结果

幸运搜中的影音幸运搜和安装文件幸运搜，见图 5.4。

图 5.4 幸运搜结果

搜索结果的进一步处理，见图 5.5、图 5.6。

图 5.5 短信息和联系人的结果的处理

图 5.6 文件打开的不同方式

第六章　结论以及总结展望

第一节　结论

本文对于如何在 Android 手机上实现快速、准确的信息搜索在各个方面做了详细的阐述。 从需求到设计，从设计到实现都遵循了良好的软件工程的思想。最大限度的保证了系统的可扩展性和可靠性。虽然在信息搜索的内容上还有待于添加，但在设计结构上已经为此做好了准备，以便在后期（添加要搜索的内容）的顺利进行。

本文对 AnySearch，Android 手机信息搜索系统，进行了全面的描述。覆盖了从需求分析，系统设计，到软件实现，功能测试等各个方面。本文基于 Android SDK 开发，充分利用系统的 SQLite 来获得相应信息，精心设计了用户界面，较好的展示了信息搜索结果。整个系统使用了设计模式中的多个模式，MVC 模式体现了数据，表现和控制的完美分离，为将来可能的功能扩充，提供了结构上的支持。基于面向对象的设计与实现，保证了系统的可扩充性。多线程技术和 Cache 技术，保证了搜索功能的快速实现。对手机硬件资源的合理应用，为用户的良好体验奠定了坚实的基础。多种搜索模式的选择和切换，方便了用户及时快速的找到所需要的信息，并提供了对搜索结果的进一步处理的途径，对广大 Android 手机用户，有着普遍实用的意义。

第二节　存在的问题与未来的展望

随着计算机技术的飞速发展，手机上的信息越来越多，信息种类也是五花八门。如何快速准确方便的搜索信息，越来越成为一个迫切的需求，虽然本文已经较好的实现了信息，联系人，本地文件等内容的搜索，并对系统的可扩展性做了前期的考虑，但是面对种类繁多，日新月异的信息，仍然显得不足。

后期的工作，本文认为，可以从以下几个方面进行：

(1) 彩信息(MMS)的搜索：本文针对 SMS 短消息进行了搜索实现，但是对于另一种信息－彩信，考虑的不多，短信息和彩信，是完全不同的。短信是 7

号信令网为载体，传输的文字信息。信息内容的大小限制在 70 个字（140 个字节）左右。短信主要以手机端对端的发送和接收，以及手机与互联网之间的发送和接收为主，彩信并不是带有多媒体特征的短信息，技术上和短信息毫无关系，它是在 GPRS 网络的支持下，以 WAP 无线应用协议为载体，传输不仅仅包括文字，还有图片、动画、声音等信息。在 Android 系统中，其存储方式也是与短信有所区别。彩信的基本信息如发送者手机号码（或端口号）、发送日期时间等也是存储在 mmssms.db 数据库文件中。但是彩信的附件却是以文件方式存储在/data/data/com.android.providers.telephony/app_parts 目录下，该目录可以看到形如 PART_126269369827 的文件，这些文件就是彩信的图片、音乐、文本。其相关信息也是存储在 mmssms.db 中。因此，要实现搜索彩信内容，势必要对其附件文件进行处理。

(2) 文件内容的搜索：当前本文实现了针对文件名进行搜索，一个更为实用的需求，就是实现基于文件内容的搜索，特别是针对诸如 pdf、word、Excel 等文档。这大大提高了软件的复杂度和难度，不但要实现相关文件的内容解析，还要实现全文搜索的需求。这是一个巨大的挑战。

(3) 其他信息的搜索：当前，除了系统所带的短信息，联系人，文件等，Android 手机往往安装了许多其他程序，这些程序应用势必会产生自己的有关数据，例如微博，微信，实现针对这些应用的信息搜索，也是有着实际的意义。

(4) 搜索结果的排序：AnySearch 的当前实现，并未实现用户可选择的排序需求。对搜索结果，提供必要的排序方法，如按照时间，大小，类型，频率等等，也是后期需要考虑的重要内容。

参考文献

[1] Mary Meeker，Liang Wu．INTERNET TRENDS D11 CONFERENCE 2013-05-29．

[2] S. Singh．An Analysis of Android Fragmentation［EB/OL］．2012，Available: http://goo.gl/gNDHC

[3] Bruce Eckel．Java 编程思想［M］．陈吴鹏，饶若南等译．北京：机械工业出版社，第 3 版．2005：30-100．

[4] Chunyue BI．Research and Application of SQLite Embedded Database Technology．WSEAS Transactions on Computers［J］，2009，8(1/3)：83-92．

[5] The Architecture of SQLite［EB/OL］．http://www.sqlite.org/arch.html

[6] 万玛宁，关永，韩相军等．嵌入式数据库典型技术 SQLite 和 Berkeley DB 的研究．微计算机信息［J］，2006，22(2)：91-93．

[7] 倪红军．基于 Android 系统的数据存储访问机制研究［J］．计算机技术与发展，2013，23(6)：90-93．

[8] 吴倩，赵晨啸，郭莹．Android 安全机制解析与应用实践［M］．北京：机械工业出版社，2013．04．34-80.

[9] 宋杰，党李成，郭振朝等．Android OS 手机平台的安全机制分析和应用研究．计算机技术与发展［J］，2010，20(6)：152-155．

[10] Ronald Packer II．Android developer offers insights. Network World［J］，2012，29(6)：6-6．

[11] 周兵．Android 中 UI 线程与后台线程交互的探讨［J］．郧阳师范高等专科学校学报，2013，33(3)：14-16．

[12] 杨杰，基于 Android 的多线程处理技术．电脑知识与技术［J］,2013,(18):4251-4254．

[13] 王军伟，武岩，易勇等．浅论 Android 线程模型．中国电子商务［J］，2009，(9)：71-72．

[14] 李君，王华军．浅析 Android Handler 的使用误区与避免．软件[J]，2013，34(3)：147-148．

[15] 刘帅奇，胡绍海，肖扬等．基于 Android 平台的动态消息客户端架构．通信技术应用［J］，2012，(2)：24-30.

[16] 黄蓉．Android 消息处理机制研究．黑龙江科技信息［J］，2012，(33)：87-87．

[17] Android Developer Document［EB/OL］．http://developer.android.com/reference/java/lang/ref/SoftReference.html

[18] Darren Shields. How to use Java Soft References For Caching［EB/OL］．http://java.sys-con.com/node/36434．

[19] 张晓培，李祥．从 Unicode 到 GBK 的内码转换．微计算机应用［J］，2006，27(6)：757-759．

[20] 邱发林，李伟，周绍景等．Unicode 及中文到 Unicode 转换．科技信息［J］（科技教育版），2006，(3)．

[21] 倪红军，钱昌俊．基于 Android 平台的自发短信系统设计与实现．电子技术应用［J］，2012，38(12)：126-129．

[22] 魏化永．基于 3GAndroid 的 SMS 应用研究．计算机光盘软件与应用［J］，2011，(11)：181-183．

[23] 杨小见，任家富，黄美传等．Android 平台 MMS／SMS 的 PDU 编码研究．中国集成电路［J］，2012，(9)：83-87．

[24] Erich Gamma，Richard Helm 等．Design Patterns： Elements of Reusable Object-Oriented Software［M］．李英军等译．北京：机械工业出版社，2000-9：157-170

[25] 刘海岩，锁志海，吕青等．设计模式及其在软件设计中的应用研究．西安交通大学学报［J］，2005，39(10)：1043-1047．

[26] 夏浩波．单例模式的设计与应用．电脑开发与应用［J］，2011，24(1)：58-59．

[27] Metsker S J． Java 设计模式［M］．第 2 版．电子工业出版社，2012．09．32-76.

[28] Tomas Chlouba．Design Patterns in Mobile Architectures［J］．Advances in Communications， Computers， Systems， Circuits and Devices．2010：286-289．

[29] 陈刚．Eclipse 从入门到精通［M］．北京：清华大学出版社， 2005，4．45-200.

[30] Reto Meier． Android2 高级编程［M］ 王超译．北京：清华大学出版社，2010．30-120.

[31] Jason Morris．Android User Interface Development：Beginner's Guide．2011，2．

[32] 王家林．大话企业级 Android 应用开发实战．北京［M］：电子工业出版社，2011．08：20-50.

[33] 单锦辉，姜瑛，孙萍等．软件测试研究进展［J］．北京大学学报（自然科学版），2005，41(1)：134-145．

[34] 万年红，李翔．软件黑盒测试的方法与实践［J］．计算机工程，2000，26(12)：91-9．

致谢

本文研究工作从选题到完成，无不倾注了导师贾春福教授诲人不倦的关怀、指导和教诲。在攻读硕士学位期间，导师渊博的学识、严谨的治学态度、敏锐的科学洞察力、高尚的道德情操、实事求是的学术作风以及为人师表的风范给了我巨大的启迪、鼓舞和鞭策，并将成为我人生道路上的楷模。值此论文完成之际，谨向导师表示我崇高的敬意和衷心的感谢，并诚挚地祝愿导师工作顺利，身体安康，阖家幸福！

在本人攻读硕士学位期间，同时得到了南开大学数学学院科学计算研究所所长梁科教授和谭玉琪老师的大力支持和帮助，在此对他们表示衷心的感谢！

感谢南开大学数学学院领导给了我这次深造的机会，感谢我办公室的同事们在我做论文期间对我的各方面照顾。感谢跟我同期毕业的同学们，你们的督促使我按期完成学位论文。

同时，尤其感谢多年来一直给予我鼎力支持和无私奉献的父母和我的妻子，尤其是我的小学一年级的女儿。没有他们的付出与牺牲，我的学位论文工作就不可能顺利完成，再次真心地感谢和祝福他们！

最后，谨向所有在攻读硕士学位期间曾经关心和帮助过本人的老师和同学表示最诚挚的谢意！衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授!

个人简历

本人生于 1970 年 3 月，1991 年毕业于南开大学数理统计专业。现在南开大学数学学院科学计算所工作，从事集群管理工作和并行算法研究工作。