系统设计文档

目录

[1. 概要设计 2](#_Toc418381781)

[1.1. 引言 2](#_Toc418381782)

[1.1.1. 编写目的 2](#_Toc418381783)

[1.1.2. 背景 2](#_Toc418381784)

[1.1.3. 参考资料 3](#_Toc418381785)

[1.2. 任务概述 5](#_Toc418381786)

[1.2.1. 目标 5](#_Toc418381787)

[1.2.2. 开发环境 5](#_Toc418381788)

[1.2.3. 运行环境 5](#_Toc418381789)

[1.2.4. 项目依赖 5](#_Toc418381790)

[1.3. 总体设计 7](#_Toc418381791)

[1.3.1. 处理流程 7](#_Toc418381792)

[1.3.2. 总体结构和模块外部设计 8](#_Toc418381793)

[2. 详细设计 10](#_Toc418381794)

[2.1. 引言 10](#_Toc418381795)

[2.1.1. 背景 10](#_Toc418381796)

[2.1.2. 工作条件与限制 10](#_Toc418381797)

[2.2. 模块设计 11](#_Toc418381798)

[2.2.1. 模块划分依据 11](#_Toc418381799)

[2.2.2. 模块结构与功能描述 11](#_Toc418381800)

[2.3. 数据库设计 13](#_Toc418381801)

[2.3.1. 数据库总体结构 13](#_Toc418381802)

[2.3.2. 数据库逻辑设计 13](#_Toc418381803)

[2.3.3. 数据库物理设计 13](#_Toc418381804)

[2.4. 用户界面设计 14](#_Toc418381805)

[2.4.1. 用户界面设计原则 14](#_Toc418381806)

[2.4.2. 布局设计 16](#_Toc418381807)

[2.5. 程序模块设计 24](#_Toc418381808)

[2.5.1. 短信监听与拦截模块 24](#_Toc418381809)

[2.5.2. 中文分词引擎模块 25](#_Toc418381810)

[2.5.3. 日期时间与地点分析识别模块 31](#_Toc418381811)

[2.5.4. 事件记录处理与提醒模块 31](#_Toc418381812)

[2.5.5. 用户数据存储与备份还原模块 32](#_Toc418381813)

[2.5.6. 事件日历查询模块 33](#_Toc418381814)

# 概要设计

## 引言

### 编写目的

本文档是概要设计书，可以为测试、开发以及接下来的文档做指导作用

### 背景

现代生活中，智能手机占据了越来越重要的地位。所谓智能手机，在初始狭义定义下是指可以自由安装软件（Software）或称应用（Application）的移动数字电话机。明显，这一定义其实和“智能”本身并无关系。究其原因，在2007年初代iPhone面试之前，市面上的智能手机性能明显不足，保有量也不算大，并且没有一个规范的市场。在此现状下，实现所谓“智能”确实十分困难。

这一切被摩尔定律和时间渐渐改变了，悄无声息。2007年当我很兴奋的看完《电脑报》上关于iPhone的报道后，无比兴奋的向旁人介绍那多点触控的高分辨率屏幕、有着400MHz几乎是我当时台式机CPU一半主频的ARM芯片时，只能得到一片茫然。时至今日，苹果的发布会被人们当做娱乐盛典谈论，地铁上几乎就是新品展销会。现在想看到一台非智能手机的难度，已经高过开售日首日在地铁上遇到新款iPhone了。

所以我们的手机，更应该名副其实一些，“智能”，对吗？

而“智能”的基础则是理解。

手机芯片性能和蜂窝数据网络的发展，使得手机应用的大规模计算成为了可能。苹果公司的个人数字助理Siri即是通过网络在线处理数据实现。而Google中文输入法的离线语音输入则更令人惊叹。

对于一般开发者而言，中文语言处理却并没有很好的解决方案。虽然现在基于Java语言的开源分词系统很多，但是经过大量测试，这些系统都使用了庞大的语料库，从而导致没有可能直接在低性能的手机平台使用。而在线的网络延迟和流量消耗，对于一款应用的使用体验有着非常大的影响。

### 参考资料

[1]Qian, J.,Zhu, Q.,Wang, Y. et al.Bloom Filter Based Associative Deletion[J].IEEE Transactions on Parallel and Distributed Systems: A Publication of the IEEE Computer Society,2014,25(8):1986-1998.

[2]Automatic Extraction Of New Words Based On Google News Corpora For Supporting Lexicon-based Chinese Word Segmentation Systems[J].Expert Systems with Application,2009,36(2 Pt.2):3641-3651.

[3]Zhang MY,Lu ZD,Zou CY et al.A Chinese word segmentation based on language situation in processing ambiguous words[J].Information Sciences: An International Journal,2004,162(3/4):275-285.

[4]Sun, X.,Zhang, Y.,Matsuzaki, T. et al.Probabilistic Chinese word segmentation with non-local information and stochastic training[J].Information Processing &amp;amp; Management: Libraries and Information Retrieval Systems and Communication Networks: An International Journal,2013,49(3):626-636.

[5] Fu, GH,Kit, C,Webster, JJ et al.Chinese word segmentation as morpheme-based lexical chunking[J].Information Sciences,2008,178(9):2282-2296.

[6] Richard Tzong-Han Tsai.Chinese text segmentation: A hybrid approach using transductive learning and statistical association measures[J].Expert Systems with Application,2010,37(5):3553-3560.

[7]封春升,郝爱民.基于模式匹配的自然语言识别[J].计算机工程与应用,2006,42(19):144-146,166.DOI:10.3321/j.issn:1002-8331.2006.19.045.

[8]孔世明.基于自然语言识别的智能检索研究与实现[D].重庆大学,2006.DOI:10.7666/d.y1023045.

[9]罗智勇,宋柔.基于多特征的自适应新词识别[J].北京工业大学学报,2007,33(7):718-725.DOI:10.3969/j.issn.0254-0037.2007.07.010.

[10]高啸.基于自然语言处理的空间信息检索优化应用研究[D].浙江大学,2014.

[11]周雅倩,郭以昆,黄萱菁等.基于最大熵方法的中英文基本名词短语识别[J].计算机研究与发展,2003,40(3):440-446.

[12]傅爱平.汉语信息处理中单字的构词方式与合成词的识别和理解[J].语言文字应用,2003,(4):25-33.DOI:10.3969/j.issn.1003-5397.2003.04.004.

[13]唐雄明.基于隐马尔可夫模型的中英文句子分析与研究[D].武汉理工大学,2012.DOI:10.7666/d.y2098340.

[14]李庆虎,陈玉健,孙家广等.一种中文分词词典新机制--双字哈希机制[J].中文信息学报,2003,17(4):13-18.DOI:10.3969/j.issn.1003-0077.2003.04.002.

[15]周俊,郑中华,张炜等.基于改进最大匹配算法的中文分词粗分方法[J].计算机工程与应用,2014,(2):124-128.DOI:10.3778/j.issn.1002-8331.1203-0002.

[16]郭玲,孟祥逢,张峰等.基于最大匹配法的中文分词技术改进[J].舰船电子工程,2009,29(12):159-163.DOI:10.3969/j.issn.1627-9730.2009.12.044.

[17]丁振国,张卓,黎靖等.基于Hash结构的逆向最大匹配分词算法的改进[J].计算机工程与设计,2008,29(12):3208-3211,3265.

[18]陈鹏宇.现代汉语自动分词研究及几个算法的实现[D].电子科技大学,2008.

[19]张李义,李亚子.基于反序词典的中文逆向最大匹配分词系统设计[J].现代图书情报技术,2006,(8):42-45,30.DOI:10.3969/j.issn.1003-3513.2006.08.009.

[20]刘威,郭渊博,黄鹏等.基于多维布隆过滤器的模式匹配引擎[J].计算机应用,2011,31(1):107-109,114.DOI:10.3724/SP.J.1087.2011.00107.

## 任务概述

### 目标

本项目希望能开发这样一款应用：

当手机接收到通知短信的时候，可以对短信全文进行快速离线分词，应用可以自动识别其中的时间、地点等信息，进而自动创建闹钟提醒。让用户不会错过任何一次约会安排。

### 开发环境

操作系统：Windows 8.1 Enterprise With update Build 9600

JRE：Java(TM) SE Runtime Environment (build 1.8.0\_31-b13)

JVM：Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)

Android SDK：Android 5.0.1（API 21）SDK Platform

IDE：IntelliJ IDEA 14.0.2 Build #IU-139.659

版本控制：Git version 1.9.0.msysgit.0

版本库托管：Git@OSC

### 运行环境

Android 操作系统 4.0以及上版本。

### 项目依赖

[Action Bar Sherlock](https://github.com/JakeWharton/ActionBarSherlock)

（<https://github.com/JakeWharton/ActionBarSherlock>）

License

Copyright 2012 Jake Wharton

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

Android Switch Preference Backport

（<https://github.com/BoD/android-switch-backport>）

Licence

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

## 总体设计

### 处理流程



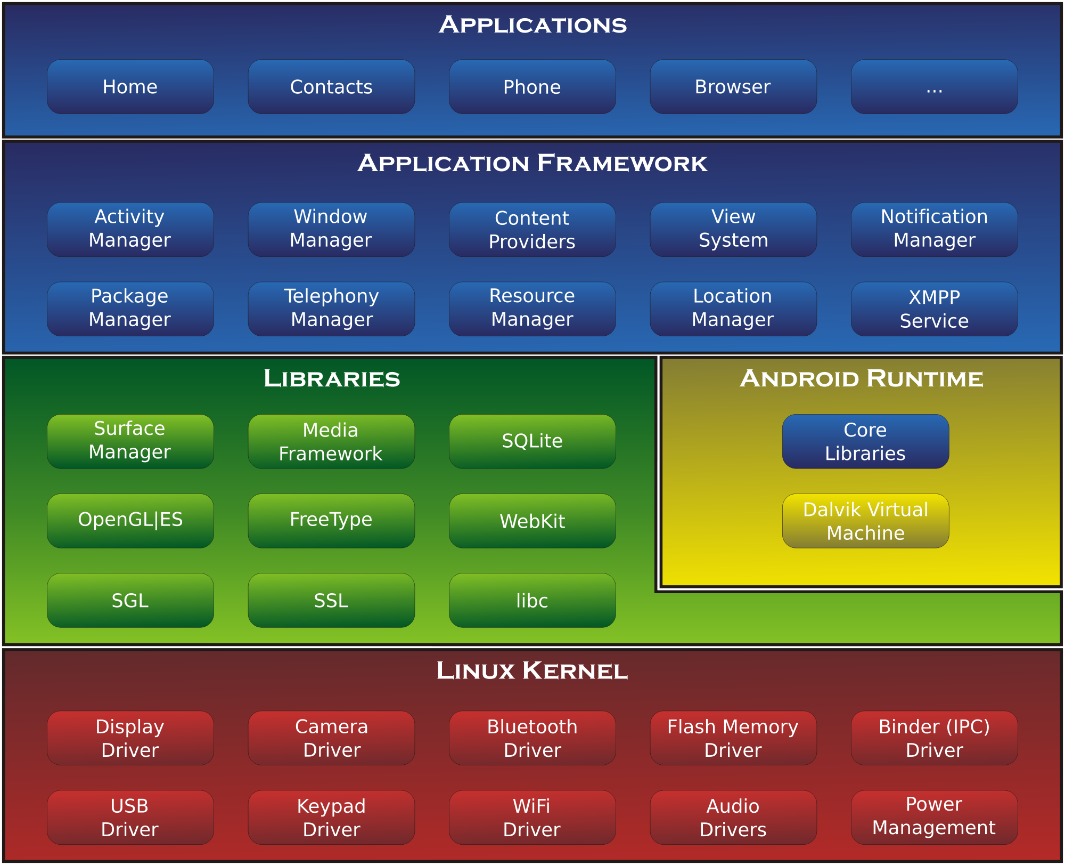
### 总体结构和模块外部设计

本应用基于Android平台开发。

在Application Framework中使用Activity进行窗口管理，Content Providers对配置数据进行持久化操作，Resource Manager实现字典访问与读取。

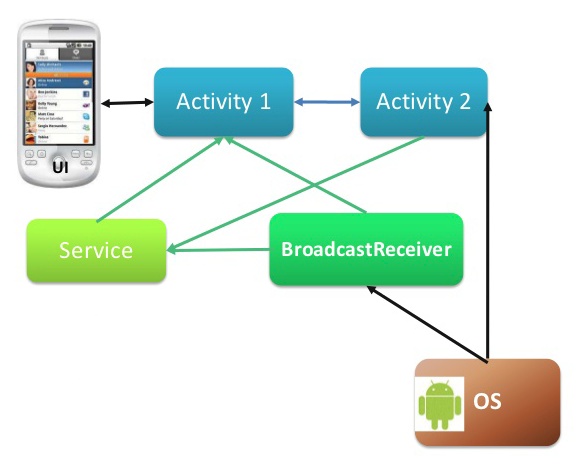
在Libraries层调用SQLite提供的数据库管理

Android平台架构图：



在项目里，由Application提供操作界面。Server与BroadcastReceiver提供了短信监听功能。

应用总体结构图：



# 详细设计

## 引言

### 背景

本项目需要在Android平台实现如下目标：

当手机接收到通知短信的时候，可以对短信全文进行快速离线分词，应用可以自动识别其中的时间、地点等信息，进而自动创建闹钟提醒。让用户不会错过任何一次约会安排。

### 工作条件与限制

本项目工作环境为Android智能手机。依据前期市场调查和综合实现成本，要求本项目可以正常工作在Android 操作系统 4.0以及上版本的市售最低端型号。所有功能必须离线实现，联网只能用于信息统计。

## 模块设计

### 模块划分依据

模块依据具体功能实现进行划分，各模块间解耦合，之间通过API进行通讯。

本系统共分为七个模块：

* 短信监听与拦截模块；
* 中文分词引擎模块；
* 日期时间与地点分析识别模块；
* 事件记录处理与提醒模块；
* 用户数据存储备份还原模块；
* 事件日历查询模块。
* UI显示模块

### 模块结构与功能描述

短信监听与拦截模块

中文分词引擎模块

事件记录处理与提醒模块

日期时间与地点分析识别模块

用户数据存储备份还原模块

事件日历查询模块

UI显示模块

每一个模块的详细功能介绍如下。

* **短信监听与拦截模块**
  + 可以监听并拦截下可能为会议通知的所有短信。对判断非会议通知短信不进行操作。
  + 在手机上安装有安全软件以及短信增强应用的情况下，依然可以及时获取到新增短信。
  + 允许用户设置关闭监听。
* **中文分词引擎模块**
  + 对给定任意长度文字，在搭载1GHz单核ARM v6以下CPU的移动Android设备上在1s内返回分词结果。全过程不允许联网。
* **日期时间与地点分析识别模块**
  + 对给定任意长度文字，返回可能包含的时间数据以及地点数据
* **事件记录处理与提醒模块**
  + 通过识别出的信息，自动生成日历事件。按期发出提醒。
* **用户数据存储与备份还原模块**
  + 所有用户数据要求可以备份及还原。
* **事件日历查询模块**
  + 可以对事件进行查询。要求可以模糊查询，并实时反馈
* **UI显示模块**
  + 将数据和UI Layout进行绑定

## 数据库设计

### 数据库总体结构

数据库使用Android默认支持的SQLite数据库。

### 数据库逻辑设计

命名：数据库表和字段名全部小写，下划线分割

用户自定义位置，共有字段两个：\_id以及location。

### 数据库物理设计

User\_location表

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 类型 |
| \_id | ID | Int |
| location | 位置 | varchar |

## 用户界面设计

### 用户界面设计原则

* **吸引我**
  + **以令人惊讶的方式取悦我**

漂亮的界面，精心设置的动画，或是恰到好处的声音效果，都是快乐的情感体验。各种细微的效果精妙地组合在一起往往效果惊人，不仅显得轻松明快，更可以营造出丰富强大的氛围。

* + **真实的物件远比按钮和菜单更有趣**

使用现实世界中的对象和操作来构建应用中的对象和操作。这样不仅能够帮助用户，尤其是新手，迅速地理解应用程序的操作方法和作用，减少认知成本，同时也使他们能获得更多的情感满足。

* + **把它变成“我的”**

人们喜欢添加个性化风格，因为这让他们感觉像在家一样一切尽在掌握，也让这个应用显得和自己一样与众不同。应用应该提供明快合理且漂亮的默认值，同时最好也考虑添加一些有趣但不妨碍主要任务的自定义项目。

* + **了解我**

随着使用时间的推移，应用应该了解并记住用户的行为和使用偏好。不要让用户一遍又一遍作出相同的选择，最好能让他们能够方便地使用上一次的选择。

* **简化我的生活**
  + **用语简洁**

使用简单的单词和短句进行说明——用户极有可能跳过冗长复杂的句子。

* + **图片比文字更直接**

尽量使用图片来进行说明。图片更吸引用户的目光，往往也比文字更有效。

* + **为我选择，但最终是我说了算**

智能地推断用户的行为，并推荐相应的解决方案，而不是一开始就询问用户想干嘛——太多的选择和决定让用户烦躁。不过为防万一推荐得不对，记得给出“撤消”操作。

* + **只在我需要的时候显示**

同时看到太多的东西让人感到不堪重负，所以最好把复杂的任务和信息分解成一段段的便于操作和理解的步骤。隐藏当前不需要的选项，并让当用户学会在需要时到哪儿去找它们。

* + **我应该知道我在哪里**

让用户清楚地知道他们在应用中的位置和操作路径。让你的应用中各个部分看起来区分明确，并在界面切换时使用适当的过渡效果，以表明它们之间的关系。对正在进行的任务提供反馈。

* + **不要丢失我的内容与设置**

保存用户花费时间创建的内容与设置，让它们在任何时间任何地点都可以被访问，并在手机、平板电脑和计算机之间保持同步一致。这也可以让升级变得更加容易与方便。

* + **如果它看起来一样，那它的功能也应该一样**

尽量使用明显是视觉差异来帮助用户区分功能上的不同。避免使用看上去相同或相似的模式、操作来实现其实并不相同功能。

* + **只在重要的时刻打断我**

如同拥有一个优秀的私人助理一样，用户不应该被无关紧要的旁枝末节所打断。人们希望能集中注意力完成手上的任务，除非有十万火急的事情需要处理。任务中的一次打断真是让人觉得费力又沮丧。

* **让我惊艳**
  + **给我各处皆适用的技巧**

当用户自己摸索出使用技巧时会感觉非常良好。在你的应用程序里使用其他安卓应用广泛采用的交互技巧——包括相同的视觉模式和肌肉动作，可以让用户更容易上手也更容易获得满足。比如，“轻扫”就是一个在安卓应用中广泛使用的良好的导航快捷手势。

* + **不是我的错**

友好礼貌地提醒用户改正错误——在使用你的应用时，用户当然不希望自己显得很蠢。如果出了差错，请清楚简明地给出解决方案，以便用户能够快速地进行修正，避免提及让用户看不懂的技术细节。如果应用能够在后台默默处理那就更好了！

* + **积极反馈**

把复杂的任务分解成较小的步骤可以让用户更轻松地完成。给每个行动都提供反馈鼓励，即使只是一个微妙的光芒也会让用户大受鼓舞。

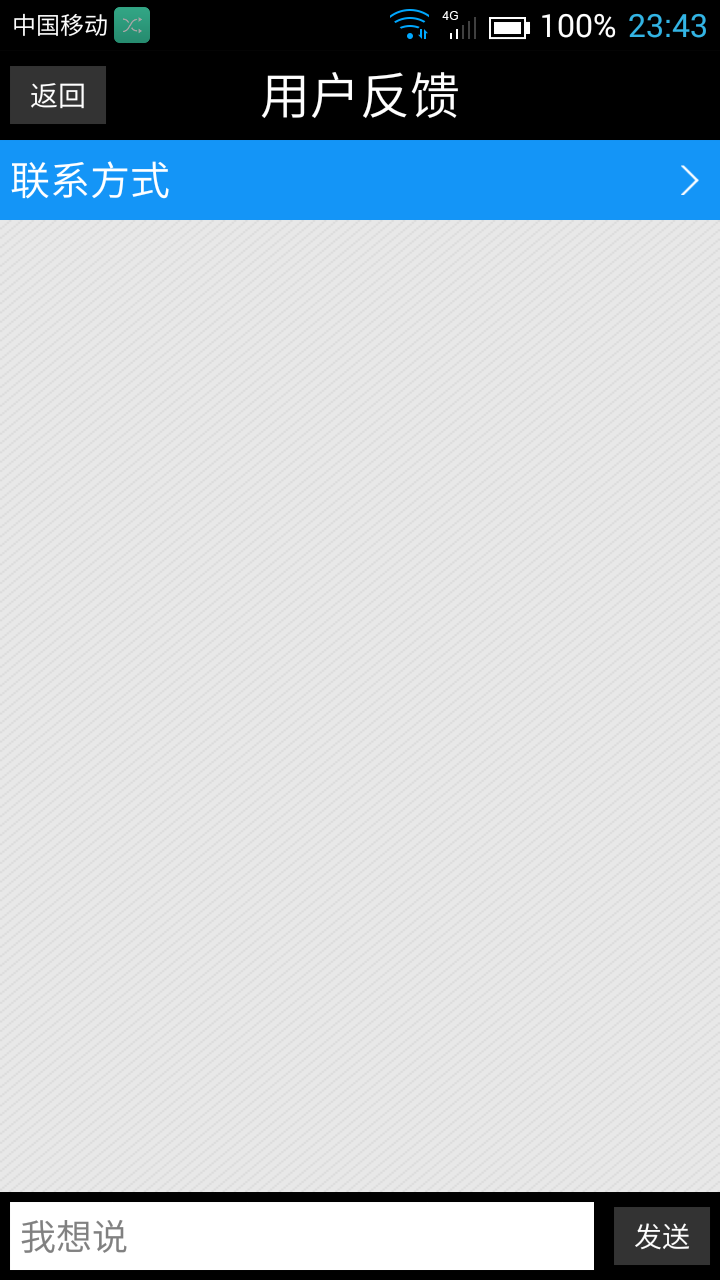
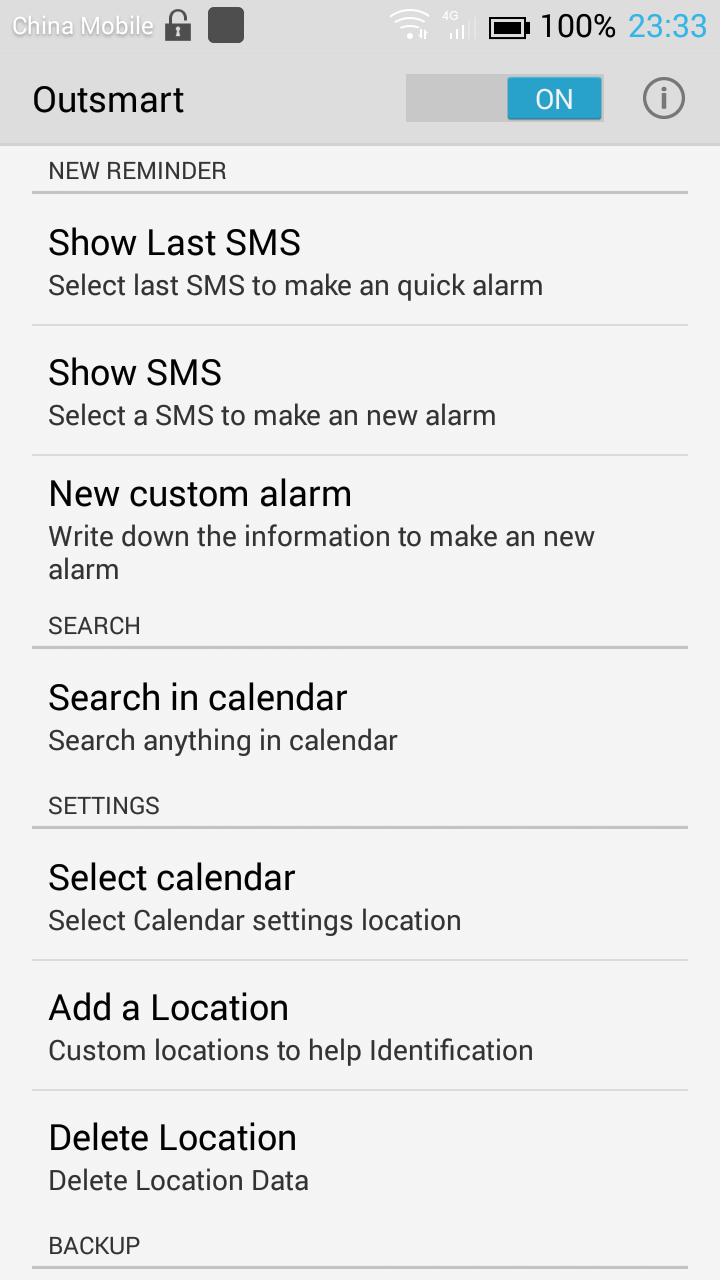
* + **为我完成复杂的任务**

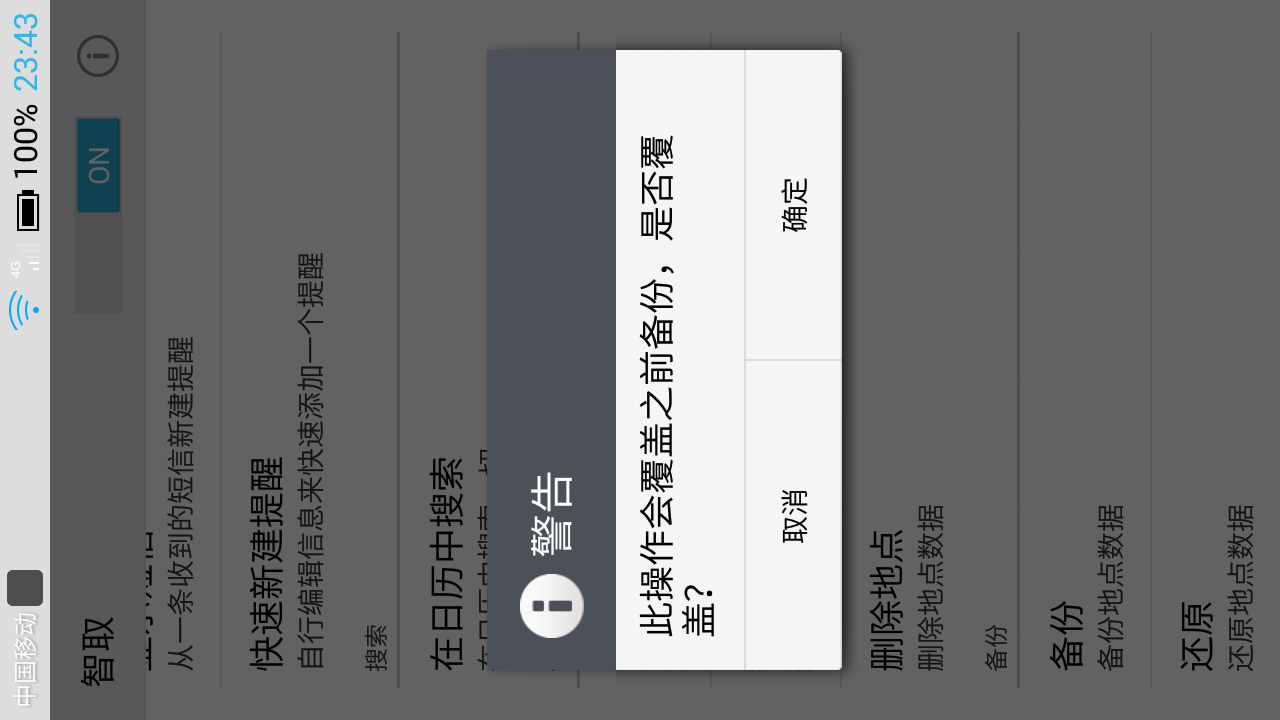
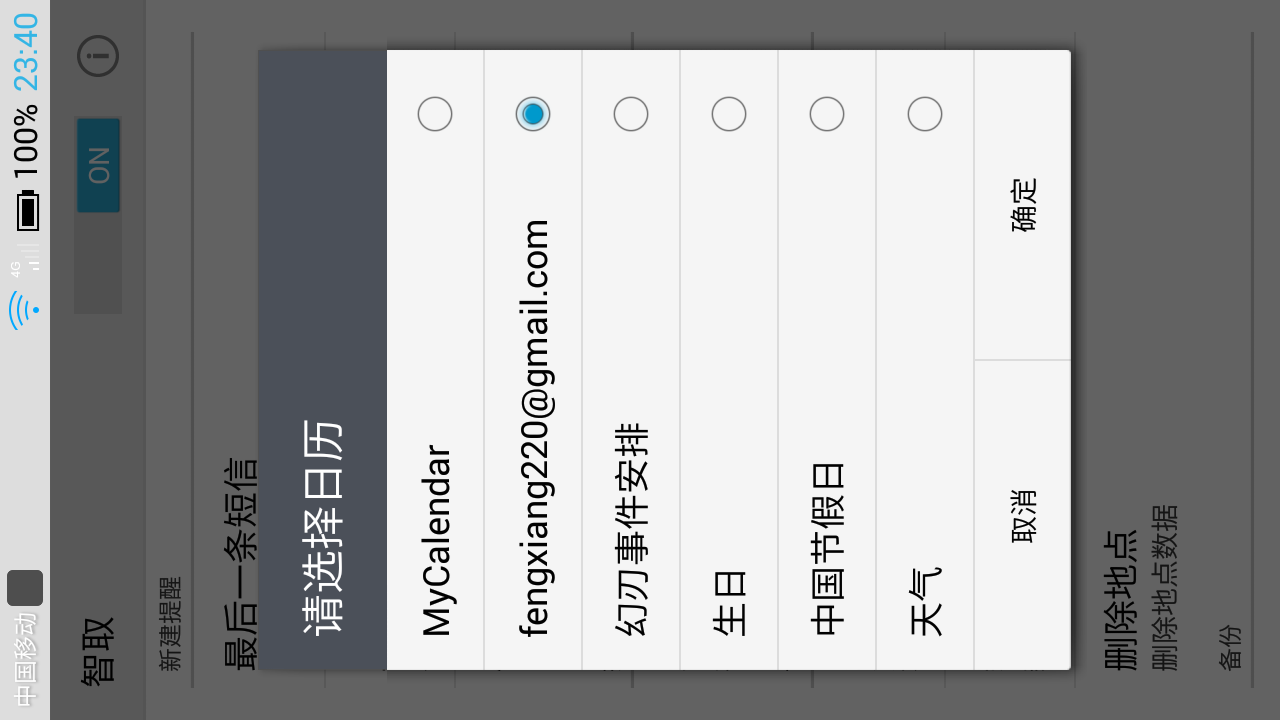
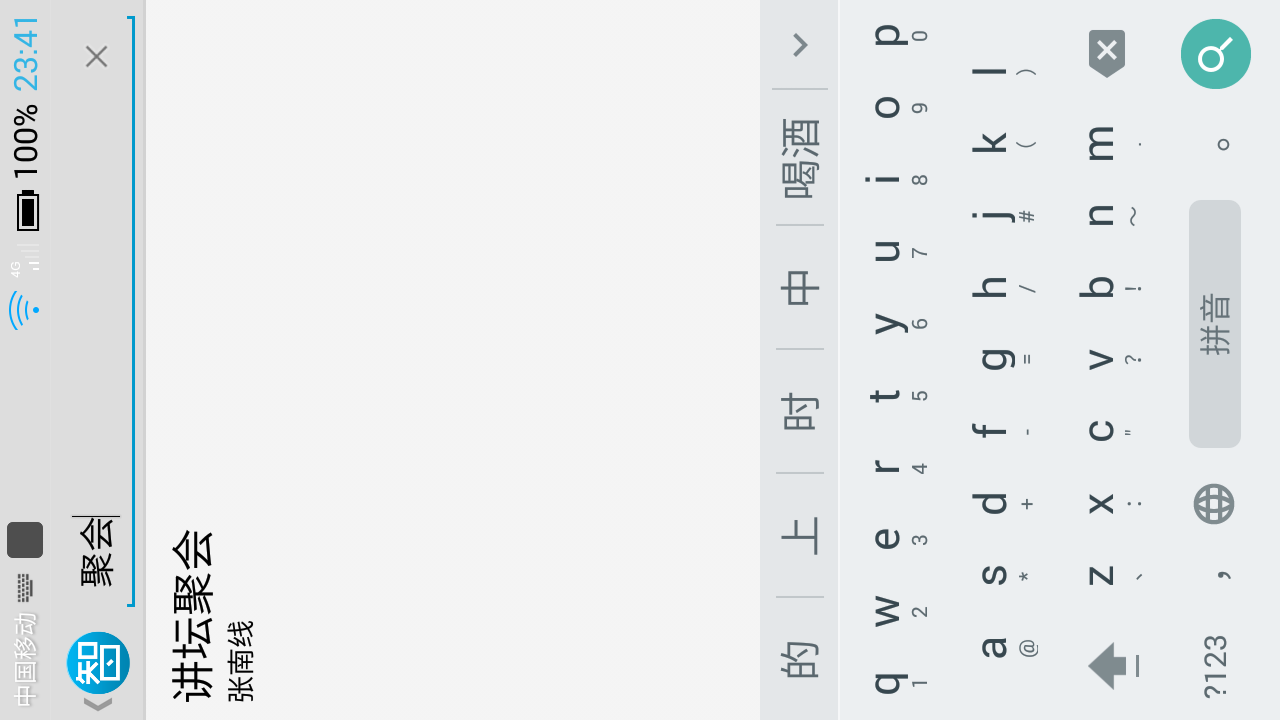
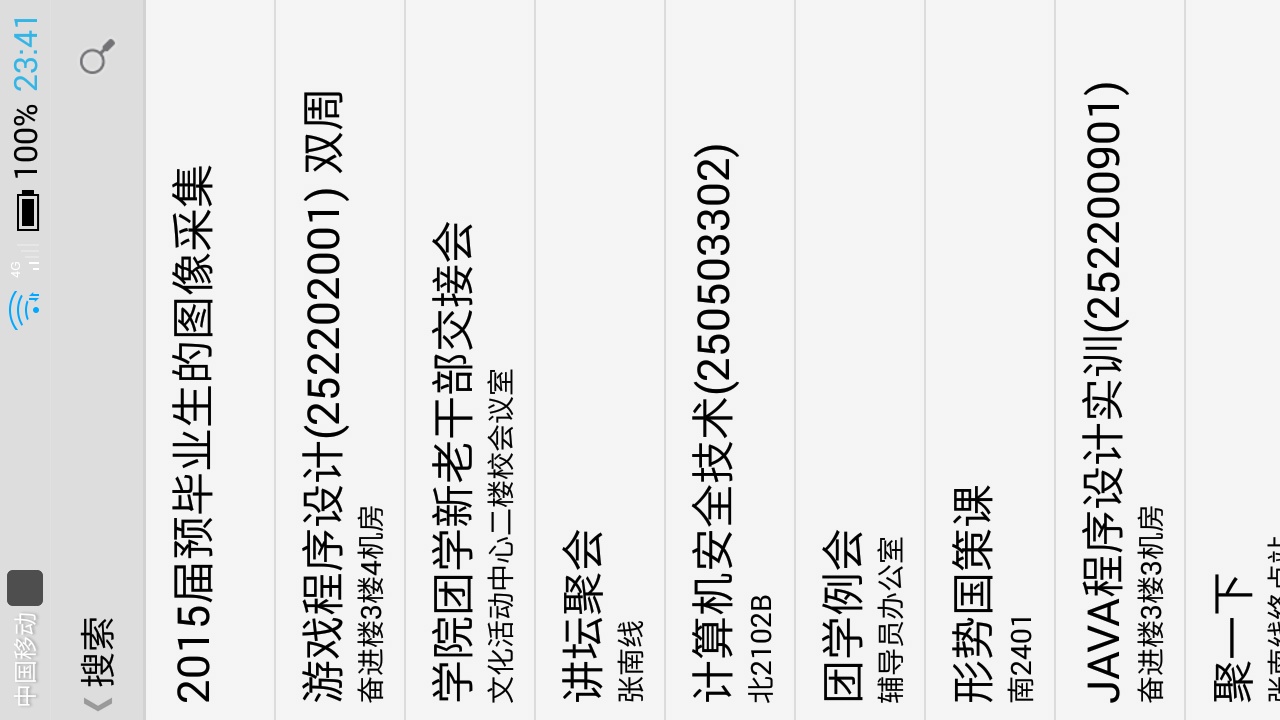
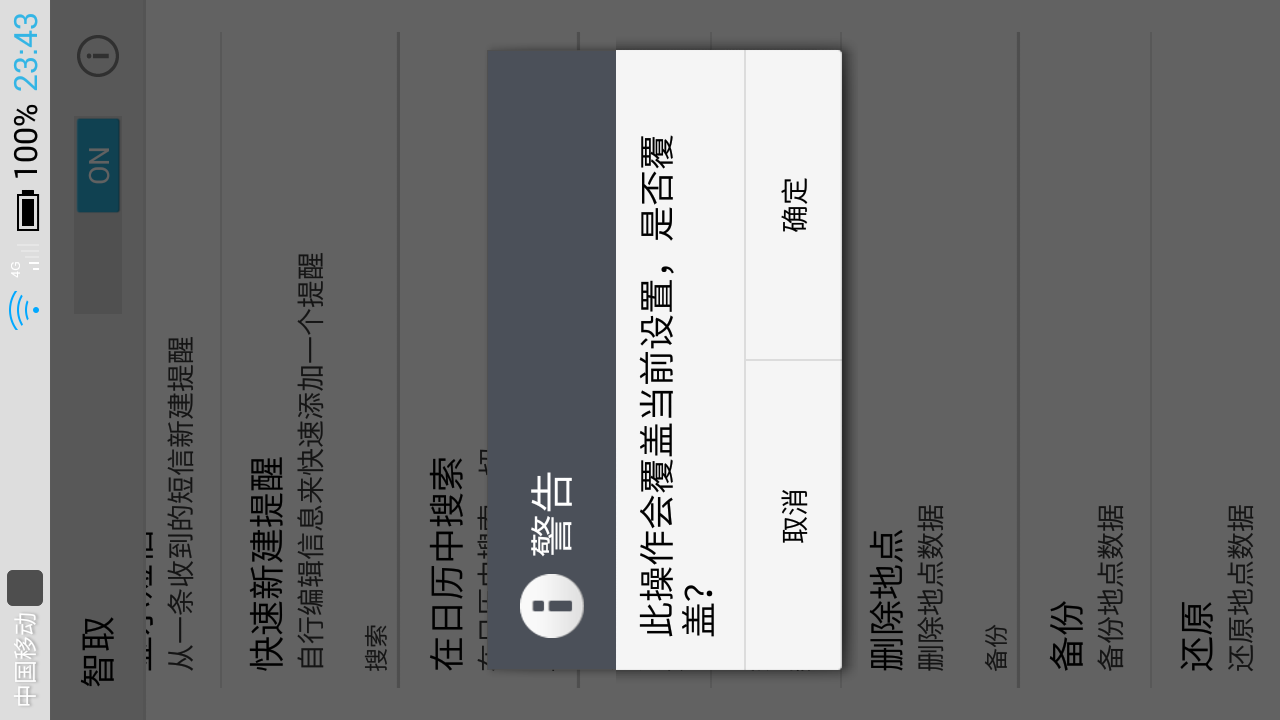
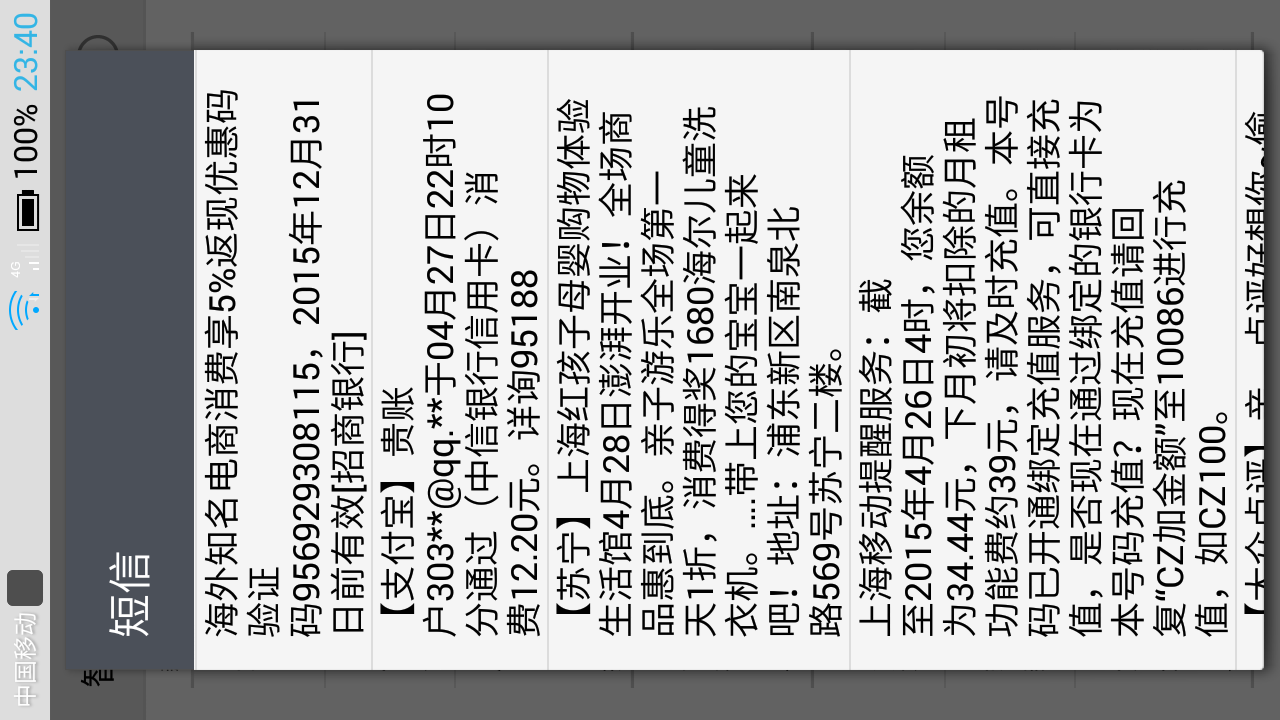
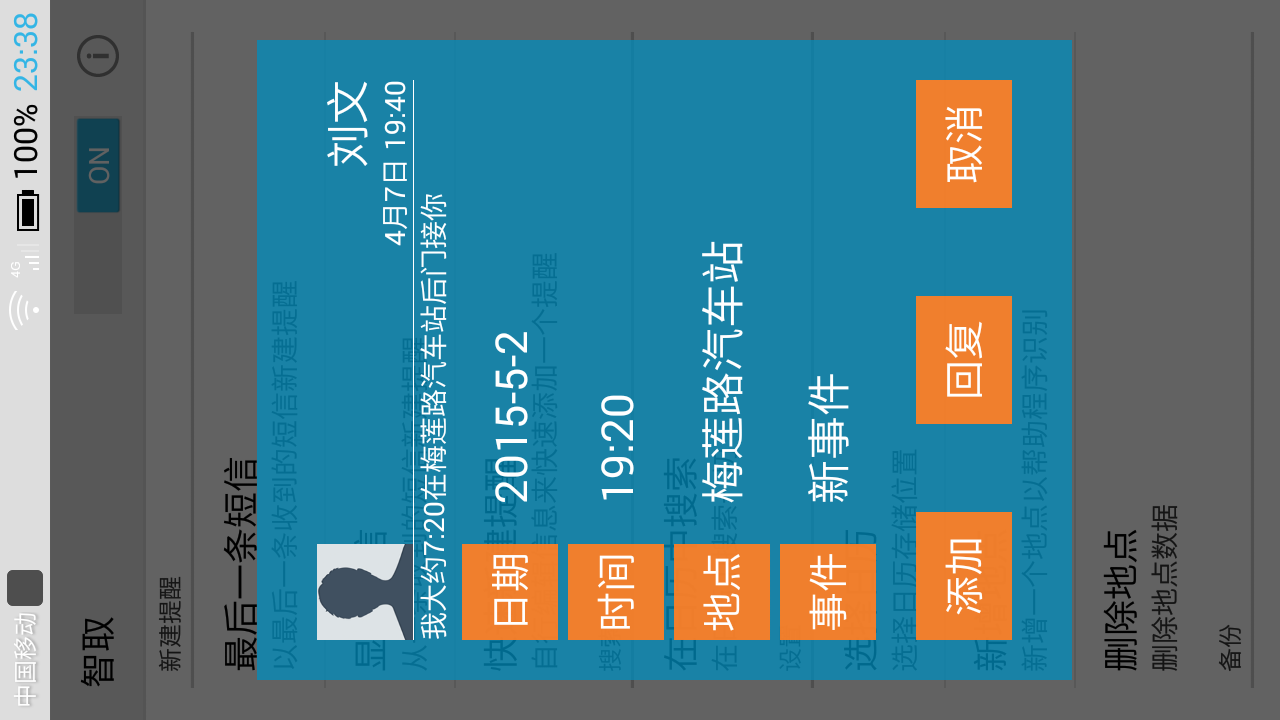
让新手使用他们能够做的简单操作，完成他们从未奢望过可以处理的专家级任务。例如，结合多种照片效果的快捷方式，只需几步操作，就可以使业余的照片看起来让人惊艳。

* + **让更重要的事情更快**

并非所有的操作都是平等的。决定在你的应用程序什么是最重要的，并让它能更容易更快速地被找到和使用——比如照相机的快门按钮，或是音乐播放器的播放/暂停按钮。

### 布局设计





## 程序模块设计

### 短信监听与拦截模块

#### 短信拦截的两种方式

Android平台短信监听有两种方式。分别是通过的广播接收者（BroadcastReceiver）以及自行通过ContentObserver监听短信数据库。

#### 广播接收者

当系统收到短信时，会发出一个action名为android.provider.Telephony.SMS\_RECEIVED的广播Intent，该Intent存放了系统接受到的短信内容，我们可以使用名称"pdus"即可从Intent中获取到短信内容。

广播可以分为普通广播和有序广播。普通广播是完全异步的，可以在同一时刻(逻辑上)被所有接受者接收到，相对于有序广播消息传递的效率比较高，但缺点是：接受者不能将出黎结果传递给下一个接受者，并且无法终止广播Intent的传播；有序广播是按照接受者声明的优先级别，被接受者依次接受广播。优先级别在<intent-filter>的android:priority属性中声明，数值越大优先级别越高，取值范围为-1000到1000，优先级别也可以调用IntentFilter对象的setPriority()进行设置。有序广播的接受者可以终止Intent的传播，广播Intent的传播一旦终止，后面的接受者就无法接收到广播。另外，有序广播的接受者可以将数据传递给下一个广播。例如A得到广播后，可以往它的结果对象中存入数据，当广播传给B时，B可以从A的结果对象中得到A存入的数据。

Context.sendBroadcast()

发送的是普通广播，所有订阅者都有机会获得并进行处理。

Context.sendOrderedBroadcast()

发送的是有序广播，系统会根据接受者声明的优先级别按顺序逐个执行接受者，前面的接受者有权终止广播（BroadcastReceiver.abortBroadcast()），如果广播被前面的接受者终止，后面的接受者就再也无法获取到广播。对于有序广播，前面的接受者可以将数据通过setResultExtras(Bundle)方法存放进结果对象，然后传给下一个接受者，下一个接受者通过代码：Bundle bundle = getResultExtras(true)可以获取上一个接受者存入在结果对象中的数据。

系统收到短信，发出的广播属于有序广播。如果想阻止用户收到短信，可以通过设置优先级，让自定义的接受者先获取到广播，然后终止广播，这样用户就接受不到短信了。

#### 广播接收者为什么不适用于本项目

用户手机中安装的系统安全应用（如360安全卫士）以及短信增强应用（包括短信替代应用）一般都会以高权限直接拦截短信，所以普通方法根本无法正常监听短信。

故此，使用直接通过ContentObserver监听短信数据库的方式。

#### ContentObserver实现细节

为防止系统清理后台进程，该监听器由服务启动，并且该服务开机自动启动。

为进一步保证稳定，还应该注册BroadcastReceiver，当系统接收到短信时，判断服务是否存在，如果不存在则启动。

ContentObserver注册之后，在短信数据库有更改的时候发生回调。所谓更改，当然有新增和删除两种情况了，所以必须判断。方法是在应用第一次启动的时候统计当前短信数据库条数，并持久化存储。当发生回调的时候，将当前短信数据库条数与持久化的数据对比。获得当前状态，并更新持久化数据。

### 中文分词引擎模块

#### 中文分词的主要实现方式

中文分词算法大概分为两大类，基于字符串匹配和基于统计以及机器学习的分词方式。

* **基于字符串匹配**，即扫描字符串，如果发现字符串的子串和词相同，就算匹配。

这类分词通常会加入一些启发式规则，比如“正向/反向最大匹配”, “长词优先” 等策略。

这类算法优点是速度块，都是O(n)时间复杂度，实现简单，效果尚可。缺点则是对歧义和未登录词处理不好。

歧义的例子很简单"长春市/长春/药店" "长春/市长/春药/店".

未登录词即词典中没有出现的词，当然也就处理不好。

现有分词项目中IKanalyzer,Paoding等就是基于字符串匹配的分词。

* **基于统计以及机器学习的分词方式**

这类分词基于人工标注的词性和统计特征，对中文进行建模，即根据观测到的数据（标注好的语料）对模型参数进行估计，即训练。 在分词阶段再通过模型计算各种分词出现的概率，将概率最大的分词结果作为最终结果。常见的序列标注模型有HMM和CRF。

这类分词算法能很好处理歧义和未登录词问题，效果比前一类效果好，但是需要大量的人工标注数据，以及较慢的分词速度。

ICTCLAS是基于HMM的分词库。

#### 本项目对分词方案的选择

考虑到需要在一个低性能平台的实现中文分词，在对准确性要求一般的情况下，基于字符串匹配的方式是比较好的选择。

经过对各种分词方法的效率与准确性平衡实验，我最终确定采用逆向最大匹配法进行分词。字典则使用了来源于网络的共计收录23万词的词典。

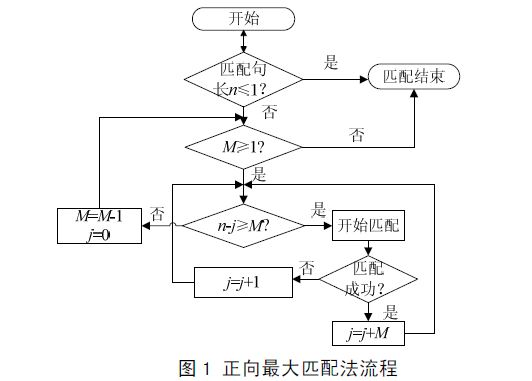
在对字典查询的前期研究中，我使用了直接String匹配、数据库匹配、Hash查询等方法，均远远超过Android可以承载的极限。在尝试现有所有基于Java的分词引擎后发现全部都无法在Android平台正常运行。经过长期研究实验，最终使用Bloom过滤器算法成功达到所需性能要求。

#### 最大匹配法分词算法

基于字符串匹配的方法又称为机械分词方法或字典匹配方法，它主要依据词典的信息，而不使用规则知识和统计信息，按照一定的策略将待切分的汉字串与词典中的词条逐一匹配，若在词典中找到该词条，则匹配成功，否则做其它相应的处理。机械分词法依据待切分文本扫描的方向不同，分为正向匹配、逆向匹配以及双向匹配；依据分词过程是否与词性标注过程相结合，又可分为单纯分词方法和分词与标注相结合的一体化方法；依据每次匹配优先考虑长词还是短词，分为最大匹配和最小匹配。常用的基于字符串匹配的分词方法通常是将上述几种单一方法组合起来使用，例如：基于字符串的正向最大匹配、逆向最大匹配、双向最大匹配以及最少切分等。

* 正向最大匹配分词算法

根据匹配不成功时重新切取的策略区分，机械匹配分词法又分为增字匹配法和减字匹配法。增字法一般与最小匹配相结合，而减字法常与最大匹配相结合。因此正向最大匹配法采用减字匹配法较为常见，其基本思想是：假设己知机器词典中最长词条的长度为N，则以N作为减字开始的长度标准，首先将待扫描的文本串S从左向右截取长度为N的字符串W1，然后在词典中查找是否存在该字符串W1的词条。如果匹配成功，则W1标记为切分出的词，再从待扫描文本串的N+1位置开始扫描；如果匹配失败，将截取长度减1后，再从S中截取此长度的字符串W1’，重复上述匹配过程,直至截取长度为1为止。以扫描完句子作为整个匹配过程结束。其算法流程如图所示，经过这一流程处理后，原本的句子S将被切分成W1W2…Wn的词序列，每一个Wi均为词典中的词条或者是原子。



* 逆向最大匹配分词

逆向最大匹配分词法，其基本思想与正向最大匹配分词法大体一致，只是扫描方向换成了从右至左。换句话说，当扫描汉语句子时，根据词典中最长词条的长度，从句末开始向左截取出汉语字符串与词典中的词条匹配，匹配流程与减字法相同，直至扫描到句首为止。例如，待切分字串为“他说的确实在理”时，正向最大匹配分词法的分词结果为“他/说/的确/实在/理/”，逆向最大匹配分词法的分词结果为“他/说/的/确实/在理/”，根据汉语原意，逆向最大匹配的分词结果是正确的，而正向最大匹配是错误的。据统计结果表明，单纯使用正向最大匹配法的错误率为1/169，单纯使用逆向最大匹配法的错误率为1/245，显然逆向最大匹配分词法较正向最大匹配分词法在切分准确率上有了较大提高，这一结果与汉语中心语偏后有一定的关系。为了节省处理待匹配字符串的时间，逆向最大匹配通常将词典中的词条也组织成逆序，例如“逆向”这一词条，在逆向最大匹配的分词词典中以“向逆”形式存储。

#### 逆向最大匹配分词算法的实际代码实现

ArrayList<String> list = new ArrayList<String>();

String subString = null;

int max = 5;

int i;

int point = str.length();

while (point != 0) {

if (point - max < 0)

subString = str.substring(0, point);

else

subString = str.substring(point - max, point);

for (i = 0; i < subString.length(); i++) {

String tempString = subString.substring(i, subString.length());

int[] backhash = GH.getHashCode(tempString);

if (isInDicByBloom(tempString, backhash)

|| i == subString.length() - 1) {

list.add(tempString);

point = point - subString.length() + i;

break;

}

}

}

// 把分词结果倒序使其正序

Stack<String> stack = new Stack<String>();

for (i = 0; i < list.size(); i++)

stack.push(list.get(i));

list.clear();

while (!stack.empty())

list.add(stack.pop() + "");

#### Bloom过滤器

上文已经解决了分词的算法问题，但是还留有一个小疑问：我们是怎样判断一个词是否匹配的呢？换言之，我们是怎样知道一个词是否在给定字典中的。

这个问题本身是个很简单的问题，甚至都没必要使用KMP算法，直接BF匹配即可。可惜的是，我们将要面对的是一个拥有27万词的大型词典。而运行代码的，只是一台普通的手机。最终，解决这一切的英雄，就是Bloom过滤器。

布隆过滤器（英语：Bloom Filter）是1970年由布隆提出的。它实际上是一个很长的二进制矢量和一系列随机映射函数。布隆过滤器可以用于检索一个元素是否在一个集合中。它的优点是空间效率和查询时间都远远超过一般的算法，缺点是有一定的误识别率和删除困难。

布隆过滤器的原理是，当一个元素被加入集合时，通过K个[散列函数](http://zh.wikipedia.org/wiki/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0" \o "散列函数)将这个元素映射成一个位[数组](http://zh.wikipedia.org/wiki/%E6%95%B0%E7%BB%84)中的K个点，把它们置为1。检索时，我们只要看看这些点是不是都是1就（大约）知道集合中有没有它了：如果这些点有任何一个0，则被检元素一定不在；如果都是1，则被检元素很可能在。这就是布隆过滤器的基本思想。

相比于其它的数据结构，布隆过滤器在空间和时间方面都有巨大的优势。布隆过滤器存储空间和插入/查询时间都是常数（O(k)）。另外, 散列函数相互之间没有关系，方便由硬件并行实现。布隆过滤器不需要存储元素本身，在某些对保密要求非常严格的场合有优势。布隆过滤器可以表示全集，而其它任何数据结构都不能。

k和m相同，使用同一组散列函数的两个布隆过滤器的交并差运算可以使用位操作进行。

但是布隆过滤器的缺点和优点一样明显。误算率是其中之一。随着存入的元素数量增加，误算率随之增加。但是如果元素数量太少，则使用散列表足矣。

另外，一般情况下不能从布隆过滤器中删除元素. 我们很容易想到把位数组变成整数数组，每插入一个元素相应的计数器加1, 这样删除元素时将计数器减掉就可以了。然而要保证安全地删除元素并非如此简单。首先我们必须保证删除的元素的确在布隆过滤器里面. 这一点单凭这个过滤器是无法保证的。另外计数器回绕也会造成问题。

#### Bloom过滤器Hash表大小选择

我们现在知道了Bloom过滤器需要一个表来存储字典，那么这个表应该多大呢？

按照通常我们使用Hash表的经验，应该选择比需要存储个数更大的表，但是经过尝试，在选区长度为10万的时候，已经获得了非常好的表现。在普通台式机进行测试后发现，在使用了三个Hash函数的情况下，已经获得了97%以上的正确率，已经非常令人满意了。

#### Hash算法选择

在对6个号称“世界上最好的Hash函数”经过交叉实验后，我选取了如下三个函数

// 1

// 从Robert Sedgwicks的 Algorithms in C一书中得到了。

// 我(原文作者)已经添加了一些简单的优化的算法，以加快其散列过程。

public long RSHash(String str) {

int b = 378551;

int a = 63689;

long hash = 0;

for (int i = 0; i < str.length(); i++) {

hash = hash \* a + str.charAt(i);

a = a \* b;

}

return hash;

}

// 4

// 这个算法是Daniel J.Bernstein 教授发明的，是目前公布的最有效的哈希函数。

public long DJBHash(String str) {

long hash = 5381;

for (int i = 0; i < str.length(); i++) {

hash = ((hash << 5) + hash) + str.charAt(i);

}

return hash;

}

// 5

// 由伟大的Knuth在《编程的艺术 第三卷》的第六章排序和搜索中给出。

public long DEKHash(String str) {

long hash = str.length();

for (int i = 0; i < str.length(); i++) {

hash = ((hash << 5) ^ (hash >> 27)) ^ str.charAt(i);

}

return hash;

}

#### 其他细节

为了获得更好的效率，分词引擎还应该去除标点符号以及停词（Stopword），并且英文不应该被逐字分开。

### 日期时间与地点分析识别模块

时间日期分析采用正则表达式匹配与特殊情况处理结合的方式完成，通过不断增加的样本库进行迭代开发。

对于匹配时间是过去时间，则自动延后。因为不可能通知过去时间开会的。

对于歧义问题，如今晚12:30，实际上应该是第二天，参照上条处理。

实际开发中遇到此类问题总原则是，根据人们习惯以及第一映像来理解。

具体实现代码过于复杂，故不在此说明。

### 事件记录处理与提醒模块

经过上述模块处理的数据，在得到用户认可后，自动格式化存储到系统日历中。

event.put(Events.CALENDAR\_ID, calId);

Calendar mCalendar = Calendar.getInstance();

mCalendar.set(Calendar.HOUR\_OF\_DAY, 10);

long start = time.getTimeInMillis();

mCalendar.set(Calendar.HOUR\_OF\_DAY, 11);

long end = time.getTimeInMillis() + 3600000;

event.put(Events.DTSTART, start);

event.put(Events.DTEND, end);

TimeZone tz = TimeZone.getDefault();

event.put(Events.EVENT\_TIMEZONE, tz.getID());

event.put(Events.HAS\_ALARM, 1);

Uri newEvent = getContentResolver().insert(

Events.CONTENT\_URI, event);

long eventID = Long.parseLong(newEvent.getLastPathSegment());

ContentValues values = new ContentValues();

values.put(Reminders.MINUTES, 10);

values.put(Reminders.EVENT\_ID, eventID);

values.put(Reminders.METHOD, Reminders.METHOD\_ALERT);

Uri uri = getContentResolver().insert(Reminders.CONTENT\_URI, values);

Toast.makeText(DialogActivity.this, "添加提醒成功!!!", Toast.LENGTH\_SHORT)

.show();

### 用户数据存储与备份还原模块

用户存储在数据库中的数据将随着卸载应用或者手机更换等情况而丢失。为了避免这些情况，需要将数据库中的资料备份到SD卡上，并可以随时还原。

// 获得正在使用的数据库路径

// 获取SD卡下的用

// Environment.getExternalStorageDirectory().getAbsolutePath()+"\*.db."

// 默认路径是 /data/data/(包名)/databases/\*.db3

File dbFile = mContext.getDatabasePath("/data/data/"+packagename+"/databases/user.db3");

File exportDir = new File(Environment.getExternalStorageDirectory(),

"/findix/Backup");

if (!exportDir.exists()) {

exportDir.mkdirs();

}

File backup = new File(exportDir, dbFile.getName());

String command = params[0];

System.out.println(backup.getAbsolutePath());

if (command.equals(COMMAND\_BACKUP)) {

try {

backup.createNewFile();

fileCopy(dbFile, backup);

return Log.d("backup", "ok");

} catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

return Log.d("backup", "fail");

}

} else if (command.equals(COMMAND\_RESTORE)) {

try {

fileCopy(backup, dbFile);

return Log.d("restore", "success");

} catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

return Log.d("restore", "fail");

}

} else {

return null;

}

### 事件日历查询模块

系统日历只可以按顺序查看而无法查询，所以该模块需要允许用户输入任意字符，并实时获得反馈。匹配原则是事件名称和地点。

UI使用Adapter模式实现。