手机安全卫士

摘 要

手机安全卫士属于系统安全方面的工具类软件。通过监控手机状态让用户可以清晰的掌握手机的实时状况，用户可通过后台进程清理、缓存清理等优化操作，达到提升手机运行速度、降低电耗等功效。另一方面对手机病毒查杀检测手机是否中毒，通过应用锁来保护个人隐私，通过管理黑名单来拦截垃圾短信或骚扰电话，全方位保障手机安全。我选择该课题是因为，此类型应用涉及到了Android的各个领域，覆盖面广、功能针对性强，而且应用人群广泛，处理效果显著，对手机安全，尤其对隐私数据的保护做到了绝对的保障。目前国内各大知名手机厂商都在自家定制系统中集成了相关功能。可见此系统的重要性。

本课题项目采用Android4.4为核心开发平台，最低兼容Android4.0，由Android客户端，Web Server和后台DB组成了一套完整的手机安全系统。开发的核心要点是对系统的体系结构进行划分，本项目采用软件工程思想，结构化分析方法分解项目模块，并严格按照软件开发生命周期步骤进行开发，系统最终实现了手机防盗、骚扰拦截、应用和进程管理、流量管理、病毒查杀、缓存清理、高级工具和贴吧等核心功能。用户最关心的手机防盗的核心是实现了对已丢失手机的远程控制，让小偷无懈可击；而病毒查杀采用的是百度统计的最新应用病毒库，囊括种类丰富，覆盖领域全面，病毒无处可藏。

本设计实现了上述提及的九大核心模块，基本满足了手机安全卫士的所有需求。课题项目使用了较多三方开源框架，内容较多，本文仅对项目核心功能的实现以及相关内容进行阐述。

关键词：Android，Servlet，Web Server，DB，软件工程，结构化分析

MOBILE SECURITY GUARD

ABSTRACT

Mobile security guards belong to the system security aspects of the tool class

**KEY WORDS：**android operate system，servlet，web server，database，software engineering，structural analysis

目　录

[前　言 1](#_Toc482470742)

[第1章 系统概述 2](#_Toc482470743)

[§1.1 开发背景 2](#_Toc482470744)

[§1.1.1 Android系统的发展史 2](#_Toc482470745)

[§1.1.2 Android系统的发展前景 4](#_Toc482470746)

[§1.1.3 安全卫士与Android系统结合的优势 5](#_Toc482470747)

[§1.2 国内外研究现状 5](#_Toc482470748)

[§1.2.1 国内的研究现状 5](#_Toc482470749)

[§1.2.2 国外的研究现状 6](#_Toc482470750)

[§1.2.3 结合相关应用分析手机卫士的优势 7](#_Toc482470751)

[§1.3 系统创新点 8](#_Toc482470752)

[§1.3.1 提供给用户两种优化手机的方式 8](#_Toc482470753)

[§1.3.2 分模块的用户操作接口 8](#_Toc482470754)

[§1.3.3 实时性的信息交互 8](#_Toc482470755)

[§1.4 系统意义 8](#_Toc482470756)

[§1.4.1 对用户的意义 9](#_Toc482470757)

[§1.4.2 对网络安全的意义 9](#_Toc482470758)

[第2章 系统需求分析 10](#_Toc482470759)

[§2.1 系统可行性研究 10](#_Toc482470760)

[§2.1.1 技术可行性 10](#_Toc482470761)

[§2.1.2 经济可行性 10](#_Toc482470762)

[§2.1.3 操作可行性 10](#_Toc482470763)

[§2.2 系统需求分析及要点 10](#_Toc482470764)

[§2.2.1 应用的推广需要广大用户的参与 11](#_Toc482470765)

[§2.2.2 简洁的UI和通信功能是该应用的创新突破 12](#_Toc482470766)

[§2.3系统功能结构分析 12](#_Toc482470767)

[§2.4系统数据库的分析 13](#_Toc482470768)

[§2.5 系统关键技术 13](#_Toc482470769)

[§2.5.1 系统开发技术 13](#_Toc482470770)

[§2.5.2 系统开发环境 14](#_Toc482470771)

[第3章 系统总体设计 16](#_Toc482470772)

[§3.1 系统设计目标 16](#_Toc482470773)

[§3.2 系统设计原则 16](#_Toc482470774)

[§3.3 系统框架结构 16](#_Toc482470775)

[§3.4 具体功能模块设计 17](#_Toc482470776)

[§3.4.1 检测新版本流程 17](#_Toc482470777)

[§3.4.2 手机防盗设置流程 18](#_Toc482470778)

[§3.4.3 骚扰拦截流程 19](#_Toc482470779)

[§3.4.4 软件管理 20](#_Toc482470780)

[§3.4.5 进程管理 20](#_Toc482470781)

[§3.4.6 流量统计 20](#_Toc482470782)

[§3.4.7 手机杀毒 21](#_Toc482470783)

[§3.4.8 缓存清理 22](#_Toc482470784)

[§3.4.9 常用工具 22](#_Toc482470785)

[§3.4.10 注册登录流程 24](#_Toc482470786)

[§3.4.11 贴吧相关操作 24](#_Toc482470787)

[第4章 数据库设计 25](#_Toc482470788)

[§4.1 概念结构设计 25](#_Toc482470789)

[§4.1.1 具体E-R图 25](#_Toc482470790)

[§4.2 逻辑结构设计 29](#_Toc482470791)

[§4.3 物理结构设计 30](#_Toc482470792)

[§4.3.1 数据表实现 30](#_Toc482470793)

[§4.3.1 数据相关事务实现 34](#_Toc482470794)

[第5章 详细设计与实现 36](#_Toc482470795)

[§5.1 应用框架 36](#_Toc482470796)

[§5.1.1 UI框架选型 36](#_Toc482470797)

[§5.1.2 其它框架选型 36](#_Toc482470798)

[§5.2 模块功能的实现 37](#_Toc482470799)

[§5.2.1 应用主界面实现 38](#_Toc482470800)

[§5.2.2 指令短信的处理 38](#_Toc482470801)

[§5.2.3 骚扰拦截处理 39](#_Toc482470802)

[§5.2.4 获取应用详细信息 40](#_Toc482470803)

[§5.2.5 获取运行进程详情信息以及处理过程 41](#_Toc482470804)

[§5.2.6 缓存清理处理 43](#_Toc482470805)

[§5.2.7 短信备份还原 44](#_Toc482470806)

[§5.2.8 应用加锁解锁 45](#_Toc482470807)

[§5.2.9 贴吧模块 47](#_Toc482470808)

[第6章 系统测试 49](#_Toc482470809)

[§6.1 功能性测试 49](#_Toc482470810)

[§6.2 性能测试 49](#_Toc482470811)

[§6.3 压力测试 50](#_Toc482470812)

[§6.4 冒烟测试 50](#_Toc482470813)

[结 论 52](#_Toc482470814)

[参考文献 53](#_Toc482470815)

[致 谢 54](#_Toc482470816)

# 前　言

在这智能移动设备飞速发展的时代，与日俱增的用户最贱接触使用到各样类型的移动智能设备。但同时迎来了越来越多各式各样的恶意网络攻击，因此保护个人设备的安全愈加的重要。智能手机被频繁用于电信通信、网络通信、甚至作为公司的数据传输和存储的载体。一旦手机被恶意病毒程序入侵，会导致隐私数据泄漏，可能造成财产上的损失。所以为了预防和避免这种意外情况，解决手持移动设备安全相关问题的产品应运而生。

我选择手机安全卫士这个题目是因为，这种类型的应用，几乎涉及到了安卓系统相关的各个领域，覆盖面广、功能针对性强，而且应用人群广泛，处理效果显著，目前国内各大知名手机厂商，都在自家的定制系统中集成了相关的功能。本课题研究的项目能够实现对用户数据的安全保障，骚扰信息等意外情况的保障和对提升手机运行效率提升的保障。

本课题以Android4.4为核心开发平台，最低兼容到Android4.0，由Android客户端，Tomcat Web服务端和MySQL后台数据库组成一个完整的手机安全卫士的胸膛，课题采用软件工程的思想，结构化分析的方法来分解项目。其中主要解决了手机防盗、骚扰电话和垃圾短信的拦截、软件和进程的管理、手机流量监控、病毒应用查杀、应用缓存清理、常用高级工具等问题。

目前国内关于这方面的应用已经相当成熟，从最初的360，到如今的各大手机安全卫士，这些产品都有体检、杀毒、备份、防盗、流量、拦截、软件管理等功能。对比这些知名产品的特色功能，综合分析，只能说，它们各自有各自针对的应用场景。而对比国外的同类型产品，它们更偏向于对硬件的保护，国外的同类型产品像ESET Mobile Security、**BullGuard Mobile Security，它们对手机应用的内存分配、数据碎片化整合方面做了充足的优化。对于国内产品而言，存在的主要问题是对未知问题的处理，以及模块功能扩充方面的不合理性。比如有些模块最后扩充到这个模块完全可以独立出来成为一个新的应用的程度；而对于手机出现的没见过的问题，多多少少会有一些，对于这些目前系统无法处理的问题，单单是通过用户反馈还不够，所以要提供一个能够让用户进行交流讨论的地方，说不定有人就遇到过这些问题。**

# 第1章 系统概述

## §1.1 开发背景

随着移动智能设备逐渐步入大众的生活，人们获取信息的主要途径、获取设备也随之改变。但随之而来的是与日俱增的网络攻击，保护个人设备安全愈加的重要。因此，手机安全服务类应用将会非常可观的扩展空间，安全卫士必然是其中一员。另一方面，由于Android平台的开放性导致它能够提供给任何用户快速便捷的数据查阅功能，这给安全卫士的产品设计与实现提供了绝佳的服务于支持。

Android是一套货真价实的应用在移动设备上的开发性综合服务平台，它包括开源的操作系统、中间过渡驱动以及应用层的平台性应用。Android是以Linux为内核使用Java语言进行封装的开源嵌入式系统，允许任何厂商和个人在其基础上进行改造。Android平台的开放性等特点既促进了技术（包括平台本身）的创新，又有助于降低开发成本。运营商能可以根据自己的需求定制专属个性化系统。因此，它的发展潜力绝对是空前的。

进一步讲，Android平台的绝大部分系统级支持库也是开源免费的，这实现了Android应用多元化。在节约成本的同时，也让开发者能够轻松地将Android应用到各个领域，也为手机安全卫士服务的普及做出贡献。

### §1.1.1 Android系统的发展史

安卓系统最初并不是应用在手机、平板这种移动设备上的，最初它是提供早期的数码相机智能化管理的一套解决方案，那时的Android系统还没有里程碑版本，但随着网络技术的革新与时代变迁的需求，最终安卓系统演化成了主要用于手机平板上的系统。

Android这一词语起源自近代法国文学家利尔亚当在1886年发表的科幻小说《未来夏娃》，作品中的外表同人类相似的机器被作者命名为Android， Android机器人这一词语由此流传下来。

Android系统起初是由以安迪鲁宾为首以及他的团队共同开发的一套用于解决数码相机智能化的解决方案，他的公司名就叫Android。从此Android系统开启了它五彩斑斓的生命里程。

由于数码相机在当时并不被市场看好，Android公司那时发展的并不景气，但那时圈内对嵌入式设备的研究热情确实空前的高涨，于是在2005年8月17日，谷歌收购了刚刚成立的高科技企业Android公司和安迪鲁宾的团队。安迪被任命为谷歌公司工程部副总裁，继续开发Android项目。经过两年的努力，谷歌于07年11月5日，正式对外公开全新的嵌入式操作系统Android。当天谷歌正式建立了世界上第一个全球开放性手持设备联盟（Open Handset Alliance）来共同研究完善Android OS，该联盟会支持谷歌发布的嵌入式操作系统以及相关软件，而后谷歌获得了阿帕奇免费开源许可证的授权，并正式发布了Android的源代码。

2008年的谷歌I/O大会上，谷歌首次提出了Android HAL架构图，同年8月18号，安卓顺利获得了美国联邦通信委员会的批准，同年9月，谷歌正式发布了Android 1.0系统，这就是Android系统最早的版本。

2009年4月，谷歌继续推出新版本Android 1.5，从1.5版本开始，由于版权问题，谷歌决定使用甜品的名字对Android的版本进行命名，而Android 1.5被命名为Cupcake（纸杯蛋糕）。

2009年9月,Android 1.6正式发布，同时推出了搭载Android 1.6正式版的手机HTC Hero（G3），G3凭借着其卓群的外观和新概念式的Android 1.6操作系统，当仁不让的成为当时全球最受欢迎的手机。Android 1.6的版本代号是Donut（甜甜圈）。

2010年2月份，Linux的驱动内核发生了重大改革，Android的驱动程序被Linux内核开发者Greg Kroah-Hartman从Linux内核“状态树”（“staging tree”）中去除，从此，Android与Linux开发主流将分道扬镳。在同年5月份，谷歌正式发布了Android 2.2操作系统。其版本代号为Froyo（冻酸奶）。

2010年12月份，谷歌正式发布了Android 2.3版本，并宣称Android 已经达到了第一个里程碑，因为在10月份，应用市场上Android APP获得Official digital certificate的数量已经超过了10万，由此可见，Android APP增长率非常之高，发展异常迅猛。Android 2.3的版本代号是Gingerbread （姜饼）。

2011年1月，谷歌宣称称Android设备日新用户数量已经达到了30万部，到2011年7月，这个数字一路增长到55万部，而使用Android设备的用户总数达到了1.35亿，Android OS已经成为当时智能手机领域占有量最高的系统。截止到同年8月2日，Android手机在全球智能机市场的占有率达到了48%，并在亚太地区市场占据统治地位，终结了塞班系统的霸主地位，跃居全球第一。同年，有大量Java ME开发者转战安卓阵营。

2011年9月份，谷歌发布了全新的Android4.0操作系统，其系统版本代号为Ice Cream Sandwich（冰激凌三明治）。4.0版本加入了许多新特性，对系统进行了深度的优化，屏蔽了一些系统操作，用户体验方面做得真是细致入微。

2012年1月6日，谷歌应用商店已有近10万开发者推出超过40万活跃应用，大多数应用免费。谷歌应用商店目录在新年首周周末突破40万，距离突破30万应用仅4个月。

2013年11月1日，谷歌正式发布了Android4.4，从具体功能上讲，Android4.4提供了各种实用小功能，更多的Emoji表情图案，UI方面的改进也与时俱进，如状态栏的半透明效果，弹簧列表等效果。尤其是各种小控件的功能令人为之振奋。从开发者的角度，13年的谷歌I/O大会上提出了一些新的开发标准，给出了更多的实用性API，大大简化了开发周期，提升了开发质量。Android4.4的版本代号是KitKat，是一个谷歌和雀巢合作的版本。

2014年10月15日, 谷歌正式发布全新操作系统Android 5.0， 5.0在UI表现上采用了独创设计风格Material Design。这种设计的初衷在于统一 Android 设备的外观和使用体验，不管使用哪种Android设备，都感觉用的是同一设备。用户可以自行选择沉浸式、流式等风格，这款系统被谷歌命名为Lollipop(棒棒糖)

Android 6.0版本，即盛传已久的Android M在2015年的 谷歌I/O大会上正式发布。在发布会上代号为Marshmallow(棉花糖),Android M最大的一个亮点是：系统提供给用户两套相互独立的解决方案。一套专门用来储存用户的工作资料，另一套专门用来存储用户的个人信息。并且这两套系统完全相互独立。当然用户也可以DIY这两套解决方案，使用方面更加细致，用户可控性也做了十足的优化。

### §1.1.2 Android系统的发展前景

安卓系统的几大优势：

1. 层次性强，低耦合

Android与Linux并不完全等同，只能说是Android基于Linux的，它的内核是一个Linux OS，但Android并不是Linux系统，而是对Linux的二次封装，相当于它使用了Linux提供的一个framework，但对应用层的表现又是另一种简单易懂的形式。在此基础上，开发者可以轻松的开发各种应用程序，开发者不需要去关注framework层的东西，仅需掌握应用层的开发方法即可。

1. 系统开源，利于创新

苹果的闭源导致的局限性，使其创新必须源自内部。而Android对开发者能用到的部分都是开源的，开发者随意修改，这对于许多手机厂家来说是一大福音，因为他们可以根据自己的硬件配置情况对驱动做出调整，进而更好的适应硬件，从而与系统形成良好的结合，打造出厂家独特的安卓手机。

1. 开发的难易程度

Android开发一般说的是应用层的开发，也就是移动客户端的开发。安卓系统的四层框架中，应用层的开发语言是Java，而Java是最容易上手的，不像 ObjectC，C#，VB这种，需要熟悉他的很多框架。

### §1.1.3 安全卫士与Android系统结合的优势

Android作为一款具有很大的潜在用户的智能手机操作系统，其前景自是一片大好，而现在人们每天大部分时间不管是上网还是通讯都是在与手机打交道，因此对手机数据的保护刻不容缓，安全卫士正是完全符合这种需求，而又缺乏的资源，它几乎面向所有群体，因为现在从小学生到看门的大爷，都在使用智能手机了。

因此，安全卫士与安卓系统的结合，可以借助安卓这一平台，既保障了手机的安全，又提高了用户保护个人信息的意识。

## §1.2 国内外研究现状

### §1.2.1 国内的研究现状

目前国内的研究中，在信息安全这一方面，用到的信息技术有图像变形技术、遥感技术、远程控制技术、生物信息识别技术等，回首我国二十多年的移动通讯发展过程，在手机市场的发展速度和发展规模方面无不令世人瞩目，我国的手机行业的发展又是大批量、超常规、跳跃式的。日前，中国信息通信研究院发布预测称，2017年，我国4G用户数将有望超过10亿户，2021年我国4G用户在总移动用户中的渗透率将接近85%，其中我国市场将以TD-LTE为主，TD-LTE用户规模占4G规模的比例保持在70%以上，2021年有望达到8.7亿户。目前,国内手机卫士类的产品已向多元化方向发展，各大知名软件公司都有自己的卫士产品，功能上各有各的偏好，但其基本功能和解决的问题基本一致，即：

1. 手机防盗功能，即手机丢失后能够远程控制自己的手机
2. 对骚扰电话和垃圾短信的拦截，预防意外情况的发生
3. 实时监控手机流量，通知用户恶意扣费的应用
4. 对手机进行杀毒，防止用户隐私泄露
5. 清理手机应用的缓存与后台耗电的进程，提升手机的性能
6. 提供常用工具方便用户使用手机，如应用锁、公共号码、归属地查询等等

目前许多手机恶意程序，如4CallLog、**Spycall**等等，只要他们被安装在智能电话中，就会自行记录这个手机收发的所有短信、联系人信息以及详细的通话记录，接着在其内部将这些数据进行压缩打包，最后通过在手机联网的情况下自动上传到指定的服务器。而控制后台的黑客可以通过这些个人隐私信息，在受害者毫不知情的情况下做一些违法的事情，造成当事人财产上的损失。去年7月份，国家计算机病毒应急处理中心截获了最新款的变种手机病毒“frame share”，此病毒已感染用户超过170万。

另一方面，这种病毒应用不单单是侵犯个人隐私，同时也严重破坏了用户的合法经济利益。因此，为手机的安全，维护个人信息安全，手机卫士这种系统安全类软件随之迅速发展起来。时至今日市场上发展较好的比较出名的卫士软件有奇虎的360手机安全卫士，King Soft的手机卫士，Tencent的软件管家，Lenovo的乐安全等。

### §1.2.2 国外的研究现状

在西方国家，尤其是欧美，在信息安全方面发展时间较长，这方面的研究经验也是十分成熟，其信息安全技术圈已形成了一定的规模。安全卫士这种类型的产品更是不言而喻。

国外做系统安全类产品的公司像 AVG**、**Kaspersky、ESET**、BullGuard 这些在世界上很知名的，也都有各自的基于移动设备上的产品，同样是各自有各自针对性的解决方案。不过国外的情况和国内不同，国外产品的普遍的特点是： 在移动设备的硬件方面做了非常细致的安全管控处理并提供给用户十分人性化的调整存储空间的接口，用户甚至可以 控制手机的电量，分配内存区域，比如BullGuard的Battery Release 功能，它会在后台执行耗电操作，以此来快速放电。而国内的手机安全软件不可能出现这种功能，国内的做的都是怎样给手机省电的功能。另一个普遍性特点是：可以对不同类型的用户授予不同的操作权限，国外的有些品牌的手机在开机时，是可以选择用户的，就像PC一样，可以创建多个用户使用同一个系统，比如早期的BlackBerry（不过黑莓现在用的都是安卓系统了），总之就是国外的手机安全类应用，在开发的时候，是参考了PC客户端来开发的。**

### §1.2.3 结合相关应用分析手机卫士的优势

从系统的发展来说，Android无疑是占据着绝对的优势的。而本应用在Android系统上的推广和其他与手机安全相关的应用，比如驱动精灵、鲁大师、蓝盾、安卓小歪（国外软件）等相比则体现出了优势。从以下几方面来分析手机卫士相较于这些应用的优势：

1. 作用范围

像鲁大师这种以主打硬件的产品来说，手机卫士的功能可以算是附加功能，他们的用途已经远超出手机卫士应有的范围，用户既可以检测手机硬件，又能在安全方面做出一些列调整，但是很明显，这些应用的针对性不强，它们集成的手机卫士的功能也有所缺失，并不能像专门的手机卫士那样，做到确实能够对手机安全起到一定的保障作用。

1. 数据流通性

在相关后台支持数据库方面，这些应用的后台支持库是集成在本地的，也就是随着版本升级一起安装在手机中的，它的更新需要随着版本更新而更新，并不是实时的。但手机卫士的相关支持库是实时更新的，无需更新版本，就能使用最新的支持库，比如后台的病毒库，后台的黑名单库等等。这些支持库都是放在服务端的后台数据库中的。

本系统使用的模式是既有本地支持库，又用到了服务端的后台支持库这样在一定程度上能够缓解服务器的压力。

## §1.3 系统创新点

### §1.3.1 提供给用户两种优化手机的方式

对于不同方面的优化处理，手机安全卫士分出来各个不同的处理模块，用户能够根据自己手机的实际情况，选择进行处理哪些问题，优化哪些方面，这些提供出来的模块，专门为用户提供这种需求；还有些用户的手机，可能用得比较久了或者其他原因，可能使用一段时间，就需要对手机做一次全方位的清理优化，这时就提供了这个一键检测、一键修复的功能。实际上就是同时开启多个线程，每个线程执行一个优化任务。给用户的感觉就是在同时检测，同时优化手机。

### §1.3.2 分模块的用户操作接口

手机安全卫士提供给用户不同的操作模块，用户不仅能够对手机进行优化操作，还能够查询修改自己的信息，或者查看其他用户的简单信息。提供给用户一个可选项，在不同的模块执行自己需要的操作。

### §1.3.3 实时性的信息交互

在使用手机卫士时，用户不再只是一个个体，系统会将相关用户关联起来形成一个关系群体，在群体中用户可以互相交流讨论，说说自己遇到的问题，还可以为提问者提供些个人建议或者自己遇到相同问题时的经验等等，解决了问题的用户可以为帮助自己解决问题的用户打赏。

这个模块相当于提供了一个类似贴吧的服务，提供给用户一个专门用于交流讨论的地方。

## §1.4 系统意义

### §1.4.1 对用户的意义

对用户来说，手机安全卫士扮演了一个PC无法替代的角色，就是帮助用户对用户的隐私做出最大的保障，对用户的手机安全做出最大的保障，对用户的手机状况及相关各方面的问题做出最大的保障。

人们不会随时随地带着自己的个人电脑，但是却时时刻刻拿着自己的手机。当你在公交车上聊微信时，当你在地铁站看购物车时，当你在公司发消息时，这一切的一切都在进行着数据的传输。因此对手机的安全防护是必要的。

作为手机上的安全卫士，它能够满足用户的一切有关手机安全和手机优化方面的需求，应该能够对后台应用、后台进程、应用缓存的清理，对垃圾短信和骚扰电话的拦截，对重要的应用加锁，对丢失的手机远程控制，对中毒的手机一键杀毒等基本操作。

该应用还提供的有登录和分享的功能，可以让用户在遇到未知问题而且系统目前无法解决的时候进行交流，通过贴吧的形式，实现用户之间的通讯。

### §1.4.2 对网络安全的意义

通过手机卫士对个人信息进行保护，从而使得个人隐私信息不再暴露出去，网络中的信息相关信息也就没有了，对整个网络来说，即清洁了网络，又让一些想要获取他人隐私数据的不法分子失去了机会，这样一举三得的事情是非常有必要做的。

# 第2章 系统需求分析

## §2.1 系统可行性研究

### §2.1.1 技术可行性

选择基于Android OS的开发平台，以Java为主要开发语言进行开发。使用J2EE版的Eclipse作为开发Android客户端应用以及后台服务端的IDE。使用Android移动开发相关技术开发出用户界面良好，功能完善，操作简洁，运行稳定的基于Android OS的手机安全卫士软件，使用Java Web相关技术开发出完善的后台操作支持。综上所述，基于Android的手机安全卫士软件在技术方面是完全可行的。

### §2.1.2 经济可行性

基于Android的安全卫士是一款移动终端应用，和PC端的同类产品相比，移动终端应用的优势在于：它可以方便、快速的随时随地获取信息，不受时间和地理的约束。因此市场开发空间较非常大，而且需求用户群庞大，容易推广，在开发方面的成本低廉，开发周期能达到上应用市场应用上架的周期的平均值。

### §2.1.3 操作可行性

基于Android的手机安全卫士，其开发风格采用目前主流的沉浸式风格，所有界面内容使用原生Android组件进行开发，开发模式采用适合于移动开发的MVP和MVVM对项目进行设计和实现。

基于以上的分析，该应用的开发是完全可行的，本项目整体综合运用Android原生API，以MVP和MVVM开发模式以及安卓主流的UI框架构建项目。

## §2.2 系统需求分析及要点

基于Android的手机安全卫士软件在4.0版本以上的Android系统设备上皆可运行，对于庞大的用户群体，必须完全满足用户的以下需求：

1．UI简约大方：时间是用钱换不来的，非常宝贵，用户不会浪费时间去玩懂一款从UI上就非常难以理解的应用。假如你的APP没有让人一目了然的UI界面，除了开发者，谁都不可能上手即用，不能做到即开即用那就说明你的产品在设计方面是有问题的，最终会流失大量的用户。所以本应用不仅要有简约的交互界面，还要有人性化的可控性设置操作，能够即开即用，还能够DIY设置，而不是需要专门去看一看应用的使用说明后，再去使用。

2．运行稳定：如果你的APP在核心支持框架的选型或设计方面都不够好不够完善的话，那么非常可能在APP运行时碰到一些导致应用崩溃而且无发预料的错误。倘若用户在使用的过程中，应用一会儿崩一下两会儿崩一下，用户从内心自然而然就会对应用产生反感，接下来就直接卸载应用，这也是用户流失的重要原因之一。本应用对应用稳定性的要求是通过最基本的冒烟测试。

3．操作简便：随着应用功能越来越多，越来越完善，在系统的操作上就变复杂了，但是这个复杂只能面向开发者而不是用户，如果用户在使用你的APP时都感到很复杂，找个啥都找半天的话，就会想这么复杂的应用我是用还是不用，最后也可能卸载应用。应用的可操作性也是衡量软件质量的标准之一。所有的细节都要从用户的角度来设计开发，以最大程度的满足用户需求。

### §2.2.1 应用的推广需要广大用户的参与

“应用做的再完美，没用户来用的话那也是永久的硬伤。”

基于这个道理，安全卫士与Android相结合，通过自己的宣传和努力，让那些手机出现过安全问题的朋友注册为用户参与进来，他们能够提供最为真实的一手手机安全问题信息、还能对以前用过的安全卫士类产品的功能、用户体验方面做比较，作为系统最具价值的信息的来源，来帮助开发者更好的完善卫士系统。

另一方面也要保证可以让没有什么经验的用户参与进来，在解决手机问题的同时，能够将自己的问题分享出去。

### §2.2.2 简洁的UI和通信功能是该应用的创新突破

本应用的UI体现出一种简约大方形象，功能模块一目了然，无需使用说明，用户一看就懂，完全不用担心不会用。

本应用嵌入了以贴吧的形式展现出来的交互模块，用于提供用户之间交流的一个平台，在遇到未知问题的时候，通过提问，能够获得他人的帮助，算是从另一方面解决了问题。相比于其他手机卫士，并没有提供此功能。

## §2.3系统功能结构分析

项目整体采用Android原生API，以MVP和MVVM开发模式和目前安卓主流的UI框架搭建项目，本项目提供一键检测和单独检测两套方案，让用户根据手机的实际情况做出针对性处理，以最为迅速的方式解决用户的问题。兼具功能的完备性。

主要功能如下：

1. 防盗功能：通过绑定安全号码，在不慎丢失手机的情况下，能够通过安全号码发送指令短信，来远程控制丢失的手机，能够远程锁屏、发送手机最新位置信息、播放报警铃声、远程销毁数据、保证数据不外露，不丢失。
2. 拦截功能：对标记的黑名单的拦截类型进行短信或者电话实施拦截操作，还用户清净。
3. 软件管理功能：对当前手机系统已安装的所有应用，进行管理，允许用户在此对列出的应用进行，详情查看、打开、卸载货分享应用等操作。
4. 进程管理功能；对系统当前正在运行的所有进程，进行管理，用户在这里可以对列出的所有运行时进程进行清理操作，同时能够开启锁屏清理进程等服务。
5. 流量统计功能：对所有已安装的应用进行流量的统计，让用户清晰的看出每个月哪个应用上传使用了多少流量，下载使用了多少流量，并在流量超过预警的情况下禁止该应用使用流量。
6. 手机杀毒功能：以最新病毒库为基础，对系统内的所有应用进行扫描比对，检测是否有病毒应用潜伏在系统内，如检测到，提示用户进行杀毒。
7. 缓存清理功能：扫描系统内所有应用，检测每个应用的缓存并统计大小，最后提示用户清理缓存，清理缓存，能有效提升手机运行速度，但是缓存过的内容需要重新缓存。
8. 常用工具：这个模块提供了号码归属地查询、公共号码查询、短信备份还原、应用程序锁等功能。
9. 贴吧讨论功能：这个模块专门为注册用户提供了一个交流的平台，注册用户可以在此互相交流、沟通，是应用的一大亮点。

## §2.4系统数据库的分析

依据此应用的功能分析和操作流程的分析，该应用主要涉及的实体数据应该包括：

1. 病毒库：

病毒应用的签名信息、病毒应用的包名信息（Android应用的签名信息是唯一的，但是包名信息有可能重复）。

1. 骚扰拦截库：

骚扰电话号码、拦截类型。

1. 应用锁库：

应用的包名信息，应用的签名信息。

1. 公共号码库：

公共号码类型，公共号码。

1. 归属地号码库：

归属地区号、地区标识位、归属地信息。

1. 贴吧相关库：

这部分涉及到用户信息，发帖信息评论等多种信息，比较复杂，放到详细设计备份说明。

## §2.5 系统关键技术

### §2.5.1 系统开发技术

众所周知，Android主要是应用在智能移动设备上的嵌入式操作系统，系统搭载arm架构的CPU，时刻都体现了与其分布式的特性。鉴于这方面的硬件差异以及CPU架构差异，致使Android OS与Windows这种PC上的操作系统的差异非常大。所以只有充分了解Android系统特性才能开发出适合应用在移动端的产品。下面对本系统采用的开放平台版本Android4.4的主要特性进行简要的总结：系统采用针对移动设备改良后的ART虚拟机；集成了改造开源Web kit引擎的轻量级浏览器；搭载Custom-2D图形库与基于OpenGL 的ES 3.0库，同时提供多种硬件加速可供选择；继续使用超轻量级关系型数据库SQLite；同时支持大多数主流MIME，如JPG、PNG、GIF、MP3、MP4、AVI、WMA、RMVB，通信方面支持GSM电话；还有其他的4.4的主流硬件标配：3G、4G、Wi-Fi、Bluetooth、GPS定位、GPRS-EDGE、感应加速器。

### §2.5.2 系统开发环境

该应用使用的集成开发工具是目前流行的Eclipse，使用最新的Android SDK和Android开发插件ADT在其上配置Android开发环境。同时，Android OS内嵌的DB是超轻量级微型数据库SQLite，占用的资源非常低，在嵌入式的设备中，仅需几百KB的内存就足够了，同时该数据库还兼容多种开发语言相，其中就有本应用开发所应用的Java开发语言，比起SQL Server、Oracle这些企业级的DBMS来说，在响应速度方面没有比SQLite更快的，因此本系统在移动终端采用SQLite数据库存储用户添加更新的内容，而在服务端则

采用小型数据库MySQL作为后台支持。

表2-1 系统开发环境

|  |  |
| --- | --- |
| 操作系统 | Windows操作系统 |
| 硬件要求 | 至少512MB内存，30GB硬盘 |
| 支持工具 | JDK、Android SDK、ADT 2.3.0.7 |
| 开发平台 | EE 版Eclipse |
| 数据库 | SQLite、MySQL |
| 开发语言 | Java语言 |

JDK（Java Development Kit）是一个提供Java开发环境的支持工具包，是由SUN公司开放的开源免费的API，可用于开发所有Java相关的产品。早期的JDK中有少量特性应用在商业领域，这部分API是闭源的。为实现Java的扩展自由化， SUN公司于2007年发布了基于GPL协议的开源JDK，从此Java彻底走向自由。JDK 可以说是Java的心脏，它包含JRE（Java Runtime Environment）、而JRE又包含了开发用到的基础类库以及各种测试

、编译、修复等以批处理命令的形式存在的工具以及JVM（Java Virtual Machine），JVM是运行Java程序的虚拟机，所有Java程序最终都要运行在与系统匹配的Java虚拟机上。而Java应用服务器的内核就是一个具体版本的JDK。

Android SDK（Android Software Develop Tookit）安卓软件开发包是由谷歌提供的专用于安卓开发的工具包，同样也是开源免费的。和JDK的作用一样，提供一个支持安卓平台的开发环境。不过由于安卓本身是基于Java语言的，在使用Android SDK之前，必须先配置好Java开发环境，也就是说要先配置好JDK。

# 第3章 系统总体设计

## §3.1 系统设计目标

从软件工程的角度出发，软件生命周期的起点是对产品的需求分析，目的是为了明确用户的需求，从而确定系统要实现何种功能，怎样划分模块等。本系统的设计目标是：设计出一款界面友好、操作简单、功能全面、运行稳定的基于Android OS的安全卫士移动客户端。

## §3.2 系统设计原则

在所有Android机上的安全卫士类的产品，必须做到在使用上安全可靠，操作简便，功能强大。如果新功能不完善，即使不添加它，也不要添加一个可能存在问题的功能模块。模块化设计既提高了独立模块的高内聚，同时降低了不同模块之间的耦合度，自然的提高了系统扩展性，使得代码更加清晰明了，可读性增强，对模块更新扩展提供了相当大的便利。

## §3.3 系统框架结构

本项目的系统功能结构细节描述如图3-1所示，顶层由主界面构成，第二层由卫士模块、论坛模块、个人中心组成，其中卫士模块包含了对手机安全、手机性能等方面优化防护的各种功能模块。

项目根节点界面采用GitHub上的开源项目SlidingMenuLibrary作为核心框架，UI框架上用到了GitHub上的PullToRefreshListView、StickyListView、

CircleProgressBar、SpringListView等优秀的开源项目，而网络框架采用的是老牌的多功能集成框架xUtils，网络图片加载框架采用的是目前使用方面对内存开销最小的Glide。

主界面

设置中心

个人中心

贴吧模块

手机防盗

骚扰拦截

软件管理

进程管理

流量统计

手机杀毒

缓存清理

常用工具

卫士模块

图3-1 系统功能结构图

## §3.4 具体功能模块设计

经过系统的分析，该软件总体上是由一个主界面和主界面上的一些模块组成，其核心模块卫士模块应包含手机杀毒、手机防盗、进程清理、缓存清理、骚扰拦截、软件管理、流量统计、高级工具以及设置中心等九个模块。该软件可以对手机中的所有软件进行基本的管理，满足用户安全性和使用性的要求。

### §3.4.1 检测新版本流程

在应用欢迎界面，即第一次打开应用的第一个界面上，检测应用有没有新版本，执行流程如图3-2所示：

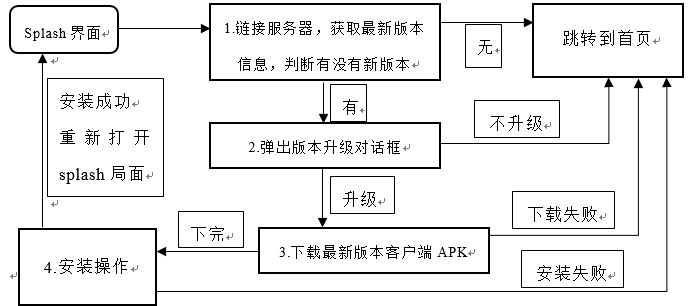


图3-2 版本检测流程图

### §3.4.2 手机防盗设置流程

手机防盗模块通过绑定的SIM卡判断手机是否被盗，如果发生手机被盗事件,手机将自动通过给安全号码发送报警短信，来通知手机被盗。手机防盗设置流程图如图3-3所示：

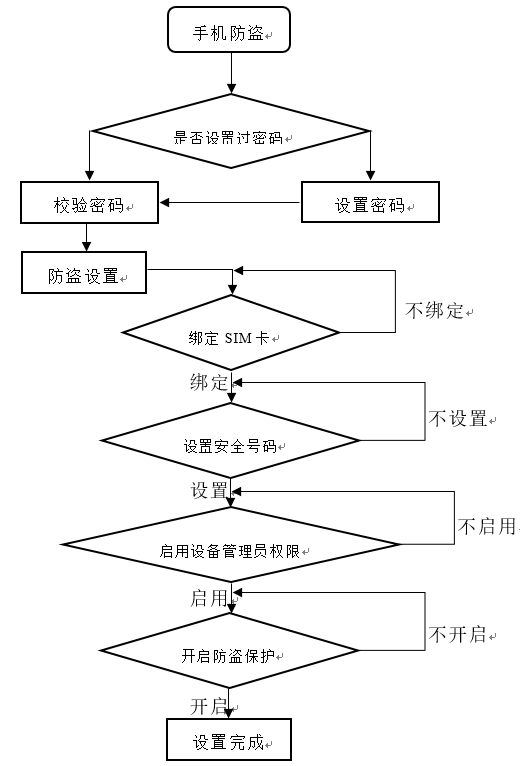


图3-3 手机防盗设置流程

手机防盗的核心是通过建立Android下的设备管理员功能，识别并解析安全号码发来的指令短信，从而远程控制已丢失的手机，可以实现给已丢失手机设置锁屏密码保护手机不被小偷乱搞，还可以控制格式化内置或外置内存卡，恢复出厂设置，具有超级管理员特权的应用，系统禁止卸载，小偷卸不掉安全卫士，能极大的保护用户的隐私安全。最重要的还能够远程定位丢失手机，获取其地理位置并发送到安全号码。短信指令控制流程图如图3-4所示：

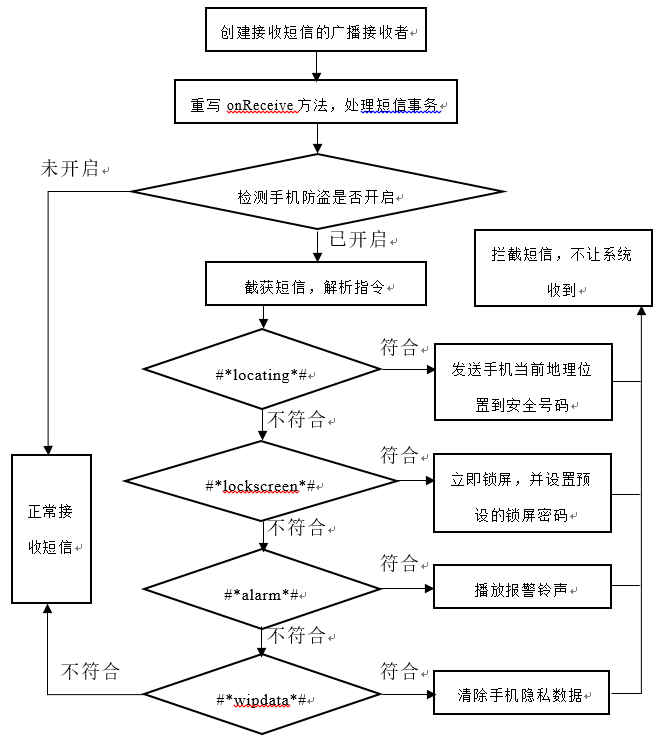


图3-4 解析指令短信流程

### §3.4.3 骚扰拦截流程

通过记录骚扰号码到黑名列表，实现下次再收到相同号码的电话或短信时进行拦截操作，其流程如图3-5所示：

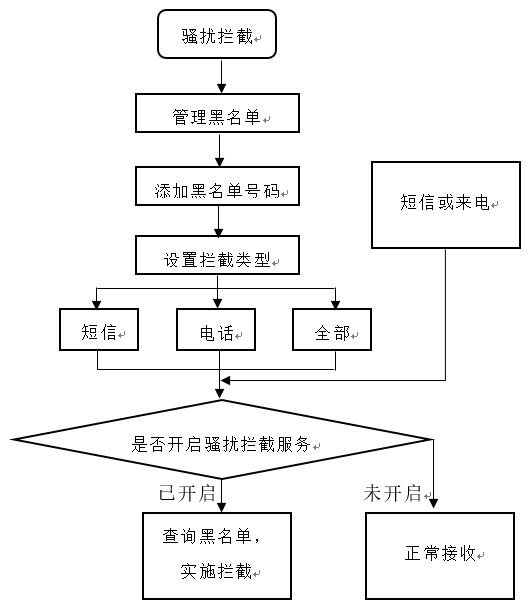


图3-5 骚扰拦截流程

### §3.4.4 软件管理

在该界面，用户能够看到自己的手机中安装的所有应用，包括系统应用，并能够对主观应用执行打开应用、打开应用详情、卸载应用、分享应用等操作，并能够查看，系统当前已使用了多少存储空间，SD卡当前已使用多少存储空间，还有多少空间可用等操作。

### §3.4.5 进程管理

允许用户在此界面对正在运行的进程进行清理操作，可用设置是否锁屏自动清理后台进程，是否自动区分耗电进程。

### §3.4.6 流量统计

该界面对手机系统中的所有应用的流量进行了详细的记录，可用看到每个应用在本月的流量使用情况，包括上传使用的流量和下载使用的流量。并在总流量超过预警值时，提醒用户流量即将用尽。

### §3.4.7 手机杀毒

在Android系统中，任何可执行程序都是以APK的形式安装到手机中执行的，手机病毒也不列外，其实它就是一个应用，只不过这个应用会对用户的隐私数据造成威胁。判断应用是不是病毒的实质是将应用的数字签名信息与病毒库中的记录做比较，如果相同的就视为病毒。而杀毒的过程就是卸载病毒应用，删除病毒应用创建的一切数据和配置文件。

具体实现需要先用PackageManager遍历手机上的所有程序，将扫描到的包名以及数字签名与数据库中的病毒应用的数字签名进行比对，发现相同的就视为病毒，并将病毒应用展示在列表首部提醒用户进行杀毒，完成后，返回手机杀毒的界面。手机杀毒流程图如图3-6所示：

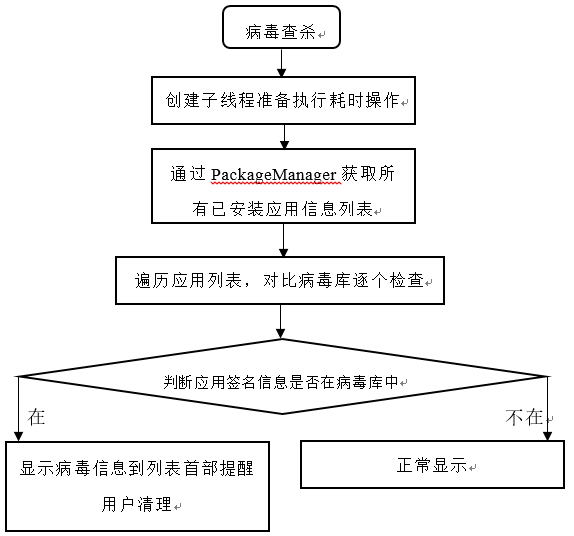


图3-6 手机杀毒流程图

### §3.4.8 缓存清理

和手机杀毒类似，都需要用到PackageManager这个包的管理者，来获取所有已安装应用的信息，通过aidl访问系统内部的方法，来实现对缓存的清理。安卓4.4的API本身并没有直接清理应用缓存的方法，所以只能另辟他径，通过系统aidl可以访问系统的一个方法，这个方法的作用是：重新申请缓存空间。

假如缓存容量总共100MB，当前已用了70MB。假如此时又申请了30MB，系统就会分配出去30MB；但是如果现在申请使用110MB空间，那当前剩余空间不足，这种情况下，系统就会释放掉已经使用的缓存空间，再加上剩余的缓存空间大小一同分配给最新的缓存空间请求。但是问题来了，就算已经将用掉的70MB释放掉，再加上之前剩余的30MB，一共100MB，那也还不够110MB，此时系统有多少缓存空间，就分配多少缓存空间，至于不够的，系统不作处理。因为每个应用的缓存空间固定就那么大，不管你申请使用多大的空间，系统最大只能给出这么大了，再多也给不了了。

那么这里就可以利用这一点，直接申请使用一个非常大的值，间接的让系统清理掉应用的缓存。

### §3.4.9 常用工具

1. 归属地查询流程：

通过将手机号的后三位，与归属地数据库中的记录作比较，如果查到，则显示出对应的归属地，如果没查到，说明不是手机号，可能是座机号码或者客服电话等等，此时就要对比号码的前四位或者是前三位，因为是对比区号，首位是0的话在数据库中是省略的，对比结果可能查到xx-xx-联通、xx-xx电信等等。

1. 常用号码查询流程：

常用号码数据库，记录的是一些客服类的公共号码，给出这些号码，是为了方便用户外出遇到一些常见的旅行方面问题时，没有求助对象时使用的缓解措施。

1. 短信备份还原流程：

短信备份过程是：读取系统短信数据库，读出目标号码、目标类型、创建时间和短信内容这四个字段的内容，每读出一条，就记录到一个bean类中去，并将其添加到一个集合中，最后读取完毕时，将保存了短信的集合转换成一个json串，写入本地SD卡内，如果没有安装SD卡，则保存到手机自带存储空间中。

短信还原过程正好相反，先从手机内存或者SD卡中读取，保存了短信内容的json串，然后解析出来一个个短信实体，通过内容解析者，将短信内容重新插入系统数据库。

这个操作需要手机完全root，而且是刷成开发者版本的root，因为操作系统短信数据库，是系统级的操作，因为Android4.4推出了一个叫AppOns的模式，就是说4.4以后，系统中，同时只允许存在一个默认的短信应用，只有默认的短信应用，才有权限收发短信。

1. 程序锁加锁解锁流程：

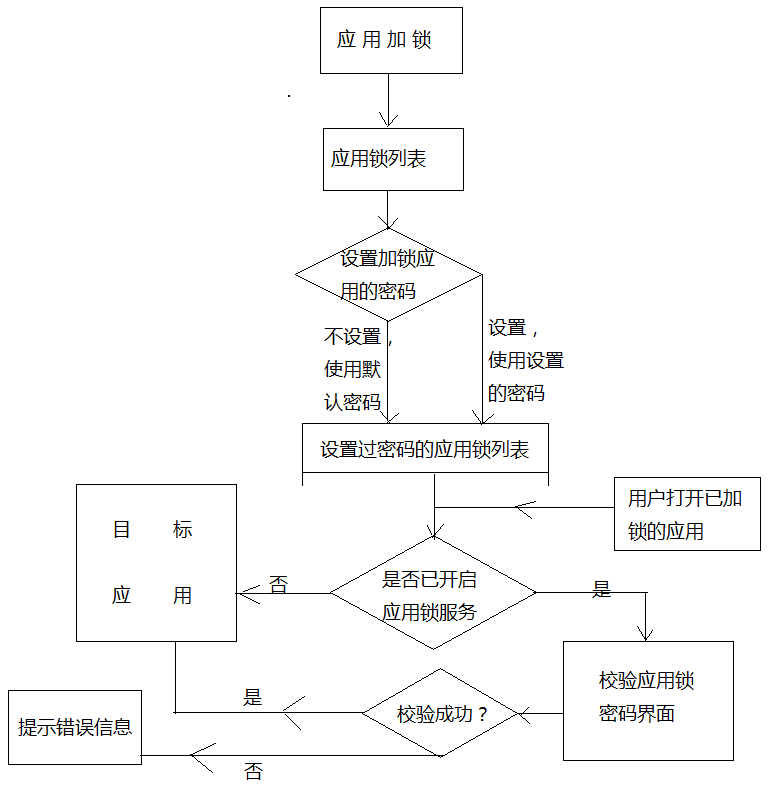


图3-7 应用加锁解锁流程图

### §3.4.10 注册登录流程

系统采用本地搭建的Tomcat服务器和MySQL作为后台数据库进行开发，用户注册后，信息保存到本地的MySQL数据库中流程如下图所示：

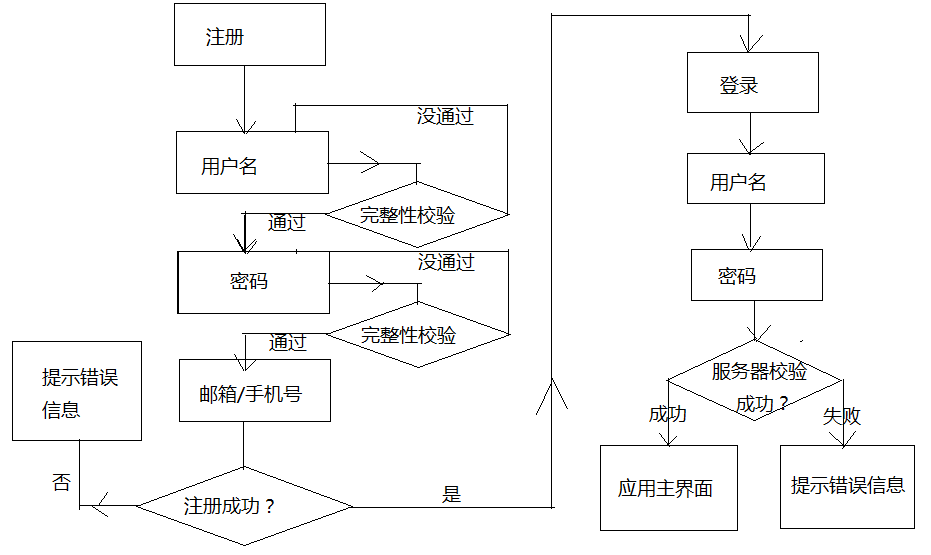


图3-8 注册登录流程图

### §3.4.11 贴吧相关操作

用户发帖，首先需要用户登录，然后才能发帖，用户发的每一个帖子都记录了用户ID，发帖内容，发布时间等信息。对于用户发过的帖子，其他用户可以对其进行赞踩评或者收藏帖子或者关注发帖人等操作，在评论列表中还能继续追加评论。如果是自己发的帖子，那么自己不能对其做如何的像赞踩评这类的操作；对于同一个帖子，用户不能重复收藏或者赞踩评；对于同一个发帖人，其他用户不能对其重复关注；对于关注过的用户，登录用户可以直接浏览关注用户的相关信息，可以查看TA收藏过的帖子或者TA赞踩评过的帖子，或者就是TA自己发的帖子以及TA关注的人。

最后发布的帖子，会放在最上面，之前的帖子追个往后排，如果有新发的帖子，每次刷新出来还是放在最上面。如需浏览以前的帖子，每次刷新10条数据进行展示。而每次获取最新数据的时候，也是请求10条数据，目的是为了节省浏览，以及节省内存。

# 第4章 数据库设计

## §4.1 概念结构设计

应用的后台服务数据是支撑整个项目设计与开发的基础，所以说数据库关系设计的是否合理健壮性强不强都会直接影响到软件的性能和可扩展性。说白了也就是说一个应用，首先要有数据的保证。结合该应用所要实现的功能，设计实现符合业务需求的数据表。首先在Android的模拟器上创建数据库进行测试。

基于上一章分析的各个实体之间的关系，以及主要实体的具体内容，要设计规范化的数据库，首先就要根据之前学过的数据库设计中的范式理论和事务原则来进行设计，具体要求如下：

1. 数据信息要求：主要黑名单信息、号码归属地信息、常用号码信息、程序锁信息、手机病毒信息等数据信息。

2. 数据信息处理要求：系统主要完成黑名单信息、号码归属地信息、常用号码信息、程序锁信息、手机病毒信息等信息的添加、删除、修改、查看功能，系统中号码归属地信息和手机病毒信息使用频率比较高，因此在处理方式上要求效率比较高。

3. 数据信息安全性与完整性要求：程序锁信息属于重要信息，因此要求保证数据信息安全，数据信息不能外泄。

### §4.1.1 具体E-R图

通过前面对系统的需求分析和系统分析，抽象出各个有关实体的属性图和E-R图：

1. 黑名单、号码归属地、常用号码、程序锁、病毒应用等实体，这些实体之间是相互独立的，如图4-1 ~ 4-4所示。
2. 用户、帖子、赞踩评、追评、收藏、关注等实体如图4-5 ~ 4-12所示，这些实体之间是存在依赖关系的，而它们之间的关系如图4-13所示。

黑名单信息

图4-1 黑名单实体属性图

号码归属地信息2

号码归属地信息1

图4-2 号码归属地实体属性图

类型x

常用号码类型信息

图4-3 常用号码实体属性图

病毒信息

图4-4 手机病毒实体属性图

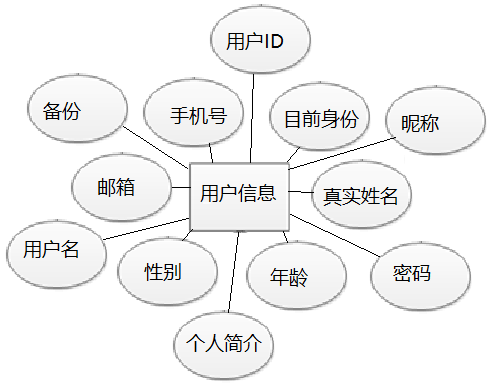


图4-5 用户信息实体属性图

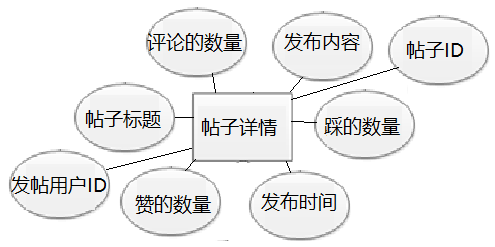


图4-6 帖子信息实体属性图

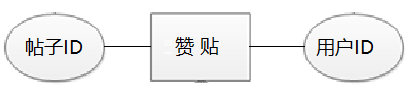


图4-7 赞贴实体属性图



图4-8 踩帖实体属性图

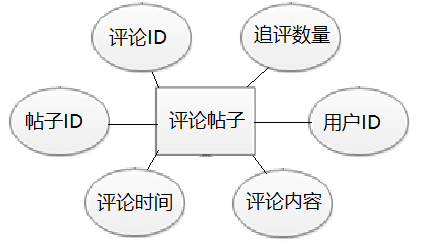


图4-9 帖子评论实体属性图

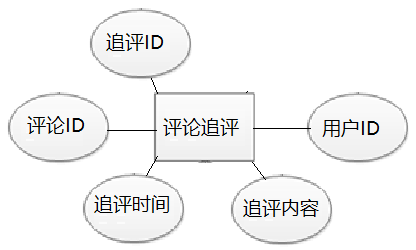


图4-10 帖子追评实体属性图

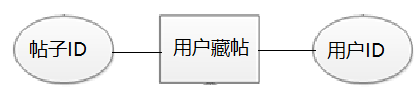


图4-11 藏帖实体属性图



图4-12 用户关注实体属性图

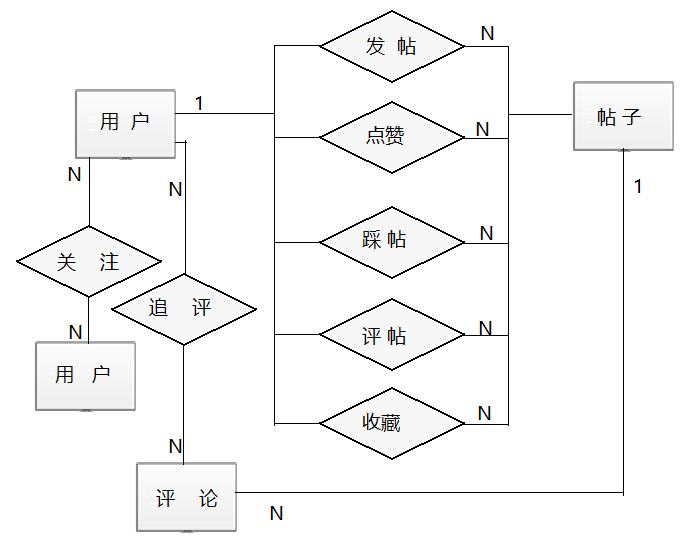


图4-13 发帖相关E-R图

综上所述，该应用的概念结构设计要以满足应用的功能为目的，结合应用的各个功能来进行数据库的设计。要以功能模块为中心来进行数据库的设计，但是也应该考虑到各个功能之间的逻辑独立性以及联系性。

在考虑功能实现的前提下，要结合数据库设计中的范式要求，以更好的设计的更加合理表结构，减少冗余数据，做到在提升数据库的存储效率的同时，还能够确保数据的一致性、完整性以及可扩展性。

## §4.2 逻辑结构设计

此阶段将概念结构设计阶段产生的各E-R图转换为本系统服务端采用的MySQL以及客户端采用的SQLite所支持的关系数据模型，并对模型进行优化，优化标准精确到第三范式。

鉴于系统在客户端采用的是SQLite数据库，而在服务端采用的是MySQL数据库，两端在对同一类型的数据长度的识别上有所区别，其文本格式和索引起始位置与前端网站常用用数据库有所不同，因此需要预先在代码上转换数据库中有差异的数据类型的预设。与此同时，为了提高应用的开发效率，以及后期应用的运行速度，对数据库的结构设计上，对事务的时间点处理上以及在触发器存储过程等引用性时间的合理性上，该应用还是要遵循以下数据库逻辑设计规范：

1. 表和字段的命名采用驼峰命名法，不过不是用大写区分，而是用下划线“\_”来分割，增强了数据表的可读性，便于后期的开发与维护。
2. 需要用Bool类型的都使用int类型，使程序员在数据库和程序间对Bool类型的数据不用在经过数据类型的转化，而能直接对数据处理。
3. Integer类型的主键使用自增ID字段。
4. 字符串统一用varchar，因为varchar必须只定一个长度，默认是20，这样可以限制发帖内容占据物理的存储空间，提高存储空间的利用率。
5. 日期类型的字段使用datetime类型，不用字符串的数据方式处理，因为datetime类型有时间精度的控制，采用字符串的方式时间的精度不便于控制。

同时，为了配合该应用的离线和在线的使用，为其创建独立的数据库，如果用户在联网情况下使用过此应用，那就该应用将用到的数据库拷贝到本地，在没有联网的时候，使用之前备份到本地的数据库；而当再次联网时，如果有新的病毒，会直接将其更新到病毒库中。

## §4.3 物理结构设计

本阶段要将逻辑结构设实施到一个具体的工作环境中去，即数据库的管理系统。服务端采用的DBMS是SQLYoga，它专门处理MySQL的图形化的数据库管理系统；而客户端没有DBMS，如果硬要说的话那就是Android系统本身，但项目开发需要查看数据库，那么SQLite对应的PC端的DBMS是SEP，可以通过在SEP上处理完数据，再导入到手机中去。而物理结构设计就是要在DBMS中完善数据存储结构与数据存取方法。

### §4.3.1 数据表实现

1. 黑名单、号码归属地、常用号码、程序锁、病毒应用这些表是创建在客户端的SQLite中的，具体见表4-1 ~ 4-7所示：
2. 用户、帖子、赞踩评、追评、收藏、关注等实体，这些互相之间存在联系的表是创建在服务端的MySQL中的，具体见表4-8 ~ 4-16所示。

表4-1 黑名单信息表（blacknumber）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| \_id | integer | NOT NULL | PRIMARY KEY |
| number | varchar(20) | NOT NULL |  |
| type | integer | NOT NULL |  |

黑名单信息表中的字段：\_id(黑名单编号)为主键；number(拦截号码)；type(拦截模式)。

表4-2 号码归属地信息表1（data1）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| id | integer | NOT NULL | PRIMARY KEY |
| outkey | varchar(20) | NOT NULL |  |

号码归属地信息表1中的字段：id为主键；outkey(号码尾号后三位)；

表4-3 号码归属地信息表2（data2）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| id | integer | NOT NULL | PRIMARY KEY |
| location | varchar(10) | NOT NULL |  |
| area | varchar(10) | NOT NULL |  |

号码归属地信息表1中的字段：id是主键，同时还是外键，参考引用了data1的outkey；location（归属地信息）；area（区号）。

如果是手机号，就查询data1，根据id查找表1 中与其对应的outkey，再到data2中查询归与此outkey一致的id对应的location。

如果不是手机号，那就是座机电话或长途电话，需要判断区号，直接查询表2，因为区号是4位的，但在sqlite中，首位是零会自动省略，所以直接用户名前三位或者前四位与data2中的area字段进行比较，如果查到，则读出对应的location。

表4-4 常用号码类型信息表1（classlist）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| name | text | NOT NULL | PRIMARY KEY |
| idx | integer | NOT NULL |  |

常用号码类型信息表中的字段：idx(号码分类编号)为主键；name(分类名称)。

表4-5 常用号码具体信息表2（tablex【x=1…8】）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| \_id | integer | NOT NULL | Reference (idx) |
| number | text | NOT NULL | PRIMARY KEY |
| name | text | NOT NULL |  |

常用号码具体信息表中的字段：\_id(类型编号)为外键，引用了classlist表中的idx；number(电话)为主键；name(公共号码名称)。

具体信息表有8张，一张对应一个大类型。

表4-6 程序锁信息表（applock）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| \_id | integer | NOT NULL | PRIMARY KEY |
| packagename | varchar(20) | NOT NULL | UNIQUE |
| psw | varchar(20) |  |  |

程序锁信息表中的字段：\_id(编号)为主键；packagename (应用程序的包名，唯一)。psw(应用锁密码，如果设置了就用设置的，如果没设置，缺省值是123)。

表4-7 病毒应用信息表（antivirus）

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 是否允许空 | 约束条件 |
| \_id | integer | NOT NULL | PRIMARY KEY |
| md5 | text | NOT NULL |  |
| type | NUMERIC | NOT NULL |  |
| name | text | NOT NULL |  |
| desc | text | NOT NULL |  |

手机病毒信息表中的字段：\_id(病毒编号)为主键；md5(应用程序签名)；type(病毒类型)；name(病毒名称)；desc(描述信息)。

表4-8 用户信息表（user）

|  |  |  |  |
| --- | --- | --- | --- |
| 列表 | 数据类型 | 是否允许空 | 约束条件 |
| id | int(11) | NOT NULL | PRIMARY KEY |
| usr | varchar(20) | NOT NULL | UNIQUE |
| psw | varchar(20) | NOT NULL |  |
| realname | varchar(20) | NULL |  |
| sex | varchar(4) | NULL |  |
| age | varchar(3) | NULL |  |
| email | varchar(50) | NULL |  |
| phone | varchar(11) | NULL | UNIQUE |
| status | varchar(10) | NULL |  |
| nickname | varchar(20) | NULL |  |
| pcintroduce | varchar(100) | NULL |  |

表4-8 帖子信息表（post）

|  |  |  |  |
| --- | --- | --- | --- |
| 列表 | 数据类型 | 是否允许空 | 约束条件 |
| postid | int(11) | NOT NULL | AUTO\_INCRE |
| uid | int(11) | NOT NULL |  |
| pubdate | datetime | NOT NULL |  |
| title | varchar(30) | NOT NULL |  |
| content | varchar(100) | NOT NULL |  |
| zan | int(11) | NULL |  |
| cai | int(11) | NULL |  |
| ping | int(11) | NULL |  |

表4-9 赞踩、收藏帖信息表（post\_zan、post\_cai、post\_collect）

|  |  |  |  |
| --- | --- | --- | --- |
| 列表 | 数据类型 | 是否允许空 | 约束条件 |
| post\_id | int(11) | NOT NULL | Reference（postid） |
| uid | int(11) | NOT NULL | Reference（uid） |

表4-10 评帖信息表（post\_cai）

|  |  |  |  |
| --- | --- | --- | --- |
| 列表 | 数据类型 | 是否允许空 | 约束条件 |
| comment\_id | int(11) | NOT NULL | PRIMARY KEY |
| post\_id | int(11) | NOT NULL | Reference（postid） |
| uid | int(11) | NOT NULL | Reference（uid） |
| content | varchar(100) | NOT NULL |  |
| pubdate | datetime | NOT NULL |  |
| chase\_num | int(11) | NOT NULL |  |

表4-11 关注信息表（post\_comment）

|  |  |  |  |
| --- | --- | --- | --- |
| 列表 | 数据类型 | 是否允许空 | 约束条件 |
| watching\_user\_id | int(11) | NOT NULL | Reference（uid） |
| watched\_user\_id | int(11) | NOT NULL | Reference（uid） |

表4-12 追评信息表（post\_collect）

|  |  |  |  |
| --- | --- | --- | --- |
| 列表 | 数据类型 | 是否允许空 | 约束条件 |
| chase\_comment\_id | int(11) | NOT NULL | PRIMARY KEY |
| comment\_id | int(11) | NOT NULL | Reference（comment\_id） |
| uid | int(11) | NOT NULL | Reference（uid） |

### §4.3.1 数据相关事务实现

1. 触发器：

使用触发器是为了提高系统处理速度，因为数据最终是存储在数据库中的，所以在数据库中完善逻辑要远比用服务端代码来的快。原因是服务端与数据库之间还有一层JDBC桥接器，而数据库事务则是直接操作数据。

帖子上的触发器，每当成功插入一条评论时，对应帖子的评论数加一。

评论上的触发器，每当成功插入一条追评时，对应评论的追评数加一。

赞踩评的触发器，每当成功插入一条赞踩评，对于帖子的赞踩评数加一。

这三个触发器的实现方式一样，这里只列出一个追评的作为参考。

CREATE TRIGGER `insert\_chase\_comment` AFTER INSERT ON `post\_chase\_comment` FOR EACH ROW BEGIN

IF EXISTS (SELECT comment\_id FROM post\_chase\_comment WHERE comment\_id = new.comment\_id) THEN UPDATE post\_comment SET chase\_num = chase\_num + 1 WHERE comment\_id = new.comment\_id;

END IF;

END;

1. 视图：

使用视图也是为了加快处理速度，视图的原理是根据实际需求选择多张表的其中一个字段，然后把它们组合在一起，形成一个虚拟的表，这个表在数据库中是不存在的，它只是引用了这些表的某一个字段，所以称之为视图。

用户信息视图，选取用户名以及用户的赞踩评收藏关注的数量作为字段来创建视图。

DEFINER VIEW `user\_view1` AS SELECT `user`.`usr` AS `usr`,`user`.`nickname` AS `nickname`,COUNT(`post\_collect`.`post\_id`) AS`count(post\_collect.``post\_id``)`,COUNT(`user\_watch`.`watched\_user\_id`) AS `count(user\_watch.``watched\_user\_id``)`,COUNT(`post`.`postid`) AS `count(post.``postid``)`,COUNT(`post\_zan`.`post\_id`) AS `count(post\_zan.``post\_id``)`,COUNT(`post\_cai`.`post\_id`) AS `count(post\_cai.``post\_id``)`,COUNT(`post\_comment`.`post\_id`) AS `count(post\_comment.``post\_id``)` FROM ((((((`user` JOIN `post\_collect`) JOIN `user\_watch`) JOIN `post`) JOIN `post\_zan`) JOIN `post\_cai`) JOIN `post\_comment`) WHERE ((`user`.`id` = `post\_collect`.`uid`) AND (`user`.`id` = `user\_watch`.`watching\_user\_id`) AND (`user`.`id` = `post`.`postid`) AND (`user`.`id` = `post\_zan`.`post\_id`) AND (`user`.`id` = `post\_cai`.`post\_id`) AND (`user`.`id` = `post\_comment`.`post\_id`))

# 第5章 详细设计与实现

## §5.1 应用框架

地基的质量往往决定了大楼能盖多高，在移动开发领域中，应用框架就相当于地基，选择优秀的开发框架能够使应用的执行效率大幅提升，本项目使用所有的框架都源于GitHub上的开源项目。

### §5.1.1 UI框架选型

1. 项目最外层UI展现采用的是Jeremy Feinstein的SlidingMenu，其效果是在应用主界面两边各有一个隐藏菜单，通过左右滑动进行展现，每个部分是一个Fragment，Fragment在Android中的作用是分页加载，每个Fragment加载一个布局。而在其中的核心模块及顶层Fragment的布局采用的是ViewPager + 底部RadioGroup。
2. 首页模块展示界面的仿QQ空间的头部视差效果是自定义的ScrollView。
3. 列表展现采用的是handmark的PullToRefreshListView、emilsjolander的stickyListView、gjiazhe的SpringListView。其中PullToRefreshListView实现了下拉刷新最新数据、上拉加载历史数据的效果，用到了论坛的展示帖子的列表上，stickyListView实现了不同类型的列表项切换的粘性效果，用到了展示运行进程的列表是，SpringListView实现了仿iOS的弹簧列表效果其他所有的列表上。
4. 手机杀毒模块使用的类似360扫描的自定义进度条采用的是lyzysd的circleProgressBar。
5. 个人中心的Pager指示器使用的是dance的PagerSlidingTabIndicator。其效果和网易新闻的头部新闻分类指示器效果一样。

### §5.1.2 其它框架选型

这里包括了网络框架、依赖注入框架、网络图片加载框架。

对于这三个框架，有一个非常好的选择，GitHub上有一个叫做xUtils的优秀开源项目，是国人开发的，这个牛人的ID是wyouflf，这个框架集成了网络框架、依赖注入、图片加载以及数据库的快速创建等功能，封装的非常合理，耦合度低，虽然是比较老的框架，但是目前依然有再用它的项目。

不得不说，xUtils在网络图片加载方面做的相当出色，提到网络图片加载，首先想到的是图片的三级缓存，其内部的封装，无论是在运算时间还是在响应速度上可以说是做到了极致，使用上的感受就是非常省流量。

而它在网络请求方面也使用了网络连接池，在实现方面就是减少http请求次数，将类型一致的多个短暂连续的请求封装在一个http请求中，一同发送，在响应处理时再逐个解析，逐个响应。

## §5.2 模块功能的实现

首先对Android整体框架做一个简要介绍，这里介绍一下安卓的四大核心组件：Activity、Service、 Broadcast Receiver、Content Provider。

在Android中，每个界面对应的包装类叫做Activity，它封装了一个布局界面以及在这个界面上的逻辑处理，而辅助activity实现业务逻辑的其他核心组件还有Service、Broadcast、ContentProvider，这几个被称为Android的四大组件，所有的Android应用，都是依靠这四大组件的配合使用而工作的。

Service在Android中起到的作用和activity类似，它也封装了一个逻辑处理模块，但是Service没有封装布局，可以说Service就是一个没有界面的activity，它经常用于处理一些用户不需要看，但是完成用户关心的耗时操作，而这些操作一般适合放在后台让系统自行处理，比如说用手机播放音乐、用手机下载东西、定时闹钟等操作，用的都是后台服务。

Broadcast Receiver是实现Android中进程间通信的一个工具类，顾名思义，它的工作原理和现实中的广播如出一辙，有发送方，有接收方。通过注册广播，然后发送匹配的广播，在广播接受者中就能够收到发送的广播从而进一步的处理。Android系统本身就是一个广播机制的体现，对于任何系统级的操作，系统都会发送一个广播通知手机，下一步应该执行什么操作。发送广播的地方通常是Activity和Service。

Content Provider也是实现跨进程通信的一个工具类，不过它是用于将自己的用于的数据库暴露出去，供其他应用于来访问的，这个需求一般用的比较少，除非用于做的比较大了而且需要提供自己的数据库供其他应用使用的时候，才会用到内容提供者，而实际开发中常用的是内容解析者ContentResolver，内容解析者是专门用于访问其他应用的数据库的工具类，它提供了基本的CURD功能。

这四大组件的配合使用，组成了一个完整的Android应用，任何单独的一个组件都无法自食其力，那只能算是一个Demo。

本章提到的更新UI指的是更新手机界面显示，UI在移动开发领域指的一般就是手机界面，本章不对如何写布局逻辑方面做阐述，只对模块实现的核心引擎做出解释。由于篇幅有限，这里就只对几个核心功能的实现原理进行讲解，对无法通过截图展示的效果不进行截图展示。

### §5.2.1 应用主界面实现

应用主界面采用Fragment的方式实现，因为系统由3个核心模块组成，即：卫士模块、贴吧模块、个人信息模块。Fragment配合SlidingMenu实现了多级菜单的效果，左菜单显示登录用户简要信息，右菜单根据用户操作显示不同内容，而中间主界面则显示这三大核心模块。实现效果如下图所示：

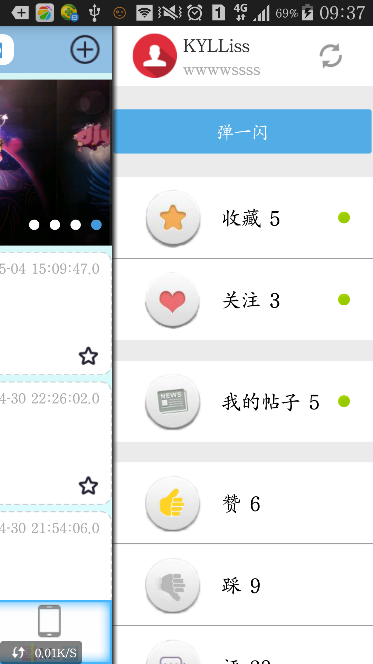
  

图5-1 左菜单（左）、主界面（中）、右菜单（右）

### §5.2.2 指令短信的处理

在Android中，对于系统短信的处理，在4.4之前没有特殊的限制，使用常规的做法，就能实现短信的发生、拦截等操作。而4.4+就变了，4.4+是Android对系统优化上的一次较大的改良，4.4+全部采用一个叫APPONS的机制，这个机制，限制了系统级的操作只允许默认应用程序进行操作，意思就是说，像收发短信这种系统级的操作，在系统中只有手机自带的短信应用能够进行处理，原因是手机自带的短信应用就是系统默认短信应用，所以它能够操作短信。那么这就需要让本应用也成为默认短信应用，虽然4.4+给这个操作隐藏了，但是还是能够通过调用系统代码打开选择默认应用界面的。这个操作需要手机root过才能生效。

这一步处理完成，就能对4.4+的手机短信进行处理了，手机防盗模块提供了四个防盗功能，分别是：远程锁屏、远程销毁数据、发送手机位置给安全号码、播放报警铃声这四个功能。其指令格式是：#\*code\*#的形式，实际上就是在收到短信后，判断短信是不是和指令匹配，如果匹配，那么就执行相关操作，并且拦截短信，拦截是为了不让系统收到指令短信，因为如果小偷看到了这样的奇怪的指令，就会引起注意。

其中远程锁屏，通过开启设备管理者的权限，并且注册短信监听广播，实现远程锁屏。当收到该指令短信，广播接收者就执行设备管理者的核心API：lockNow，进行锁屏操作。

远程擦除数据也和远程锁屏应用，使用设备管理者的核心API：wipeData进行手机数据的擦除。

播放报警铃声使用了系统组件MediaPlayer，播放指定好的报警铃声。

通过构造器传入报警铃声，然后调用MediaPlayer的start方法开始播放。

远程定位使用了系统的GPS组件，当解析到定位指令时，开启一个定位服务，在服务中，监听手机的位置变化，一旦发生改变，就发送当前手机位置信息到安全号码。这里解析经纬度坐标采用的是免费网站GPSspg。

### §5.2.3 骚扰拦截处理

该模块实现的核心是通过开启骚扰拦截服务实现对黑名单的骚扰拦截。

在服务中，动态注册了一个接受短信的广播，当来短信时，判断短信号码，如果和黑名单中对应拦截类型匹配，那就拦截短信。同时注册了一个电话状态监听器，在来电时对比黑名单列表，如果有匹配项，那么就自动挂断电话，并且删除通话记录。

值得一说的是挂断电话这个操作被系统屏蔽的，虽然能看到，但是不能直接调用，该方法是ITelephony的方法，ITelephony是一个AIDL文件，Android中提供了一种进程间通信的方式，就叫做安卓接口定义语言(Android Interface Define Language)，简称AIDL。它的实现原理相当复杂，要解释它需要做大量的铺垫，抛出许多专业概念并解释其原理，然后才能引出AIDL的实现原理，这里不对它的实现原理做解释，只说用法，就是使用这个定义好的AIDL文件（这类系统专用文件可以去谷歌Android SDK上下载）去跨进程调用系统的方法，从而实现挂断电话，这里就是通过ITelephony调用了系统的挂断电话的方法endCall。

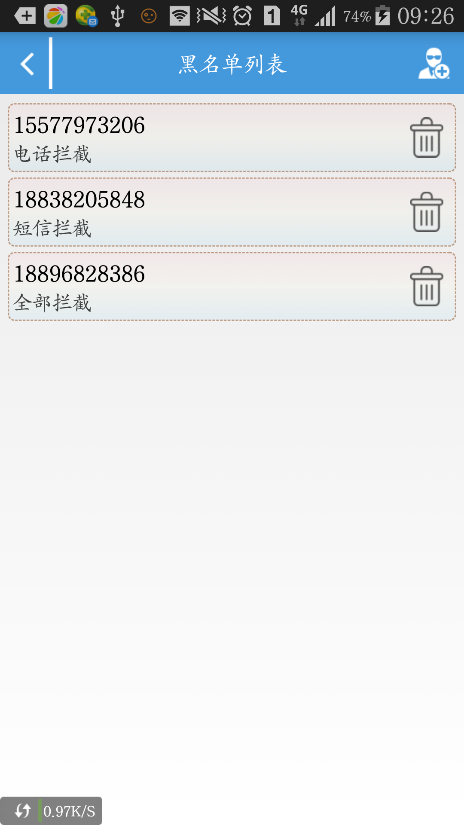
 

图5-2 骚扰拦截，黑名单（左）、服务开关（右）

### §5.2.4 获取应用详细信息

应用详情信息的获取，用到了一个核心的工具类，叫做PackageManager，这个类通过当前上下文获取，当前上下文就是指当前activity或者当前Service，不过这里使用的是Application的上下文，Application、Activity、Service都是Context的子类，都可以称之为上下文，不过Application和它俩的区别在于Application是一个Android跑起来的第一个上下文，是全局上下文，而且一个安卓应用有且仅有一个唯一的Application。通过它获取到PackageManager对象后，再通过该对象获取系统中所有已安装的应用的包的信息的集合，集合类型是PackageInfo，

这个类就是一个bean类，就封装了安装包的所有信息，遍历该集合，通过每一个安装包对象来获取安装包中需要的信息，然后保存到自己的bean类中，这样就实现了对应用详情信息的获取。核心代码如下：

PackageManager pm = MyApp.context.getPackageManager();

List<PackageInfo> ins = pm.getInstalledPackages(PackageManager.GET\_SIGNATURES); // 额外 获取 应用 签名信息（杀毒 模块 要用 应用 签名信息的MD5值 与 病毒库 进行比对）

for (PackageInfo pi : ins) {

……//通过pi获取应用所有信息

}

用到该引擎的有软件管家、缓存清理、应用锁等模块。其中软件管家实现效果如下：

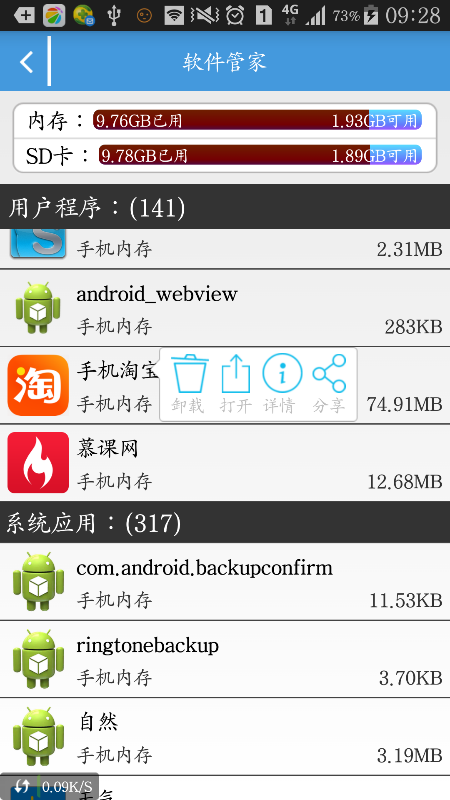


图5-3 软件管家界面

### §5.2.5 获取运行进程详情信息以及处理过程

进程详情信息的获取同应用安装包信息类似，都需要用到PackageManager，不过获取进程信息还需要用到进程管理者ActivityManager。在Android中，一个应用对应的进程的数量是：该应用四大组件数量的总和再加一，安卓应用的四大组件，每一个都是重量级组件，需要独占一个进程，而最后再加一是因为每个安卓应用还有一个全局的上下文Application。

通过上下文对象拿到系统进程管理者ActivityManager对象，再通过该对象获取正在运行的进程的信息详情，由此拿到进程的所有信息的列表，列表类型为RunningAppProcessInfo，它也是一个bean类，只不过封装的是正在运行的进程的信息，同样需要遍历集合，操作一样。其核心代码如下：

ActivityManager am = (ActivityManager) MyApp.context.getSystemService(Context.ACTIVITY\_SERVICE);

PackageManager pm = MyApp.context.getPackageManager();

List<RunningAppProcessInfo> runningAppProcesses = am.getRunningAppProcesses();

for (RunningAppProcessInfo rapi : runningAppProcesses) {

……//通过PackageManager的killBackGroundProcess方法杀死后台进程。}

实现效果如下：



图5-4 进程管理界面

### §5.2.6 缓存清理处理

缓存清理主要用到的是PackageManager中的一个被隐藏的方法，该方法能够实现的功能是：重新请求分配空间，通过请求一个无限大的缓存空间，让系统释放掉所有已用的缓存空间然后在分配出去最大的缓存空间，通过这种方式变相的实现对应用缓存的清理操作。这么做的原因是系统没有提供清理缓存的方法，这也是通过研究系统源码，分析出来的一种变相的解决方案。

在Java中，被系统隐藏的方法，即添加了@Hide注解的方法是不能继续调用的，如果一定要用，只能通过反射机制来做，这里就使用反射去调用那个被隐藏的方法，方法名叫：freeStorageAndNotify，而该方法中，又用到了一个AIDL，这个AIDL是一个回调，当空间请求成功的时候会走这个回调。而在此之前，需要先获取应用缓存的大小，这里参考的是系统获取缓存信息的做法，又是查看源码，最后查到了系统也是用了一个AIDL来获取的，这个AIDL叫做IPackageStatsObserver，这里就直接复制系统代码过来，然后根据项目需求，获取缓存大小，并保存到自己的bean类中去。

还有一点，AIDL在Android中的执行都是在子线程中进行的，如果要更新UI的话，需要使用UI线程的Handler完成。

核心代码如下：

1. 获取应用缓存信息：

final IPackageStatsObserver.Stub mStatsObserver = new IPackageStatsObserver.Stub() {

public void onGetStatsCompleted(PackageStats stats, boolean succeeded) { // 获取 应用 状态信息成功时调用的方法

long cacheSize = stats.cacheSize; //获取应用缓存大小

……

}};

1. 清除应用缓存：

Method met = PackageManager.class.getDeclaredMethod("freeStorageAndNotify", long.class, IPackageDataObserver.class);

met.invoke(pm, Long.MAX\_VALUE, new IPackageDataObserver.Stub() {

@Override // 该方法 是在 缓存清理 完成 时 调用的

public void onRemoveCompleted(String packageName, final boolean succeeded) throws RemoteException {

runOnUiThread(new Runnable() { // 所有的 aidl 都是 运行在 子线程 public void run() { …更新UI }});}});

实现效果如下：

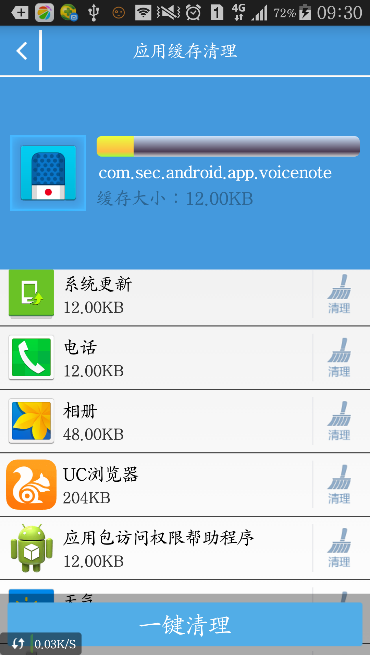
 

图5-5 缓存清理界面，清理中（左）、清理完成（右）

### §5.2.7 短信备份还原

Android中的短信备份需要读取系统短信数据库，而读取系统相关的数据库在Android中并不需要手写SQL语句去实现，而是使用谷歌封装好的一套API，来对系统数据库进行存CRUD操作。谷歌封装它是有道理的，安卓系统每一次升级，都可能对原有的数据库数据表的结构或者字段名进行修改，而且确实有这样的例子，就是说手写的SQL语句在低版本能用，在高版本用不了的。所以谷歌对此问题，提供了一套解决方案，由谷歌内部将对系统库进行操作的所有SQL操作进行封装成一套对外可用的公共API，版本升级后，API内部使用的SQL语句也会相应发生变化，就是说不管怎么升级，只要调用这套API，就能使用对应版本的SQL语句对系统数据库进行操作。这样，既简化了操作，又降低了开发者对版本SQL细节的记忆的要求。

而实现短信备份，无非要读这么几个字段：目标号码、短信类型（是收的还是发的）、短信时间、短信内容。接下来用内容解析者进行解析即可，内容解析者同样需要使用上下文对象进行获取，最后将获取的短信内容通过Gson工具类拼接成json字符串，存储到本地即可。

短信还原的过程与之相反，区别仅在于备份是读取，而还原是插入。核心代码如下：

1. 获取短信内容解析者URI对象

Uri uri = android.provider.Telephony.Sms.CONTENT\_URI;

1. 通过API确定要操作的表字段

String[] projection = {android.provider.Telephony.Sms.Inbox.ADDRESS,//号码android.provider.Telephony.Sms.Inbox.DATE,//短信时间

android.provider.Telephony.Sms.Inbox.TYPE,//短信类型

android.provider.Telephony.Sms.Inbox.BODY};//短信内容

1. 备份核心：当前短信数据库

ContentResolver cr = MyApp.context.getContentResolver();

Cursor c = cr.query(uri, projection, null, null, null);

while (c.moveToNext()) {c.readXxx(index); // 字段挨个读取}

1. 还原核心：往短信数据库中插入记录

ContentResolver cr = MyApp.context.getContentResolver();

ContentValues values = new ContentValues();

values.put("address", sm.address);values.put("date", sm.date);

values.put("type", sm.type);values.put("body", sm.body);

Uri insert = cr.insert(Uri.parse(ConstC4.SMS\_DB\_URI), values);

### §5.2.8 应用加锁解锁

应用锁的实现思路是：开启应用锁服务后，在服务中开启一个子线程，子线程中实现高频率的定时包名检查，比如200ms检查一次，检查当前打开的应用的包名信息是否记录在应用锁的数据库中，如果在应用锁数据库中，这说明该应用已被加锁，需要解锁后才能打开此应用，那么就跳转到应用解锁界面，让用户进行解锁操作；如果没有查到，说明用户没有对这个应用加锁，不进行处理，直接打开应用。

其核心之一就是一个定时检查，实现定时检查的方式有三：

1. JavaSE提供的Timer和TimerTask类的配合使用；
2. AndroidSDK提供的AlarmManager；
3. 循环控制语句处理。

这里为了实现简洁，使用第三种方式手动处理。

而核心之二则是如何获取最顶层的应用的包名，这里需要先获取当前的Activity任务栈，任务栈这个概念描述了一个应用的界面使用情况，举个例子：手机淘宝买东西流程：1.手机淘宝主页；2.目标商品详情；3.订单核对界面；4.购买结果界面，那么当前的任务栈的顺序从栈底到栈顶就是：1234的顺序，然后按返回键返回的顺序则是4321。任务栈的特点就是栈中的所有Activity的包名一致，因为是同一个应用。那就是先获取到顶层任务栈，然后在获取到任务栈中任意一个Activity的包名即可。核心处理逻辑如下：

while (isOpenService1) { // 死循环

List<RunningTaskInfo> runningTasks = am.getRunningTasks(1);//1.任务栈获取

for (RunningTaskInfo rti : runningTasks) {

ComponentName cn = rti.baseActivity;//2.获取任务栈中的栈底Activity的组件名

topPackageName = cn.getPackageName();//3.根据组件名获取应用包名

if (list.contains(topPackageName)) {

if (!topPackageName.equals(unlockPackageName)) {//4.判断应用是否已加锁，即判断应用包名是否存在于应用锁数据库,如存在则跳转到 解锁界面 } } }

SystemClock.sleep(200); // 每隔200ms 检测一次}

实现效果如下：

图5-6 应用锁界面，应用锁列表（左）、解锁界面（右）

### §5.2.9 贴吧模块

该模块实现的功能和百度贴吧一样，能够满足用户的发帖、评帖、赞踩评、收藏、关注、私信等一些列操作。该模块涉及到的业务逻辑比较多，代码方面各个驱动各个适配器之间都有联系，而且既要处理客户端，又要处理服务端。客户端主要是处理在用户操作后UI上的实时更新显示等逻辑操作，而服务端则是要在收到客户端的请求后通过JDBC对数据库进行更新操作，然后返回相关数据给客户端，客户端在根据服务端响应来更新UI。客户端一个操作，两端都要进行逻辑处理，这里就说一个评帖的执行流程。

用户评论帖子的流程：

1. 获取登录用户ID、评论内容、评论时间、被评论的帖子的ID
2. 发送评论请求，将第一步的参数信息封装到请求体发送给服务端
3. 服务端解析用户请求参数，执行安全性判断
4. 判断成功则通过DAO，将参数内容插入评论库
5. 插入成功后返回给客户端成功信息，否则返回错误代码
6. 客户端判断服务端响应信息，如果是成功，则更新UI，否则根据错误代码提升相应的错误信息

其中，客户端需要回显帖子评论数量，即更新当前帖子列表，显示最新数据，如果是在帖子详情界面进行评论，则应该显示出最新的评论并且更新评论数量。而服务端在执行安全性判断时需要检测用户发送的传送的格式是否正确，检测用户或帖子是否存在，以及帖子编号是否重复。除此之外，客户端和服务端之间需要解决一个编码问题，因为Tomcat服务器默认采用ISO-8859-1这种西文的编码格式，而Android系统默认使用UTF-8的编码格式，所以，所有的请求必须先转换文字编码，再进行相关操作。

实现效果如下图所示：



图5-7 帖子列表

# 第6章 系统测试

## 

运行稳定、健壮性强、逻辑通顺的应用一定是经过重重测试筛选出来的，任何一款界面友好、运行稳定、响应速度迅速的产品，都是经过了后期各种的功能、性能、压力和冒烟测试的，而且并没有出现什么问题。只有通过这些测试的狂轰乱炸之后，还能依然不出问题，能够像平常一样正常运行，才能说，这是一款合格的产品。

## §6.1 功能性测试

本应用各个模块相互独立，每个模块可以做一个单元测试，这里就使用黑盒测试的方法，对每个模块单独进行测试。

测试范围：卫士模块的各个功能、贴吧相关各个操作、用户注册登录改密相关操作。这方面测试基本上是写完就测，测完改，改完再测，测完再改，单元测试是贯穿整个开发流程的。随着功能的完善，可能改动了之前写好的某些内容，所以改了哪一块，那包括与这一块相关的都要重新测试，以确保手数据完整性。

其中最令人头疼的无非是贴吧的相关操作了，改一点儿代码，其关联的几个类都要改，改完关联类，很可能改动到与关联类相关的类，这一点如果不注意的话，确实头疼的很，因为你根本不知道问题的出处。

这一块遇到的问题也是最多的，关于这方面的问题我总结出：在关联处理的逻辑上，每一个流程点都打上log，这样测试流程走向的时候，可以通过log信息，检测出走到哪出问题了，这样就能避免盲目的找bug，而且节省了开发时间。

## §6.2 性能测试

由于Android系统的开放性机制，在获取系统应用列表的时候，能够获取到应用的所有信息，其信息量比较大，所以需要一定的时间的。像获取应用进程信息这种操作是获取系统中所有已安装应用的前台进程、后台服务、后台广播等所有组件，属于耗时操作，而安卓中的所有的耗时操作，在Android4.0以后规定必须在非UI线程中执行，这么做是为了防止ANR（Android Not Response）异常。因为子线程在执行的时候可能会引起线程阻塞，所以在获取应用详细信息这种耗时操作一定要放到非UI线程去做，所以出现延迟的起因就在于此。目前对此问题并没有很好地解决策略,这毕竟是获取系统详细需要以处理事务的方式去执行，而不能像用sax方式解析xml那样边解析边执行。

而服务端在DAO的CRUD操作方面，虽然没有使用连接池这种资源调度工具类，但是也对其进行了相关优化。给DAO工具类中的每个操作都加上一个是否保持连接的标识，请求完成后是否断开数据库连接则就看这个标识。如果不加这个标识，那么每次执行CRUD操作都要：建立连接->执行->断开连接，而建立连接和断开连接会浪费大部分执行时间，所以，数据库连接原则上只建立一次，只有关闭连接后，才会建立新的连接，而断开连接则是根据这个标识来决定，一般是应用退出时，客户端发给服务端一个消息，告诉服务端，该用户不需要再与服务端保持连接了，然后服务端就断开数据库连接即可。

## §6.3 压力测试

这方面测试的核心在于应用能否经得起多用户并发访问，并发访问这就牵扯到服务端是否健壮。本应用的服务端采用的是Servlet + JDBC + MySQL，基本满足了少量用户的并发访问的需求，因为是自己做的项目，不可能有大量真实用户来进行这方面的测试，而且服务端也是在本地部署的，并没有部署到外网上。这里请了班里的十几个同学安装本应用，协助进行了这项测试，不过对于班同学等少数用户并发访问，Tomcat Web服务器是完全承受得住的，使用上并没有出现什么问题，说明了本应用的服务端的健壮性有一定的保障，如果将服务端应用部署到更强的服务器上，肯定能够支持用户量更大的并发访问。

## §6.4 冒烟测试

这类测试是在项目即将上线之前必须做到测试，核心就是使用开发工具，对应用进行快速连续而且位置不确定的点击，工具的点击速率大约是每秒40 ~ 100下不等（这个值可以自行调节，但是一般就使用这个区间最为合适），直到把这个应用点崩溃为止。对真实项目的要求是能够在至少连续的1000次冒烟测试中，能够通过95%以上，才允许应用上架。而现实中的用户，不可能点的像工具那么快速，工具都点不挂，那用户也基本上不可能把应用给点挂掉了。

另一方面，这种测试，对于有些老手机来说，是无法实现的，因为一些老手机的硬件配置可能达不到目前的主流配置的标准，达不到标准的话，那对于用户的点击，对系统来说就是触摸事件的响应方面就达不到要求，可能你在1秒内点了10次，系统只响应了3次，剩下的7次，系统没反应过来所有没有处理，那么对于这种情况，这种测试是没有意义的。

本应用虽然没有上线，但是也做了这方面的测试，Android SDK本身就集成了这个工具，叫做Monkey，使用Monkey命令，可以对应用进行冒烟测试。命令如下：monkey 项目包名 点击次数 [点击速率：次数/秒]。后期测试连续执行了50次默认参数的Monkey命令，应用并没有出现崩溃之类的问题。截取了其中两张执行命令时的效果图：

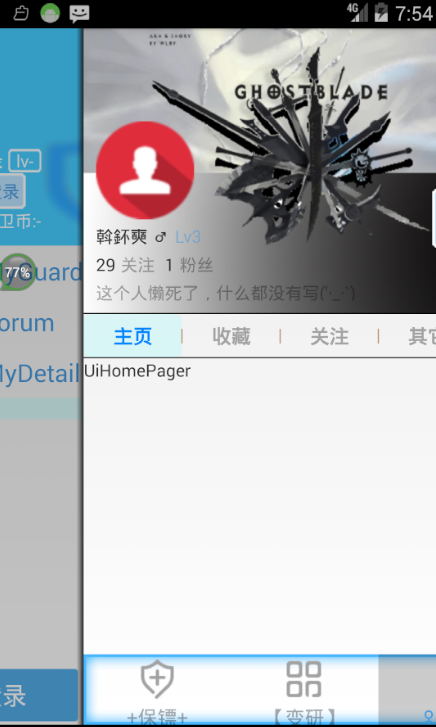
 

图6-1 执行Monkey命令过程中的手机状态

# 结 论

本应用严格遵循软件工程的思想，根据软件生命周期的各阶段的特点与需求进行开发，首先对应用的系统整体进行了专一严谨的定义，确定了该系统要解决的问题及存在的意义，并实施了可行性研究，接着进行了细致的需求分析，将本应用划分为九个模块，确定了每个模块应该实现的功能，根据系统结构图和数据流图设计了系统的软件结构，限定了系统框架，各模块之间基本上都做到了高内聚、低耦合。接下来就继续对软件进行数据逻辑设计与系统详细设计，确定了应用在数据处理方面使用的数据结构、算法以及UI展现方面使用的框架以及各个模块之间的处理流程。而且后期还对系统做了标准的项目测试，保证了系统的稳定性。

从结果上看，本应用实现了全部的预期要求，实现了一个界面友好、稳定性强的功能完整的系统，系统也确实实现了能够在一定程度上保护手机的安全，提升手机的性能，保护手机的数据等效果，合理的帮助用户管理自己的手机，对手机做出不同程度的优化，同时也给游客提供了一个交流、分享的平台，不仅让用户能够使用系统功能解决问题，还能够通过这个交流的平台寻求帮助，并且能够很自然地对应用进行推广。

该应用虽达到了预期目标，实现了基本的功能，但是同时，系统还有一些功能和设计上的不足：

1. 部分页面的处理速度比较慢。
2. 在用户隐私模块的UI效果没有做的很细致，没有达到真实的项目的个人中心的标准。
3. 服务端的JDBC桥接方面以及服务端框架选型方面，没有做特别的处理，只是用了简单的Servlet进行处理（毕竟主要精力都放在Android客户端的开发上，服务端没法做的那么细致，而且也没有使用Spring、Hibernate等框架）。

接下来需要将这些地方进行改良，让系统愈加完善的同时，也为本应用的用户提供更好的用户体验。

# 参考文献

1. [美]Bruce Eckel, 陈昊鹏译. Java编程思想.北京：机械工业出版社, 2011
2. 佘志龙. Google Android SDK开发范例大全.北京：人民邮电出版社, 2010
3. America J, Madeira H and Silva J G. Xception: A technique for the experimental evaluation of dependability in modern computers. IEEE Transactions on Software Engineering, 2008, 24(2): 125-136
4. Koopman P and DeVale K. Comparing the robustness of POSIX operating systems. In: Proceedings of the IEEE 29th International Symposium on Fault-Tolerant Computing, Madison, IEEE CS Press, June 2009, pp. 30-37
5. Cay S.Horstmann, Gary Coenell, 叶乃文译. Java核心技术卷二：核心技术. 北京：机械工业出版社, 2008
6. 托普雷. J2EE技术手册. 北京：中国电力出版社, 2004:123 ~ 165
7. 史济民.《软件工程原理、方法与应用》.高等教育出版社，2002年
8. 萨师煊、王珊. 数据库系统概论. 北京：高等教育出版社, 2003
9. 福林布朗. 移动应用的设计与开发. 北京：电子工业出版社，2010
10. [美]Brett D.Mclaughlin.Head.First Object-Oriented Analysis & Design[M]. O'Reilly Media，2009.
11. 程浩鹏. Java核心技术卷. 机械工业出版社, 2008
12. 李兴华. Java开发实战经典. 清华大学出版社, 2009
13. 结城浩. Java多线程设计模式. 北京：中国铁道出版社，2005
14. 盖索林. Google Android开发入门指南. 人民邮电出版社, 2009
15. [美]Mark Murphy. 李雪飞、吴明晖译. Beginning Android 2. 北京：人民邮电出版社，2010
16. 墨菲. 《Android开发入门教程》. 人民邮电出版社，2010:15 ~ 86
17. 梅尔、王超. Android 2高级编程(第2版). 清华大学出版社
18. 韩超、梁泉. Android系统原理及开发要点详解. 电子工业出版社
19. 李刚. 疯狂Android讲义. 北京：电子工业出版社，2011
20. 陈文、郭依正. 深入理解Android网络编程：技术详解与最佳实践. 北京：机械工业大学出版社，2013

# 致 谢

时光飞逝，转眼四年的大学生活即将结束，这四年不管是在学校的经历还是实习的经历都让我受益匪浅。对我来说这篇论文从四年前我刚迈入科大的那一刻起就已经开始了。国教和信工的老师们，为我们大学几年的学习、成长营造了一个适合充实自我的环境，因此我们才可以想现在这样充分利用校内的一切可利用资源，不断地去探索、去发掘自我。在此，我怀着一颗感恩的心，正式向你们道谢，向所有殷勤奉献的教师致敬。

写论文的这一个月里，从选题到定题再到查阅资料到最后形成提纲，从构思内容到初稿完成再到反复打磨最后到定稿，过程中仿佛经历了人生的大起大落，过程一波三折，跌宕起伏，没想到写毕业论文的感受竟如此复杂。但在论文最终成稿之际，那种复杂的感受不安的心情也随之涣然冰释，看着自己比较满意的作品，心情一片大好。我要对所有帮助过我的可爱的你表示诚挚的谢意：

首先要感谢的是我的导师张晓玲老师。她教过我们面向对象和编译原理这两名课，和我们很熟。平时她就像同窗室友那样平易近人，但她治学严谨，思维缜密，在我写论文期间，老师对我的专业指导，对论文的内容要点分析，其针对我的问题所提出的批判之准、概要之精，都令我为之佩服。从最开始不知道从何下手，到论文选题定题，到查资料列提纲，再到后期对论文的无数次的改版，老师自始至终都严格认真地审视批判，总能以她独特的方式来我开拓我的思路，打开我继续创作的灵感。论文能够如期完成，还要归功于老师对我细致入微帮助。

然后要感谢我最敬佩的学长，田苗同学。在公司项目即将上线之际，还在百忙之中硬挤时间来给我讲解我选的毕设项目的框架原理，还给我分享了他自己写毕业论文时的经验，大大拓展了我写论文的思路。我深知这些宝贵的经验来之不易，所以在此对学长对我的帮助表示衷心的感谢。

最后还要感谢图书馆的老师们。他们总能热心的为我们这些即将毕业的学生提供与我们所关心的内容最最相关资料的一手信息，让我们能够不像无头苍蝇一样一排排的找资料，节省了我大量的宝贵时间，让我非常感动，在此对图书馆老师们的支持和帮助表示万分感谢。

寥寥数句，虽不足以表达我的谢意，但是我要表明我的诚心。如果没有导师的殷勤指导，如果没有最懂我的学长的建议，如果没有图书馆老师们的热心帮助，就不会有现在的我。在得到帮助的同时我还理解了为人处事的道理，交了更多的朋友，学到了除了专业知识外的各种很细节也很实用的东西。最后，衷心的感谢所有帮助过我的大伙儿。感谢你们在炎炎夏日的辛勤付出，愿你们一切工作顺利，身体健康。

**SIoT: Giving a Social Structure to the Internet of Things**

Luigi Atzori, Senior Member, IEEE, Antonio Iera, Senior Member, IEEE, and Giacomo Morabito, Member, IEEE

Abstract—The actual development of the Internet of Things (IoT) needs major issues related to things’ service discovery and composition to be addressed. This paper proposes a possible approach to solve such issues. We introduce a novel paradigm of “social network of intelligent objects”, namely the Social Internet of Things (SIoT), based on the notion of social relationships among objects. Following the deﬁnition of a possible social structure among objects, a preliminary architecture for the implementation of SIoT is presented. Through the SIoT paradigm, the capability of humans and devices to discover, select, and use objects with their services in the IoT is augmented. Besides, a level of trustworthiness is enabled to steer the interaction among the billions of objects which will crowd the future IoT.

Index Terms—Internet of things, social networks, ubiquitous computing.

I. INTRODUCTION THE realization of an effective and reliable IoT requires the deﬁnition of a complex architecture that takes into account the issues of sensing the real world, transmitting data, and managing the relevant services to build applications [1]. Recently, major architectural solutions have been proposed within ITU, EPCGlobal, the CASAGRAS initiative, and the uID research group [2]. The list is not exhaustive: the plethora ofproposalsfromECfundedprojectscoordinatedby the IERC cluster must be added [3]. From the analysis of these efforts, emerging priorities are: (i) enable full connectivity of things to the Internet by operating at sensing and network levels, (ii) provide middleware and application functionality and protocols to ease the exploitation of things-related services. Our contribution ﬁts into the latter line of research, indeed we propose an innovative paradigm of interaction among objects. Basic ideais the deﬁnitionof a “social networkof intelligent objects”, named Social Internet of Things (SIoT). In analogy with Social Networks Services (SNS) for human beings, the novel paradigm introduces the concept of social relationships among objects. Advantages are the possibility to: ∙ Give the IoT a structure that can be shaped as required to guarantee network navigability, so as that object and service discovery is effectively performed and scalability is guaranteed like in human social networks (see [4], for example). ∙ Extend the use of models designed to study social networks to address IoT related issues (intrinsically related to extensive networks of interconnected objects).

Manuscript received June 23, 2011. The associate editor coordinating the review of this letter and approving it for publication was I.-R. Chen. L. Atzori is with the University of Cagliari, Italy (e-mail: l.atzori@diee.unica.it). A. Iera is with the University of Reggio Calabria, Italy (e-mail: antonio.iera@unirc.it). G. Morabito is with the University of Catania, Italy (e-mail: giacomo.morabito@diit.unict.it). Digital Object Identiﬁer 10.1109/LCOMM.2011.090911.111340

∙ Create a level of trustworthiness to be used for leveraging the level of interaction among things that are “friends”. Ours is a distributed approach, which is expected to guarantee a higher scalability and a better reaction to frequent state changes characterizing objects involved in the IoT. As for the identiﬁcation of a whole Architecture for SIoT, deriving it from “human” social network models is quite commonsense; although it is not exactly the same thing. The exploitation of social networks in the context of the IoT has been investigated in [5]. In this paper, it was proposed to exploit (human) social network relationships to share the resources offered by a given smart thing (smart things enabled to support web services usable by friends of their owner). Also the research in [6] tries to lay the groundwork for the establishment of a so-called Ubiquitous IoT architecture inspired by social organizations of human beings. Authors of [7] investigate on the potential of combining social and technical networks and discuss about the implication of socalled “socio-technical networks” in the context of the IoT. The Pachube platform is close to the idea of a social network of objects, allowing developers to connect sensor data to the Web to build applications. What the platform does not allow is objects to form social groups autonomously, for the beneﬁt of human beings but without their intervention. Actually, SIoT focuses on the latter aspect, in line with the notion of IoT devices with social connections introduced in [7]. A complementary perspective is considered in [8], where the focus is on solutions that enable smart wireless devices – mostly wireless sensors – to establish temporary connections and their owners to control such a process. The approach we propose differs from the literature for three major reasons. (i) We are interested in establishing and then exploiting social relationships among things, not among their owners. The owner mediation can be foreseen but objects have to play the key role, as it happens in novel IoT-based applications. (ii) Through social relationships things can crawl the IoT and discover services and resources; this provides a distributed solution that is expected to be effective, efﬁcient and, most important, relieves the humans from doing it. (iii) The envisioned IoT architecture is not a mere service platform centered on the concept of web of things, yet a real platform for SNSs with suitable components introduced to cope with the presence of objects instead of human beings.

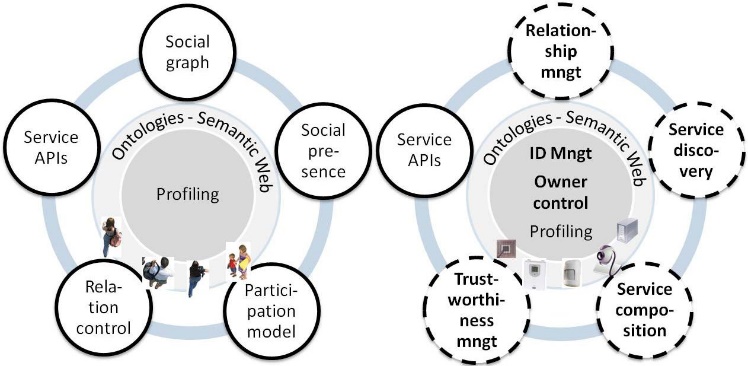
II. THE SOCIAL STRUCTURE In this section, we analyze the types of social relationships in which things can be engaged. Like for human beings, ﬁrst form of socialization among objects we foresee is a parental object relationship, deﬁned among similar objects, built in the same period by the same manufacturer (the role of family isplayed by the productionbatch). This relationship is easily implemented during the item production, it will not change over time and is only updated by events of disruption/obsolescence of a given device. Moreover, objects can establish co-location object relationship and co-work object relationship, like humans do when they share personal (e.g., cohabitation) or public (e.g., work) experiences. These relations are determined whenever objects (e.g., sensors, actuators, RFID Tags, etc.) constantly reside in the same place (e.g., to offer home/industrial automation services) or periodically cooperate to provide a common IoT application, such as emergency response and telemedicine. These relationships are established as part of the initialization/implementation of either a “location-based application” proﬁle or a “situation-based application” proﬁle. Changes are frequent and usually based on time duration of co-location/coworking, frequency of the interaction, and reputation. These are the sort of relationships considered in [8]. A further type of relationship is deﬁned for objects owned by the same user (mobile phones, game consoles, etc.). We name this ownership object relationship. Associating one another all devices of the same user is already a common procedure. A ownership object relationship is the logical generalization of this concept through a richer device proﬁle. The last relationship is established when objects come into contact, sporadically or continuously, for reasons purely related to relations among their owners (e.g., devices/sensors belonging to friends). We name this social object relationship. Similarly to people exchangingtheir contacts (phone numbers, e-mail addresses, etc.), the device, if properly authorized, autonomously exchanges its social proﬁle. The driving idea is that devices with similar characteristics and proﬁle can share best practice to solve problems already faced by “friends”. Policies, exploiting ad-hoc metrics, measure the opportunity of maintaining a given relationship. Accordingly, the relationships among objects in the SIoT evolve towards social structures that need to be studied to maximize the beneﬁts of the SIoT in service discovery and exploitation. Sociology, Anthropology, or Cognition studies can provide useful hints in this direction. Alan Fiske “relational models” theory [9], furnishes four basic relational structures that can be applied to objects as well. In Communal sharing relationships, equivalence and collectivity membership emerge against any form of individual distinctiveness. These can be deﬁnitely associated with behaviors of objects that have collective relevance only. This is, for example, the case of “swarms” of objects for which is only important the service that the whole swarm can provide to users. Equality matching, based on egalitarian relationships characterized by in-kind reciprocity and balanced exchange, may represent all forms of information exchange among objects that operate as equals during the IoT service provision while maintaining their individuality. In communal sharing relationship the service is associatedto the wholegroup;in thissecond case each object has associated a service that it advertises. Authority ranking relationships are asymmetrical, based on precedence,hierarchy,and status. These are established among objects of different complexity and hierarchical levels (RFID reader and tags, Bluetooth master and slaves, etc.) exchanging

Fig. 1. Basic components of social network platforms for humans (on the left) and for objects (on the right).

information asymmetrically. The service advertised is usually associated to the group or to the object of the highest rank. Market pricing relationships are based on proportionality, with interactions organized by referring to a common scale of ratio values. They can be established among objects working together to achieve a mutual beneﬁt. In many IoT applications, this implies that the participation in this relationship is considered only when it is worth the while to do so.

III. SIOTARCHITECTURAL MODEL In Fig. 1 (left side) we show a common architectural model of SNS for humans [10]. This is not applicable to the SIoT, but must be modiﬁed to take its speciﬁc features into account. In this context, criteria to consider are related to the main SIoT objectives, that is, object-related service discovery and composition as well as object trustworthiness management.

A. Components of SIoT Accordingly, Fig. 1 compares the main components of a SNS and of SIoT. Differences in the novel architecture are shown with bold fonts and dashed contours. Three basic components can be envisioned: ID management (ID): to assign an ID that universally identiﬁes all object categories and to maintain existing object identiﬁcation schemes, a simple XML-based protocol can be implemented, which allows to specify the ID mechanism adopted other than the ID itself. This system includes at least: IPv6 addresses, Universal Product Code (UPC), Electronic Product Code (EPC), Ubiquitous code (Ucode), OpenID, URI. Object proﬁling (OP): it includes static and dynamic information about the object. Objects should be organized into classes on the basis of the main object features. Owner control (OC): speciﬁc policies need to be deﬁned by the owner to rule any possible operation the object performs (information to share, allowed relationships, etc.). To this aim, different security and access control policy deﬁnition languages already available can be used. Owner control includes the SNS functionality of the Relation control component. We foresee the satellite componentslisted in the following1. Relationship management (RM): it introduces into the SIoT the “intelligence” that allows objects to start, update, and

1Social graph is a minor functionality of the SIoT. However, this tool may still be implemented to allow humans to visualize objects relationships.

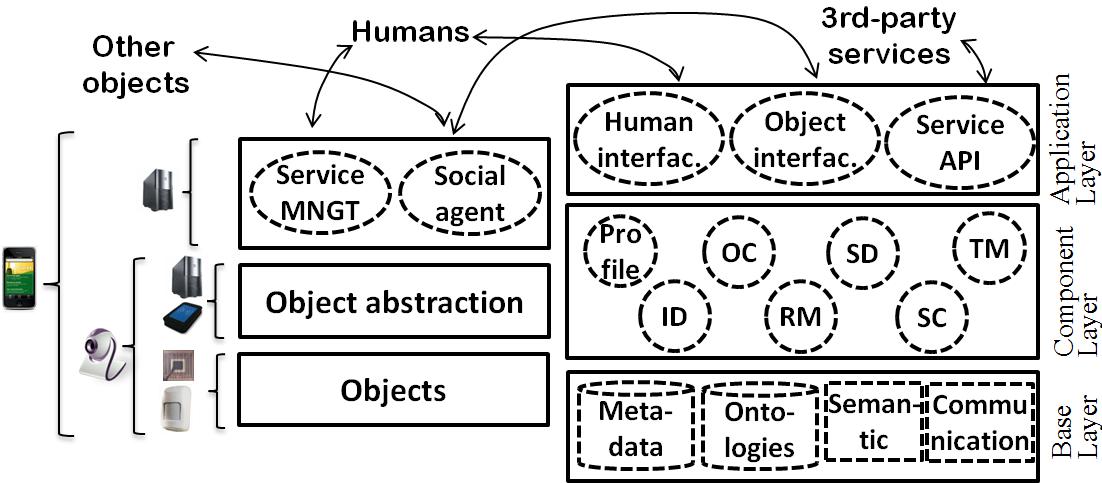


Fig. 2. Architecture for the SIoT: client side (left) and server side (right).

terminate relationships. The selection of which friendship to accept is based on human control settings. The rules described in Section II are implemented in this component. Service discovery (SD): it replaces the Social presence and is ﬁnalized to ﬁnd which objects can provide the required service in the same way humans seek for friendships and information in the SNSs. Indeed, to discover the service, the object queries its social relationship network. Service composition (SC): it enables the interaction among objects and replaces the Participation model. The Service discovery exploits the object relationships to ﬁnd the desired service, which is then activated by this component. Both a reactive or a proactive approach to service composition are envisaged. This component will also include the functionality of crowd information processing, to process the information obtained from different objects and obtain the most reliable answer to a query on the basis of different visions. Trustworthiness management (TM): this is aimed at understanding how the information provided by other members has to be processed. Reliability is built on the basis of the behavior of the object and is strictly related to the RM module. Trustworthiness can be estimated by well-known SNS notions, such as centrality and prestige, and, again, built on the basis of the object social structure we propose. Service APIs: this component is analogous to the one required in SNSs.

B. The SIoT architecture We propose a system architecture made up of three main layerson the server side (Fig. 2). TheBase layer encompasses: the database for storage and management of data with relevant descriptors, ontologies database, semantic engines, and communications. The Component layer hosts tools for basic and satellite component implementation. Interfaces to objects, humans, and third-party services are in the Application layer. On the object side, the ﬁrst architectural layer – named the Object layer – is where the physical objects are located and are reached through their speciﬁc communication interfaces. An Object abstraction layer is thus needed to harmonize the communication of the different devices through common languages and procedures. In the case of elementary objects, such as an RFID-tagged object, a gateway is required to implement this abstraction layer, while for more complex objects this layer can be implemented in the object itself. In the third layer, the Social agent is devoted to the communicationamongobjectsandwith SIoT serversto update proﬁle and friendships and to discover/request services from the social network. Finally, the Service management is the interface of humans to control the object behavior in the SIoT.

IV. CONCLUSION In this paper we introduced the novel concept of Social Internet of Things (SIoT), based on a sort of social relationship among objects, analogously to what happens for human beings.Currently,we are statistically analyzingthe structureof the SIoT network through simulations that model the mobility of objects andtheir relationships.Preliminaryresultsshow that most of SIoT features are similar to those observed in social networks of humans. Based on the results of this analysis, we will investigate whether network navigability can be achieved in SIoT and we will identify techniques in the set up of social links that can improve navigability. Possible application scenarios are those where objects share best practices. For instances, PCs in the same local area network can establish social relationships that can be used to ﬁnd solutions to common setting problems, such as those related to the conﬁguration of a tricky network printer or an AP. Similarly, cars of the same brand, model and year can provide information about possible solutions to frequent and common mechanical/electrical concerns. In other scenarios, devices that visit the same geographical area can establish friendships to exchange useful information on the physical world. This is the case of handsets that provide data on the radio coverage to new visitors so as improve their connectivity service (providing useful information to the user/owner).

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, “The Internet of things: a survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] L. Zheng et al., “Technologies, applications, and governance in the Internet of things,” Internet of Things - Global Technological and Societal Trends. River Publisher Ed., 2011.

[3] www.internet-of-things-research.eu, “European research cluster on iot.”

[4] J. Kleinberg, “The small-world phenomenon: an algorithmic perspective,” in Proc. ACM Symposium on Theory and Computing, 2000.

[5] D. Guinard, M. Fischer, and V. Trifa, “Sharing using social networks in a composable web of things,” in Proc. IEEE PERCOM, 2010.

[6] H. Ning and Z. Wang;, “Future Internet of things architecture: like mankind neural system or social organization framework?” IEEE Commun. Lett., vol. 15, no. 4, pp. 461–463, 2011.

[7] M. Kranz, L. Roalter, and F. Michahelles, “Things that Twitter: social networks and the Internet of things,” in What can the Internet of Things do for the Citizen (CIoT) Workshop at Pervasive, May 2010.

[8] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H. Gellersen, “Smart-its friends: a technique for users to easily establish connections between smart artefacts,” ACM Ubicomp’01.

[9] A. P. Fiske, “The four elementary forms of sociality: framework for a uniﬁed theory of social relations,” Psychological Review, vol. 99, 1992.

[10] D. M. Boyd and N. B. Ellison, “Social network sites: deﬁnition, history, and scholarship,” J. Computer-Mediated Commun., vol. 1-13, 2007.

**SIoT：给物联网赋予社会结构**

**摘要—**物联网的实际发展需要解决与物联网服务发现和组成相关的重大问题。本文提出了一种可能的方法来解决这些问题。我们引入一个新的“智能对象”的社会网络范式，即事物的社会网络（SloT），基于对象之间的社会关系的概念。以下对象之间的一个可能的社会结构的定义，对缝隙的实现初步的体系结构。通过缝隙的范式，人和设备的能力去发现、选择、物联网中的与他们的服务对象是使用增强。此外，水平的可信性是能够引导数十亿的对象，这将拥挤未来的物联网之间的相互作用。

**术语索引—**物联网，社会网络，云计算。

**I. 简介**

一个有效的和可靠的物联网的实现需要一个复杂的架构，考虑到现实世界的问题，传输数据，并管理相关的服务，以建立应用程序的定义[ 1 ]。最近，主要建筑方案已经提出在ITU，EPCglobal的casagras倡议和UID，研究组[ 2 ]。这个列表并不详尽：从欧共体资助项目由印度教育资源中心集群协调建议过多必须加上[ 3 ]。从这些努力的分析，新兴的优先事项是：（i）通过感知和网络层面的操作，使整个连接到互联网的东西，（ii）提供中间件和应用程序的功能和协议，以减轻开发的东西相关的服务。我们的贡献适合后一行的研究，我们确实提出了一个创新的对象之间的互动范式。基本的想法成为一个“智能对象”的社会网络的定义、命名事物的社会网络（SloT）。在类比与社交网络服务（SNS）对人类来说，新范式介绍社会关系的概念对象之间的优势存在如下几种可能：

·给物联网赋予一种结构可以形成所需的保证网络的导航结构, 因此，该对象和服务发现被有效地执行和可扩展性，保证在人类社会网络(举例见[4]).

·通过设计模型的扩展用法来研究社会网络作品，以解决物联网相关问题。(与互联对象的广泛网络本质相关).

2011年6月23日收到了手稿。副编复审这封信之后, 批准I- R. Chen是它的出版。

L.Atzori与卡利亚里大学，意大利 (e-mail: l.atzori@diee.unica.it).

A.Iera与卡拉布里亚大学，意大利 (e-mail: anto-nio.iera@unirc.it).

G.莫拉比托与卡塔尼亚大学，意大利 (e-mail: gia-como.morabito@diit.unict.it).

创造一个用来改变物物交互的信誉等级,那便是”朋友”。我们的是一个分布式的方法，这将保证更高的可扩展性和较好的频繁反应参与进来的物联网对象的状态变化特征。作为一个物联网整体的结构缝隙鉴定，根据“人”的社会网络模型是相当的常识；虽然并不是完全一样的事情。

在物联网的背景下，社会网络的探索已经调查研究一段时间了[ 5 ]。在本文中，它提出了利用（人类）的社会网络关系，通过某一智能硬件来共享资源（智能的东西，支持他们的所有者的朋友支持的Web服务）。此外，这项研究也尝试奠定了基础，建立一个所谓的无处不在的物联网建筑灵感来自人类社会组织。作者[ 7 ]调查的潜力，结合社会和技术网络，并讨论所谓的“社会技术网络”的含义，在物联网的背景下。在Pachube平台接近一个对象的社会网络的概念，使开发人员能够将传感器数据到Web构建应用程序。该平台不允许的对象是社会团体自发形成的，为人类的利益，但没有他们的干预。事实上，SIoT更关注后一方面，与社会的联系介绍物联网设备的概念[ 7]。一个互补的角度被认为在[ 8 ]，其中重点是解决方案，使智能无线设备-主要是无线传感器-建立临时连接和他们的主人来控制这样一个过程。

我们提出的方法不同于文学的三个主要原因。

1. 我们致力于建立然后利用事物之间的社会关系，而不是在他们的所有者之间。持有者的可能妥协可

以预见，但原则它必须发挥关键作用，因为它发生在新的以物联网为基础的应用中。

1. 通过社会关系，事情可以爬行物联网，发现服务和资源，这提供了一个分布式的解决方案，预计将是有效的，高效的，最重要的是，减轻人类这样做。
2. 物联网架构的设想不仅仅是一个围绕物联网概念的服务平台，没有一个真正的平台为合适的组织介绍应对代替人类的对象存在的SNS网站。

**II. 社会结构**

在本节中，我们分析了可以参与的社会关系的类型。与人类一样，我们预见到的对象之间的社会化的第一种形式是父母对象关系，缩小范围来讲，指的是由同一个制造商在同一时期内建立的（家庭的角色由生产批次扮演）。

这种关系是很容易实现的项目生产过程中产生的，它不会随时间而改变，它的变化更替有两方面原因,要么是事件中断,要么是一个给定的设备陈旧。

此外，对象可以建立合作位置的对象关系船和合作的工作对象的关系，像人类一样，当他们分享个人（如同居）或公众（如工作）的经验。这些关系被确定时，对象（例如，传感器，执行器，RFID标签等）就会不断驻留在同一个地方（例如，提供家庭/工业自动化服务）或定期合作，以提供一个共同的物联网应用，如应急响应和远程医疗。这些关系都是建立在初始化/实施一个“位置应用”形或“情境”中的应用简介。变化是频繁的，通常的基础上的持续时间的合作位置/合作工作，频率的互动，和声誉。这些是在[ 8 ]中考虑的那种关系。

对于同一用户拥有的对象（手机、游戏机等），定义了一种更深层次的关系类型。我们命名这种所有权对象关系。将同一用户的所有设备关联起来已经是一个常见的事情.。所有权对象关系是这个概念的逻辑概括，通过更丰富的设备外观。

最后的关系是建立在对象偶尔或持续的接触，纯粹的原因是因为存在了相关关系的持有者（例如，设备/传感器属于朋友）。我们将此命名为社会对象关系。相似的,人们如果交换他们的联系人（电话号码，电子邮件地址等），而且该设备也有相应授权，那么设备就会自主交流其社会概况。驱动理念则说的是，具有相似的特性和外形的设备可以分享最好的做法，以解决问题已经面临的“朋友”。政策，利用特设的指标，衡量维持一个给定的关系的机会。

因此，在物联网的对象之间的关系的发展对社会结构需要研究的最大好处的缝隙中的服务发现和开发。在这个方向上,社会学，人类学，或认知研究可以提供有用的提示。Alan Fiske的“关系模型”理论[ 9 ]，提供四种基本关系结构,而这些结构都可以应用到对象上。在共享关系中，等价和集体成员出现反对任何形式的个人独特性。这些可以肯定与只有集体相关的对象的行为相关。例如，这是“群集”的对象的情况下，这是唯一重要的服务，这个服务的整个群体可以提供给用户使用。

基于平等关系的平等匹配通过这种互惠和平衡交换而越来越特征化，这种匹配可以代表所有形式的对象之间的信息交换，而这种交换又是物联网在提供服务的过程中保持自己的个性的唯一手段。在共享关系的服务中与整个集团相关；在此情况下，每个对象都有关联的一种服务，这便是它的广告。

基于优先级，层次和地位的权威排名关系是不对称的。它们是建立在不同的复杂性和层次的对象（RFID读写器和标签，蓝牙主机和附属机等）之间的交换广告服务通常与集团或最高级别的对象相关联。

市场定价关系是基于比例的，与相互作用的组织参照一个共同的比例值。它们可以建立在对象之间, 在一起工作实现互利。在众多的物联网应用面前，这意味着这段关系只有被认为是当它值得这么做才会参与。

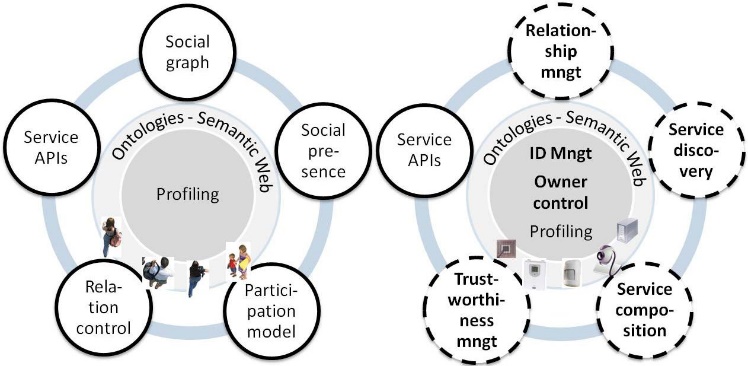


Fig 1. 适用于人（左图）和对象（又图）的社会网络平台的基本组件

上图可以明显看出，信息的不对称。广告服务通常涉及到团队或接触对象的最高层级。市场价格关系是基于比例性的，互动性的，并参考一个公共的比值而定的。他们能够建立在在一起工作的对象中并实现一种互惠互利。许多物联网应用中都有这样的体现，这也就暗示了与此相关的角色只有在值得去这么做的情况下才会去考虑。

**III. 社会架构模型**

图一(左侧) 我们展示一个对于人类的常见的SNS结构模型 [ 10 ]。这个并不适用于物联网，必须进行修改，以考虑其具体特征录入报告。在这样的背景下，标准的考虑是主要的物联网相关的目标，是相关的服务发现和组合对象以及对象的诚信管理。

**A.社会物联网的组件**

因此，图1比较了SNS的主要组件和缝隙。在新的架构的差异显示与大胆的字体和虚线轮廓。

三个基本组成部分的设想：

ID管理（ID）：分配一个ID，普遍标识所有的对象类别，并保持现有的对象识别方案，一个简单的基于XML的协议可以实现，它允许指定ID机制以外的ID本身。这个系统至少包括：IPv6地址，通用产品代码（UPC），电子产品代码（EPC），无处不在的代码（UCODE），OpenID，URI。

对象分析（OP）：包括对象的静态和动态形成。对象应该根据对象的主要特征组织成类。

所有者控制（OC）：具体政策需由业主定义规则的任何可能的操作对象执行（信息共享，允许关系，等）。为了这个目的，不同的安全和访问控制策略定义语言已经可以用。业主控制包括控制组件的SNS功能的关系。

我们预见到在以下列出的卫星部件。关系管理（RM）：介绍到缝隙的“情报”，允许对象开始，更新，和终止的关系。

社交图是一个小功能的缝隙。然而，这个工具仍然可以实现，让人类可视化对象的关系。

选择接受哪种友谊是基于人类控制设定。第二节中描述的规则在这个组件中实现.。

服务发现（SD）：它取代了社会的存在和最终找到的对象可以提供所需的同种方式的人的服务,即在SNS上寻求友谊和社交网站信息。事实上，为了发现服务，对象就要查询它的社会关系网络。

服务组合（SC）：它确保了对象之间的交互，并取代参与模型。服务发现利用对象关系找到所需的服务，然后激活该组件。无论是活性的还是主动的方式实现的服务组合都还是设想。该组件还包括人群信息处理的功能，处理从不同的对象获得的信息，并获得最可靠的答案从而在不同的愿景的基础上进行查询。

诚信管理（TM）：这是针对与理解信息如何由其他成员提供,而这些信息是这些成员必须处理的。可靠性是建立在对象的行为的基础之上的，而且与RM模块密切相关。诚信可以通过知名SNS概念估计，如中心性和威望，以及我们提出的再次建立在对象的社会结构的基础上。

服务API：这部分是类似于一个在SNSs的要求。

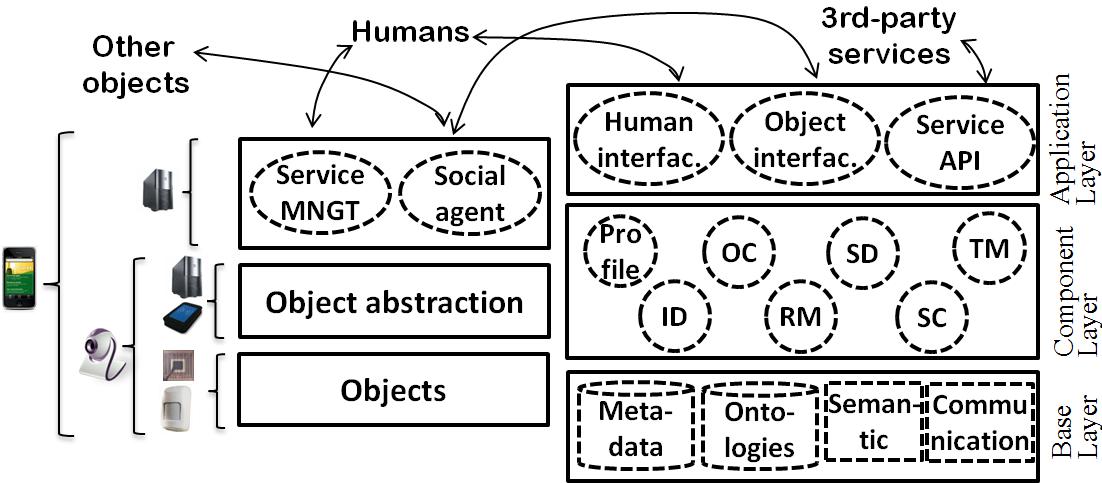


Fig. 2. 社会 物联网 的 结构：客户端（左） 服务端（右）

**B.社会物联网的结构**

我们提出了一个由三个主要层的服务器端的系统架构（图2）。基础层包括：用于数据与相应的描述符，本体语义引擎数据库，存储和管理数据库，通信。组件层为基本组件和卫星组件实现工具。在应用层中，对象、人类和第三方服务的接口。

在对象方面，第一个体系结构层（命名为对象层）是物理对象所在的位置，并通过其特定的通信接口到达。因此，需要一个对象抽象层，通过公共语言和程序的通信来协调不同的设备。在基本对象的情况下，如RFID标签对象，网关需要实现这个抽象层，而对于更复杂的对象，该层可以在对象本身实现。

在第三层，社会代理是专注于对象之间的通信和SIoT服务器更新剖面以及友谊和发现/请求服务的社会网络。最终，服务管理便成为了人类的接口控制对象行为的物联网。

**IV.结论**

在本文中，我们介绍了事情的社会物联网络的新概念（SIoT），一种基于对象之间的社会关系，类似于人类发生了什么事情。目前，我们的统计性的分析了,通过模拟SIoT网络的结构模型的对象和它们之间的关系的流动性。初步结果表明，大多数的SIoT特征类似于人类社会网络观察。在这一分析结果的基础上，我们将决定是否在网络导航领域中进行投资,它可以实现在SIoT和我们将识别技术中建立社会联系，从而提高适航性。

可能的应用场景是那些对象共享实践最优的场所。例如，在同一局域网中的个人电脑可以建立社会关系，可以用来解决常见的设置问题，如那些相关的配置一个棘手的网络打印机或AP。类似的如:同一品牌，型号和年份的汽车可以频繁提供有关和常见的机械/电气问题的可能解决方案的信息。在其他情况下，访问同一地理区域的设备可以建立友谊, 在现实的物理世界中交换有用的信息。这是有手机的情况下，提供无线覆盖的数据，覆盖新的访问者，以提高他们的连接服务（提供有用的信息给用户/所有者）。