基于 Android 的模拟炒股 APP

摘　要

随着经济的快速发展、虚拟经济的发达以及股市日渐成熟，人们在是日常生活中的投资方式也日趋多样化。股市作为当下十分火爆的投资渠道，想要了解并进行一定投资的人也越来越多。但是学习股市规则以及规律是有代价的，且投资过程具有不可逆性。因此初学者需要一款移动端的模拟炒股应用，可以随时随地地了解股市行情，并进行合理且几乎零代价的投资。

Android 平台作为目前最大的移动端平台，在用户数量上处于绝对的优势。开发一款安卓平台的软件是十分顺应市场的潮流的。本系统是基于 Android 的开发平台，使用 KOTLIN 语言，使用 IntelliJ IDEA 作为开发工具，在 Windows10 系统上进行开发。系统还使用多种第三方库主要实现了股票行情展示、买入卖出、钱包等功能。

本系统的最大优势在于市场前景明朗，能满足让不同领域的用户快速上手的需求。同时，强大的第三方图表库也使得行情更加清晰。略为可惜的是，未对系统界面进行美观处理而且并设计一些人机交互性较强的功能。除此之外，还未实现把多支股票的统计数据展现给用户。希望在未来的时间中能把这些功能一一实现，从而完善本系统的功能。

关键词：Android 开发；模拟炒股；kotlin 语言；mpandroidchart

Android-based Simulation Stock Trading APP
Abstract

With the rapid development of economy, the development of virtual economy and the maturity of stock market, people's investment methods in daily life are becoming more and more diversified. As a very popular investment channel, more and more people want to know and make certain investments. However, learning the rules and regulations of the stock market has its costs, and the investment process is irreversible. Therefore, beginners need a mobile simulated stock trading application, which can understand the stock market anytime and anywhere and make reasonable and almost zero-cost investments.

As the largest mobile platform at present, Android platform has an absolute advantage in the number of users. The development of an Android platform software is very in line with the trend of the market. This system is a development platform based on Android, using KOTLIN language and IntelliJ IDEA as a development tool, and is developed on Windows10 system. The system is also based on a variety of third-party libraries, mainly realizing the functions of stock market display, buying and selling, wallet, etc.

The biggest advantage of this system is that the market prospect is clear and can meet the needs of users in different fields to get started quickly. At the same time, the powerful third-party chart library also makes the market clearer. It is a little pity that the system interface has not been treated aesthetically and some functions with strong human-computer interaction have been designed. In addition, it has not yet realized to show the statistical data of many stocks to users. It is hoped that these functions can be realized one by one in the future so as to perfect the functions of the system.

Key words: Android development; simulated stock speculation; kotlin language; mpandroid chart

目录

插图目录

表格清单

引言

随着经济的快速发展、虚拟经济的发达以及股市日渐成熟，人们在是日常生活中的投资方式也日趋多样化。股市作为当下十分火爆的投资渠道，想要了解并进行一定投资的人也越来越多。但是学习股市规则以及规律是有代价的，且具体不可逆性。但是投资者所能使用的炒股软件无一不是付费使用的。因此，开发一款可以模拟炒股，能让用户实时查看股票行情且买入卖出的软件是十分有必要的。

Android 平台作为目前最大的移动端平台，在用户数量上处于绝对的优势。开发一款安卓平台的软件是十分顺应市场的潮流的。由于移动端的便于携带，轻量，人们使用移动端的时间已经超过 PC 端了。在安卓平台开发的软件更具有市场前景，且能让使用者更加方便地学习和锻炼炒股策略。本系统是基于 Android 的开发平台，使用 IntelliJ IDEA 作为开发工具，在 Windows10 系统上进行开发。在开发语言上选用了 kotlin 语言，这门语言作为近几年刚被谷歌宣布为安卓官方开发语言。当然在开发过程中，由于要解决一些具体的业务问题——json 字符串解析、图表构建、网络服务，也使用了许多第三方库如——mpandroidcahrt，anko，okhttp3，gson 等。

本次开发主要实现了股票行情功能、买入卖出功能、钱包功能。预期能让用户在查看到想要股票的最近一段时间的实时行情，且能进行一些基本的买入卖出的操作。当然查看当前用户所持股状态以及钱包状态也是不可缺。

就当下而言，android 系统在市场上有很大的份额，其使用群体也是相当可观。而且，股票投资这项技能在如今的生活也越发重要。对于那些不懂股市的人来说，有一款可以免费锻炼投资策略的软件具有重要绪论。

第1章 绪论

1.1 课题的研究意义

在当今的生活中，经济越来越来向着虚拟经济发展了，而股票作为现今社会存在的一种经济形式在经济领域发挥着越来越重要的地位。但是在实际操盘时，股民大会有诸多困难如：资金不足、经验缺少、投资策略差、心态差会使实际炒股中并不能为自身谋取相当的利益，而模拟炒股软件可以为股民或者学生提供了解股市规则和投资策略训练的地方，且不需要任何的代价。百度百科对模拟炒股软件的解释大致为：无代价地利用软件，进行股票的模拟交易或执行投资策略。

模拟炒股的交易规则与正常股市一样，但它是根植于虚拟平台，并实现股票买卖的一种炒股手段，一般设置市场每位选手起始资金为若干人民币，人们可以用系统提供的虚拟资金进行股票委托买卖。

由于模拟炒股使用的是虚拟货币，所以使用者在心态上可以说十分轻松。有趣的是轻松的心态反而更容易赚钱，现实中因为资金的压力，价格造成的波动往往会使投资者紧张，从而导致错误的判断。所以很多有要模拟炒股中熟悉市场后去到真实的 A 股市场时往往会吃亏，原因就在这里。

模拟炒股软件最大的好处在于激励办法，为了让更多的新股民或者准股民更认真更负责地在模拟炒股系统中学习到炒股知识，至于为什么要有模拟炒股以及模拟炒股的意义，说到这里应该很清楚了，就是为了演练自己的买卖策略，等技术成熟后再回去真实的 A 股市场里，无疑就提高了赢利的机会。特别是现在的高校大学生们，现在可能手上没有足够的资本，又想了解股市，学习炒股，为日后做准备，那么，模拟炒股就是最好的办法了，不是为了钱，仅仅是为了让自己多知道些别人不知道的金融知识。

1.2 相关领域开发及应用现状分析

在移动互联网快速发展的今天，智能手机已经成为了每个人的生活必需品，而且移动端使用占比也愈发大。因此，人们的消费理念也从 pc 端渐渐转移到手机上。随之而来的是，手机 app 开发的巨大商机，各种商家都想借助手机软件来提高自身与客户之间的联系，以带来更多的利益。

从 2013 年到现在，手机软件的发展从简单单一到现在的功能复杂且强大。但其主要由三部分构成：为企业开发应用、开发通用应用以及游戏开发。而与本系统相关程度最大的就是通用应用了，它更属于一种学习型软件。在目前国内开发环境下，因为利润少、维护成本高，所以开发学习类软件并不多。但是随着股市在国内的快速发展，了解股市并学习会使用已经渐渐成为一项必不可少的基本技能。通过以上分析可知，一款免

费的模拟炒股软件的应用前景是十分明朗的。

1.3 课题的主要研究内容

本次的毕业设计是为了在安卓移动端设计一款可以实时查看行情，进行模拟炒股的APP。由于国内学生或想了解炒股的人并不在资金上并不充裕，一款可以无乎无代价地锻炼炒股策略的软件是十分有必要的。本次的毕业设计主要是完成一款基于移动端的模拟炒股软件，使用 kotlin 语言编写并用 genymotion 器对系统的功能测试和体验来完善系统。其主要研究内容为：

1.大盘各股走势 K 线图

2.行情界面：显示有涨幅榜、跌幅榜 、分时涨幅以及各类数据统计

3.模拟交易：委托交易、撤销买单、查询

4.服务器数据：外网比特币交易信息库

第2章　开发工具及相关技术介绍

2.1　Android 操作系统简介

2.1.1　Android 简介

Android 系统：Google 于 2007 年正式发布，并可以应用于手机或者平板上的操作系统。最新的系统版本为 9.0，SDK 版本也达到了 28 了。安卓系统以其开源特性，和良好的特性，使得愈多的开发者选用安卓系统进行软件开。

2.1.2　Android 的特点

Android 三大特点——界面、应用程序、内存管理。

默认的界面支持用户直接操作，可通过对应现实动作作出输出，例如滑动、点击、捏动和反射挤压，随着虚拟键盘，以操控屏幕上的对象。

应用程序（简称 apps，即 applications）是扩展设备功能的软件，都是使用 Android 软件开发工具包(SDK)编写的。17 年 5 月，谷歌宣布安卓应用程序编写官方语言为 kotlin。

因为 Android 设备基本采用电池供电，因此 Android 旨在管理流程以将耗电降到最低，当用户的应用程序长时间未使用时，系统会暂停对其操作，虽然可以在关闭期间立即使用，但它并不会使用电池电源或 CPU 资源。而当内存不足时，系统又会自动将长时间非活跃状态下的进程。

2.1.3　Android 的系统架构

Android 的系统架构与 windows、Linux 等系统相似，在系统架构上也采用了分层的形式，从低层到高层共 5 层，分别是应用程序框架层、系统运行库层、Linux 内核层以及应用程序层。

一般情况安卓的最新版本会和同一系列的核心应用程序打包在一起发布，这些核心应用程序包括客户端、SMS 短消息程序和日历等常用程序。谷歌也开放了访问这些核心应用程序的 API 框架，使得开发者这些内容。Android 中也包含一些 C/C++语言编写的库，Android 系统中的许多组件都可以调用这些库，同时它也通过应用程序框架为开发者们提供功能服务。常用的核心库有系统 C 库，媒体库，Surface Manager，LibWebCore 等。

2.2 Android 基本组件介绍

Android 的基本有活动、服务、广播接收器、内容提供商。

在安卓中，Activity 是所有程序的根本，所有程序都运行在 Activity 之中，Activity 可以算是开发者遇到的最频繁，也是 Android 当中最基本的模块之一。在 Android 的程序当中，Activity 一般代表手机屏幕的一屏。如果把手机比作一个浏览器，那么 Activity 就相当于一个网页。在 Activity 当中可以添加一些 Button、Check box 等控件。可以看到 Activity 概念和网页的概念相当类似。

Service 是 android 系统中的一种组件，它跟 Activity 的级别差不多，但是他不能自己运行，只能后台运行，并且可以和其他组件进行交互。

在 Android 中，Broadcast 是一种广泛运用的在应用程序之间传输信息的机制。而 BroadcastReceiver 是对发送出来的 Broadcast 进行过滤接受并响应的一类组件。

Content Provider 是 Android 提供的第三方应用数据的访问方案。

2.2 koltin 语言

Kotlin 是一个用于现代多平台应用的静态编程语言，由 JetBrains 开发。 Kotlin 不仅可以编译成 Java 字节码，也可以编译成 JavaScript，十分方便在没有 JVM 的设备机器上运行 kotlin。

Kotlin 已正式成为 Android 官方支持开发语言。

2.3 相关技术

2.3.1 anko 技术

Anko 是 JetBrains 公司开发的一个功能强大的库。它的主要功能是使用代码的形式来替代以往的 xml 形式来生成 UI 布局。

2.3.2 mpandroidchart 技术

Mpandroidchart 是 github 上 16.6k star 的的图表库，能够使用它在安卓上画柱状图 、拆线图、饼状图等等，它还内置了各种监听器。本文主要使用该库画 K 线图及各种图形。

2.3.3 circleimageview 技术

一个十分快速的图层去展示主页资料的技术，支持将图片自动蒙版为圆形图片。

2.3.4 gson 技术

Gson 支持 json 字符串到对象的相互转换，本文主要采用该技术对从网面获取的数据进行解析。

2. 4 sqlite 数据库

SQLite 是遵守 ACID 的关系数据库管理系统，它包含在一个相对小的 C 程式库中。它是 D.RichardHipp 建立的公有领域项目。不像常见的客户端/服务器结构范例，SQLite 引擎不是个程序与之通信的独立进程，而是连接到程序中成为它的一个主要部分。所以主要的通信协议是在编程语言内的直接 API 调用。这在消耗总量、延迟时间和整体简单性上有积极的作用。整个数据库（定义、表、索引和数据本身）都在宿主主机上存储在一个单一的文件中。它的简单的设计是通过在开始一个事务的时候锁定整个数据文件而完成的。

2.5 IntelliJ IDEA Community 简介

intellij idea 是 java 语言集成开发环境，也是业界公认的最好的开发环境之一，具有智能代码助手、代码自动提示、重构、J2EE 支持、Ant、JUnit、CVS 整合、代码审查、创新 GUI 设计等特色，新版本内置了 decompiler，同时还新增了扩展代码检查功能等。

第3章　系统需求分析

3.1 系统的开发目标

本系统旨在了解并掌握开发 Android 应用的方法和具体流程，使用 jetbrains 旗下 IntelliJ IDEA 软件进行本系统开发。系统开发目标是开发一套能满足实际应用需求的模拟炒股系统。

3.2 系统的需求分析

系统的需求分析包括功能需求分析、性能需求分析等。

3.2.1 功能需求分析

基于 Android 的模拟炒股软件是针对 Android 平台的用户，可以进行简单模拟炒股，查看实时行情的软件。

通过需求分析，以下是本系统应当具有的功能：

（1） 用户管理功能。整个系统中使用普通用户和管理员两个级别的权限进行管理。

（2） 实时行情显示功能。需要对实时行情数据进行图表化展示给使用。

（3） 基本股票交易功能。模拟炒股者可以任意支股票进行买入、卖出，并查看当前钱包状况。

（4） 股票评论功能。在每支股票下加入可以评论功能，以方便投资者对该股票有一定的理解，合理投资。

3.2.2 性能需求分析

由于使用者的文化水平、阶层、领域不尽相同，分析该系统应该具有以下性能：

（1） 实时行情获取时间快、且数据准确。

（2） 系统运行流畅，不能让使用者在使用中产生卡顿感，或者异常退出等情况。

（3） 系统界面友好，UI 交互正常。

3.3 系统的可行性

可行性研究的主要目的是尽量使用最小的代价和短的时间确定需求问题能否解决。系统可可行性分析的主要内容是技术可行性、操作可行性、经济可行性以及法律可行性。

3.3.1 技术可行性

具体地从技术的可行性进行分析，首先需要一台闲置计算机，并且拥有较高的硬件基础以方便安装占用大内存的模拟器，除了这些之处还要有足够空间安装数据库和编程工具软件等。当然，开发人员还要掌握一定的计算机知识、安卓编程经验、以及熟悉并会使用一门安卓编程语言（如 java 或 kotlin）

技术是系统的敲门砖，其主要过程是根据客户提出的需求、系统的性能和执行的约束条件，从技术角度研究系统的可行性。该系统使用 kotlin 和 Android 技术，并用使用 Sqlite 数据库以及一些第三方库。从这些分析来看，本系统在技术上是可行的。

3.3.2 操作可行性

在当今社会中，科学技术水平越来越高。而随着智能手机的更新迭代，人们对手机上的新功能也越来越能快的上手。从操作难度看，本系统对于新手十分友好，系统使用的全部控件都是常见且简单的。并且在每一个界面都有相应的文字提示使用，看完提示，系统的操作就会非常简单。由此可以看出本系统在操作性上是可行的。

3.3.3 经济可行性

经济可行性研究的主要是目的是从成本收益的角度，分析项目在经济上是否为可行的。它通过估算或估计项目的成本，包括开发成本、将来的维护成本、运营成本，在数值上是否高于项目的预期经济收益，从而判断软件能否给用户带来足够的经济收益。

本项目开发及维护均为开发者本人，而从收益来看，并不收取任何费用。这二者虽前者大于后者，但在经济上由于成本过低的原因大体可行。

3.3.4 法律可行性

本系统的开发过程完全独立，没有任何侵犯任何知识产权，只参考了一些 github 上的项目和 csdn 的网页。因此本系统具有法律可行性。

第4章　系统概要设计

概要设计也叫总体设计，其主要目的是确词定系统的数据流图和数据字典，以及系统整体软件结构，划分软件体系结构的各子系统或模块，确定它们之间的关系。确切地说，概要设计就是完成体系结构设计、界面设计和数据设计。

4.1 体系结构设计

体系结构设计是软件设计早期应该做的事，其任务可分为两点：
一、提供软件设计师能预期的体系结构描述。
二、数据结构、文件组织、文件结构体现了软件设计的早期抉择。

经过对软件需求的分析，最终决定选定本系统结构为基于 C/S 模式的分布式结构。由于股票系统的每支股票的具体数据以及服务信息都会在服务器保存，而且使用者的手机软硬件配置不尽相同,并且要求信息共享。使用基于C/S模式的分布结构的好处在于，可以提供众多使用者一起完成对服务器数据的获取，而且可以让使用者在客户端完的操作的信息都上传到服务器。最终的数据的处理都将集中在服务端。

4.2 模块化设计

模块化设计就是通过一定的分解，将系统分解成几个模块，降低业务需求的复杂性，从而使得软件系统结构清晰，便于阅读和理解，提高软件的可理解性和可维护性。

4.2.1 系统各模块划分

原则上“尽量使用数据耦合，少用控制耦合，限制公共耦合范围，坚决避免使用内容耦合”进行模块划分。下面是本系统的模块划分图：

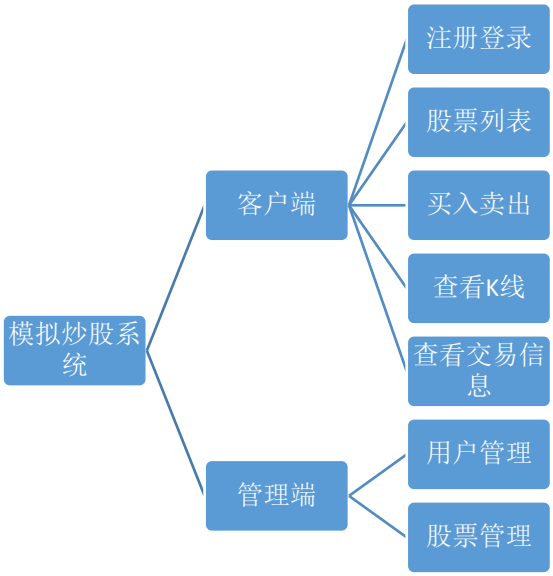

图 4-1 系统模块划分图

4.2.2 系统用例图

用例图也被称为参与者的外部用户能观察到的系统功能模型图。它通过一些指向性和图表性的关系来表示参与者和一些具体的系统用例的关系，其主要用于对子、主系统的功能进行建模。

管理员用例主要包含了用户管理用例、股票管理用例等

普通用户用例则主要包含了查看股票列表用例、查看股票 k 线图用例、登录注册用例等。下面系统的用例图

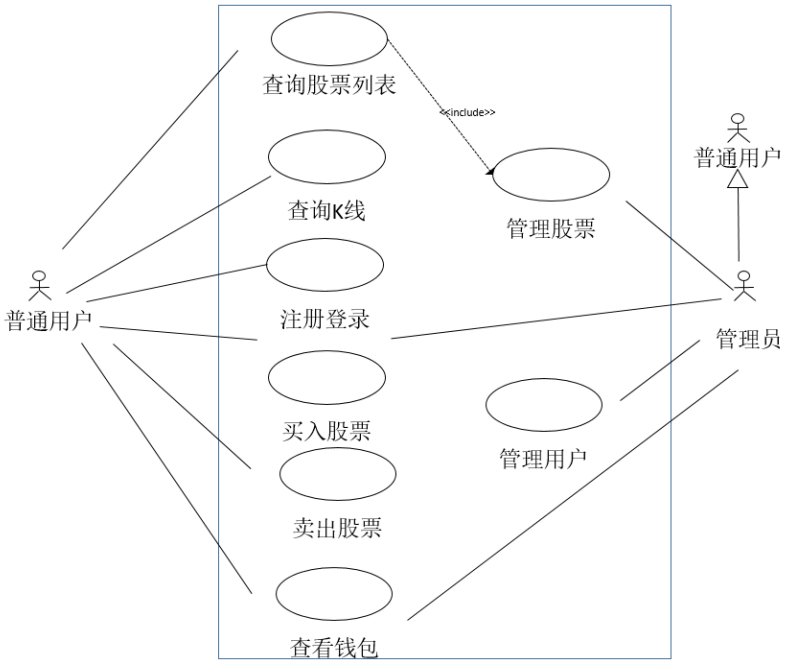

图 4-2 系统用例图

4.2.3 功能介绍

通过以上的系统功能划分，大家对相应的功能有两个大致的了解，接下来对具体的各个模块的功能进行大概的说明。

1. 用户登录注册功能：这个模块是用户能否进入股票系统的关键，有账户密码的用户才可以直接登录游戏，没有账号密码的用户只有先注册一个的用户名才能过往系统。
2. 查看股票列表功能：这个模块是用户查看系统有那些股票可供选购，同时也是查看 k 线图的入口。
3. 查询 k 线功能：这个模块是展示股票的最近一段时间股票的走势以及开、闭盘的数据。
4. 买入、卖出功能：这个模块是用户执行投资策略的地方，也是整个系统的核心。
5. 查看钱包：通过这个模块，用户可以查看当前所持股，以及所有的交易明细和收益。
6. 管理股票功能：此功能仅管理员可以使用，它可以让管理员添加、删除、查询系统所支持的股票。
7. 管理用户功能：本功能仅管理员可以使用，管理员通过本功能进行用户的添加、删除、查询等操作；同时管理员也可以通过它查询任一用户的钱包信息。

4.3 MVP 模式

MVP 模式是 MVC 模式演变而来，而 MVP 模式是使用在安卓领域的模式。其好处在于结构清楚、扩展性高、也方便单元测试，同时也提高代码的复用性。其区别于 MVC 模式的关键地方在于 presenter，presenter 完全把 model 和 view 分离，程序的逻辑代码在 presenter 实现的。View 和 model 之间的并不直接交互，它们之间仅能通过 presenter 交互。其工作原理如图 4-3。

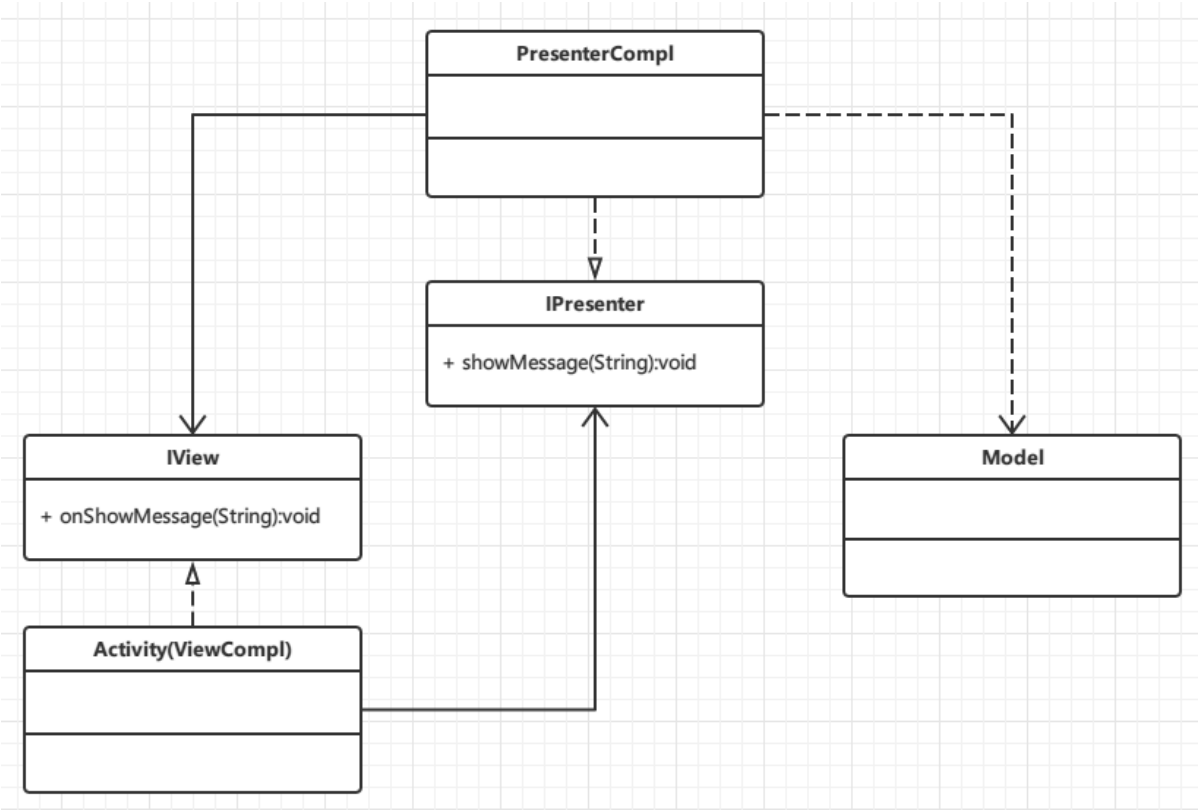

图 4-3 MVP 模式

4.4 数据设计

4.4.1 系统数据库表

字段名	数据类型	是否主键	是否为空	描述
uid	Int	是	否	用户 ID
uname	Text	否	否	用户名
pwd	Text	否	否	用户密码
Right	Int	否	否	用户权限
Total_property	Real	否	否	用户总资产
Left_property	Real	否	否	用户剩余资产
Market_value	Real	否	否	股票市场资产
Rate_of_retrun	Real	否	否	收益率

表 4-1 用户表

字段名	数据类型	是否主键	是否为空	描述
Timestamp	Int	是	否	秒级时间戳
Amount	Real	否	否	交易额
Close_price	Real	否	否	收盘价
Max_price	Real	否	否	最高价
Min_price	Real	否	否	最低价
Open_price	Real	否	否	上一个收盘价
Usdt_amout	Real	否	否	对应 usdt 交易额
Volume	Real	否	否	交易量

表 4-2 K 线数据表

字段名	数据类型	是否主键	是否为空	描述
Timestamp	Int	是	否	秒级时间戳
Current_volume	Real	否	否	至今交易量
Max_price	Real	否	否	最高价
Min_price	Real	否	否	最低价
Open_price	Real	否	否	上一个收盘价
Price_base	Real	否	否	交易区对应价格
Price_change	Real	否	否	价格变化百分比
Symbol_id	Int	否	否	股票代码
Total_amount	Real	否	否	总交易额
Total_volume	Real	否	否	总交易量
Usdt_amount	Real	否	否	对应 usdt 交易量

表 4-3 实时价格表

字段名	数据类型	是否主键	是否为空	描述
Id	Int	是	否	股票编号
Coin_id	Text	否	否	股票 id
Coin	Text	否	否	股票名称

表 4-4 股票表

字段名	数据类型	是否主键	是否为空	描述
Id	Int	是	否	交易 id
Uid	Int	否	否	用户 id
Code	Text	否	否	股票名称
Profit_loss	Real	否	否	盈亏
Chicang	Real	否	否	持仓
Buy_price	Real	否	否	成本
Coin_id	Text	否	否	股票 id
Profit_loss_rate	Real	否	否	盈亏比率
Keyong	Real	否	否	可用股数
Now_price	Real	否	否	现价
Time	Text	否	否	时间

表 4-5 交易条目表

第5章 系统详细设计

5.1 用户注册模块

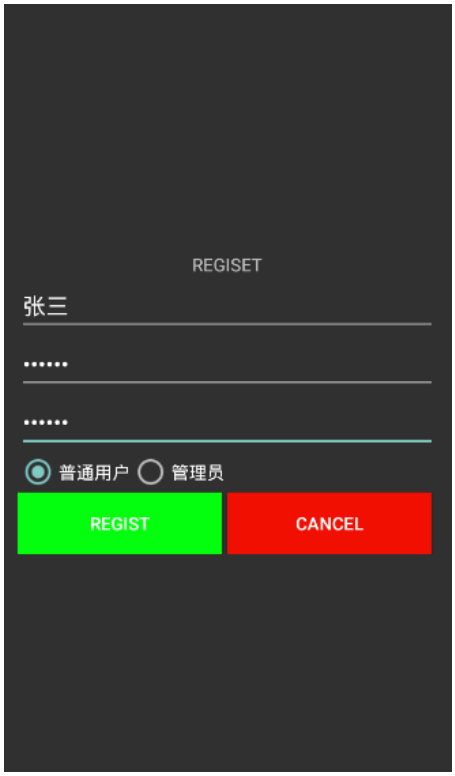

图 5-1 注册界面

用户注册部分代码如下：

```
override fun onClick(v: View?) {
    when(v?.id){
        R.id.btn_regist -> {
            val username=editUser.text.toString()
            val password=editPsw.text.toString()
            val c_password=c_psw.text.toString()
            Log.e("regist","$username $password $c_password")
            if(username.equals("")||password.equals("")||c_password.equals("")){
                Log.e("regist","error")
            }else if(!password.equals(c_password)) {
                Log.e("regist","error")
            }else{
                val right=if(radio_group.checkedRadioButtonId==rb_1.id)
                    1
```

```
                    else
                        2
                    iRegistPresenter.doRegist(username, password,right)
                    Log.e("regist", UserDb().queryUser("1=1").toString())
                    startActivity(Intent(this@RegistActivity, LoginActivity().javaClass))
                    finish()
                }
            }
            R.id.btn_cancel -> onBackPressed()
        }
}
override fun doRegist(name: String, pwd: String,right:Int)=
userPresenter.registUser(name,pwd,right)
override fun registUser(username: String, pwd: String,right:Int):Boolean {
    if(checkLogin(username,pwd,right).isNotEmpty()){
        Log.d("regist","on user found")
        return false
    }
    Log.d("regist","no user found")
    return UserDB.insertNewUser(user(-1,username,pwd,right,10000f,10000f,0f,0f))!=-1L
}
```

5.2 用户登录

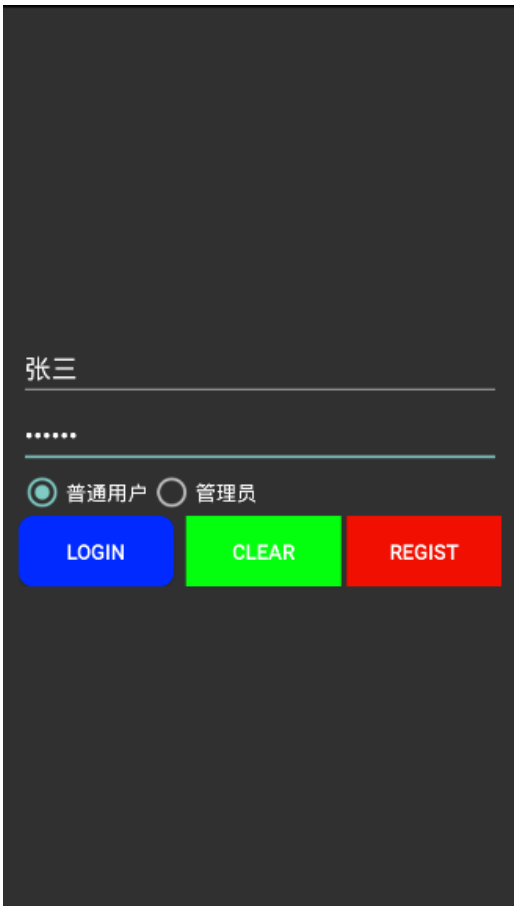

图 5-2 登录界面

登录模块部份代码如下：

```
override fun onClick(v: View?) {
    when(v?.id) {
        R.id.btn_login_clear -> loginPresenter.clear()
        R.id.btn_login_login -> {
            changeButtonState(false)
            loginPresenter.doLogin(editUser.text.toString(),editpwd.text.toString(),if
(radio_group.checkedRadioButtonId==rb_1.id) 1 else 2)

        }
        R.id.login_regist ->{
            startActivity(Intent(this@LoginActivity, RegistActivity().javaClass))
        }
    }
}
```

```
override fun doLogin(name: String, pwd: String,right:Int) {
    val code=iUserDbPresenterImpl.checkLogin(name,pwd,right)

    Log.e("query user",code.toString())
    handler.postDelayed({
        if (code.isNotEmpty()){
            //SharedPreUtil.putUser(code.get((0)))
            iLoginView.onLoginResult(true, code[0])
        }else {
            iLoginView.onLoginResult(false,null)
        }
    },1000)
}
override fun checkLogin(username: String, pwd: String,right: Int): List<user> {

    val condition:String=" ${userTable.UNAME}='$username' and ${userTable.PWD}
='$pwd' and ${userTable.RIGHT}='$right'"
    val result=UserDB.queryUser(condition)
    Log.e("checkLogin",result.toString())
   return result
}
```

5.3 股票列表模块

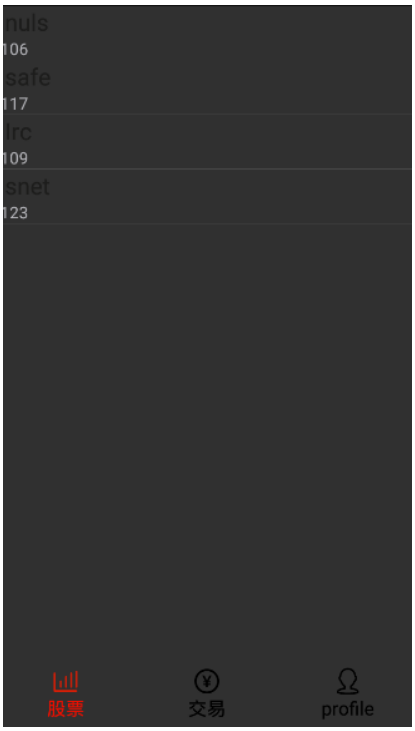

图 5-3 股票列表界面

部分关键代码如下：

```
private fun putStockList(view: View){
    val list_stock=view.findViewById<ListView>(R.id.stock_list)
    val coinList=CoinDb().queryCoin("1=1")
    Log.e("coinList","${coinList.size}")

    list_stock.adapter=StockListAdapter(App.instance(),coinList)
    list_stock.onItemClickListener = object :AdapterView.OnItemClickListener,
View.OnClickListener {
        override fun onItemClick(parent: AdapterView<*>?, view: View?, position: Int, id:
Long) {
            val ListView = parent
            val temp = ListView?.getItemAtPosition(position) as coin

            val bundle = Bundle()
            bundle.putString("coin_id", temp.coin_id)
            bundle.putString("coin", temp.coin)
            val Intent = Intent(App.instance(), KLineActivity().javaClass)
```

```
                Intent.putExtras(bundle)
                startActivity(Intent)
            }

            override fun onClick(v: View?) {
                Log.d("item","onclick")
            }
        }
}
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
        val holder: Holder
        if(convertView==null){

            val c:View=LayoutInflater.from(context).inflate(R.layout.stock_list,null)
            holder= Holder(c.find<TextView>(R.id.stock_name), c.find(R.id.scode))
            holder.sname.text=mData.get(position).coin
            holder.scode.text=mData.get(position).coin_id
            c.tag=holder
            return c
        }else {
            holder = convertView.tag as Holder
            holder.sname.text = mData.get(position).coin
            holder.scode.text = mData.get(position).coin_id
            return convertView
        }
}
```

5.4 k 线图功能

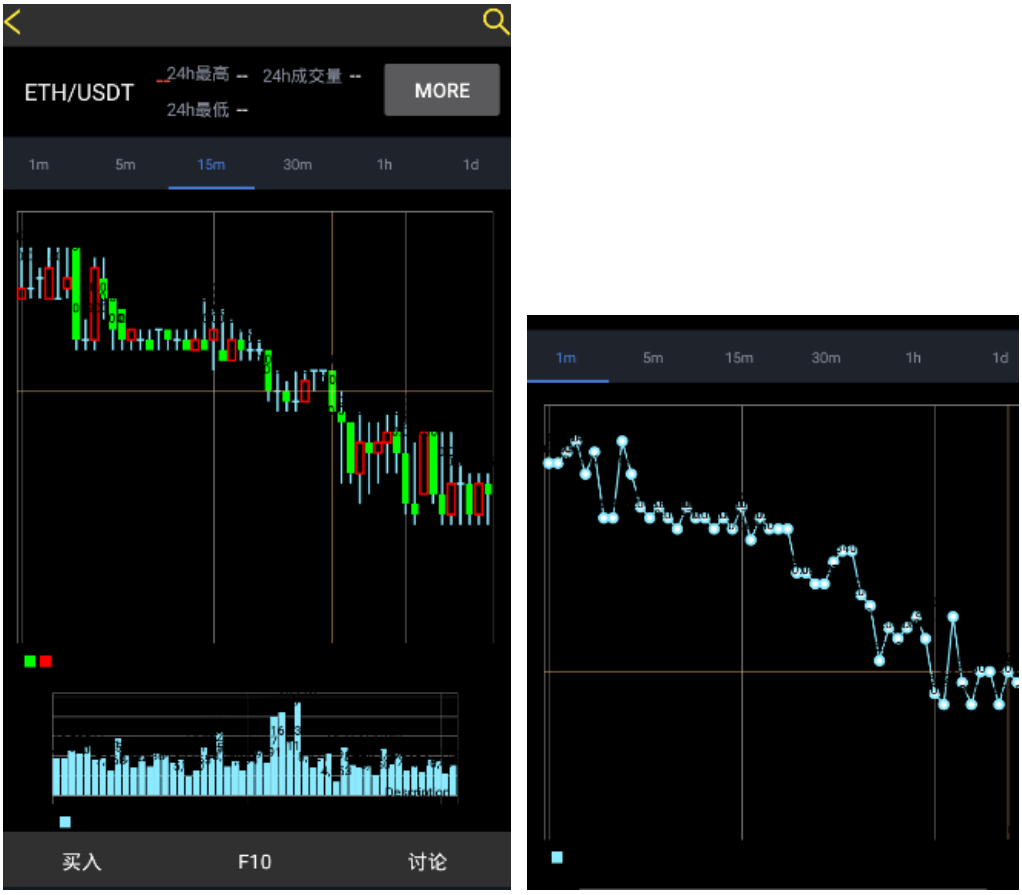

图 5-4 K 线图界面

部分代码如下：

```
private fun loadData() {
        clearChart()
        toLeft = true
        getData("0")

}
private fun getData(time: String) {
        Log.e("index",index.toString())

        val url = String.format(URL,coin_id, time, if (toLeft) "2" else "1", KL_TYPE[index])
        doAsync {
            val result=Internet_util.senRequestForData(url)
            uiThread { onRes(url,result) }
        }
```

```
}
private fun onRes(url: String,json: String){
    Log.e("onResponse", "Kline: $json")
    val kl= Gson().fromJson<KLineEntity>(json,KLineEntity::class.java)
    dataList.addAll(kl.data!!.lists)
    formateData()
    Log.e("stocklist.size",stockList.size.toString())
    setData()
}
private fun loadData() {
    clearChart()
    toLeft = true
    getData("0")

}
private fun formateData(){
    var temp=0L
    var i=0

    if (index==0){//分时图，仅画 line
        for (data in dataList){
            temp=data[6].toLong()*1000
            val x = SimpleDateFormat("yyyy/MM/dd HH/mm/ss",
Locale.getDefault()).format(Date(temp*1000))
            lineXVals.add(x)
            minValues.add(Entry(data[1].toFloat(),i))
            lineBarEntries.add(0,BarEntry(data[8].toFloat(),i ))
            i++
        }
    }else{
        for (data in dataList){
            temp=data[6].toLong()*1000
            val x = SimpleDateFormat("yyyy/MM/dd HH/mm/ss",
Locale.getDefault()).format(Date(temp*1000))
            candleXVals.add(x)
            stockList.add(CandleEntry(i, data[2].toFloat(), data[3].toFloat(),
data[4].toFloat(), data[1].toFloat(), x))
            candleBarEntries.add(BarEntry(data[8].toFloat(),i ))
```

```
                i++
            }
        }
    }

private fun setData(){
    if (index==0){
        combinedData= CombinedData(lineXVals)
        lineSetMin= LineDataSet(minValues,"1m")
        combinedData!!.setData(LineData(lineXVals,lineSetMin))
        barSet= BarDataSet(lineBarEntries,"bar")
        barData=BarData(lineXVals,barSet)
    }else{
        combinedData= CombinedData(candleXVals)
        candleSet=CandleDataSet(stockList,"kline")
        candleSet!!.increasingColor=Color.RED
        candleSet!!.decreasingColor=Color.GREEN
        combinedData= CombinedData(candleXVals)
        combinedData!!.setData(CandleData(candleXVals,candleSet))
        barSet= BarDataSet(candleBarEntries,"bar")
        barData=BarData(candleXVals,barSet)
    }
    candle_chart.data=combinedData
    barChart.data=barData
    barChart.setVisibleXRange(52f,52f)
    candle_chart.setVisibleXRange(52f,52f)
    barChart.invalidate()
    candle_chart.invalidate()
}
```

5.5 买入功能

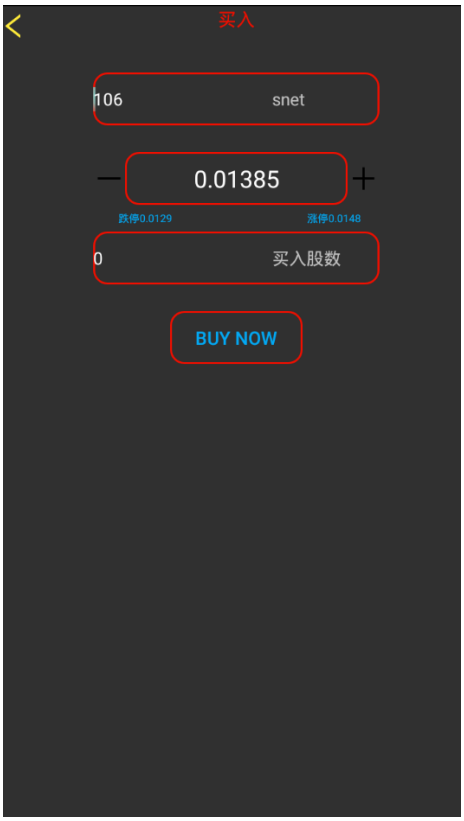

图 5-5 买入功能界面

部分代码如下：

```
private fun getDetails() {
    val url = String.format(url, coin_id)
    Log.e("url", url)
    doAsync {
        val result = Internet_util.senRequestForData(url)
        Log.e("buy result", result)
        uiThread { onRes(result) }
    }
private fun onRes(json: String) {
    val rl = Gson().fromJson<real>(json, real::class.java)
    matchValue(rl)
}
private fun makeDeal(): Boolean {
    val buy_in_price = buy_price.text.toString().toFloat()
    val v = volume_et.text.toString().toFloat()
    if(v<0f||buy_in_price<0f)
```

```
            return false
        else{
            val user=App.getLoginUser()
            if (user==null){
                return false
            }else{
                val market_value=buy_in_price*v
                user.market_value=user.market_value+market_value
                user.left_property=user.left_property-market_value
                App.putLoginUser(user)
                UserDb().updateUser(user)
                return     TradeListDb().insertTradeItem(trade(-1,     App.getLoginUser()!!.uid,
code.text.toString(), 0f, v, buy_in_price, stock_code_edit.text.toString(), 0f, v, buy_in_price,
"sb")) == -1L

            }
        }
}
```

5.6 卖出功能

图 5-6 卖出功能界面

如下源码如下：

```
class SoldActivity:AppCompatActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sold)
        val sold_list=find<ListView>(R.id.sold_list)

sold_list.adapter=SoldAdapter(App.instance(),TradeListDb().queryTradeList("${TradeTable.
UID}=${App.getLoginUser()!!.uid}"))
    }
}
```

5.7 钱包功能

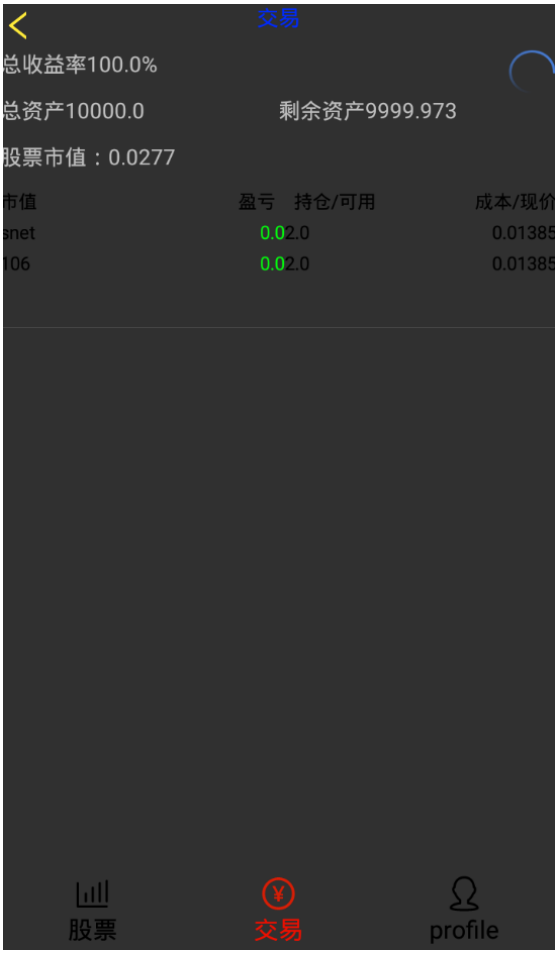

图 5-7 钱包界面

部分代码如下：

```
private fun setData(user:user?){
    var u=user
```

```
    if (u==null) {
        Log.e("getuser", "failed")
        u=App.getLoginUser()
    }
    rate_of_return.text="总收益率"+u?.rate_of_return.toString()+"%"
    total_property.text="总资产"+u?.total_property.toString()
    left_property.text="剩余资产"+u?.left_property.toString()
    market_value.text="股票市值："+u?.market_value.toString()
    val mData=TradeListDb().queryTradeList("${TradeTable.UID}=${u!!.uid}")
    Log.e("mData",mData.isEmpty().toString())
    trade_list.adapter=TradeListAdapter(App.instance(),mData)
    trade_list.onItemClickListener = object :AdapterView.OnItemClickListener{
        override fun onItemClick(parent: AdapterView<*>?, view: View?, position: Int, id:
Long) {
            val map = parent?.getItemAtPosition(position) as trade
            val coin_id = map.coin_id
            val bundle = Bundle()
            bundle.putString("coin_id", coin_id)
            val intent = Intent(App.instance(), KLineActivity().javaClass)
            intent.putExtras(bundle)
            startActivity(intent)
        }
    }
}
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
    val holder: Holder
    if (convertView == null) {
        val cv: View = LayoutInflater.from(context).inflate(R.layout.trade_list, null)
        holder = Holder(
            cv.findViewById(R.id.code),
            cv.findViewById(R.id.profit_loss),
            cv.findViewById(R.id.chicang),
            cv.findViewById(R.id.buy_price),
            cv.findViewById(R.id.coin_id),
            cv.findViewById(R.id.profit_loss_rate),
            cv.findViewById(R.id.keyong),
            cv.findViewById(R.id.now_price)
        )
```

```
        with(mData.get(position)) {
            holder.code.text = code
            holder.profit_loss.text = profit_loss.toString()
            holder.chicang.text = chicang.toString()
            holder.buy_price.text = buy_price.toString()

            holder.coin_id.text = coin_id
            holder.profit_loss_rate.text = profit_loss_rate.toString()
            holder.keyong.text = keyong.toString()
            holder.now_price.text = now_price.toString()
        }

        cv.tag = holder
        return cv
    } else {
        holder = convertView.tag as Holder
        with(mData.get(position)) {
            holder.code.text = code
            holder.profit_loss.text = profit_loss.toString()
            holder.chicang.text = chicang.toString()
            holder.buy_price.text = buy_price.toString()

            holder.coin_id.text = coin_id
            holder.profit_loss_rate.text = "$profit_loss_rate%"
            holder.keyong.text = keyong.toString()
            holder.now_price.text = now_price.toString()
        }

        return convertView

    }
```

5.7 用户信息界面

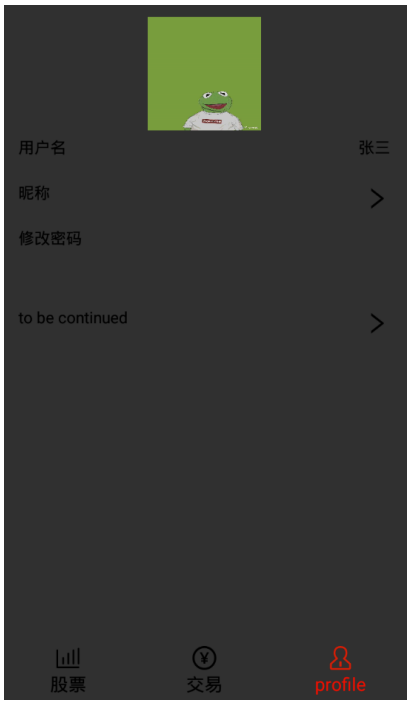

图 5-8 个人信息界面

部分源码如下：

```
private fun setNormalPage(view: View) {
    val username = view.find<TextView>(R.id.p_username)
    pic_head = view.find<CircleImageView>(R.id.pic_head)

    val user = App.getLoginUser()
    val pic_head_string = SharedPreUtil.getString("pic_head")
    if (pic_head_string != null) {
        val byteArray = Base64.decode(pic_head_string, Base64.DEFAULT);
        if (byteArray.size > 0) {
            val byteArrayInputStream = ByteArrayInputStream(byteArray);

            //第三步:利用 ByteArrayInputStream 生成 Bitmap
            val bitmap = BitmapFactory.decodeStream(byteArrayInputStream);
            pic_head.setImageBitmap(bitmap);
        }
    }
    pic_head.setOnClickListener(object : View.OnClickListener {
        override fun onClick(v: View?) {
```

```
                val intent = Intent()
                intent.setAction(Intent.ACTION_PICK)
                intent.setData(MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
                startActivityForResult(intent, 11)
            }

        })
        username.text = user?.uname

        val nikename_line = view.find<RelativeLayout>(R.id.nikname_line)
        val password_line = view.find<RelativeLayout>(R.id.password_line)
        nikename_line.setOnClickListener(object : View.OnClickListener {
            override fun onClick(v: View?) {
                Log.d("nikename_line", "action")
            }
        })
        password_line.setOnClickListener(object : View.OnClickListener {
            override fun onClick(v: View?) {
                Log.d("password_line", "action")
            }
        })
        password_line.removeView(view.find(R.id.icon_right_2))
}
```

第6章 系统测试

由于本系统是基于安卓平台的软件,UI 交互均为用户对界面的各种点击操作事件。所以本系统的系统测试均为对每个界面的具体触控事件和输出测试，并不严格遵守上章的系统详细设计的功能的顺序。

6.1 用户注册模块测试

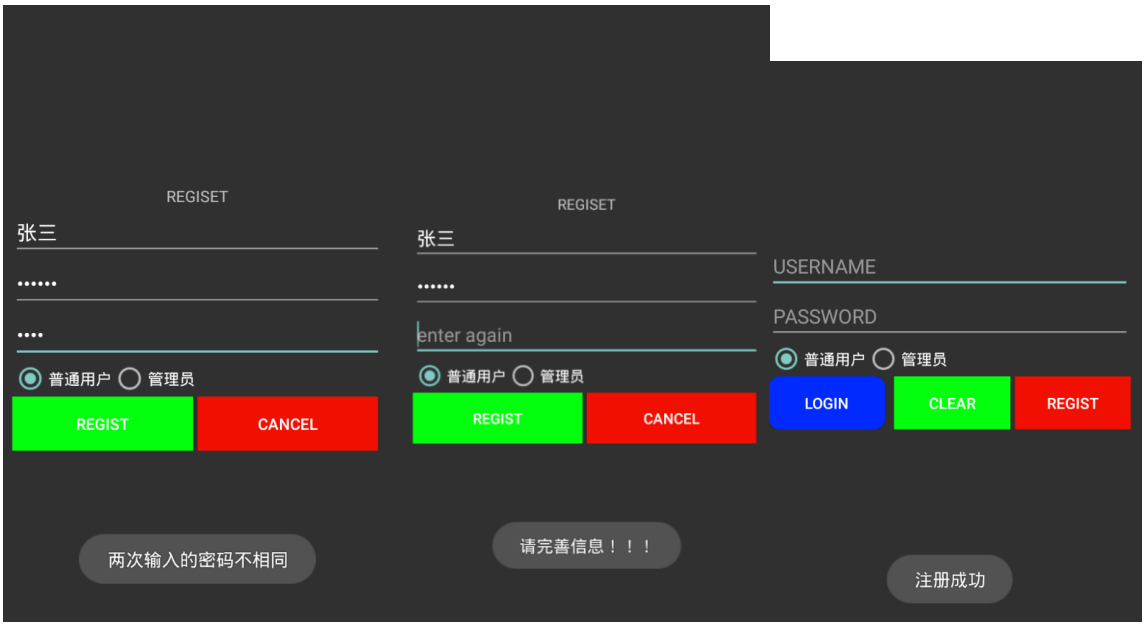

图 6-1 注册功能测试图

测试 1：用户名：张三

密码：341266

重复密码：3412

结果：提示——两次输入密码不相同

测试 2：用户名：张三

密码：341266

重复密码：空

结果：提示——请完善信息!!!

测试 3：用户名：张三

密码：341266

重复密码：341266

结果：提示：注册成功

注册功能主要检测用户注册时信息的填写正确与否，由上面的测试可知，系统对三个输入控的非空检测和密码重复检测均有效。因此本功能完成符合预期。

6.2 用户登录模块测试

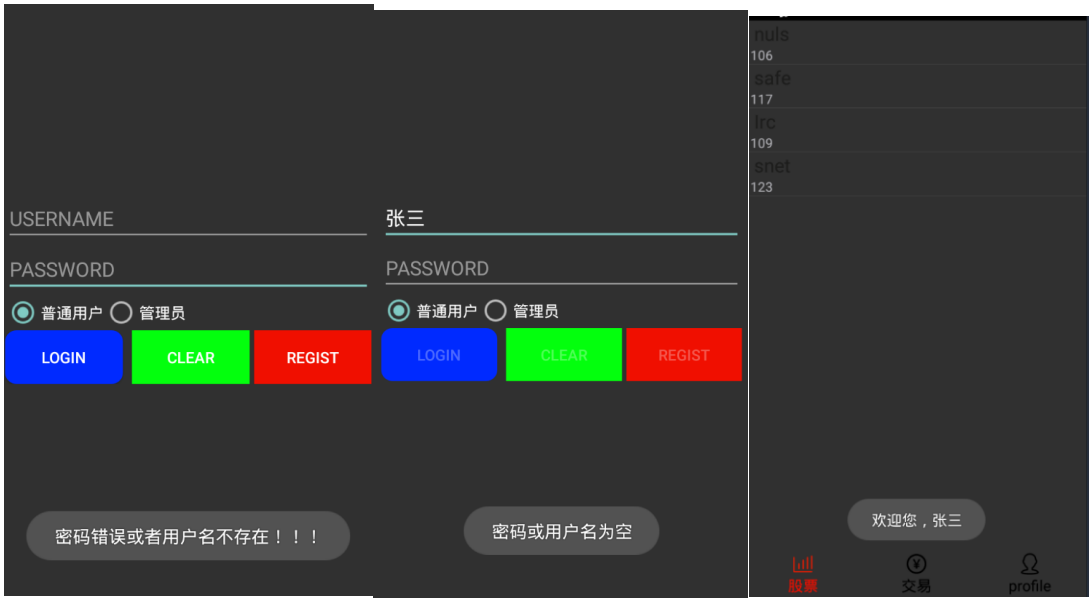

图 6-2 登录注册测试图

测试 1：用户名：张三

密码：3412

结果：提示——密码错误或用户名不存在!!

测试 2：用户名：张三

密码：空

结果：提示——密码或用户名为空!!

测试 3：用户名：张三

密码：341266

结果：进行主界面并提示——欢迎您，张三

用户登录功能和用户注册功能类似，不同的地方在于，它是在后端检测用户是否存在以及密码是否正确，然后把结发送给客户端。由上面的测试可知，本功能对密码和用户名匹配、以及输入框非空检测均有效，因为本功能符合预期。

6.3 主界面的各项功能

6.3.1 股票列表功能

图 6-3 k 线功能测试图

测试 1：点击 103/eth 股票

结果：跳转至股票行情界面

测试 2：点击 149/ont 股票

结果：跳转至股票行情界面

股票列表功能主要实现的是在主界面，点击任一股票，就会跳转至股票行情界面。上面随机对两个股票进行点击，并进入了相应的股票行情界面。由此可知，本功能的测试成功。

6.3.2 钱包功能

图 6-4 钱包功能测试图

测试 1：在买入界面，买入一支股票

结果：跳转至钱包，并将买入的信息展示出来。

测试 2：点击刷新按钮

结果：系统会自动查找实时行情，并更新钱包具体信息。

钱包界面的主要功能是将使用者的所有购买信息以列表形式，把具体信息展示给用户看。且右上角的刷新按钮实现了，更新当前用户所持股票的状态。由上面两图可知，这些功能的测试成功。

6.3.3 个人信息界面

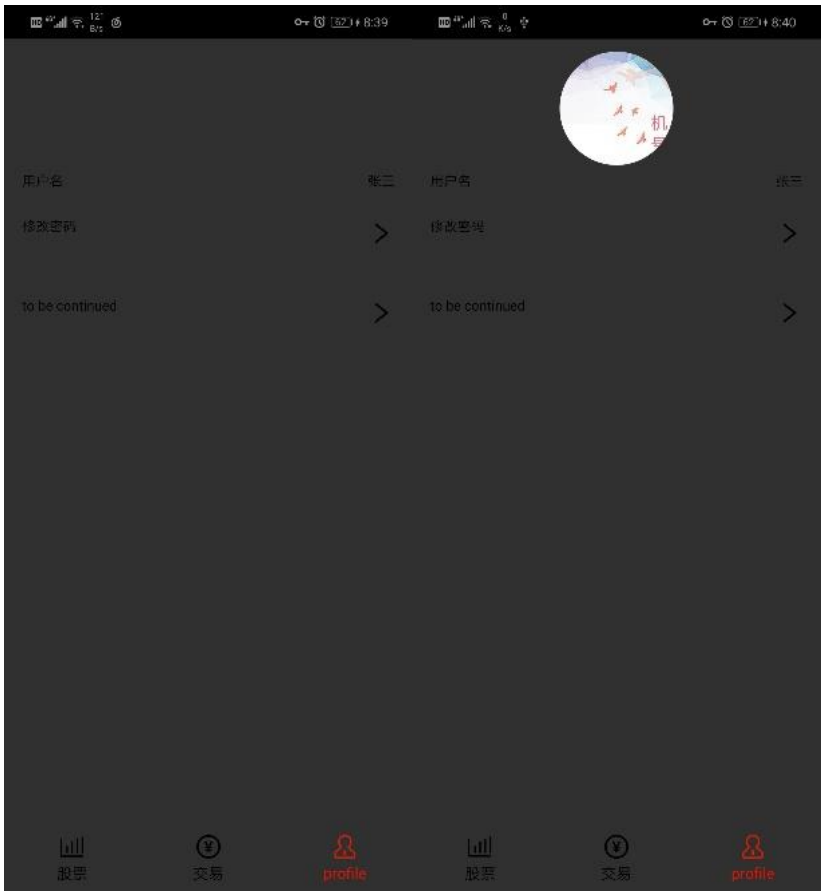

图 6-5 个人信息功能测试图

测试 1：点击右下方 profiel 按钮

结果：界面显示用户具体个人信息。

测试 2：点击上方头像按钮

结果：跳转至相册，选中相册后，主页会将图片剪切成圆形头像。

个人信息界面主要的功能是将登录用户的具体信息展示在屏幕上。在界面上方有一个可以从相册选择头像的功能。由上面的两张图片可知，这两个功能均被完整实现。

6.4 k 线图功能测试

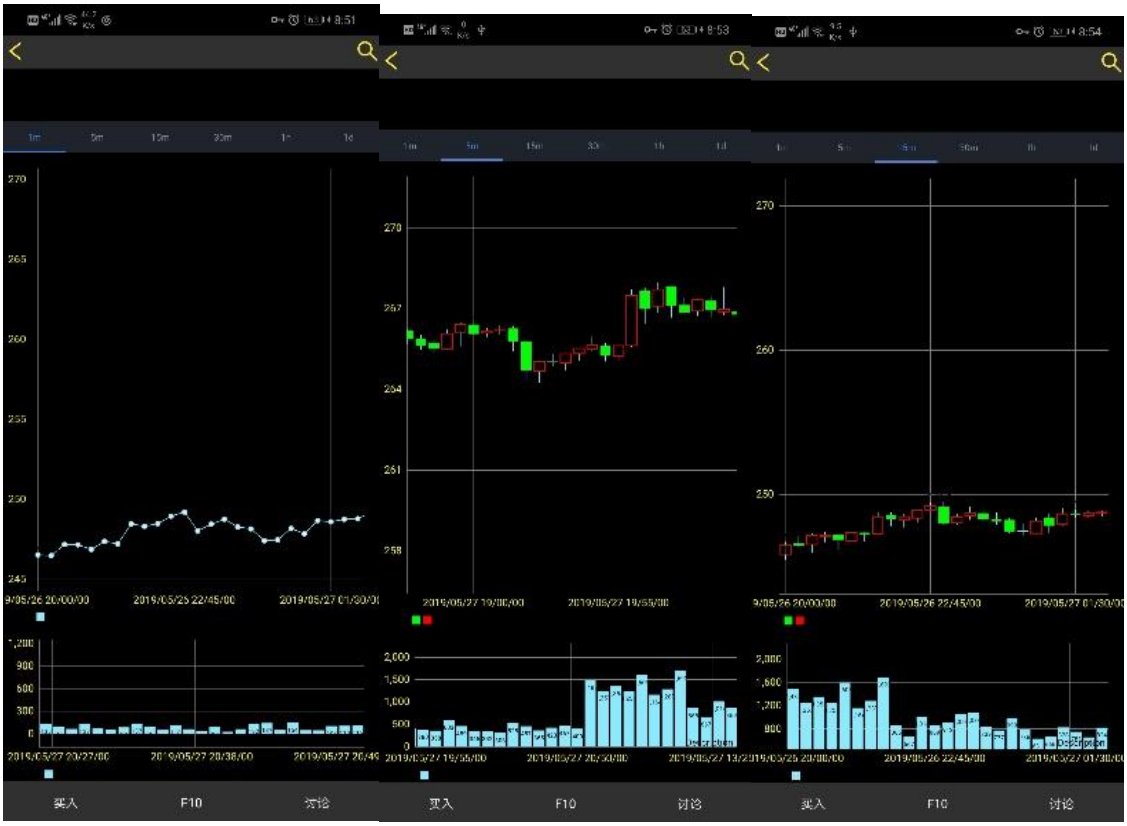

图 6-6 k 线功能测试图

测试 1：点击上方 1m

结果：片刻等待后，系统加载出分时图

测试 2：点击上方 5m

结果：片刻等待后，系统加载出蜡烛图

测试 2：点击上方 15m

结果：片刻等待后，系统加载出蜡烛图

股票行情界面的主要功能是展示股票的最近一段时间的行情。另外，在图表上方有 tab 按钮，可通过点击，选择观看不同种类的行情。如 1m、5m、15m 等。上图仅展示了三种行情，可知，本功能正确。

6.5 买入功能测试

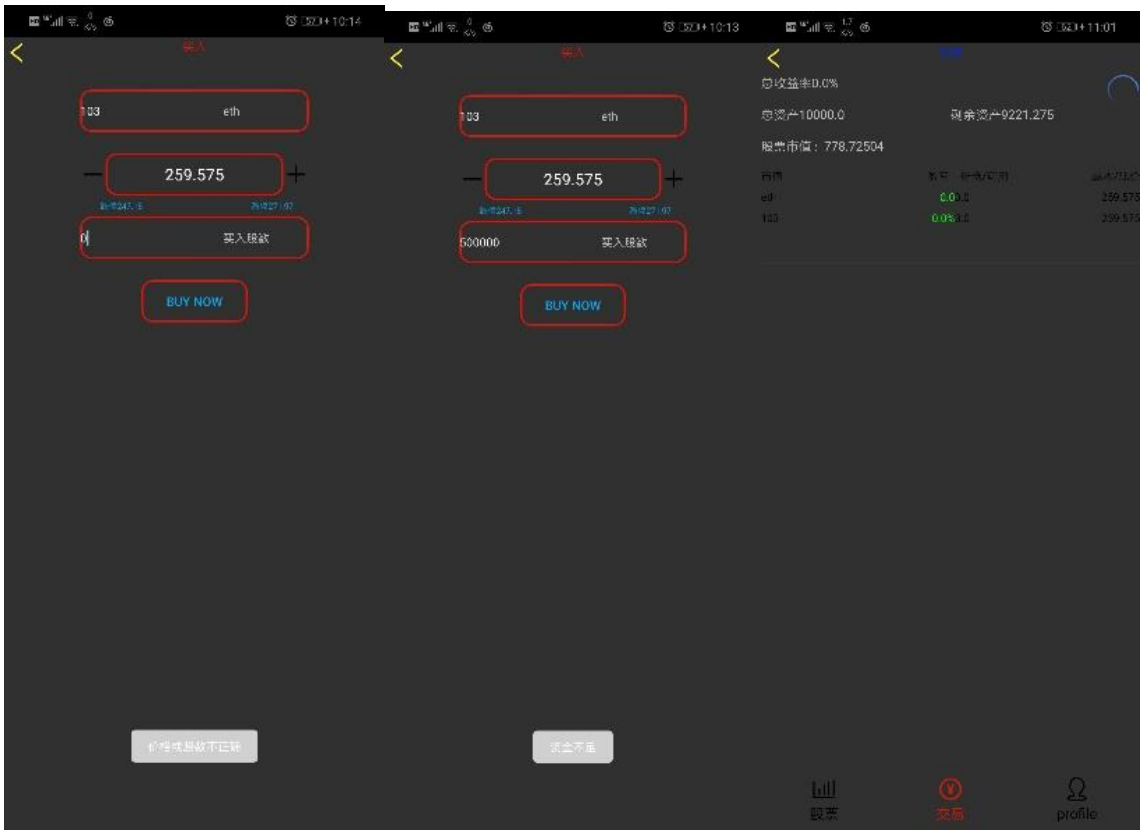

图 6-7 买入功能测试图

测试 1：输入股数 0

结果：提示，股数过少

测试 2：购买价格*股数>可用资金

结果：提示——资金不足

测试 3：购买价格*股数<=可用资金

结果：提示——买入成功

买入界面主要是让投资者对心仪的股票进行策略投资，当然在选择购买股票时，会对输入的信息进行检测。其针对价格和股数以及用户可用资金进行算法检测。

由上三图可知，本功能完整。

6.6 卖出功能测试

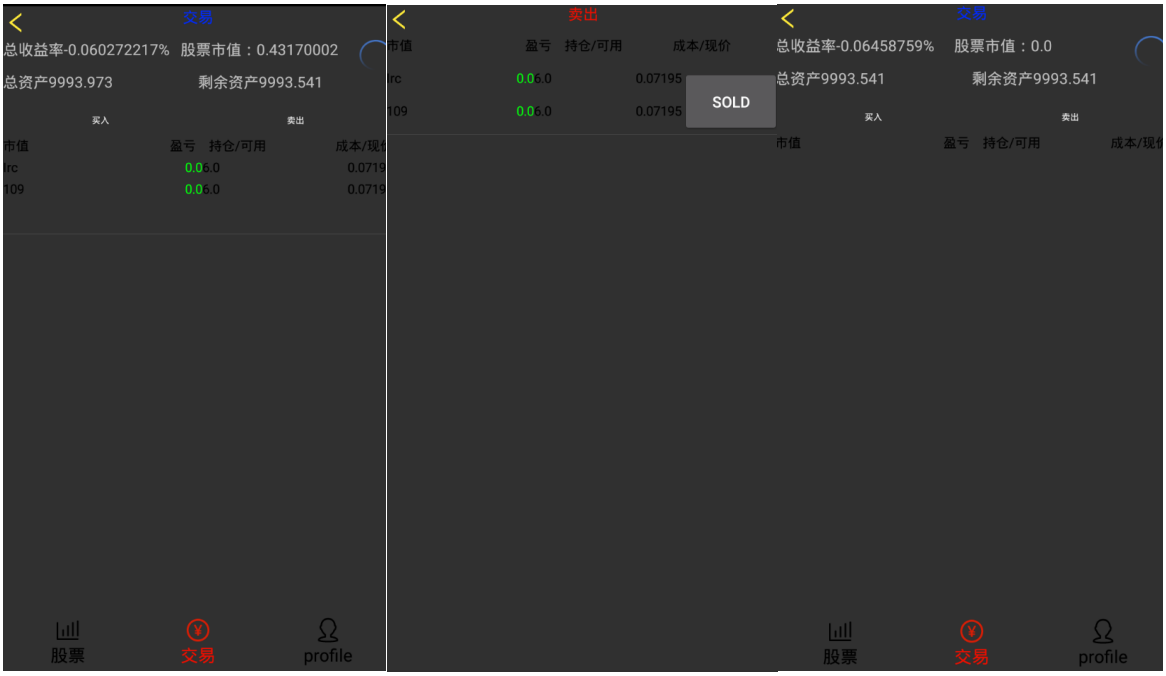

图 6-8 卖出功能测试图

上面的测试为卖出一支股票，由结果可以看出，卖出功能测试正确。

总结与展望

至此，本次的毕业设计已经大体完成了。在这毕设的几个月中，查阅过诸多网站，也翻看过许多教学、书籍、论文，从对移动端软件设计一无所知，一步步对地成长，到现在，已经对安卓软件的设计流程和技术有一定的了解。

但是，本次的系统设计过程中也遇到了许多困难。这里简单列出几点。首先是数据问题。由于本系统是模拟炒股系统，股票数据是最能体现本系统的地方，也是系统的关键地方。如果不能给用户展示完整且清晰数据，那么用户就不能从图表中获取出有效信息进行策略投资。但是本系统使用的数据是外网下的比特币数据，数据最小间隔为一分钟，且许多比特币数据不完整。另外本系统采用的是第在方库 mpandroidchar 描画股票行情图，但是由于使用时间过短，对于良好的 UI 设计并不能完善地实现。其次，系统 UI 设计不美观，色调单一也是本系统一大问题。此外，对于本系统选用 kotlin 语言编写并不满意。虽然 kotlin 不久前被谷歌宣布为安卓开发官方语言，但是，大部分安卓第三方库还都是 java 语言编写的，使用过程中，在出现 bug 的地方时常需要查看函数的 java 源码。除此之外，我在参加网上的一些教程时，作者使用的均为 java 语言，这让我编写系统时有种绕远路的感觉。

虽然在编写本系统过程遇到了诸多困难，但一步步走过来，也在途中收获累累。Kotlin 在当下的开发环境虽然不好，开发者数量规模不足，但是在未来 Kotlin 的发展可期。在使用 kotlin 的过程中，也认识到这门语言的优势——语言的简洁、lamda 表达式、数据类、委托属性等等。另外，一次完整的编写系统的经验也是十分宝贵的，这次经历让我对一个系统的构建有了一个基本的认识。在开发的过程，也正如《软件工程基础》一书说讲授的一样，良好的开发流程会让工作量大幅度减少。而毫无章法地“东拼西凑”的开发模式只会让业务流程变得没有条理，只会让系统的扩展性和功能耦合性更高。在本次毕设过程中最宝贵的经验莫过于对业务逻辑的实现的理解。在这之前，我对于业务需求的理解就是算法。而现在，我对业务需求的理解不仅仅是将业务需求转化为算法，还有对业务需求转化为具体的业务流程。先要设计好一个业务流程框架，然后在这个框架下写具体的算法才是有用的，否则只是纸上谈兵。

致 谢

历时两个多月，在老师的帮助下，我终于完成了此次毕业设计。我非常感谢老师给我提供的帮助，老师渊博的知识、严谨的教学态度和工作作风、认真的科研精神给我树立了良好的榜样，激励着我更认真的完成此次设计。从论文的选题到论文定稿，老师花费了大量的精力，在此，向老师表达我最由衷的感谢。同时也向设计与答辩中给过我帮助的所有同学、舍友和老师，表达我最真挚的感谢！

参考文献

[1]龙达鑫.接口回调实现安卓列表控件适配器和 Fragment 的通信[J].信息技术与信息化,2019(03):85-88.
[2]许庆建.个人投资者股票投资风险管理思考[J].合作经济与科技,2019(10):68-70.
[3]龙达鑫.运用 JAVA 异常机制分析安卓应用程序崩溃[J].信息技术与信息化,2019(04):160-162.
[4]宋金牛,姜颖.投资者情绪状态对股票收益的影响[J].经济研究导刊,2019(11):128-130.
[5]姚 远,钟 琪,姚贝贝.投资者情绪与股票市场波动关系研究——基于噪声交易与股票市场价格非理性波动关系的分析[J].价格理论与实践:1-3.
[6].JetBrains 开源其 Kotlin 语言,基于 JVM 的新编程语言[J].硅谷,2012(04):77.
[7]Krill, Paul. Kotlin project adds another language option to JVM[J]. InfoWorld.com,2011.
[8]Zhao Limei,Ma Xiaotie. Design and Implementation of Body Quality Index App Based on Android[J]. Journal of Physics: Conference Series,2019,1187(5).
[9]谢莉莉.基于安卓仿真的支付宝云数据取证方法[J].电脑知识与技术,2018,14(32):297-298.
[10]应柠泽.基于安卓的校园旧书交易 app 的设计与实现[J].电子世界,2019(04):141-142.
[11]钟亚妹,郑志恒.安卓手机移动办公 APP 软件的开发及应用[J].电脑知识与技术,2019,15(05):69-70.
[12]唐瑜冲.基于叩富网大学理财课程实践平台的模拟炒股探究[J].管理观察,2018(30):146-147.
[13]郑豪,姜荣信,李燕.基于文献的大学生炒股行为研究调查[J].现代商业,2016(26):181-182.
[14]严瑜. 股票行情分析系统的设计与实现[D].吉林大学,2014.
[15]郑森. 股票交易模拟软件的设计与实现[D].山西大学,2013.
[16] 郭霖.第一行代码 Android[M].北京：人民邮电出版社，2014.8:15-20
[17] Enck W, Octeau D, Mcdaniel P, et al. A study of android application security[C].Usenix Conference on Security. USENIX Association, 2011:1-3
[18] Lin Deng,Jeff Offutt,Paul Ammann,Nariman Mirzaei. Mutation operators for testing Android apps[J].Information and Software Technology,2017
[19] Yan Zhuang. The performance cost of software obfuscation for Android applications[J]. Computers & Security,2018
[20] 王忠群.软件工程 [M].中国科学技术大学出版社,2009:26-29

附　录

附录 C

A security framework for mHealth apps on Android platform

Abstract

Mobile Health (mHealth) applications are readily accessible to the average users of mobile devices, and despite the potential of mHealth applications to improve the availability, affordability and effectiveness of delivering healthcare services, they handle sensitive medical data, and as such, have also the potential to carry substantial risks to the security and privacy of their users. Developers of applications are usually unknown, and users are unaware of how their data are being managed and used. This is combined with the emergence of new threats due to the deficiency in mobile applications development or the design ambiguities of the current mobile operating systems. A number of mobile operating systems are available in the market, but the Android platform has gained the topmost popularity. However, Android security model is short of completely ensuring the privacy and security of users' data, including the data of mHealth applications. Despite the security mechanisms provided by Android such as permissions and sandboxing, mHealth applications are still plagued by serious privacy and security issues. These security issues need to be addressed in order to improve the acceptance of mHealth applications among users and the efficacy of mHealth applications in the healthcare systems. The focus of this research is on the security of mHealth applications, and the main objective is to propose a coherent, practical and efficient framework to improve the security of medical data associated with Android mHealth applications, as well as to protect the privacy of their users. The proposed framework provides its intended protection mainly through a set of security checks and policies that ensure protection against traditional as well as recently published threats to mHealth applications. The design of the framework comprises two layers: a Security Module Layer (SML) that implements the security-check modules, and a System Interface Layer (SIL) that interfaces SML to the Android OS. SML enforces security and privacy policies at different levels of Android platform through SIL. The proposed framework is validated via a prototypic implementation on actual Android devices to show its practicality and evaluate its performance. The framework is evaluated in terms of effectiveness and efficiency. Effectiveness is evaluated by demonstrating the performance of the framework against a selected set of attacks, while efficiency is evaluated by comparing the performance overhead in terms of energy consumption, memory and CPU utilization, with the performance of a mainline, stock version of Android. Results of the experimental evaluations showed that the proposed framework can successfully protect mHealth applications against a wide range of attacks with negligible overhead, so it is both effective and practical.

1. Introduction

In the early days, mobile phones were only used for making phone calls. Nowadays, mobile phones have come to be known as smartphones because of their increasing functions and intelligence. Smartphones are equipped with powerful operating systems that enable users to install additional software, more storage and processing capabilities, and multiple options of network connectivity. Due to their improved functionalities and computing capabilities, smartphones are increasingly viewed as handheld computers (Boulos et al., 2011), and their adoption by people is rising due to their ease of use (Park and Chen, 2007).

Among the available smartphone operating systems (OS) in the market, Android OS has the topmost popularity, with a market share of above 85% (International Data Corporation, 2017), and more than 3.5 million applications available on Google Play ("Number of Android applications", 2017). Categories of those applications range from the basic trivia game applications to serious business and financial applications. One active area of smartphone applications that has witnessed an astonishing growth is the healthcare systems. Under the category of medical applications, Google Play and similar online stores of smartphone applications are providing large collections of applications that can be used for various healthcare-related functions. Adopting the notion of mobile Health (mHealth) as a reference to the use of mobile devices in medicine and public health, smartphone medical applications are referred to in this research article as mHealth applications.

In general, mobile health is a medical and public health practice using mobile devices, such as smartphones, personal digital assistants (PDAs), patient monitoring devices (Abdulnabi et al, 2017, Kalid et al, 2018, Salman et al, 2017) and other wireless devices (World Health Organization, 2011). Mobile health is an emerging field which has the potential to make healthcare professionals more efficient, increase patient satisfaction and reduce the healthcare cost. The general concept of mHealth includes medical applications. There are several types of medical applications (Alanazi et al, 2010, Alanazi et al, 2015, Mat Kiah et al, 2013, Mat Kiah et al, 2014a, Mat Kiah et al, 2014b, Mat Kiah et al, 2014c, Nabi et al, 2011, Zaidan et al, 2015a, Zaidan et al, 2015b), some are using external devices such as medical sensors, and some applications are using smartphone resources, such as the camera for the treatment of the patient. The use of mHealth applications among physicians and patients has grown significantly since the introduction of mobile phones. Physicians can access patients' data and medical knowledge at the point of care, and they can also monitor patient health through mHealth applications.

mHealth applications have the potential to improve the availability, affordability and effectiveness of healthcare services for patients (Bateman et al, 2017, Mirza et al, 2008). They have become incorporated into the health informatics field as tools that maintain a patient-centred model of healthcare by enabling users to monitor their health related problems, attain personal fitness goals, and understand specific medical conditions. Patients can use smartphones to access and update their medical records (Zaidan et al, 2011, Zaidan et al, 2015a, Zaidan et al, 2015b), monitor their health, and to view their prescriptions as well (Brennan et

al., 2010). Physicians, on the other hand, can use smartphones to access patient records and test results, monitor patient health and to prescribe medications (Burdette et al, 2008, Luxton et al, 2011, Ozdalga et al, 2012). mHealth applications can also improve the way in which physicians interact with patients and provide healthcare services.

Similar to other new trends, mHealth applications have to face a number of challenges despite their compelling benefits.

mHealth applications face the usual security requirements of enforcing confidentiality, integrity, and availability via authentication, authorization, and access control (Adhikari et al, 2014, Dehling et al, 2015, He et al, 2014, Mitchell et al, 2013, Müthing et al, 2017, Plachkinova et al, 2015). Such protection is necessary to facilitate the adoption of these applications by the healthcare systems. The following points summarize some of the major threats that mHealth applications are facing:

Users of mHealth applications are susceptible to privacy threats, such as identity theft, disclosure threats, privilege escalation attacks and side channel threats, among others (Davi et al, 2011, He et al, 2014, Kotz, 2011, Müthing et al, 2017, Plachkinova et al, 2015).Leakage of information is a major challenge for mHealth applications (Dehling et al., 2015), where these applications may leak information in numerous ways. For example, applications usually declare their components as public (He et al., 2014), so malicious applications can easily access their information.

Besides, applications usually store unencrypted data on smartphone external storage (He et al, 2014, McCarthy, 2013, Mitchell et al, 2013), so any application that has the permission to access external storage can easily access the user's data.

Usage of third party services and sharing of information with social networks or other third parties are also raising threats to mHealth applications (Adhikari et al, 2014, Dehling et al, 2015, He et al, 2014, Plachkinova et al, 2015).

In addition, mHealth applications use external devices to enhance the functionality of the phone. These devices also impose serious threats to user's data, such as external-Device MisBonding (DMB) attacks that include data-stealing and data-injection attacks (Naveed et al., 2014), since Android permission system does not provide permission-based protection for external devices and sensors.

Existing smartphone operating systems, particularly Android, are not sufficient to ensure privacy and security of users' data, particularly in the case of mHealth applications. One major issue in the security model of Android is that the permission mechanism is too coarse-grained and the user might not be aware of the full implications when granting permissions to applications (Zhou et al., 2011).

It should be noted that in addition to the traditional threats found in other software and information systems, mHealth applications introduce new security and privacy threats to mobile computing (He et al., 2014). Even when compared to other health information systems, mHealth

applications are different in various perspectives. First, mHealth applications have the potential to collect large amounts of data from patients because mobile devices are most of the time carried by the patients and can collect data over long time intervals. Second, mHealth applications collect much broader range of data besides physiological measurements and direct medical data; this includes patients' activities, location, lifestyle, social interactions, diet details, eating habits and so on. Third, the nature of communication between the patient and healthcare professionals is different (He et al., 2014); e.g. healthcare professionals can remotely access and monitor patients' health conditions.

Based on the above facts, there is a need for a better solution to protect the security of mHealth applications, and ensure the confidentiality, integrity and availability of their data. Data associated with mHealth applications are particularly of sensitive nature, and unauthorized leakage or manipulation of these data do not only threaten the privacy of the patients, but might also threaten their health or even lives. The intended protection is two-way; meaning it protects the mHealth application and its corresponding data from potential threats on the system, and also protects the system and its resources from installed mHealth applications that can unintentionally or otherwise bring new threats by means of poor design, or ill will. The focus of this research article is to propose such a solution in the form of a security framework for mHealth applications on Android platform. The proposed framework ought to address the aforementioned security and privacy issues on Android, with a special focus on threats associated with mHealth applications, such as the revealed vulnerabilities in literature, including information leakage; and the published attacks, such as DMB, privilege escalation, and side channel attacks.

2. Related work

The sensitive nature of mHealth apps' purpose and consequence of use—in relation to human health—imposes several questions about their reliability, authority, and compliance to regulations. Aside from the functional requirements, issues related to non-functional requirements have also to be addressed, such as the usability of the apps by users from different age groups, the security of the system and its apps, as well as the privacy of their users. The latter is the main concern in this work. In particular, it soon became clear that mHealth apps carry substantial risks to the security of users' sensitive medical data as well as their privacy (Adhikari et al, 2014, Dehling et al, 2015, Gill et al, 2012, He et al, 2014, Plachkinova et al, 2015, Zhou, Jiang, 2012). Developers of these apps are usually unknown, and users are unaware of how their data are being managed and used (Federal Trade Commission, 2016). In mHealth, users can easily enhance the functionalities of their smartphones by connecting them to external devices, such as medical devices, sensors and credit card readers. This introduces many new threats along with the useful applications in various domains, including healthcare information systems and retail (Anokwa et al, 2012, Avancha et al, 2012, Backes et al, 2014, Federal Trade Commission, 2016, Istepanian et al, 2006, Murthy, Kotz, 2014, Naveed et al, 2014).

In response to these concerns, researchers have been actively involved in studying the security and privacy of mHealth apps. For example, Mitchell et al. (2013) investigated the security and privacy challenges of mHealth apps; He et al. (2014) raised the security concerns of Android mHealth apps; and Plachkinova et al. (2015) proposed a taxonomy of mHealth apps' security and privacy concerns. Nevertheless, beyond the identification and investigation of the problem itself, there is no actual solution for the security and privacy of mHealth apps specifically, except one policy framework (Mitchell et al., 2013). This framework provides some guidelines to secure mHealth apps; however, these policies are not enough and even not implemented to secure mHealth apps. In addition, Android-provided security features are still insufficient to protect user data against few security attacks that are equally applicable to mHealth apps and their data, such as side channel threats, privilege escalation attacks, sensors-based covert channels and DMB attacks (Al-Haiqi et al, 2014, Davi et al, 2011, He et al, 2014, Hussain et al, 2015, Hussain et al, 2016, Naveed et al, 2014).

mHealth apps are a new and revolutionary development in healthcare systems, and a huge number of people can access this new system at a very low cost. Considering the great utility and impact of this phenomenal development, and the detrimental effect that security and privacy issues might cause to its successful deployment, those issues need to be addressed to improve mHealth apps' effectiveness and alleviate any barriers to their rapid integration into the healthcare systems.

3. The design of “mHealth apps security framework”

This section presents the detailed design of the proposed mHealth Apps Security Framework, named as “MASF”. This framework comprises several components that are divided into two major layers, each of which is discussed below.

3.1. MASF overall architecture

As discussed earlier, mHealth apps are facing several security and privacy challenges. To participate in addressing these issues, this article proposes a security framework (MASF) that can provide special measures to protect the users of medical apps and their data from malicious or otherwise incompetently written mHealth apps on the Android mobile. In order to achieve such a goal, several functions are expected from the framework.

The framework is expected to inspect apps before and during execution, starting from the point of installation. It is also required to provide security measures beyond the capability of the host system (Android in this case). For example, MASF is required to provide fine-grained access control to sensitive resources that is more effective than the coarse-grained permission system offered by Android. The concept of context should also be considered when controlling access to private resources or data. For example, collection of certain medical data is not expected at certain times or places, and the framework should be able to discern the allowed and disallowed actions based on the particular context at which the actions occur.

Furthermore, scenarios that are specific to mHealth apps should be paid special attention.

For example, reading to or from medical sensors and devices should be monitored and checked against the current context, and against any known attacks as well. It is also important to enable the user of the framework to dictate a set of policies that is used to derive the decisions of the framework; for example, policies related to what actions are allowed in which context.

In the design of MASF, several components are assigned different tasks to ensure the above functionality. A main component is responsible for performing the security checks and taking decisions related to attempted actions by the various mHealth apps in the system. This main part is called the manager. To support its operation, the manager refers to a set of software tools called checkers. These tools perform specialized checks on the installation of new mHealth apps, context checks, malware checks, taint analysis and checks related to the communication of external devices. The manager refers to a policy-database and makes use of those special checks to form a complete idea about the adherence of a specific app or actions of an app to the stated policies. It then delegates the enforcement of the policies to an action-performer. Another component called the user-interactor allows the user to provide policies to protect his/her security and/or privacy.

It is obvious that the functions performed by the various checkers and by the action-performer need special access to the underlining operating system, as they interfere with and control low-level operations of the host system. To modularize the design of MASF, the above described parts including the manager and its supportive components are grouped in one layer, named as the Security Module Layer (SML) since it is directly concerned with the security-related functions, and then this layer relies on another layer for interfacing to the Android internal parts. This latter layer is aptly called the System Interface Layer (SIL). The general structure of the proposed MASF is illustrated in Fig. 1. Further details on SML and SIL are given in the subsequent sections (4.2.1 and 4.2.2, respectively).

Download high-res image (163KB)

Download full-size image

Fig. 1. The overall architecture of the proposed framework of MASF.

3.2. MASF layered components

This section details the individual components of MASF. The main two layers of MASF are shown in white colour in Fig. 1. Most contributions of this research lie within the first layer (SML), in which the building blocks are shown in light grey.

3.2.1. Security module layer

Security Module Layer (SML) is the main part of MASF, which encompasses the essential functions to protect the system's security and the user's privacy. In order to fulfil its purpose, this layer needs to be able to:

Monitor references and actions made by mHealth apps

Examine static and dynamic attributes of mHealth apps

Receive policies defined by the user of the system
Enforce user policies

One or more components are dedicated for each of these function categories. Several modules depicted in Fig. 1 as checkers are responsible for providing the necessary security checks and examinations. A user-interactor component interfaces with the user to receive user policies (the form of which to be defined later in Section 4.2.4), and a policy-database stores them for later use by the framework. The central component that monitors the apps' references, rationalizes their actions in line with the user's policies, and then makes the required decision is the manager, which enforces the made decisions through the action-performer. The following subsections elaborate the design of each of those components.

3.2.1.1. Security checkers

Security checkers are used by the manager to essentially provide necessary information for making security decisions. In order to provide such information, a checker might perform operations such as simple collection of sensor data, up to sophisticated analysis of the apps code or behaviour. Five types of checks are deployed in SML. The corresponding checkers are explained below.

a)
Context-checker

According to Conti et al. (2011), a context could be defined as one of the following aspects: status of some variables (e.g., time, geographical location, temperature, light and noise), the presence of other devices and sensors, a particular type of interaction between the smartphone and user, or a combination of all these aspects. Some security rules depend on the context. In MASF, only time and geographical location are considered for context-aware access control.
The context-checker is responsible for detecting and reporting the context of the device and its apps. This job requires the context-checker to subscribe to Android system services, such as the LocationManager. Basically, this type of checks is based on the concept of context-aware access control. The context-checker collects the physical location parameters (GPS, Cell ID, Wi-Fi parameters) through the device sensors and reports those parameters to the manager upon request. These context checks enable the manager to allow the user of imposing runtime constraints on the usage of sensitive resources based on different contexts (e.g., location and time). The users can describe their constraints using a simple interface (i.e., app), implemented by the user-interactor. Possible rules and policies related to context-aware access control are defined in Section 4.2.4.

b)
Malware-checker

A modern smartphone provides capabilities that are comparable to low-end computers. As such, it is also facing nearly the same security and privacy issues. Indeed, mobile malware is growing exponentially, and hence the process of malware detection is becoming an essential

part of mobile security frameworks. The SML manager performs some checks against malware apps with the help of a malware-checker that is dedicated to identifying malicious apps.

Basically, the malware-checker invokes an anti-malware app that is installed on the smartphone, and scans the target apps. The results of the scan are reported to the manager. If any malware is detected, then the manager sends a notification to the user about that particular malware. The malware app is marked as an "untrusted" alongside its source, and this information is kept in the policy-database so that future installations of new apps from the same source can be prevented by the installation-checker. The untrusted app cannot access other mHealth apps, nor can it access any sensitive system resources.

c)

External devices checker

The use of medical sensors and other external devices is growing among smartphone users, and mHealth apps in particular are increasingly expected to support communication with external sensors and monitoring devices. Therefore, SML includes another special tool used by the manager specifically to check connections to external devices. For example, the device-checker tests against the Device MisBonding (DMB) attack that was exposed by Naveed et al. (2014).

The device-checker monitors the apps that use sensors or external devices (any device outside the smartphone), and checks the information that is being transferred between the smartphone and the external sensors or devices. It is invoked by the manager when an app accesses the external device (e.g., heartbeat reader) through Bluetooth or Wi-Fi, the two most common connectivity options on smartphones. On the basis of security and privacy policies, the manager takes the decision whether or not the app is allowed to communicate (send and receive) that information with the external device. The device-checker also provides the manager with information on whether the communication with the external device is encrypted or not.

d)

Installation-checker

Another tool in the arsenal of the SML manager is the installation-checker. This checker is unlike the above checkers, this works only at the time of installation of a particular app. Some security rules have been developed for the installation-checker regardless of the installed app or the user-provided policies. When users want to install a new app, installation-checker first checks its permission and requested resources, as well as other meta-data that the manager needs to decide whether to accept the new installation or raise an alarming notification for the user.

When users install an app, they grant the app all permissions it requests, and have to trust on the way the app uses the granted permissions and smartphone's resources. Although there is now an option available since Android 6.0 that enables users to revoke permissions, still users

are not aware of how the app is using the device resources after granting the permission. Apps can easily misuse smartphone's resources to compromise user privacy, and can reveal user sensitive medical information in case of mHealth apps. Presently, after granting the permissions to an app, there is no way to impose extra constraints on how, when and under what circumstances those granted permission can be used. The actual problem lies in the inability of users to really understand the implications of the granted permissions and correctly judge their approval.

Installation-checker checks the newly requested permissions against few predefined combinations of permissions that can be dangerous for mHealth apps. It is really difficult for mHealth app users to understand the requested permissions and most importantly how a combination of some requested permissions can misuse medical information or can leak sensitive medical data. So, installation-checker refers to some dangerous combination of permissions and performs analysis at install time.

At the time of installation, installation-checker first extracts the security configuration of an app from the target package manifest, and reports the result to the manager, which evaluates the configuration against a collection of security and privacy policies. If an app's security configuration fails to pass all the policies, then the manager has two options. The more secure choice is to reject the installation of that particular app. Otherwise, installation can continue after giving the user a warning notification that the app could be harmful for the sensitive medical data. Obviously, this is a less secure option for the users who usually install apps ignoring any warnings.

For example, the manager might take the decision to either allow or deny the installation of a particular app based on the following dynamics: 1) do not accept any app from other than Google Play store, 2) do not install an app if the developer(s)' name(s) appear in a blacklist (i.e., users can add a developer's name in that list on the basis of their own experience with apps from that particular developer), 3) do not install an app if it has some strange features (i.e., strange features are a list of some features, such as an app wants to access camera, however, it does not need it to fully perform its functionality).

The policies related to installation-checker are defined in Section 4.2.4.

e)

Taint analyser

Determining how an app uses and reveals privacy-sensitive information is achievable using fine-grained taint analysis, commonly called “taint tracking”. A taint is simply a label on a data item or variable. The label assigns a semantic type (e.g., geographic location of device) to the data, and may simultaneously use multiple such types (commonly called a taint tag). It is the task of the taint tracking system to (1) assign taint labels at a taint source, (2) propagate taint labels to dependent data and variables, and finally (3) take some action based on the taint label of data at a taint sink.

The taint-analyser traces the flow of information, and if there is disclosure of sensitive information, for example via transmission of information from an app's component to the Internet, then it notifies the manager. The taint-analyser can be invoked by the manager in several occasions and in tandem with other checkers for tracking sensitive information and comparing against the context, connected device, and security policies.

The static taint analysis technique can keep track of sensitive tainted information through the app by starting at any one from a list of sources and then following the data flow until it reaches any one from a list of sinks. So, it reveals which sensitive data is being leaked to which sink channel. Static taint analysis has been used in many previous research works (Arzt et al, 2014, Lu et al, 2012, Rasthofer et al, 2014). Dynamic taint analysis has also been used in some works such as TaintDroid (Enck et al., 2014). Both dynamic and static analysis can be used to achieve taint tracking. However, dynamic analysis may require many test runs to reach appropriate code coverage. In case of dynamic analysis, however, a malicious app can be developed to be able to recognize the behaviour of dynamic analysis and pose itself as a benign app to bypass the detection. Dynamic analysis also entails a heavy overhead on the performance of the system if used in real time. For these reasons, static analysis is chosen over dynamic analysis for data leakage detection in MASF.

Implementing static analysis on Android is very challenging due to the special design of Android OS. Existing data flow analysis techniques are not directly applicable to Android apps due to the Android programming paradigm's special multiple-entry points. Unlike a Java program, Android app does not have a single entry point, and many entry points can be defined for an Android app. As Android apps have four main components (i.e., activities, services, content providers and broadcast receivers), Android framework can call the methods associated with these components to start and stop the components. To be able to effectively predict the data flow, static analysis need not only precisely model the life-cycle of components but also need to integrate callbacks for system-event handling, UI interaction and so on.

Fig. 2 shows the design of the taint-analyser. This design is basically based on FlowDroid (Arzt et al., 2014) and extended by He (2014). Tainting searches through the app for lifecycle and callback methods by parsing various Android-specific files, including the manifest file, layout XML files, Java source files and so on. Then, a list of sources and sinks is constructed from labels defined by developers in the source code. After that, tainting generates a main method as a single entry point for the Android program from the list of lifecycle and callback methods. This main method is used to generate a call graph for the taint analysis. The taint analysis reports to the manager any possible links between the sources and sinks as warnings of potential vulnerabilities.

Download high-res image (185KB)

Download full-size image

Fig. 2. Static taint-analysis system design.

3.2.1.2. SML manager

The manager component is the heart of the Security Module Layer. It performs the core function of monitoring references and actions taken by the mHealth apps and uses other components of SML to both examine those references and actions against the defined policies in the policy-database and to enforce the policies through the action-performer component. There are many types of actions that action-performer can take based on the manager direction, such as data shadowing, blocking access to data, granting access to data, revoking permissions, installation control, saving state, and disabling intents.

MASF extends Android's middleware through the System Interface Layer (SIL). For example, the PackageManager in Android is responsible for the installation of new apps. The PackageManager is extended by the SIL to interface with SML manager, so the manager can interact with the apps, and can enforce the necessary security and privacy rules and policies for mHealth apps.

Android uses the mechanism of Inter-Component Communication (ICC) as the primary method of communication between apps. However, ICC is technically based on IPC at the kernel level, and it can be seen as a logical connection in the middleware. Access control on ICC is important for the enforcement of security and privacy policies in the middleware.

Different types of ICC can be used by apps for communication. First, the most common way for apps to communication through ICC is to establish direct communication links, known as Direct ICC. For example, an app can send an Intent to another app, query its content provider, or connect to its service. The manager detects this communication through hooks provided by the SIL and prevents it in case the sender and receiver apps of the ICC are not allowed to communicate or exchange specific information according to the corresponding policies. However, system apps form an exception and direct ICC is not prohibited if either the sender or receiver of the ICC is a system app.

Besides the direct ICC, apps can also send Broadcast Intents, which are delivered to all registered receivers. Similar to the approaches followed by Bugiel et al. (2011) and Ongtang et al. (2012), the manager filters out all the receivers of a broadcast intent who are not allowed according to the policies before the broadcast is delivered. Again, system apps have an exception and are not filtered from the receivers list.

附录 D

题录【1】
【作者】龙达鑫
【题名】接口回调实现安卓列表控件适配器和 Fragment 的通信
【摘要】列表控件和 Fragment 控件都是安卓开发中极为常用的两种控件，列表控件是将多项内容相似的布局以列表滚动的形式进行展示，而 Fragment 是一种嵌套在活动中的 UI 组件。对比 Fragment 和列表控件适配器的各种数据通信方式，提出利用接口回调实现 Fragment 和列表控件适配器之间的通信。针对 Fragment 的特点和列表适配器各自的特点，进一步阐述了如何使用接口回调实现数据通信，并给出利用接口回调实现 Fragment 与列表适配器之间通信的案例。
【关键字】安卓; Fragment; 通信; 接口回调;
题录【2】
【作者】许庆建
【题名】个人投资者股票投资风险管理思考
【摘要】复利是投资的关键,投资损失是负复利效应的外部显现,其本质是个人投资者对认知风险、人性风险、市场风险、公司风险、极端风险等股票投资风险缺乏理性认知和有效管理。股票投资是一个建立在风险管理基础上,追求稳健持续长期复利增值的过程,而保证这一过程实现的有效投资策略是 50：50 再平衡策略。
【关键字】股票投资; 风险管理; 复利; 再平衡策略;
题录【3】
【作者】龙达鑫
【题名】运用 JAVA 异常机制分析安卓应用程序崩溃
【摘要】安卓手机已成为日常生活的必需品,科技的日新月异使得手机的应用程序更为多元化。而安卓程序也是我们每天都会接触到的,包括各种社交软件、金融支付类软件,但是你是否经历过安卓应用程序崩溃。究其原因,安卓应用程序崩溃是源程序代码有缺陷所致,安卓源程序的上层 JAVA 代码抛出了异常且未处理,或者处理的不完善所致。本文将以某安卓程序崩溃为切入点,浅析如何运用 JAVA 异常机制分析安卓程序崩溃的原因及如何进行优化。
【关键字】JAVA; 异常; 安卓;
题录【4】
【作者】宋金牛 姜颖
【题名】投资者情绪状态对股票收益的影响
【摘要】从行为金融视角出发,选取封闭式基金等 6 个情绪代理指标,通过主成分分析法并剔除宏观经济因素的影响构建了投资者情绪指数,运用多因素模型划分了不同投资者情绪状态,对投资者情绪指数与股票收益进行回归分析。结果表明,投资者情绪与股票收益之间是正相关关系,在加入三因子控制变量以后,二者之间的拟合度变高;同时还发现,投资者处于情绪高涨期相较于情绪低迷期对股票收益的影响更大

【关键字】投资者情绪; 主成分分析; 股票收益;
题录【5】
【作者】不祥
【题名】JetBrains 开源其 Kotlin 语言,基于 JVM 的新编程语言
【摘要】近日,JetBrains 宣布将其 Kotlin 编程语言开源,这是一个针对 Java 和 JavaScript 平台的现代编程语言,该语言的设计目的是成为 Java 语言的替代品。其主要设计目标如下:兼容 Java。比 Java 更安全,能够静态检测常见的陷阱,如:引用空指针。
【关键字】

题录【6】
【作者】郑森
【题名】股票交易模拟软件的设计与实现
【摘要】股票市场数据显示,中国股民 82%都是亏损的,仅有 18%的股民盈利。这充分说明,股票投资风险大,炒股时需要对股票市场产生的大量历史交易数据进行分析以选择投资方向。现在国内外的一些模拟炒股软件,虽然能满足广大新股民的一些需求,但是这些软件功能复杂,不易操作,并且还存在一个最大的问题,那就是它们都具有荐股功能。炒股新手因此也将被动的选择股票,从而逐渐失去主动分析和研判股票的能力。所以一个能让初学者简单,快捷,并且能达到训练他们能力的虚拟炒股环境是非常有必要的。它可以为初学者提供一个更好的缓冲阶段,促使他们对股票进行研究与判断,增加他们对股票的了解以及炒股的信心,从容入市。本论文就是一个能让初学者简单,快捷,并且能达到训练他们能力的虚拟炒股软件的设计与实现方法论述。该方法实现的模拟炒股软件采用了先进的计算机技术,包括了服务器编程,数据库访问控制,安全策略设计和保证等等。本软件功能主要包括三个部分,分别是系统选项、炒股功能和特色功能。系统选项包括交易提示音设置,快捷键设置,便利计算器以及帮助文档几个部分；炒股功能包括行情、用户信息、交易以及交易查询等功能：特色功能包括股票盘点,交易成功率,选股记事本,综合能力评估。此外,为保证信息传输的准确性,本软件还引入了安全设置,保证了系统的操作安全和数据安全。
【关键字】模拟炒股; 股票分析; 信息安全;
题录【7】
【作者】严瑜
【题名】股票行情分析系统的设计与实现
【摘要】我国的股票市场已有二十多年历史，经历了从无到有，从少数人炒股的低潮阶段到现今全民炒股鼎盛时期，从仅有一家上市公司到现今全国有 2000 多家上市公司。我国的股票市场得到了快速发展，并正向成熟化市场前进。股票市场的发展对我国的经济发展也起到了促进作用，推动了我国经济的迅速发展，为国有企业改革和国民经济的健康发展也做了重要的贡献。在大环境的影响下，炒股已经成为大众的理财的基本方式之一。因此开发一款适合大众使用的股票分析软件是极为必要的，对于我国股票市场的发展是有现实意义的。随着股票市场的发展，开发股票分析软件仍具有广阔的市场前景。

目前，市场上有很多种类的股票分析软件，但都有各自的优缺点，存在各种各样的不足，无法满足所有股民的需求。通过对众多股票软件研究分析发现，大多股票分析软件都具有股票数据分析，股票数据显示，相关参数设置等功能，但也存在操作过程复杂、使用手册不完备、股票数据分析的不准确等问题。对于初学者而言，无疑会降低他们学习的兴趣或对初学者的投资产生误导，这对于股票市场的发展无疑是不利的。本文针对炒股初学者群体，设计了一款简单易懂，易操作的股票分析软件。本软件的操作界面设计简洁，有助于用户迅速抓取信息，对于刚开始炒股的股民来说初期的练习是很重要的，而本软件恰好加入模拟炒股这一功能。其主要功能可分为三大模块：设置模块，视图模块和功能模块，这三大模块中的设这模块实现了对股票参数的设置，视图模块实现了股票数据的视图显示，可供用户对数据进行分析。本软件包含了炒股需要的基本功能，如基本参数设置，个股均线图、K 线图显示，历史记录图片保存和模拟炒股功能，对于一些用户炒股需要分析的基本参数都包含在本软件中；同时为了有一个良好的用户界面，本软件设计了用户注册界面、用户登陆界面、用户登出界面和用户管理界面。本软件能够帮助炒股初学者熟悉炒股流程，熟练的掌握炒股的基本技巧。本文首先介绍了股票的基础知识，然后通过需求分析和系统设计对软件的开发进行了详细的阐述，在需求分析部分给出了软件所需实现功能的描述，在系统设计部分对软件的功能模块进行了模块划分，并针对每一部分功能的实现给出了设计思想和约束，使用 C++实现了本软件的基本功能模块，并用 MFC 实现股票分析软件界面，最后对软件进行了功能测试。

【关键字】证券市场; 股票分析; MFC;

题录【8】

【作者】郑豪 姜荣信 李燕

【题名】基于文献的大学生炒股行为研究调查

【摘要】近年来随着我国资本市场不断进步,证券交易日趋活跃,越来越多的社会群体开始投身股市,其中经管类大学生炒股现象引起了我们高度的重视。大学生投身于股市的原因,社会大众对大学生炒股现象的态度,大学生炒股的行为的特点都是值得研究的议题。本文在阅读大量国内学者的文献资料后对前人的相关研究进行了梳理,以便为后续调查提供了相关前期基础。

【关键字】文献; 大学生; 炒股;

题录【9】

【作者】唐瑜冲

【题名】基于叩富网大学理财课程实践平台的模拟炒股探究

【摘要】大学生作为社会群体中较为激进而又纯粹的消费者且兼具高素质的特殊人群,他们会通过在课堂上老师传授的或课外自学的投资知识与炒股技术,而尝试着以锤炼自身的理财意识、投资能力与财富追逐为动机加入到现实版的炒股大军中,但模拟炒股和现实炒股的体验感是截然不同的。本文基于叩富网大学理财课程实践平台样本数据,以模拟炒股这一事件作为切入点,研究大学生模拟炒股参与情况及炒股绩效,并提出相应的建议。

【关键字】叩富网; 大学理财课程; 实践平台; 模拟炒股;

题录【10】

【作者】谢莉莉

【题名】基于安卓仿真的支付宝云数据取证方法

【摘要】随着手机支付宝软件的不断普及,越来越多的人使用支付宝进行线上或线下的金融交易,除了提供支付功能外,支付宝还提供了第三方服务,包括出行打车、快递查询等服务,因此对支付宝的用户数据进行取证,能获取丰富的用户信息。由于对用户数据安全性的保护,支付宝中大部分重要数据都存储在云端服务器,需要登录后才能获取,因此,该文提出一种基于安卓仿真的支付宝云数据取证方法,基于智能终端设备信息及支付宝数据,定制化虚拟机环境,将智能终端中的支付宝数据迁移至虚拟机,实现免密码登录,获取用户云端服务器数据,为取证人员提供有力线索。

【关键字】安卓仿真; 支付宝; 手机取证;

附录 E

主要源码：

```
package com.example.stock.myapplication

import android.content.Intent
import android.graphics.Color
import android.graphics.Paint
import android.os.Bundle
import android.support.design.widget.TabLayout
import android.support.v7.app.AppCompatActivity
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.widget.RadioButton
import android.widget.TextView
import com.example.stock.App
import org.jetbrains.anko.find
import com.example.stock.R
import com.example.stock.bean.KLineEntity
import com.example.stock.presenter.InternetPresenter
import com.example.stock.presenter.InternetPresenterImpl
import com.github.mikephil.charting.charts.BarChart
import com.github.mikephil.charting.charts.CombinedChart
import com.github.mikephil.charting.components.XAxis
import com.github.mikephil.charting.components.YAxis
import com.github.mikephil.charting.data.*
import com.google.gson.Gson
import java.text.SimpleDateFormat
import java.util.*
import kotlin.collections.ArrayList


class KLineActivity : AppCompatActivity(), TabLayout.OnTabSelectedListener,IKlineView {

    private lateinit var tabLayout:TabLayout
    private val kltype = intArrayOf(1, 2, 3, 4, 5, 6)
    private var index = 2//TabLayout 选中下标
```

```
    private val KLINE = arrayOf("1m", "5m", "15m", "30m", "1h", "1d")
    private val URL =

"https://openapi.dragonex.im/api/v1/market/kline/?symbol_id=%s&st=%s&direction=%s&co
unt=100&kline_type=%s"
    private var toLeft=false
    private var lineBarEntries=LinkedList<BarEntry>()
    private var candleBarEntries=LinkedList<BarEntry>()
    private var dataList=LinkedList<List<String>>()

    private var lineXVals=LinkedList<String>()
    private var candleXVals=LinkedList<String>()
    private    var stockList=LinkedList<CandleEntry>()
    private var minValues=LinkedList<Entry>()
    private var candleSet:CandleDataSet?=null
    private var lineSetMin:LineDataSet?=null
    private var barSet:BarDataSet?=null
    private lateinit var candle_chart:CombinedChart
    private lateinit var barChart:BarChart
    private var combinedData: CombinedData?=null
    private    var barData: BarData?=null
    private lateinit var coin_id:String
    private lateinit var coin:String
    private lateinit var buy_in:RadioButton
    private lateinit var internetPresenter: InternetPresenter
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_multi_stock_graph)
        coin_id=intent.getStringExtra("coin_id")
        coin=intent.getStringExtra("coin")

        initView()
        loadData()
    }
    private fun initView(){
        internetPresenter=InternetPresenterImpl(this)
        candle_chart=find(R.id.cc_kl)
        barChart=find(R.id.bc_kl)
```

```
        tabLayout = findViewById(R.id.tl_kl)
        buy_in=findViewById<RadioButton>(R.id.buy)

        for (i in KLINE.indices) {
            val v = LayoutInflater.from(this).inflate(R.layout.item_tab_kline,null) as
TextView
            v.text = KLINE[i]
            tabLayout.addTab(tabLayout.newTab().setCustomView(v), i == index)
        }
        tabLayout.addOnTabSelectedListener(this)
        buy_in.setOnClickListener(object: View.OnClickListener{
            override fun onClick(v: View?) {
                val bundle = Bundle()
                bundle.putString("coin_id", coin_id)
                bundle.putString("coin",coin)
                val Intent = Intent(App.instance(), BuyActivity().javaClass)
                Intent.putExtras(bundle)
                startActivity(Intent)
                finish()
            }

        })
    }

    private fun loadData() {
        clearChart()
        toLeft = true
        getData("0")

    }
    private fun setData(){
        if (index==0){
            combinedData= CombinedData(lineXVals)
            lineSetMin= LineDataSet(minValues,"1m")
            combinedData!!.setData(LineData(lineXVals,lineSetMin))
            barSet= BarDataSet(lineBarEntries,"bar")
            barData=BarData(lineXVals,barSet)
        }else{
```

```
            combinedData= CombinedData(candleXVals)
            candleSet=CandleDataSet(stockList,"kline")
            candleSet!!.increasingColor=Color.RED
            candleSet!!.decreasingColor=Color.GREEN
            combinedData= CombinedData(candleXVals)
            combinedData!!.setData(CandleData(candleXVals,candleSet))
            barSet= BarDataSet(candleBarEntries,"bar")
            barData=BarData(candleXVals,barSet)
        }
        candle_chart.data=combinedData
        candle_chart.xAxis.textColor=Color.YELLOW
        candle_chart.xAxis.position= XAxis.XAxisPosition.BOTTOM
        candle_chart.axisRight.isEnabled=false
        candle_chart.axisLeft.isEnabled=true
        candle_chart.axisLeft.setLabelCount(5,false)
        candle_chart.axisLeft.textColor=Color.YELLOW
        candle_chart.axisLeft.setDrawZeroLine(false)



        barChart.data=barData
        barChart.xAxis.textColor=Color.YELLOW
        barChart.xAxis.position=XAxis.XAxisPosition.BOTTOM
        barChart.axisLeft.setLabelCount(5,false)
        barChart.axisLeft.textColor=Color.YELLOW
        barChart.axisRight.isEnabled=false
        barChart.axisLeft.setDrawZeroLine(false)

        candle_chart.setDrawValueAboveBar(false)
        barChart.setDrawValueAboveBar(false)
        candle_chart.setVisibleXRange(25f,25f)
        barChart.setVisibleXRange(25f,25f)
        barChart.invalidate()
        candle_chart.invalidate()
    }
    private fun getData(time: String) {
        Log.e("index",index.toString())
        val url = String.format(URL,coin_id, time, if (toLeft) "2" else "1", kltype[index])
```

```
        internetPresenter.getKlineData(url)
    }
    private fun clearChart() {
        lineBarEntries.clear()
        lineXVals.clear()
        candleXVals.clear()
        stockList.clear()
        candle_chart.setNoDataText("加载中...")
        candle_chart.clear()
        barChart.setNoDataText("加载中...")
        barChart.clear()
    }
    override fun onDataOk(json: String) {
        Log.e("klineActivity",json)
        val kl=Gson().fromJson<KLineEntity>(json,KLineEntity::class.java)
        dataList.addAll(kl.data!!.lists)
        formateData()
        setData()
    }


    override fun onTabReselected(p0: TabLayout.Tab?) {
        onTabSelected(p0)
    }

    override fun onTabUnselected(p0: TabLayout.Tab?) {
    }

    override fun onTabSelected(p0: TabLayout.Tab?) {
        index = p0!!.position
        loadData()
    }
    private fun formateData(){
        var temp: Long
        var i=0

        if (index==0){//分时图，仅画 line
            for (data in dataList){
```

```
                    temp=data[6].toLong()*1000
                    val x = SimpleDateFormat("yyyy/MM/dd HH/mm/ss",
Locale.getDefault()).format(Date(temp))
                    lineXVals.add(x)
                    minValues.add(Entry(data[1].toFloat(),i))
                    lineBarEntries.add(0,BarEntry(data[8].toFloat(),i ))
                    i++
                }
            }else{
                for (data in dataList){
                    temp=data[6].toLong()*1000
                    val x = SimpleDateFormat("yyyy/MM/dd HH/mm/ss",
Locale.getDefault()).format(Date(temp))
                    candleXVals.add(x)
                    stockList.add(CandleEntry(i, data[2].toFloat(), data[3].toFloat(),
data[4].toFloat(), data[1].toFloat(), x))
                    candleBarEntries.add(BarEntry(data[8].toFloat(),i ))
                    i++
                }
            }
        }

}
package com.example.stock.db

import android.database.sqlite.SQLiteDatabase
import com.example.stock.App
import com.example.stock.bean.userTable
import org.jetbrains.anko.db.*

class
UserDbHelper :ManagedSQLiteOpenHelper(App.instance(),UserDbHelper.DB_NAME,null,
DB_VERSION){
    companion object {
        const val DB_NAME="user_1.db"
        private val DB_VERSION=1
        public val instance:UserDbHelper by lazy { UserDbHelper() }
    }
```

```
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.dropTable(userTable.TNAME,true)
        onCreate(db)
    }

    override fun onCreate(db: SQLiteDatabase?) {
        db?.createTable(userTable.TNAME,true,
                Pair(userTable.UID, INTEGER + PRIMARY_KEY+
AUTOINCREMENT),
                Pair(userTable.UNAME, TEXT),
                Pair(userTable.PWD,TEXT),
                Pair(userTable.RIGHT, INTEGER),
                Pair(userTable.TOTAL_PROPERTY,REAL),
                Pair(userTable.LEFT_PROPERTY, REAL),
                Pair(userTable.MARKET_VALUE, REAL),
                Pair(userTable.RATE_OF_RETURN, REAL))
    }

}
package com.example.stock.bean


/**
 * K 线数据
 */
class KLineEntity(var ok: Boolean=false, var code: Int=0, var msg: String="", var data:
datatype?=null)
class datatype(val columns: List<String>, val lists:List<List<String>>)

package com.example.stock.presenter

import com.example.stock.App
import com.example.stock.bean.real
import com.example.stock.bean.trade
import com.example.stock.bean.userTable
import com.example.stock.db.TradeListDb
import com.example.stock.db.UserDb
```

```
import com.example.stock.util.Internet_util
import com.google.gson.Gson

class IDBTradePresenterImpl:IDBTradePresenter{
    private val url="https://openapi.dragonex.im/api/v1/market/real/?symbol_id=%s"
    /**
     * if the coin_id of the trade has been save ,
     * then we should search it from the db.
     * and update the item
     */
    override fun addTrade(buy_price:Float,volume:Float,code:String,coin_id:String):Boolean {
        if(buy_price<0f||volume<0f)
            return false
        if(!App.isLogin()){
            return false
        }else{
            val loginUser=App.getLoginUser()
            val left= loginUser!!.left_property-(buy_price*volume)
            if(left<0)
                return false
            else{
                TradeListDb().insertTradeItem(trade(-1,loginUser.uid,code,0f,volume,buy_price,coin_id,0f,volume,buy_price,"0"))
                return true
            }
        }

    }

    override fun updateTradeList() :Boolean{
        /**
         * TradeDb().query.where uid= &uid = list<trade>.
         * for each list of list
         * search it and update the user message and update the trade info
         *
         */
        val user=App.getLoginUser()
```

```
        if(user==null)
            return false
        val result=TradeListDb().queryTradeList("${userTable.UID}=${user.uid}")
        var url:String
        var json:String
        var rl:real
        var market_value=0f
        for ( index in result){
            url = String.format(this.url,index.coin_id)
            json= Internet_util.senRequestForData(url)
            rl= Gson().fromJson<real>(json, real::class.java)
            val now_price=rl.data.get(0).close_price.toFloat()
            index.profit_loss_rate=((now_price-index.buy_price)/index.buy_price)*100
            index.profit_loss=index.chicang*index.profit_loss_rate
            index.now_price=now_price
            market_value=market_value+index.chicang*now_price
            TradeListDb().updateTradeItem(index)
        }
        user.market_value=market_value
        user.total_property=user.market_value+user.left_property
        user.rate_of_return=((user.total_property/10000f)-1)*100
        App.putLoginUser(user)
        UserDb().updateUser(user)
        return true
    }

    override fun soldTrade(trade:trade):Boolean {
        val user=App.getLoginUser()
        if(user==null){
            return false
        }else{
            user.left_property+=trade.chicang*trade.now_price
            user.market_value-=trade.chicang*trade.now_price
            user.total_property=user.left_property+user.market_value
            return TradeListDb().deleteTradeItem(trade.id)!=-1
        }
    }
```

```
}
package com.example.stock.util

import android.annotation.SuppressLint
import android.util.Log
import com.example.stock.bean.stock
import org.json.JSONArray
import org.json.JSONObject
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL
import java.text.SimpleDateFormat
import java.util.*
import kotlin.collections.ArrayList


/**
 * @param    lastDate the time of the last line data in the databse
 * @param lastId the id of the last line data in the database
 * @return the new data from web
 */
object Internet_util {


    fun getData(lastId: Long, lastDate: String, stockname: String): List<stock> {
        val str = senRequestForData(stockname)
        //val                                    results                                    =
Gson().fromJson<List<stock>>(str,object :TypeToken<List<stock>>(){}.type)
        val jsonArray = JSONArray(str)
        //转换数据类型
        var index = 0
        var lastid = lastId
        val results = ArrayList<stock>()
        while (index < jsonArray.length()) {
            val obj: JSONObject = jsonArray.getJSONObject(index)
            if (compareDate(lastDate, obj.getString("day").toString())) {
                val newstock = stock(
```

```
                        ++lastid,
                        obj.getString("day"),
                        obj.getString("open").toFloat(),
                        obj.getString("high").toFloat(),
                        obj.getString("low").toFloat(),
                        obj.getString("close").toFloat(),
                        obj.getString("volume").toFloat())
                results.add(newstock)
            }
            index++
        }
        Log.d("getWebData", results.size.toString())
        return results
    }

    /**
     * @param stock
     * @return the function will download data from web
     *          and return data as json-string
     */
    fun senRequestForData(stock: String): String {
        val connection: HttpURLConnection?
        val reader: BufferedReader?
        //val rawUrl =
"http://money.finance.sina.com.cn/quotes_service/api/json_v2.php/CN_MarketData.getKLine
Data?symbol=$stock&scale=5&ma=no&datalen=1023"

        Log.d("url",stock)
        val url=URL(stock)
        connection = url.openConnection() as HttpURLConnection
        with(connection){
            requestMethod = "GET"
            connectTimeout = 60000
            readTimeout = 60000
        }


        val inStream = connection.inputStream
```

```
            reader = BufferedReader(InputStreamReader(inStream))
            val response = StringBuilder()
            val allText = reader.use(BufferedReader::readText)
            inStream.close()
            response.append(allText)
            return response.toString()
        }

        @SuppressLint("SimpleDateFormat")
        fun convertTODate(Str: String): Date? {
            return SimpleDateFormat("yyyy-MM-dd HH:dd:ss").parse(Str)
        }
        /**
        *if the newdate is after this date
        * the function will return true
        *
        **/
        fun compareDate(lastDate: String, newDate: String) =
                convertTODate(lastDate)?.compareTo(convertTODate(newDate)) == -1
}
package com.example.stock

import android.app.Application
import com.example.stock.bean.coin
import com.example.stock.bean.user
import com.example.stock.db.CoinDb
import com.example.stock.util.OkHttpUtil

class App:Application(){
        companion object {
            private var instance:Application?=null
            private var login_user:user?=null
            fun instance()= instance!!
            fun getLoginUser()= login_user
            fun putLoginUser(user: user){
                login_user=user
            }
            fun isLogin()= login_user!=null
```

```
    }

    override fun onCreate() {
        super.onCreate()
        instance=this
    }
}
```