令人发指的 WHOEVE 开发报告

————BY 张涵（逃

摘要： 随着网络技术地发展，使得人们进入了移动互联的时代。人们从单纯地、麻木地全盘接收媒体信息的方式转变成以微博传播为主得主动“获取信息”的方式，再到以微信为主的基于“强关系链接网”地虚拟与现实的“无缝对接”。滴滴打车是如今很火的另一款软件，通过客户端进行通信，实现乘客和司机的通信连接，极大地方便了人们的出行。如何以及怎么在类似微信的聊天软件中加入像滴滴打车软件的功能，实现乘客和驾驶员之间的即时通讯，更加方便快捷地了解相互之间的更多需求，是本次的研究课题。

本课题是基于 Android 平台开发的 Whoeve 通讯与像滴滴打车 APP 的设计与实现，主要包括需求分析、概要设计、详细设计、技术分析、项目测试以及总结等内容,将具体详细的介绍此技术开发中所涉及到的具体技术和功能的实现。

该项目使用的开发工具是 Android Studio，该工具属于原生安卓类移动应用开发。该项目主要涉及技术知识即：在线即时通讯、线程池连接、基于 Socket 的网络连接、频道分化、坐标获取与发送、数据库操作、网络编程、安卓数据存储等技术点。项目为用户提供了比较全面的 Whoeve 即时通讯及打车功能，包括在线语音通讯、即时通讯、范围寻找其他用户、GPS 定位与坐标获取、地图、社区论坛、个人中心等等几大模块。移动端是采用安卓原生开发实现界面交互与数据展示，服务端则是使用 Socket 后端云服务平台来实现接口处理，图片和视频等文件将其存储在百度云端服务器平台，实现加速访问与文件处理等复杂操作。此应用方便了用户的出行，并且与滴滴打车相比，不仅有即时通讯的方便之处，而且流量消耗较少，有较高的实用价值。

关键词： 安卓；在线即时通讯；打车；GPS 定位与地图；分频道群聊；

Design and implementation of mobile social software based on geographic location

Abstract: With the development of network technology, people have entered the era

of mobile internet. People changed from simply, numb all the way to receive the micro-blog Communication Initiative "access to information", and then to WeChat mainly based on "strong links to" virtual reality "seamless docking". Didi taxi is now a fire of another software, through the client to communicate, to achieve the communication connection between passengers and drivers, greatly facilitates people's travel. How and how similar to WeChat chat software join the taxi drops software function, realize the real-time communication between passengers and drivers, more convenient to learn more needs of each other, is the research topic of the.

This topic is based on the Whoeve communication Android platform development and design and implementation as a taxi drops APP, including demand analysis, outline design, detailed design, technical analysis, project testing and summary content, the specific details of specific techniques and functions related to the development of this technology in the.

The project uses the development tool is Android Studio, which is a native Android mobile application development. This project mainly relates to the technical knowledge that online instant communication, thread pool connection, Socket based network connection, channel differentiation, coordinate acquisition and technology transmission, database operation, network programming, Android data storage etc.. The project provides a more comprehensive Whoeve instant messaging and taxi function for users, including online voice communication, instant messaging, search other users, GPS positioning and coordinate several modules to obtain, map, community forums, personal center etc.. The mobile terminal is using Android native interface development interaction and data display, the server is to use the Socket back-end cloud service platform to realize the interface processing, picture and video files will be stored in the Baidu cloud server platform, accelerate access and file processing and other complex operations. This application is convenient for the user to travel, and compared with drops taxi, not only the convenience of instant messaging, but also less traffic consumption, has a higher practical value.**Key words:** Android; online instant messaging; taxi; GPS positioning and map; channel group

chat

第一章：课题研究背景及意义

为更好的满足用户的需求，把打车功能与在线即时通讯相结合。客户端及服务器使用开发工具 Android Studio 进行开发，而客户端与服务器之间的连接则使用基于 Socket 的网络连接。当服务器端套接字监听到客户端套接字的连接请求时，它便响应客户端套接字的请求，去建立一个新的线程，把服务器端套接字的描述信息发给客户端，服务器端套接字则继续处于监听状态，从而继续接收其他客户端套接字的连接请求。并且同时使用线程池技术使服务器能够同时建立多个线程，便于多个用户同时登入。服务器端则主要使用阿里云端服务器，用于解决云数据库问题，不需再买服务器便可实现数据存储和在线访问及一些常用的第三方服务。GPS 功能则是使用精度比较高的高德地图 API，解决坐标获取与发送等位置问题。

第二章：技术分析

其主要介绍项目在实施过程中使用的关键技术，分析讨论在线即时聊天、云数据库、自定义 View 和 View Group、服务器和客户端架设、Socket 通信连接、Android 图片与语音处理、线程池处理等方面涉及到的具体技术点。

2.1　云端服务器

随着“云”概念不断进入我们生活的方方面面，这个作为世界尖端前沿的移动平台，Android 则是赶上时代潮流，把“云”带入了 Android 的各个方面，很典型的例子就是“移动后端服务”。[(移动后端服务——Backend as a Service 也叫 Baas，即是将服务端的内容进行打包，把数据库的设计、服务器的搭建以及后台逻辑的处理等都存放在了云端，这样用户只需要调用他们提供的 API 接口便可以实现其网络通讯功能。)Baas—其概念最早来源于一些国外的网站，比较老的有 Stack Mob 和 Parse。之后 Baas 便如雨后春笋般发展壮大，而其主要为用户提供云端逻辑、文件存储、容器服务、应用统计、消息推送等功能。]

云端服务器可以给开发者提供方便、快速的云数据库服务，以便于用户浏览查询终端保存的各种信息。而后端云服务器在硬件和软件层面都为用户建立了安全机制。所以使用云端服务器则无需再打造服务端，就能轻松拥有应用开发地后

端能力支持，从而减少了开发者将应用从 idea 再到产品（成品）的时间。增加了密钥匹配、数据安全隔离、多租户虚拟化技术、分布式存储架构、访问权限控制等功能，从而进一步保障数据的安全，降低风险。阿里云平台主要的四大优势是可视化操作、全平台支持接入、多种数据类型和开放自由。

2.2 阿里云数据存储

本地服务器无法胜任的进行聊天文字和语音记录或个人信息的存储时，则采用阿里云端数据存储。

阿里云端数据存储主要使用的是 RAM 和 STS 权限管理系统。其中 RAM 的主要作用是控制账号系统的权限。即通过使用 RAM 可以在主账号的权限范围内创建子账号，然后给不同的子账号分配不同的权限进而达到授权管理的目的。STS 则是一个安全凭证（Token）的管理系统，是用来授予临时的访问权限，这样便可以通过 STS 去完成对于临时用户的访问授权的权利。

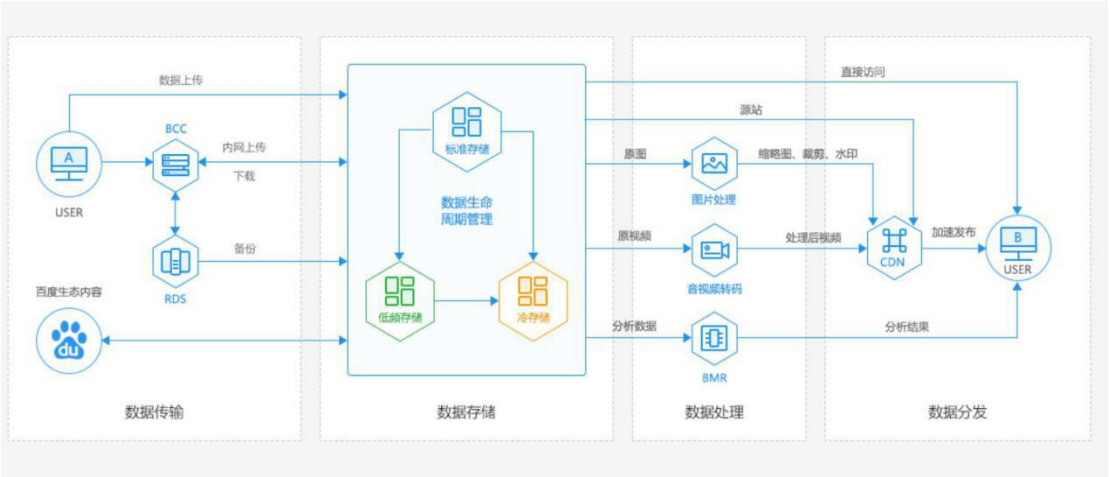
如何在不暴露主账号的 Access Key 的情况下安全地授权别人访问是 RAM 和 STS 需要解决的一个核心问题。这是因为一旦主账号的 Access Key 暴露出来，就会带来极大的安全风险，因为如果这样别人便可随意操作此账号下所有资源或盗取重要信息等。

RAM 实际上提供的是一种长期的、有效的权限控制机制，它是通过分出不同权限的子账号，然后将不同的权限分给不同的用户，一旦子账号泄露出去也不会造成全局的信息泄露。在一般情况下，子账号是长期有效的。所以，子账号的 Access Key 是不能泄露的。

相比较 RAM 提供得长效控制机制，STS 提供得则是一种临时访问授权，即通过 STS 能够返回临时的 Access Key 和 Token，而且这些信息是可以直接发给临时用户去用来访问表格存储。总的来看从 STS 获取的权限是会受到更加严格的限制，而且拥有时间限制，因此这些信息泄露之后相对于 RAM 对系统的影响比较小。

相比较传统的 IDC，访问速度则能提升数倍。当使用阿里云的镜像存储然后对源站资源（文件/图片等）进行存储后，便可使用阿里云内置得 CDN 加速进行分发访问。

数据存储过程如下图：



2.3　本地服务器与云端数据存储的对接

本地服务器，也叫独立服务器，它指的是服务器在局域网当中所担任的一种职能。服务器只仅仅向网络内的计算机提供单一得服务，它并不负责网络内计算机得管理职能。而云服务器就简单来说是 VPS 升级版。VPS 是从一台服务器上划分出来一部分的内存、硬盘从而搭建而成的一台虚拟服务器。而云服务器则是在一组集群服务器上划分出来多个类似独立服务器得部分，而且集群中的每台服务器各个都有云主机的一个镜像备份，所以当其中一台机器出现故障的时候，系统便会自动访问镜像备份，进而从根本上提高了云服务器的安全性和稳定性。

因此，按自身需求，然后自己编写本地服务器用来实现客户端基本的功能，用于便于以后进行维护和服务器升级与功能的更新。而个人信息和聊天记录等数据存储则能够使用阿里云端存储服务。即本地服务器是通过本地的代理客户端和云端进行通信，本地代理用来监视本地文件，一有变化便会即时同步到云端，这时云端便负责包括版本管理等的同步管理。

2.4　基于 Socket 的网络连接

网络上的两个程序是通过一个双向的通信连接来实现的数据交换，连接的一端则被称为一个 socket。

建立网络通信连接至少需要两个端口号(socket)。Socket 的本质是编程接口(API)，对于 TCP/IP 得封装，TCP/IP 也要提供可供程序员做网络开发所用得接口，这便是 Socket 编程接口。

Socket 的英文意思是“插座”或“孔”。它作为 BSD UNIX 的进程通信机制，取的是后一种意思。通常也叫做“套接字”，也用于描述 IP 地址和端口，它是一个通信链的句柄，是可以用来实现不同计算机或不同虚拟机之间的通信。在 Internet 上的主机上一般运行了多个服务软件，并同时提供了几种服务。每种服务都是打开一个 Socket，并且绑定到一个端口上，不同的端口对应不同的服务。

不仅如此，最重要的是 Socket 的设计是面向客户/服务器模型，可以针对不同客户和服务器程序来提供不同的 Socket 系统调用。客户可以随机申请一个 Socket（就像一个想打电话的人能够使用任何一台入网电话拨号呼叫），系统便为其分配一个 Socket 号；服务器是拥有全局公认的 Socket，所以任何客户都可以并且能够向它发出连接请求和信息请求（就像一个被呼叫的电话能够拥有一个呼叫方知道的号码）。

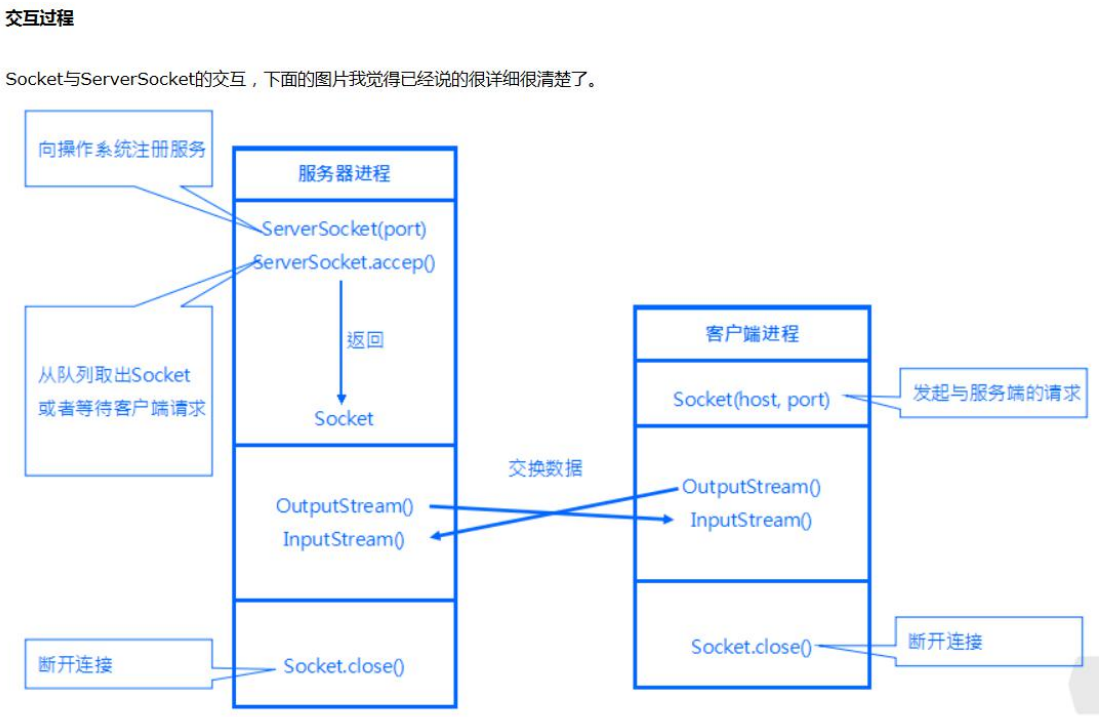
进程之间通信链接的问题被 Socket 利用客户/服务器模式机智巧妙的解决了。

根据本地套接字要连接的目标及连接启动的方式，可以将套接字之间的连接过程分为三个步骤：一服务器监听--二客户端请求--三连接确认。

（1）服务器监听：它是服务器端套接字但并不定位具体的客户端套接字，它是处于等待连接的状态，能够实时监控网络状态。

（2）客户端请求：指的是由客户端的套接字来提出连接请求，而服务器端的套接字是其要连接的目标。因此，客户端的套接字必须首先要描述它所要连接的服务器的套接字，使其指出服务器端套接字的端口号和地址，便向服务器端套接字去提出连接请求。

（3）连接确认：指当服务器端套接字接收到客户端套接字的连接请求或监听到，它便会响应客户端套接字地请求，去建立一个新的线程；然后把服务器端套接字的描述发送给客户端，客户端一旦确认了此描述，连接便建立好了。继而服务器端套接字则继续处于监听状态，继而去接收其他客户端套接字的连接请求。



2.5　线程池处理

处理器单元内多个线程执行的问题是多线程技术主要解决的问题。它能够明显减少处理器单元的闲置时间，从而去增加处理器单元的吞吐能力。

线程池作为一种多线程处理形式。在处理过程中，它先将任务添加到队列，再在创建线程后去自动启动这些任务，线程池和线程都是后台线程。其中每个线程都是使用默认的堆栈大小，然后以默认的优先级运行，使其处于多线程单元中。如果其中某个线程在托管代码中是空闲的（像是正在等待某个事件），那么线程池将会插入另一个辅助线程使其所有的处理器保持繁忙。若所有的线程池线程都是始终保持繁忙的状态，但是队列中则是包含挂起的工作，那么线程池将会在一

段时间后去创建另一个辅助线程，即使如此，但是线程的数目将永远不会超过其最大值。

简单的线程池实现：

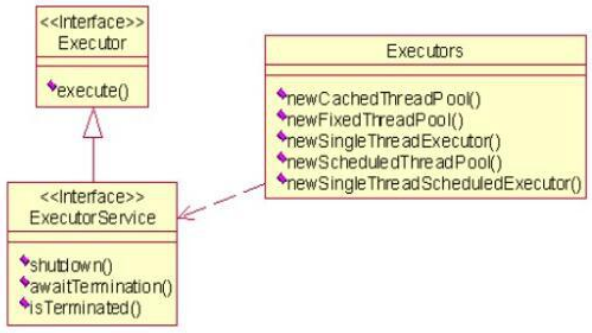
1、线程池管理器（Thread Pool Manager）:它主要是用于创建并且管理线程池。

2、工作线程（Work Thread）: 指的是线程池中线程。

3、任务接口（Task）:指的是每个任务都必须实现的接口，以便供工作线程调度任务的顺利执行。

4、任务队列：主要是用于存放当前没有处理的任务，也是提供一种缓冲机制。

- java.util.concurrent包提供了现成的线程池的实现。



JDK 类库中的线程池的类框图

Executor接口表示线程池，它的execute(Runnable task)方法用来执行Runnable类型的任务。Executor的子接口

ExecutorService中声明了管理线程池的一些方法，比如用于关闭线程池的shutdown()方法等。

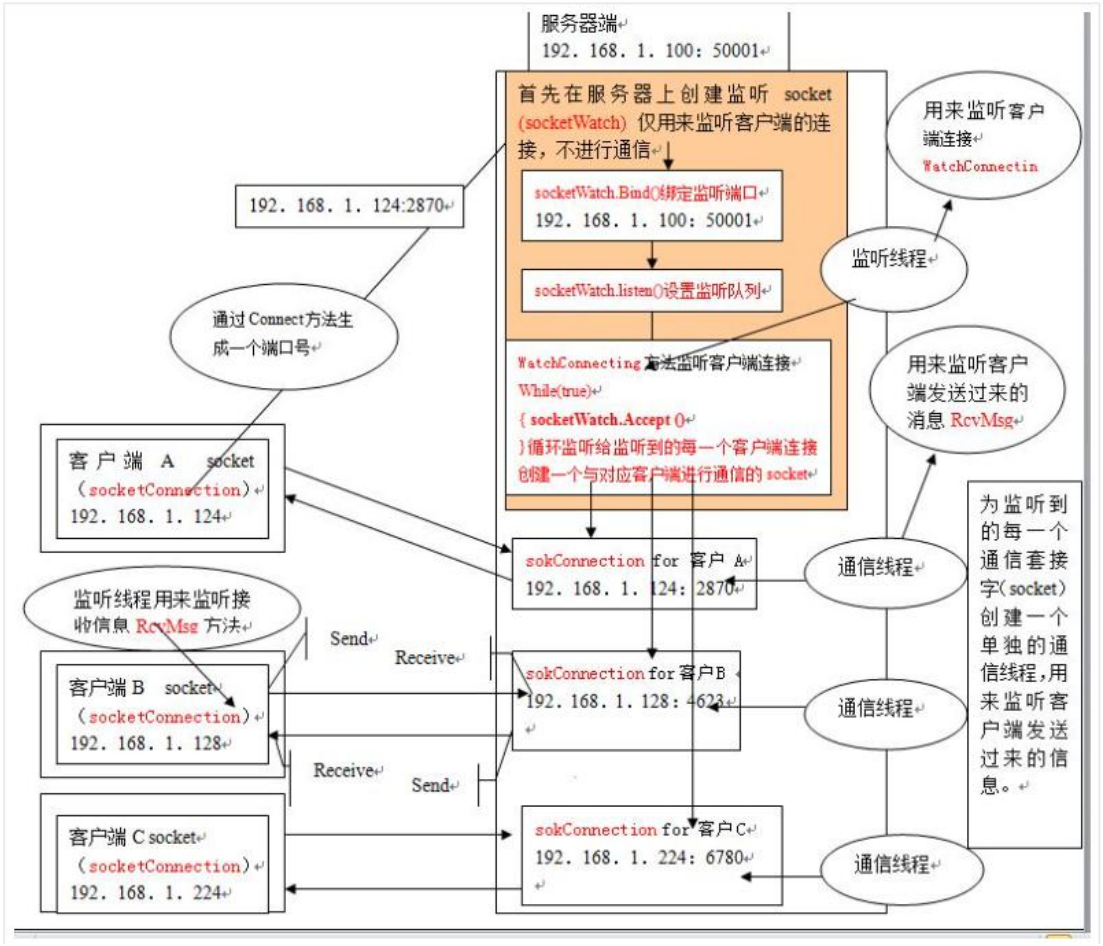
Executors类中包含一些静态方法，它们负责生成各种类型的线程池ExecutorService实例。

Executors 类的生成 ExecutorService 实例的静态方法

Executors 类的静态方法	创建的 ExecutorService 线程池的类型
newCachedThreadPool()	在有任务时才创建新线程，空闲线程被保留 60 秒。
newFixedThreadPool(int nThreads)	线程池中包含固定数目的线程，空闲线程会一直保留。参数 nThreads 设定线程池中线程的数目。
newSingleThreadExecutor()	线程池中只有一个工作线程，它依次执行每个任务。
newScheduledThreadPool(int corePoolSize)	线程池能按时间计划来执行任务，允许用户设定计划执行任务的时间。参数 corePoolSize 设定线程池中线程的最小数目。当任务较多，线程池可能会创建更多的工作线程来执行任务。
newSingleThreadScheduledExecutor()	线程池中只有一个工作线程，它能按时间计划来执行任务。

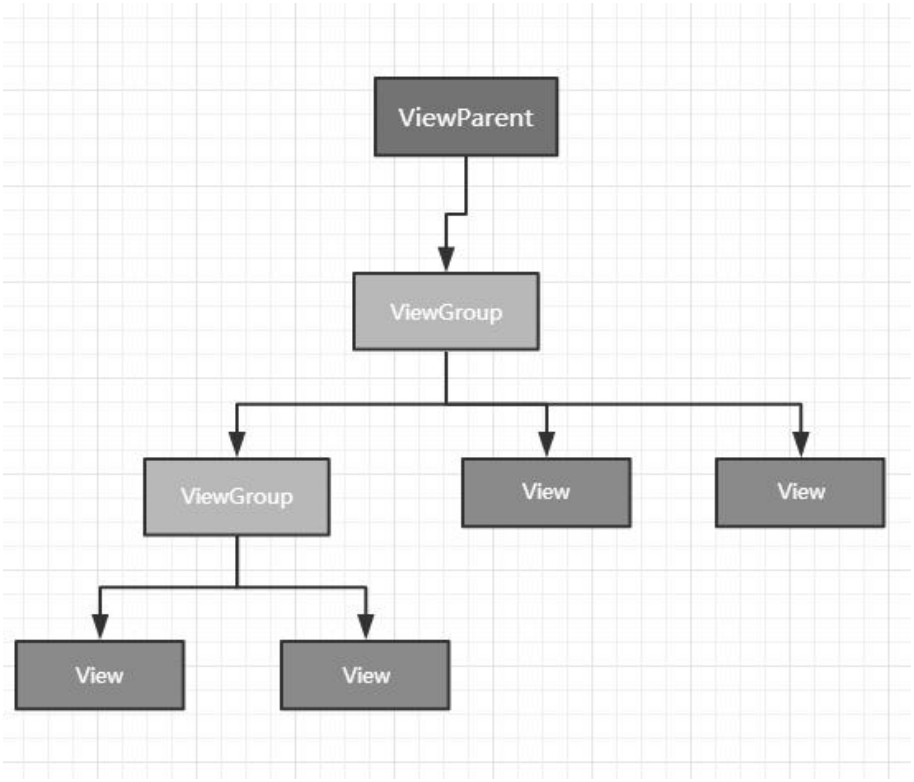
服务器与客户端通过线程池实现通信的简单示意图：

通信过程图



2.6 自定义 View 和 View Group

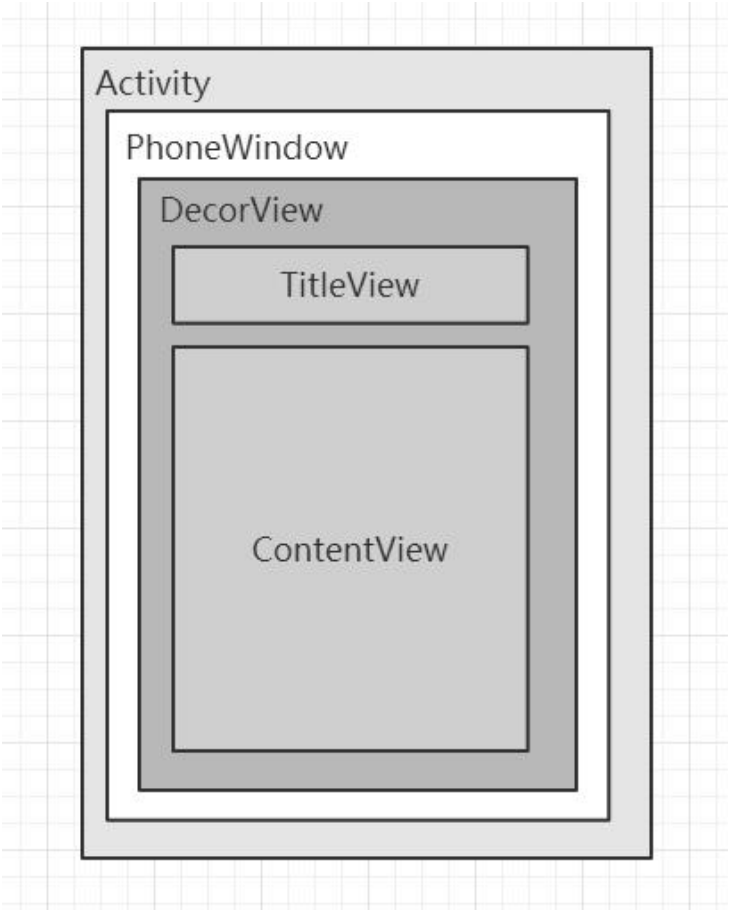
安卓中的控件主要分为两大部分：View 控件和 View Group 控件。从中可以看出，View Group 是 View 的父控件，它可以包含并且管理多个其他 View Group 或者 View。若整合 View Group 便从上到下形成了一颗树形的结构图。最上方的 View Group Wiew 的整个树形结构的父控件，便可而知，自上而下，并且每个上层控件能够负责下层控件的测量与绘制及事件的传递与交互。测量它指的是父布局所需要测量所以字控件的大小，比如高度和宽度，能够设置自己的宽高以便于容纳子控件。而绘制则是通过已经测量好的宽和高，然后将所以子 View 绘制到其相应的位置然后使其展示出来。如图所示其展示了一个 View 视图树：



View 树形结构

每个 Activity 之中都包含一个 Phone Window 对象用来实现其 Window 视图，然后 Phone Window 又将一个 Decor View 设置成为整个布局的根 View。而 Decor View 它作为视图的顶层，View 便封装好了一些非常常用的事件方法，通过 Activity 的对象来回调相对应的 on Click Listener。当 Activity 调用完 on Creat()方法中的 set Content View()方法后，系统便将整个 Decor View 添加到 Phone Widow 中，继而完成整个视图窗口的绘制。

如图展示 Activity 窗体的视图结构：



UI 界面架构图

自定义 View 作为安卓中非常重要的功能之一，它一直以来都被初学者当做代表高手的一个象征。作为一员的我们只有站在设计者的角度上，谨记不要机械的认识记忆所有的绘图 API，而是让这些 API 为你所用，只有这样才能更好地创建自定义 View。

自定义 View 主要是以下几个步骤：

1.自定义 View 它的属性。

2.通过从 View 的构造方法中取得我们自定义的属性。

3.重写 View 的 on Measure()方法进行控件大小的计算。

4.重写 View 的 on Draw()方法将其控件绘制出来。

这里需要注意的是 View 的三种测量模式：UNSPEIFIED、EXACTLY、AT_MOST 之间的区别。View Group 作为一个 View 容器，它需要对其子 View 进行遍历，然后获得其所有子 View 的大小，来决定自己的大小。当测量完子 View 的大小之后，View Group 便同样使用遍历然后来调用子 View 的 on Layout()方法将其具体绘制在它们对应的位置。在自定义 View Group 或 View 的时候，若需要支持

wrap_content 的属性，则都需要重写 on Measure()方法。然后根据其实际情况返回其相应的测量值大小。

2.7 Android 图片处理

移动端与 PC 端有很大不同，其在图片处理方面功能则比较局限，不仅如此，由于手机的运行内存比较小，但系统已经为每个应用分配好其独立的进程与空间的大小，所以一旦应用所需空间超出了其系统所能够给的大小，便会抛出 OOM 异常问题，这便是我们常说的内存溢出。所以在图片处理这个方面，我们有必要先将图片压缩成一定比例后，然后再加载入内存进行显示。并且在进行图片上传操作的时候，也应该将图片质量再进行适当压缩，这样可以节省用户流量，从而能够提高用户体验。由此而知，图片的压缩主要可以分为两个方面：像素压缩与质量压缩。

图片的存在形式有三种：

（1）文件形式（以二进制形式存在于硬盘中）。

(2)流的形式（以二进制形式存在于内存中）。

(3)Bitmap 形式。

这其中文件形式和流形式这两种形式对图片的体积大小没有影响，如一张图片的大小是 200KB，那么这张图片以流形式加载入内存之中也将会是 200KB。但若以 Bitmap 的形式加载入内存之中，占用的内存将会瞬间变大，再严重的情况便会导致手机内存溢出，继而出现 Crash 情况。

图片若以 Bitmap 形式加载到大内存之中变大是因为图片是以由像素点组合而成，每个像素点可以分为 ARGB 模式，而且每一位能够提供 0~255 种不等值，即每位需占用 8 位字节。由此看来，若一张像素非常大的图片加载入内存之中，则每个像素点占用的内存之和将会是一个非常庞大的数字。而图片像素压缩便是指通过压缩图片得横纵像素数，从而使图片的质量大小能够得以压缩，利用 Bitmap Factory 得 options 属性，然后设置其压缩比例，如此这样图片的横纵像素数便将会按其设置比例进行压缩。图片的像素数量得到压缩，那么其加载到内存之中所占用的内存大小也自然将减小。但通常像素被压缩后，图片的清晰度也将被降低，所以该压缩比例应该以手机的分辨率为基础来计算得出，这样能够保证压缩之后的图片清晰度可以处于一个非常合适的范围之内。

反言之，图片的质量压缩法不是压缩图片的像素数量，是改变图片的每个像素点的位深与透明度等值，继而可以改变文件的实际大小。根据其使用场景不同，两种方法通常是会结合使用，整张图片需要通过手机展示给用户观看时，使用像素压缩法；需要将图片上传至服务器时，通常是会使用 Bitmap.compress()方法去压缩图片质量，能够以便减少其图片的大小，可以节约用户的流量。

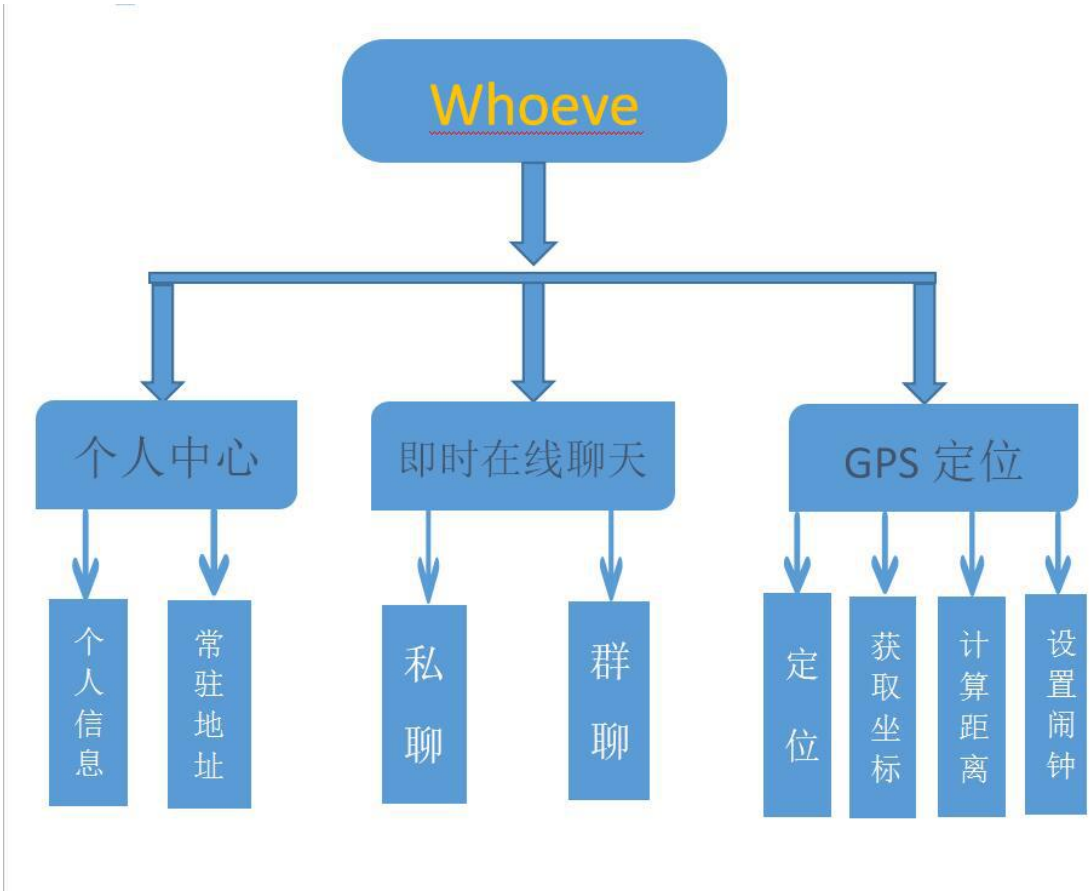
第 3 章　需求分析

本章主要讨论的是对软件整体需求进行分析描述及对一些重要的功能进行详细分析。其中包括注册登陆、个人中心、即时在线聊天、搜索附近的人、坐标的获取和发送、GPS 定位等主要功能的介绍。

3.1　主要功能结构图

项目的主要功能模块可以划分为三个部分即:个人中心、GPS 定位、即时在线聊天。在这其中个人中心的部分主要包括：注册登录、个人信息、常驻地址等等；即时在线聊天的部分主要包括：建群聊频道进行群聊、输入另一用户 ID 进行私聊等等功能；GPS 定位的部分主要包括：定位自己的位置、获取和发送自己的地理坐标及获取附近别人的地理坐标、计算距离与设置 GPS 闹钟等等功能。

主要功能结构图如下所示：



主要功能结构图

3.2　主要功能分析

1.个人中心

个人中心主要包括：登录和注册，首先 APP 需要判断用户是否已登录，未登录用户需要给出登录入口。已登录用户正常显示用户的基本信息。APP 中部分功能需要判断用户登录状态，如果用户未登录则引导用户进入登录页面。我的笔记界面通过自定义高度、背景的 item 样式实现瀑布流布局效果；个人信息，可以填写自己的常驻地址，兴趣爱好，以便好友更加了解自己，还可以修改昵称，彰显个性；在登录时 app 还可以检测手机通讯录，自动匹配账户。

2.即时在线聊天

聊天模块主要通过 Socket 连接与服务器进行通信，将客户端输入的信息传送至服务器，再转存到云端存储，然后在另一客户端显示输出。也可根据需要选择群聊或私聊，选择群聊时，可以建立不同频道，用于讨论不同话题；选择私聊时，可以直接输入对方 ID，服务端接收到的信息就会直接且仅发送至使用该 ID 登录的客户端。在聊天界面会同步显示在线用户的 ID 和信息发送时间。

3.GPS 定位

GPS 定位模块使用国内顶尖地图专家高德地图，在开发者平台申请权限与秘钥，并使用其 JDK 和 JAR 包实现了自己的 GPS 定位功能。高德地图以其领先的地图渲染技术、专业在线导航功能、AR 虚拟实景、丰富的出行查询功能而成为地

图行业佼佼者，而 app 里的地图功能还拥有获取与发送坐标、设置闹钟等功能。其中闹钟功能可以让你选取一个目的地，当你接近目的地时会实现振铃，提醒你即将到达，避免在坐车时因睡觉或其他原因而发生坐过站之类的事情。

第 4 章　界面设计

1.登录界面

登录界面是用户用于登录和注册新账户的地方,这里采用自定义 View,以(任意图片）为背景，Name 为用户账户，Password 为用户密码，点击 loginin 即可登录。

2.个人中心界面

个人中心是用户填写个人资料的模块，有昵称、兴趣爱好、年龄、性别、常驻地址等选项，点击即可填写。

3.聊天界面

聊天界面能够进行即时在线聊天,上方四分之三为聊天信息显示界面,以(任意图片）为背景图片，下方有私聊、群聊选项，在输入框输入聊天信息，在 SentToOne 里输入对方 ID 并点击按钮即可实现私聊，点击 SentToAll 按钮即可实现群聊。界面右下角有定位标志，点击即可获取自己的地理坐标（经纬度）并发送。

4.GPS 地图界面

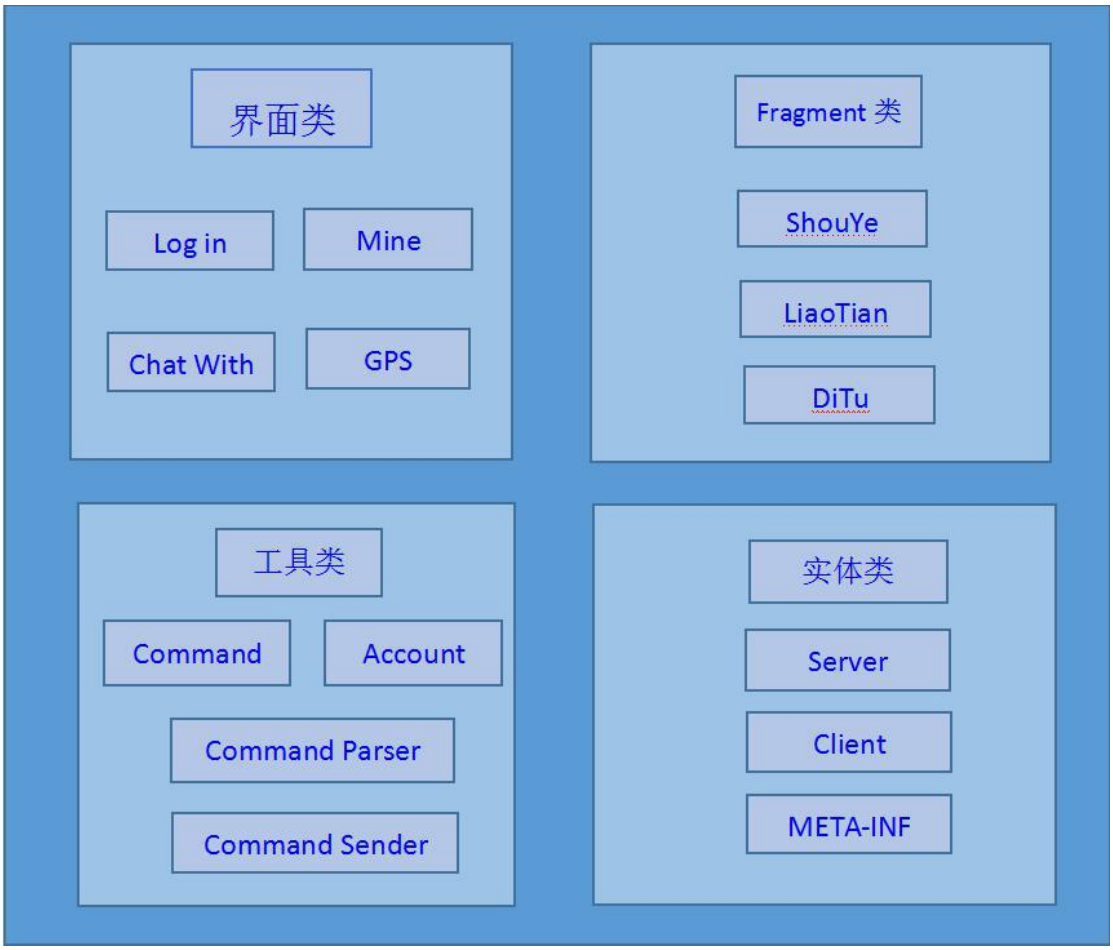
地图界面使用高德地图中的 3DMap 包，点击定位按钮即可定位到自己所在地点；可以通过手势来进行放大缩小；双击屏幕或点击左上角“设置闹钟”按钮然后在地图上点击你要去的地点,即可成功设置闹钟；点击界面左上角“清除闹钟”按钮即可取消闹钟；使用两根手指同时上滑屏幕即可变为 3D 地图。

第 5 章　详细设计

本章将介绍该项目中的主要界面和数据库表的详细设计和实现，对具有代表性的界面和逻辑功能进行详细介绍。包括即时在线聊天、文件云端存储、GPS 地图定位等主要功能，其余辅助类和工具类将进行一些简要介绍。

5.1 项目类框架结构图

项目主要分为 Activity 类、View 类、Command 类、Account 类、DataBase 类、Point 类、Client 类和 Server 类等。将各自的代码归整到不同包下，实现代码的高内聚低耦合性，遵循框架搭建规范，方便以后的代码优化和更新，主要类框架结构如图所示：

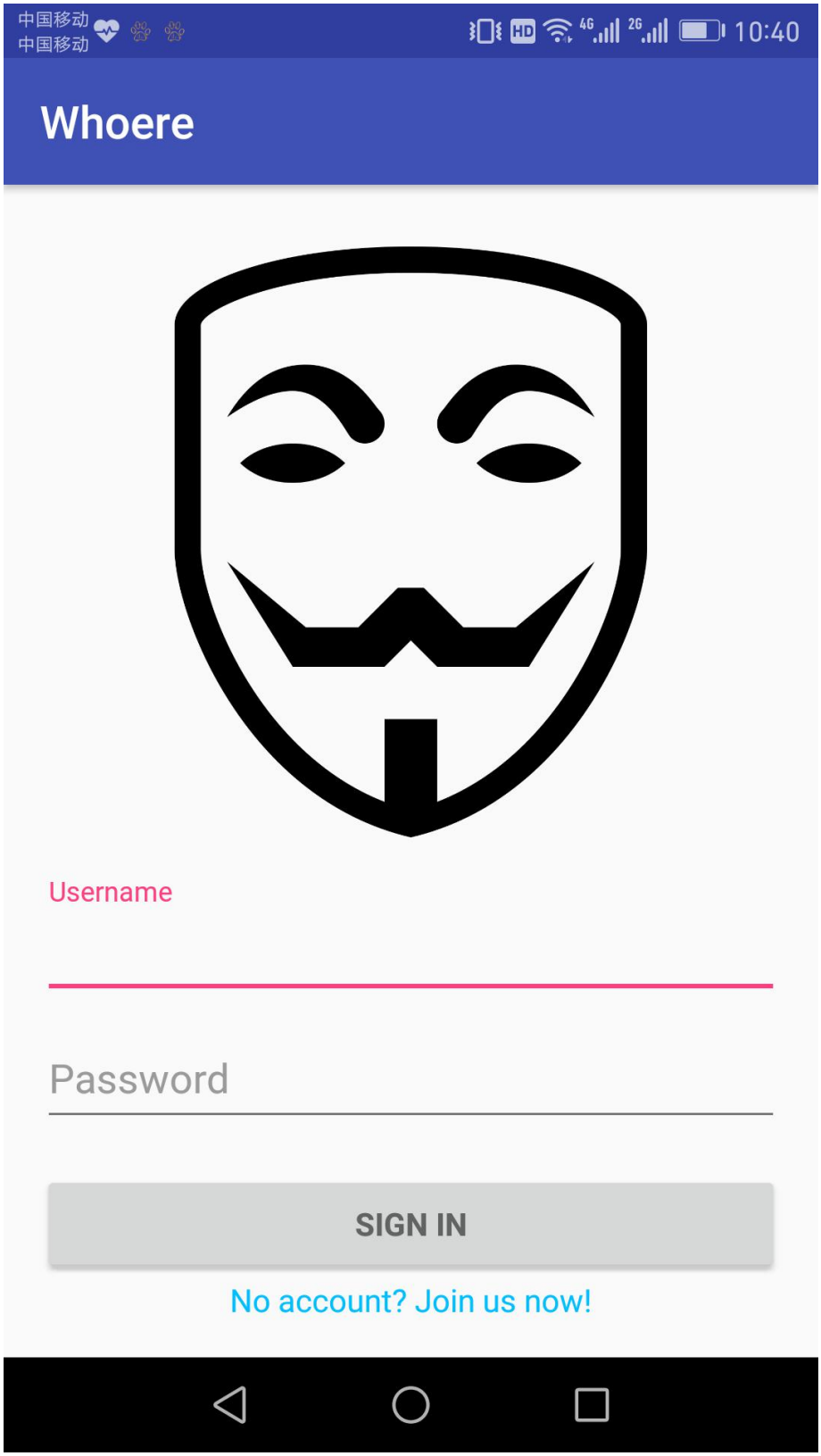


项目类框架结构图

5.2 界面设计与实现

1.登录界面

登录界面为自定义 View，设有两个按钮，可以实现登录登出功能，对界面设置其高度测量模式为 AT_MOST 来解决显示不全的问题。



2. 聊天界面

聊天界面的输入框以后方嵌入型文字来提醒用户每个输入框应该输入什么内容，并设有四个按钮，每次点击 GET 按钮即可获取一条消息，SENDMI 按钮用于私聊，调用 sendmi 方法直接将消息发送至该 IP，而 SEND TO SERVER 则是发送至群聊，所有人可见。

3.地图界面

右下角定位按钮即可获取自己的位置信息，调用高德地图 3D Map 和 Location_Demo 包，并配合 Search 方法，实现地图的展现和缩放、查找、定位等功能。


中国移动
中国移动
19:27
Instance :
SEND LOCATION
GET LOCATION
CLEAN
北县
赤城县
承德市
北京市
唐山市
武清区
保定市
沧州市
饶阳县
50公里

4.列表界面

列表界面通过两个按钮来实现查找“附近的人”和“在线用户”，并且通过地理围栏功能来限定查找范围，并且服务器会定时刷新列表，剔除不符合条件用户名，添加符合条件的新用户。

第6章　项目测试

本章主要介绍和演示项目最终的测试结果以及介绍一下该软件的整体使用流程。

6.1 登录模块测试效果

登录模块是软件正常使用必须的部分，进入登录界面后，输入自己的ID和PsW（PassWord）来进行登录或登出。

6.2 聊天模块测试效果

聊天信息由“信息内容”+“by 发送人”+“发送时间”构成，并且发送时间在内容下一行，靠后放置，条理清楚。

6.3 列表模块测试效果

获取列表的显示为“用户名”+“用户客户端接口”+“距离”组成，信息之间由“*”隔开，以免混淆。

6.4 地图模块测试效果

地图模块使用高德地图地图包和使用方法，放大极限为200米比例尺，拥有全球各地地图，可以获取和发送自己的地理坐标，也可以清除已设置的标志。


中国移动
中国移动
19:56
Instance :
SEND LOCATION
GET LOCATION
CLEAN
G107
惠济区
金水区
G310
郑州市
二七区
G3001
G4
S88
G3001
S85
龙湖镇
白寨镇
5公里
Bai du 地图


中国移动
中国移动
19:28
Instance :
SEND LOCATION
GET LOCATION
CLEAN
S293
南选村
滨海公路
南泊子村
1
鲁信和璧花园
200米
Baidu 地图

第 7 章　总结与展望

7.1 项目总结

通过近一个月的不断努力，本项目最终得以完成，其中主要界面和功能都已实现，包括即时在线聊天、GPS 定位、坐标获取等。作为一名计算机专业的学生，我一直以来都认为实战项目设计是最能锻炼人和提高专业能力的途径。课堂上学习到的始终是理论知识，作为一名合格的程序员，如果不亲自动手将代码实现一遍，只是听一听或看一看的话，效果实在不佳。

这次项目设计给了我一个很好的机会，让我能够把前期需求分析、设计，到软件框架搭建、前端后台功能的实现这么一整套流程都自己走一遍。这其中不免会遇到许许多多的问题，如逻辑设计不合理导致项目进度卡顿、一些新的技术点自己还不能完全掌握等等。但是也正是由于遇到了这些问题，让我能够亲自一个一个去解决它们，通过查阅资料、阅读博客和观看学习视频，在完成项目的过程中，我确实学到了许许多多新的知识，也掌握许多新的技能，如了解了许多当前热门的第三方服务平台，例如阿里云端存储、Bmob 后端云服务、高德地图 LBS 等，也更加熟练地掌握了许多知识点的具体实验原理，如线程池的使用、基于 Socket 的通信过程、GPS 定位与地理闹钟、多 UI 自定义设计等等。

在独立完成整个项目开发的阶段，也让我对软件项目开发的整体流程有了一个全新的认识，提高了我的代码编写能力和对待事情的严谨性、周密性，有时候前期考虑的不周到会导致后续许多隐藏问题的出现。因此，总的来说这次的项目设计对我的帮助和提高还是非常大的，我很感谢老师和学校能过给我这次机会，在以后的工作学习中，我也会再接再厉，不断提高自己。

7.2　项目改进与展望

由于准备时间仓促，加上个人能力有限，通过这段时间的代码编写，该项目的基本功能都已实现，但是还存在一些不足之处，有待我进一步完善和优化。下面简单描述一下对该项目的优化与改进。

1.个人中心改进

目前的个人中心只有兴趣爱好、年龄、性别、地址等寥寥数个选项卡，另一用户从此可以得到的信息量十分有限，后续可以添加更多选项卡，例如个性签名、上次登录地点等。让个人中心更加个人化。

2.聊天界面改进

聊天界面只有类似命令行的简单输出，人机交互不是特别友好，后续可以

改进为类似QQ和微信的对话框和文本条形式；并且增加表情包功能，可以发送网上下载的表情包；其次增加语音通讯功能，可以发送语音消息；聊天背景图片也允许用户自定义修改。使其功能更加完善，人机交互友好，智能化程度更高。

3.GPS定位改进

目前的GPS定位功能只具有高德地图的一部分功能，支持定位、获取坐标、查找地点、地图缩放、地理围栏、计算距离、设置闹钟等功能，但是并不支持导航、语音提醒、路况即时监控、出行方式推荐等高端功能，后续可以继续升级，使其能够拥有更多更好功能。

4.打车功能改进

目前的打车功能是通过和司机进行通信，互相发送地理坐标，然后约定打车地点来完成交易，但是本身并没有网上支付功能，需要另外使用微信或者支付宝来进行费用支付，后需改进可以加入网上支付功能，在应用内部就可以使用微信或支付宝进行支付，节省时间的同时也能提高该应用的实用价值。

5.隐私安全改进

目前登录账号时使用明文传输，安全等级较低，后续改进改为密文传输，提高个人隐私安全性。同时，可以再次改进，让个人隐私跳过服务器，在服务器中不留下记录，保护个人隐私不泄露。（技术较难，留待以后改进）

6.功能拓展

例如增添“随意拍”等各种有趣有用的功能。

更改客户端代码，让其可以同时支持多个服务器和协议。

参考文献

[1]梁桐川.基于强地理位置的社交软件的设计与实现[D].山东大学,2016.
[2]徐威.基于LBS的移动应用服务端的设计与实现[D].华中科技大学,2014.
[3]杨露希.基于Android的移动校友录的设计与实现[D].电子科技大学,2014.
[4]袁伟华.Java线程池的研究与实现[J].电脑编程技巧与维护,2015,01:28-29.
[5]许文勇.基于Socket的网络编程技术及其实现[J].无线互联科技,2014,05:17.
[6]洪世勇.Java中基于Socket的网络编程[J].软件工程师,2013,Z1:61-62.
[7]高翔,张金登.基于线程池的多任务并行处理模型[J].指挥信息系统与技术,2012,04:54-56.

附录

Whoeve 设计与实现部分关键代码

1.实现与服务器端连接

```
    /**
     *
     * @param ip
     *            连接 IP（自己的服务器：182.254.140.27）
     * @param port
     *            连接端口(自己的服务器：12345)
     */
    public Client(String ip, int port) {
        this.ip = ip;
        this.port = port;
    }

    /**
     * 启动客户端，开始连接
     */
    public void start() {
        thread = new Thread(this);
        thread.start();
    }

    private void connect() {
        // System.out.println("0");
        try {
            // 创建一个信道，并设为非阻塞模式
            clntChan = SocketChannel.open();
            clntChan.configureBlocking(false);
            select = Selector.open();
            // 向服务端发起连接

            // System.out.println("1");
            clntChan.register(select,          SelectionKey.OP_CONNECT          |
SelectionKey.OP_READ | SelectionKey.OP_WRITE);
            clntChan.connect(new InetSocketAddress(ip, port));
            // Thread.sleep(500);

        } catch (Exception e) {
            e.printStackTrace();
        }

        // System.out.println("2");
```

```
    }

    /**
     * 绑定数据监听器，每次你向服务器发送一条指令，会在这里得到回馈。
     * 请求码通常是用于区分各种请求。所以大部分发送信息方法中都有 requestCode 参数
     *
     * @param listener
     *            要绑定的监听接口
     */
    public void setOnReceiveDataListener(OnReceiveData listener) {
        this.ord = listener;
    }

    //private StringBuilder temp = new StringBuilder(1024 * 64);

    private void sendData(SelectionKey s, String data) {
        // 接收到的总的字节数
        // 每一次调用 read（）方法接收到的字节数
        SocketChannel sc = (SocketChannel) s.channel();
        // System.out.println("sendData");
        try {
            sc.write(charset.encode(data));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

2.发送 GPS 信息

```
public void sendGpsInfo(double latitude, double longtitude) {
        Command.Builder builder = new Command.Builder();
        builder.addParam("gps");
        builder.addParam("x");
        builder.addParam(latitude);
        builder.addParam(longtitude);
        builder.addEndFlag();
        cs.saveCommand(builder.toString());
        // System.out.println("cmd:" + builder.toString());
    }

    /**
     * 上传 GPS 信息，必须在已经登录的情况下才可以执行此操作。 服务器不会回应上传数据是否成功。
     *
```

```
     * @param pos
     *            点坐标
     */
    public void sendGpsInfo(Point pos) {
        sendGpsInfo(pos.latitude, pos.longtitude);
    }
```

3.内置线程

```
public void run() {
        connect();
        String s;
        System.out.println("connected");
        try {
            while (select.select() > 0) {

                for (SelectionKey k : select.selectedKeys()) {
                    select.selectedKeys().remove(k);
                    // select.selectedKeys().remove(k);
                    if (k.isValid() && k.isAcceptable()) {
                        // System.out.println("Accept");
                        // System.out.println("正在连接...");
                    }
                    if (k.isValid() && k.isConnectable()) {
                        // System.out.println("Connect");
                        select.selectedKeys().remove(k);
                        while (!((SocketChannel) k.channel()).finishConnect())
                            ;
                    }
                    if (k.isValid() && k.isReadable()) {
                        // System.out.println("read");
                        readData(k);
                    }
                    if (k.isValid() && k.isWritable()) {
                        // System.out.println("write");
                        s = cs.getCommand();
                        if (s != null) {
                            // System.out.println("exec server cmd-----\n" + s);
                            sendData(k, s.toString());
                        } else {
                            Thread.sleep(10);
                        }
                    }
                }
                Command cmd;
```

```
                while ((cmd = cp.cmds.poll()) != null) {
                    // System.out.println("exec local cmd " + cmd);
                    exec(cmd);
                }
                // Thread.sleep(200);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

4.服务端处理来自客户端的请求

```
while (selector.sel
ect() > 0) {
            tick++;
            try {
                for (Iterator<SelectionKey> it =
selector.selectedKeys().iterator(); it.hasNext();) {
                    SelectionKey sk = it.next();
                    it.remove();
                    selector.selectedKeys().remove(sk);
                    // 处理来自客户端的连接请求
                    if (sk.isValid() && sk.isAcceptable()) {
                        SocketChannel sc = server.accept();
                        // 非阻塞模式
                        sc.configureBlocking(false);
                        // 注册选择器，并设置为读取模式
                        SelectionKey newsk = sc.register(selector,
SelectionKey.OP_READ);
                        // cmdp.addSelectionKey(newsk);
                        //
System.out.println(newsk+"==========================================================
=======");
                        // 将此对应的 channel 设置为准备接受其他客户端请求
                        sk.interestOps(SelectionKey.OP_ACCEPT);
                        System.out.println("[Accept] IP :" +
sc.getRemoteAddress());
                    }
                    // 处理来自客户端的数据读取请求
                    if (sk.isValid() && sk.isReadable()) {
                        // 返回该 SelectionKey 对应的 Channel,其中有数据需要读取
                        SocketChannel sc = (SocketChannel) sk.channel();
                        ByteBuffer buff = ByteBuffer.allocate(1024*512);
                        // StringBuilder content = new StringBuilder();
```

```
                        int bytes = 0, by = -2;
                        try {
                            String info;
                            Account accc = (Account) sk.attachment();
                            if (accc == null) {
                                info = sc.getRemoteAddress().toString();
                            } else {
                                info = accc.getName();
                            }
                            System.out.println("[" + info + "] start");
                            while ((by = sc.read(buff)) > 0) {
                                buff.flip();
                                // content.append(charset.decode(buff));
                                cmdp.parseString(sk,
charset.decode(buff).toString());
                                buff.clear();
                                //buff.flip();
                                bytes += by;
                            }
                            if (by == -2 && tick % 500 == 0) {
                                ((SocketChannel)
sk.channel()).socket().sendUrgentData(0);
                                System.out.println("[" + sc.getRemoteAddress() +
" ]" + "test Link");
                            }

                            System.out.println("[" + sc.getRemoteAddress() + "]"
+ bytes + " Bytes");
                            // 将此对应的 channel 设置为准备下一次接受数据
                            sk.interestOps(SelectionKey.OP_READ);
                        } catch (IOException io) {
                            try {
                                closeUser(sk, true);
                            } catch (IOException ioo) {
                                ioo.printStackTrace();
                            }
                            continue;
                        }
                        // 开始执行指令
                        for  (Iterator<Command>  itc  =  cmdp.cmds.iterator();
itc.hasNext();) {
                            runCommand(itc.next(), sk);
                            itc.remove();
                        }
```

```
                        /*
                         * // 开始传递数据 if (content.length() > 0) { //
                         * 广播数据到所有的 SocketChannel 中 for (SelectionKey
key :
                         * selector.keys()) { Channel targetchannel =
                         * key.channel(); if (targetchannel instanceof
                         * SocketChannel) { SocketChannel dest = (SocketChannel)
                         * targetchannel;
                         * dest.write(charset.encode(content.toString())); } } }
                         */
                        // -------------
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

5.处理传入的指令和参数

```
public void parseString(SelectionKey sk, String s) {
        // sb.append(s);
        // 处理未完成的指令和他的参数

        //System.out.println(sk                                                    +
"-----------------------------------------");

        int nowPos = 0;// 处理到哪里了?
        int len = s.length();
        ParsingData left = lefts.get(sk);
        //System.out.println(">>>>>>>>>" + left);
        if (left == null) {
            left = new ParsingData();
            lefts.put(sk, left);
            //System.out.println(">>>>>+" + left);
        }
        //System.out.println("[parse] " + s);
        do {
            if (left.isParsing) {
                // 处理未完成的参数
                if (left.isParsingParam) {
                    int pos1 = s.indexOf(Command.PARAM_START, nowPos);
                    if (pos1 == -1) {
                        left.left.append(s);
```

```
                        return;
                    } else {
                        String   param   =   left.left.append(s.subSequence(0,
pos1)).toString();
                        if (param.equals(Command.COMMAND_END)) {
                            left.isParsing = false;
                            add();
                        } else
                            params.add(param);
                        left.isParsingParam = false;
                        nowPos = pos1 + 1;
                    }
                }
                // 到此处，已经处理好 command，准备处理参数
                int pos3;
                while (nowPos < len) {
                    pos3 = s.indexOf(Command.PARAM_START, nowPos);
                    if (pos3 == -1) {
                        left.isParsingParam = true;
                        left.left   =   new   StringBuilder(s.subSequence(nowPos,
len));
                        nowPos = len;
                        break;
                    }
                    String param = s.substring(nowPos, pos3);
                    nowPos = pos3 + 1;
                    if (param.equals(Command.COMMAND_END)) {
                        left.isParsing = false;
                        add();
                        break;
                    } else
                        params.add(param);
                }
            } else {
                // 创建新指令
                int pos4 = s.indexOf(Command.COMMAND_START, nowPos);
                if (pos4 == -1) {
                    // 无效的指令，不处理
                    return;
                }
                nowPos = pos4 + 1;
                left.isParsing = true;
            }
        } while (nowPos < len);
```

```
}

private void add() {
    Command cmd = new Command(params);
    params = new ArrayList<>(10);
    cmds.add(cmd);
}

public void reset() {
    cmds = new LinkedList<>();
    // left=null;
}
```