# 摘要：

近年来，随着手机的日趋流行，手机软件也日益发展壮大。而在全球化的背景下，对于通讯的安全也变得越来越重要。聊天应用软件作为交流和文件传输的工具，凭借其便捷、简单、高效的优势，日益受到年轻人的青睐。它通过使用联网数据库或者本地数据库来实现网上聊天，并且显示在图形用户界面上。本项目拟开发一款聊天应用程序，包括三部分:第一部分是实现第三方登录，第二部分是实现与其他用户一对一聊天（发送文本，图像，标签），第三部分是个人中心（更新个人资料和头像）。第三方登录，程序支持第三方登录，目前采用通过Google账户登录。实现与其他用户的一对一聊天，程序通过实现与其他用户的一对一聊天来发送文本，图像，标签等。个人中心部分，主要实现了更新个人资料和头像。

目录

[摘要： 1](#_Toc27939310)

[1. 引言 5](#_Toc27939311)

[1.1 项目介绍 5](#_Toc27939312)

[1.2 项目的目的和意义 5](#_Toc27939313)

[1.3 前端技术介绍 5](#_Toc27939314)

[1.1. 后端技术介绍 6](#_Toc27939315)

[2. 系统需求分析 7](#_Toc27939316)

[2.1. 软件功能 7](#_Toc27939317)

[3. 项目介绍 7](#_Toc27939318)

[3.1. 开发环境 7](#_Toc27939319)

[3.1.1. 操作系统 7](#_Toc27939320)

[3.1.2. 集成开发环境 8](#_Toc27939321)

[3.1.3. SDK选择 8](#_Toc27939322)

[3.2. 本地数据存储 8](#_Toc27939323)

[3.3. 团队协同开发工具 8](#_Toc27939324)

[4. 系统设计 9](#_Toc27939325)

[4.1. 总体设计 9](#_Toc27939326)

[4.1.1. 处理流程 9](#_Toc27939327)

[4.2. 目录结构 9](#_Toc27939328)

[4.2.1. android目录 10](#_Toc27939329)

[4.2.2. ios目录 11](#_Toc27939330)

[4.2.3. lib目录 11](#_Toc27939331)

[4.2.4. pubspec.yaml文件 11](#_Toc27939332)

[4.2.5. 资源 11](#_Toc27939333)

[4.3. 系统的具体功能 11](#_Toc27939334)

[3.2.1登录注册模块 11](#_Toc27939335)

[3.2.2聊天模块 12](#_Toc27939336)

[4.3.1. 主界面模块 12](#_Toc27939337)

[4.3.2. 个人中心模块 13](#_Toc27939338)

[5. 页面及其功能展示 13](#_Toc27939339)

[5.1. 登录界面及其功能 13](#_Toc27939340)

[5.1.1. 好友列表界面 15](#_Toc27939341)

[5.1.2. 设置页面 18](#_Toc27939342)

[5.1.3. 好友列表页面 20](#_Toc27939343)

[5.1.4. 定位页面 24](#_Toc27939344)

[5.1.5. 消息推送 27](#_Toc27939345)

# 引言

## 项目介绍

本项目拟开发一款[即时通讯](http://www.3yan.cn/topic/jishitongxun" \t "_balnk)应用程序，包括三部分:第一部分是实现第三方登录，第二部分是实现与其他用户一对一聊天（发送文本，图像，标签），第三部分是个人中心（更新个人资料和头像）。第三方登录，程序支持第三方登录，目前采用通过Google账户登录。实现与其他用户的一对一聊天，程序通过实现与其他用户的一对一聊天来发送文本，图像，标签等。个人中心部分，主要实现了更新个人资料和头像。

## 项目的目的和意义

[即时通讯](http://www.3yan.cn/topic/jishitongxun" \t "_balnk)是指能够即时发送和接收互联网消息等的业务。自1998年面世以来，特别是近几年的迅速发展，即时通讯的功能日益丰富，逐渐集成了电子邮件、博客、音乐、电视、游戏和搜索等多种功能。即时通讯不再是一个单纯的聊天工具，它已经发展成集交流、资讯、娱乐、搜索、电子商务、办公协作和企业客户服务等为一体的综合化信息平台。是一种终端连往即时通讯网络的服务。

全球化渗入到我们生活的方方面面，网络通讯在我们学习、生活和工作中扮演着重要的角色。手机通讯软件的出现给人们对于即时通讯带来了极大的便利，通过使用聊天软件，我们可以随时随地与他人进行交流。随着手机的使用日益普遍，支持各种系统系统的手机应用软件也日益增多，聊天通讯软件就是其中很好的交流工具。

## 前端技术介绍

我们团队使用了Google推出并开源的移动应用开发框架Flutter。Flutter框架主打跨平台、高保真、高性能。开发者可以通过 Dart语言开发 App，一套代码同时运行在 iOS 和 Android平台。 Flutter提供了丰富的组件、接口，开发者可以很快地为 Flutter添加 native扩展。同时 Flutter还使用 Native引擎渲染视图，这无疑能为用户提供良好的体验。

Flutter与用于构建移动应用程序的其它大多数框架不同，因为Flutter既不使用WebView，也不使用操作系统的原生控件。 相反，Flutter使用自己的高性能渲染引擎来绘制widget。这样不仅可以保证在Android和iOS上UI的一致性，而且也可以避免对原生控件依赖而带来的限制及高昂的维护成本。

Flutter使用Skia作为其2D渲染引擎，Skia是Google的一个2D图形处理函数库，包含字型、坐标转换，以及点阵图都有高效能且简洁的表现，Skia是跨平台的，并提供了非常友好的API，目前Google Chrome浏览器和Android均采用Skia作为其绘图引擎。

目前Flutter默认支持iOS、Android、Fuchsia（Google新的自研操作系统）三个移动平台。但Flutter亦可支持Web开发（Flutter for web）和PC开发，本书的示例和介绍主要是基于iOS和Android平台的，其它平台读者可以自行了解。

Flutter高性能主要靠两点来保证，首先，Flutter APP采用Dart语言开发。Dart在 JIT（即时编译）模式下，速度与 JavaScript基本持平。但是 Dart支持 AOT，当以 AOT模式运行时，JavaScript便远远追不上了。速度的提升对高帧率下的视图数据计算很有帮助。其次，Flutter使用自己的渲染引擎来绘制UI，布局数据等由Dart语言直接控制，所以在布局过程中不需要像RN那样要在JavaScript和Native之间通信，这在一些滑动和拖动的场景下具有明显优势，因为在滑动和拖动过程往往都会引起布局发生变化，所以JavaScript需要和Native之间不停的同步布局信息，这和在浏览器中要JavaScript频繁操作DOM所带来的问题是相同的，都会带来比较可观的性能开销。

## 后端技术介绍

本门课程中心是移动应用的开发和安全，故后端没有要求。所以我们组采用Google提供的Firebase平台。 Firebase是一个移动平台，可快速构建应用程序，无需管理基础架构。Firebase能提供了分析，数据库，消息传递和崩溃报告等功能。

# 系统需求分析

## 软件功能

随着科学技术的发展，人类社会日新月异，智能手机自出现以来就在人类的生产生活中饰演着重要的角色，越来越成为大家日常生活中必不可少的生活工具。在学校、在家中、在交通工具上甚至在工作中随处可见智能手机的身影。与此同时手机软件也日益发展壮大。而在全球化的背景下，对于信息交流的安全性也变得越来越重要。手机通讯软件作为人们网上交流的工具，凭借其便捷、简单、高效的优势，日益受到年轻人的青睐。

本项目APP主要针对用户对交流通讯和文件传输的安全性注重的需求，采用端到端的加密方式，即使服务器也不知道传输的信息，实现了网上交流与文件传输的安全性。

为满足目标用户的需求，基本可以去确定本聊天应用软件设计所要完成的功能目标。以下是本项目中聊天应用程序具备的功能。

1. 登录界面，用户使用第三方进行登录，保障用户信息安全；
2. 注册界面，供新用户进行注册；
3. 主菜单界面，供用户选择要执行的操作界面；
4. 聊天界面，供用户与其他用户进行聊天交流
5. 个人信息界面，供用户更新个人资料和头像；

# 项目介绍

## 开发环境

### 操作系统

本次软件开发我们使用的操作平台为Windows 10。Windows 10提供优秀的环境变量控制，且与最新的继承开发环境有较好的兼容性。

### 集成开发环境

我们选择 IntelliJ IDEA作为我们的集成开发环境。IDEA完美的兼容Android开发，拥有美观的界面和完善的功能，而且还拥有丰富的插件，可以大大加快我们的开发过程。

### SDK选择

考虑到java 8 以后的版本都不向下兼容，我们选择java 8 版本。Flutter 还在发展期，其最新版本修复了老版本的大量bug和提供更好的新特性，我们选择最新的Flutter 1.9。

## 本地数据存储

我们开发的应用并没有太多的数据需要本地存储，大部分是采用网络存储数据，需要存储本地的数据量少，且这些数据的格式都是简单的字符串型、基本类型的值。所以本软件并没有采用SQLite而是使用Android系统提供的基于持久键-值对存储的SharedPreferences。

## 团队协同开发工具

考虑到Git有一下优势，我们使用Git来团队协同开发：

1. 版本库本地化，支持离线提交，相对独立不影响协同开发。每个开发者都拥有自己的版本控制库，在自己的版本库上可以任意的执行提交代码、创建分支等行为。例如，开发者认为自己提交的代码有问题？没关系，因为版本库是自己的，回滚历史、反复提交、归并分支并不会影响到其他开发者。

2. 更少的“仓库污染”。git对于每个工程只会产生一个.git目录，这个工程所有的版本控制信息都在这个目录中，不会像SVN那样在每个目录下都产生.svn目录。

3. 把内容按元数据方式存储，完整克隆版本库。所有版本信息位于.git目录中，它是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签、分支、版本记录等。

4. 支持快速切换分支方便合并，比较合并性能好。在同一目录下即可切换不同的分支，方便合并，且合并文件速度比SVN快。

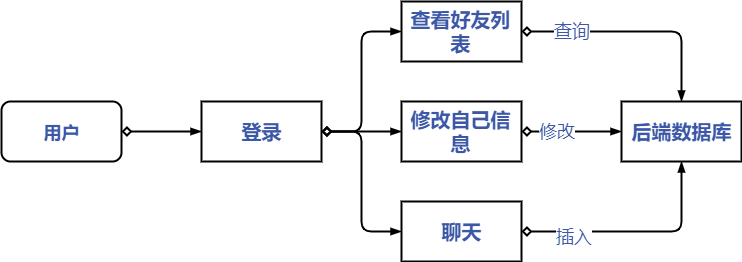
5. 分布式版本库，无单点故障，内容完整性好。内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

# 系统设计

## 总体设计

### 处理流程

根据需求分析设计子系统如下：



图表 1 子系统之间的关系图

## 目录结构

由于我们才有了Flutter框架，目录结构与原生Android开发的结构差异较大。

一下为项目主要的目录结构

Easy\_chat\_app

┬

├ android - 包含 Android 特定的文件。

├ build - 存储 iOS 和 Android 构建文件。

├ ios - 包含 iOS 特定的文件。

├ lib - 应用源文件。

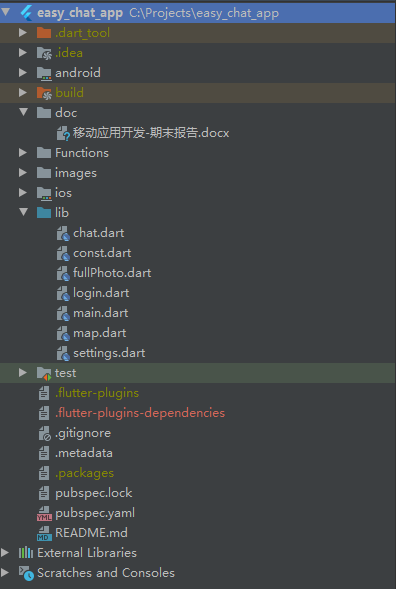
┬

└ src - 包含额外的源文件。

└ main.dart - 程序运行入口文件。

├ test - 测试文件。

└ pubspec.yaml - 包含 Flutter 应用程序的包数据。



图表 2 项目目录结构

### android目录

这里存放的是Flutter与android原生交互的一些代码，这个路径的文件和创建单独的Android项目的基本一样的。不过里面的代码配置跟单独创建Android项目有些不一样。

### ios目录

这里存放的是Flutter与ios原生交互的一些代码。

### lib目录

这里存放的是Dart语言编写的代码，这里是核心代码。不管是Android平台，还是ios平台，安装配置好环境，可以把dart代码运行到对应的设备或模拟器上面。刚才的示例中，就是运行的lib目录下的main.dart这个文件。  
我们可以在这个lib目录下面创建不同的文件夹，里面存放不同的文件，使用Dart语言写我们的自己的代码。

### pubspec.yaml文件

这个是配置依赖项的文件，比如配置远程pub仓库的依赖库，或者指定本地资源（图片、字体、音频、视频等）。  
例如刚才创建的项目的pubspec.yaml里面的：cupertino\_icons: ^0.1.2，表示项目要依赖cupertino\_icons这个库，版本号为0.1.2。

### 资源

像图片、视频、文字等这些资源文件，在 Flutter 里是可以直接引用的，不过需要对资源进行声明式说明。  
在 pubspec.yaml 里进行声明。

## 系统的具体功能

### 3.2.1登录注册模块

该模块主要解决系统中的用户登录及个人信息问题。在本系统中，采用第三方账号登陆。第三方登录之所以会被较为广泛地在产品设计上使用，是因为它有以下几个优点：

（1）对普通用户

相比于本地注册，第三方登录一般来说比较方便、快捷，能够显著降低用户的注册和登录成本，方便用户实现快捷登录或注册。

不用费尽心思地应付本地注册对账户名和密码的各种限制，如果不考虑昵称的重复性要求，几乎可以直接一个账号走遍天下，再也不用在大脑或者什么地方记住N多不同的网站或App的账号和密码，整个世界一下子清静了。

在第一次绑定成功之后，之后用户便可以实现一键登录，使得后续的登录操作比起应用内的登录来容易了很多。

对于某些喜欢社交，并希望将更多自己的生活内容展示给朋友的人来说，第三方登录可以实现把用户在应用内的活动同步到第三方平台上，省去了用户手动发布动态的麻烦。但对于某些比较注重个人隐私的用户来说，则会有一些担忧，所以龙哥所说的这个优点是有前提的。

（2）对应用

因为降低了用户的注册或登录成本，从而减少由于本地注册的繁琐性而带来的隐形用户流失，最终提高注册转化率。

对于某些应用来说，使用第三方登录完全可以满足自己的需要，因此不必要设计和开发一套自己的账户体系。

通过授权，可以通过在第三方平台上分享用户在应用内的活动在第三方平台上宣传自己，从而增加产品知名度。

通过授权，可以获得该用户在第三方平台上的好友或粉丝等社交信息，从而后续可以针对用户的社交关系网进行有目的性的营销宣传，为产品的市场推广提供另一种渠道。

### 3.2.2聊天模块

本模块为系统的主要功能模块，用于解决用户与其他用户交流需求。在该模块中，用户可选择自己需要发送消息的好友，聊天界面有文本输入框用于输入要发送的消息，主界面显示两个用户的聊天记录，同时还提供了发送表情包、图片和用户当前位置。

### 主界面模块

本模块主要显示好友列表和信息。可以点击列表中的好友来与其聊天。

### 个人中心模块

本模块主要是让用户可以对在本系统中存储的的个人信息进行管理。用户可在本模块中进行个人信息修改，修改的内容主要包括用户昵称、头像。如果修改的是用户昵称或头像，保存修改返回个人中心之后，在个人中心显示的昵称或头像已经修改。

# 页面及其功能展示

## 登录界面及其功能

用户第一次登录时，软件会自动跳转得到登录界面。登录界面上部为界面名称，提示用户这个界面的功能。中心为登录按钮，点击按钮后会激活谷歌服务，实现谷歌登录。

以下为登录界面关键代码：

try {

// 新用户，则写入数据库

FirebaseUser firebaseUser = (await firebaseAuth.signInWithCredential(credential)).user;

assert(firebaseUser.email != null);

assert(await firebaseUser.getIdToken(refresh: true) != null);

if (firebaseUser != null) {

// Check is already sign up

final QuerySnapshot result =

await Firestore.instance.collection('users').where('id', isEqualTo: firebaseUser.uid).getDocuments();

final List<DocumentSnapshot> documents = result.documents;

if (documents.length == 0) {

// Update data to server if new user

Firestore.instance.collection('users').document(firebaseUser.uid).setData({

'nickname': firebaseUser.displayName,

'photoUrl': firebaseUser.photoUrl,

'id': firebaseUser.uid,

'createdAt': DateTime.now().millisecondsSinceEpoch.toString(),

'chattingWith': null,

'aboutMe':"还没有个人简介"

});

// Write data to local

currentUser = firebaseUser;

await prefs.setString('id', currentUser.uid);

await prefs.setString('nickname', currentUser.displayName);

await prefs.setString('photoUrl', currentUser.photoUrl);

} else {

// Write data to local

await prefs.setString('id', documents[0]['id']);

await prefs.setString('nickname', documents[0]['nickname']);

await prefs.setString('photoUrl', documents[0]['photoUrl']);

await prefs.setString('aboutMe', documents[0]['aboutMe']);

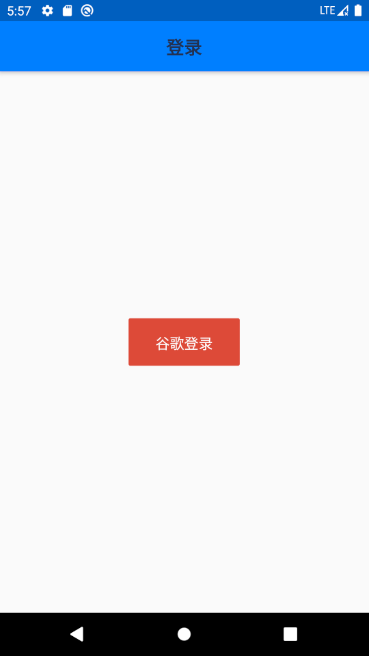
}

Fluttertoast.showToast(msg: "Sign in success");

this.setState(() {

isLoading = false;

});



图表 3 登录界面

### 好友列表界面

当用户登录后，会显示当前好友列表页面。当前页面会显示用户的所有好友。用户左上角的返回按钮实现用户注销，点击后会弹出一个消息框，提示用户是否确定退出。右上角为功能按钮，点击后用户可以选择设置自己的信息和注销。选择设置功能时会跳转到设置页面。此页面主体部分为好友列表。好友列表由一个个好友项组成，每个好友项代表一个好友。每个好友项能展示好友头像、昵称、介绍。当好友数量超过界面显示的最大数量时，用户可以通过滑动来浏览全部好友。点击好友项来与好友聊天。

以下为好友列表界面伪代码：

注销登录状态

Future<Null> handleSignOut() async {

this.setState(() {

isLoading = true;

});

await FirebaseAuth.instance.signOut();

await googleSignIn.disconnect();

await googleSignIn.signOut();

this.setState(() {

isLoading = false;

});

Navigator.of(context).pushAndRemoveUntil(

MaterialPageRoute(builder: (context) => MyApp()),

(Route<dynamic> route) => false);

}

选择功能按钮：

void onItemMenuPress(Choice choice) {

if (choice.title == '注销') {

handleSignOut();

} else if (choice.title == '设置') {

Navigator.push(

context, MaterialPageRoute(builder: (context) => Settings()));

}

}

创建用户列表

Container(

child: StreamBuilder(

stream: Firestore.instance.collection('users').snapshots(),

builder: (context, snapshot) {

if (!snapshot.hasData) {

return Center(

child: CircularProgressIndicator(

valueColor: AlwaysStoppedAnimation<Color>(themeColor),

),

);

} else {

return ListView.builder(

padding: EdgeInsets.all(10.0),

itemBuilder: (context, index) =>

buildItem(context, snapshot.data.documents[index]),

itemCount: snapshot.data.documents.length,

);

}

},

),

),

图表 4 好友界面

图表 5 注销消息框

图表 6 功能选择按钮

### 设置页面

点击左上角返回按钮会返回好友列表页，页面主题包含头像、昵称、简介。有户可以点击头像，从用户相册里选择一张图片作为头像。用户通过昵称和简介的输入框输入数据，更改昵称和简介。跟新后的头像、昵称、简介会保存在后端的数据库中。

关键代码如下：

导入本地缓存

void readLocal() async {

prefs = await SharedPreferences.getInstance();

id = prefs.getString('id') ?? '';

nickname = prefs.getString('nickname') ?? '';

aboutMe = prefs.getString('aboutMe') ?? '';

photoUrl = prefs.getString('photoUrl') ?? '';

controllerNickname = new TextEditingController(text: nickname);

controllerAboutMe = new TextEditingController(text: aboutMe);

// Force refresh input

setState(() {});

}

更新数据

void handleUpdateData() {

focusNodeNickname.unfocus();

focusNodeAboutMe.unfocus();

setState(() {

isLoading = true;

});

Firestore.instance

.collection('users')

.document(id)

.updateData({'nickname': nickname, 'aboutMe': aboutMe, 'photoUrl': photoUrl}).then((data) async {

await prefs.setString('nickname', nickname);

await prefs.setString('aboutMe', aboutMe);

await prefs.setString('photoUrl', photoUrl);

setState(() {

isLoading = false;

});

Fluttertoast.showToast(msg: "更新成功");

}).catchError((err) {

setState(() {

isLoading = false;

});

Fluttertoast.showToast(msg: err.toString());

});

}



图表 7 用户数据设置

### 好友列表页面

界面左上角为返回按钮，点击后返回好友列表页面。头部显示当前好友的用户名。

页面主体部分为消息列表，用户通过滑动可以查看所有消息。消息类型共有四种： 图片、表情包、定位和一般消息。消息列表根据不同的消息类型展现不同的样式。页面尾部有图片按钮，点击后会调用系统相册，选择图片发送；表情包按钮，可以发送软件自带的表情包；定位按钮，点击后会向系统申请定位权限，并发送用户当前位置；消息输入框，用户输入和编辑消息；发送按钮，点击发送输入框中的消息。

上传图片：

Future uploadFile() async {

String fileName = DateTime.now().millisecondsSinceEpoch.toString();

StorageReference reference = FirebaseStorage.instance.ref().child(fileName);

StorageUploadTask uploadTask = reference.putFile(imageFile);

StorageTaskSnapshot storageTaskSnapshot = await uploadTask.onComplete;

storageTaskSnapshot.ref.getDownloadURL().then((downloadUrl) {

imageUrl = downloadUrl;

setState(() {

isLoading = false;

onSendMessage(imageUrl, 1);

});

}, onError: (err) {

setState(() {

isLoading = false;

});

Fluttertoast.showToast(msg: '这不是图片文件');

});

发送消息：

// type: 0 = text, 1 = image, 2 = sticker type=3 position

void onSendMessage(String content, int type) {

if (content.trim() != '') {

textEditingController.clear();

// 写入聊天数据

// massages -> groupId(document) -> groupId(collection) -> massage

var documentReference = Firestore.instance

.collection('messages')

.document(groupChatId)

.collection(groupChatId)

.document(DateTime.now().millisecondsSinceEpoch.toString());

Firestore.instance.runTransaction((transaction) async {

await transaction.set(

documentReference,

{

'idFrom': id,

'idTo': peerId,

'timestamp': DateTime.now().millisecondsSinceEpoch.toString(),

'content': content,

'type': type

},

);

});

listScrollController.animateTo(0.0,

duration: Duration(milliseconds: 300), curve: Curves.easeOut);

} else {

Fluttertoast.showToast(msg: '请输入消息');

}

}

}

图表 8 聊天页面

图表 9 弹出表情包

图表 10 申请定位权限

### 定位页面

点击定位类型的消息后，会跳转到定位页面。通过处理用户的经纬度信息，调用谷歌地图并在地图上标记出用户的位置。

用户位置类：

class UserPosition {

final double latitude;

final double longitude;

UserPosition({@required this.latitude, @required this.longitude});

UserPosition.fromJson(Map<String, dynamic> json)

: latitude = json['latitude'],

longitude = json['longitude'];

Map<String, dynamic> toJson() =>

{

'latitude': latitude,

'longitude': longitude,

};

@override

String toString() {

return "latitude: ${this.latitude}\t longitude: ${this.longitude}";

}

显示地图：

Widget build(BuildContext context) {

final LatLng \_position = LatLng(widget.position.latitude, widget.position.longitude);

print(\_position.toString());

LatLng \_lastMapPosition = \_position;

final Set<Marker> \_markers ={};

Marker \_maker = Marker(

// This marker id can be anything that uniquely identifies each marker.

markerId: MarkerId(\_position.toString()),

position: \_position,

infoWindow: InfoWindow(

title: 'your Position',

snippet: 'your location',

),

icon: BitmapDescriptor.defaultMarker,

);

\_markers.add(\_maker);

void \_onCameraMove(CameraPosition position) {

\_lastMapPosition = position.target;

}

return MaterialApp(

home: Scaffold(

appBar: AppBar(

title: Text('位置信息'),

backgroundColor: Colors.blue[700],

),

body: GoogleMap(

onMapCreated: \_onMapCreated,

initialCameraPosition: CameraPosition(

target: \_position,

zoom: 15.0,

),

mapType: MapType.normal,

markers: \_markers,

onCameraMove: \_onCameraMove,

),

),

);

}

}

图表 11 地图界面

### 消息推送

如果用户没有使用软件，我们通过系统提供的消息推送功能向用户发送通知，提醒用户接收消息。

显示消息内容：

void registerNotification() {

firebaseMessaging.requestNotificationPermissions();

firebaseMessaging.configure(onMessage: (Map<String, dynamic> message) {

print('onMessage: $message');

showNotification(message['notification']);

return;

}, onResume: (Map<String, dynamic> message) {

print('onResume: $message');

return;

}, onLaunch: (Map<String, dynamic> message) {

print('onLaunch: $message');

return;

});

firebaseMessaging.getToken().then((token) {

print('token: $token');

Firestore.instance

.collection('users')

.document(currentUserId)

.updateData({'pushToken': token});

}).catchError((err) {

Fluttertoast.showToast(msg: err.message.toString());

});

}

图表 12 消息推送