题 目：停车场管理手机客户端软件—登陆与车位预约模块设计

**目录**

**摘 要**

**ABSTRACT**

**引言**

**1 概述**

**2 基础技术**

2.1 JAVA介绍

2.1.1 面向对象思想

2.1.2 JAVA的优势

2.1.3 JAVA的特性

2.2 Android介绍

2.2.1 Android简介

2.2.1 四大组件，五大布局以及生命周期

2.2.2 Android基本控件

2.2.3 网络通信

2.2.3.1 线程

2.2.3.2 TCP、HTTP协议

2.2.3.3 JSON

2.3 OSX操作系统下搭建Android开发环境

2.3.1 JDK,ADT,Android Studio的下载及安装

2.3.2 Genymotion的下载及创建模拟器

2.4 Git介绍

2.4.1 什么是版本控制

2.4.2 在Github上创建仓库

2.4.3 使用Android Studio提交代码

**3 软件的设计与结构-------------------------------------------------------------4**

3.1 课题介绍

3.2 优秀开源项目

3.2.1 Fresco

3.2.2 OKHttp

3.2.3 Gson

3.2 MVC

3.2.1 MVC简介

3.2.2 MVC的优点

3.2.3 MVC在本课题中的使用

3.3 功能需求与界面设计

3.3.1 所需功能

3.3.2 界面展示

3.4 项目框架搭建

3.4.1 项目结构图

3.4.2 [封装网络请求](#封装网络请求)

3.4.4 定义常量

**4 程序实现及结果---------------------------------------------------------------5**

4.1 登录与注册

4.2 首页

4.2.2 查询预约页

4.2.3 个人中心页

4.3 个人信息完善

4.4 个人车辆信息的维护

4.4.1 查询

4.4.2 增加

4.4.3 修改

4.4.4 删除

4.5 停车记录

4.6 修改密码

4.7 退出登录

4.8 预约车位

4.8.1 选择停车车位

4.8.2 选择停放车辆

4.8.3 选择停车时间

4.8.4 预约结果

4.9 取消预约

4.10 结束停车

4.11 推送

4.11.1 在推送平台创建应用

4.11.2 集成SDK

4.11.3 处理推送消息

4.11.4 推送结果

4.12 build.gradle

**5 总结与展望-------------------------------------------------------------------6**

**致谢**

**参考文献**

摘要

ABSTRACT

引言

随着移动互联网的发展以及Android设备的普及，开发出一款实用并且用户体验良好的App是当前开发者们所注重的。而Android Studio开发平台更是Google亲自推出的新的Android开发工具，采用了Google用于智能化构建项目的构建集成工具，让开发变得高效，便捷。

本软件从目前繁华地段停车困难的现状入手，分析用户的需求，设计出了帮助用户提前预约车位的系统。目前大部分停车场都只能在车开到停车场时才能知道该停车场能否停车，如果停车场已停满，车主只能继续寻找其他停车位，这不仅耽误了大量车主的时间，也在一定程度上最交通状况造成了压力。本次设计的停车场管理系统就是为了尽可能的解决这个问题。本课题就是完成这个系统的手机客户端的设计与开发。停车场只需要接入这个系统，用户只需要经过简单的注册登录，完善基本信息后，便可以随时随地预约车位，不仅极大地节约了停车场的人力成本，也节约了车主的时间。

本论文全面详细地介绍了实现本课题需要用到的基础技术、软件的设计与功能结构、在开发时的逻辑处理以及详细代码的编写和最终的实现结果。从章节上来看，第一章概括了文章结构，第二章介绍了实现软件需要用到的相关技术，第三章介绍了软件的架构以及相关界面的展示，第四章详细说明了软件的具体实现过程以及实现结果，第五章为课题总结。

# 概述

近些年来，私家车的数量急剧增加，人们对于停车场管理的要求不断增加。随着移动互联网的发展以及智能终端设备的迅速普及，使得开发基于停车场智能管理的移动客户端软件成为可能。在当前快节奏的生活中，时间越来越成为人们所关注的话题，本软件的优势便在于节约车主的时间以及停车场的人力成本。

完成本课题，具体的实现思路如下:

1. 搭建Android开发环境，安装开发工具。
2. 分析需要完成的功能，搜集相关资料，设计相应的界面，搭建项目框架。
3. 整个项目通过Http协议的post方法与服务器通信，客户端解析服务器返回的Json并处理，完成用户注册登录、完善个人信息、查询车辆、预约车位、结束停车的界面，推送等功能。还需要封装Http请求，完成项目中需要的工具类，尤其是有关时间的方法，以确保用户能够正确地预约车位。
4. 性能调优，在多图列表界面，采用压缩图片的方法，使图片内存占用减少，对列表（ListView）的每个Item进行重用，避免OOM。在容易发生崩溃的地方采用try，catch捕获异常，避免应用直接崩溃等等。

**2 技术基础**

## 2.1 JAVA介绍

### 2.1.1 面向对象思想

从世界观的角度可以认为：面向对象的基本哲学是认为世界是由各种各样具有自己的运动规律和内部状态的对象所组成的；不同对象之间的相互作用和通讯构成了完整的现实世界。因此，人们应当按照现实世界这个本来面貌来理解世界，直接通过对象及其相互关系来反映世界。这样建立起来的系统才能符合现实世界的本来面目。

从方法学的角度可以认为：面向对象的方法是面向对象的世界观在开发方法中的直接运用。它强调系统的结构应该直接与现实世界的结构相对应，应该围绕现实世界中的对象来构造系统，而不是围绕功能来构造系统。

### 2.1.2 JAVA的优势

Java 编程语言是个简单、面向对象、分布式、解释性、健壮、安全与系统无关、可移植、高性能、多线程和静态的语言。适合团队开发，软件工程可以相对做到规范。由于Java语言本身的极其严格语法的特点，Java语言无法写出结构混乱的程序。这将强迫程序员的代码软件结构的规范性。这是一个很难比拟的优势。

### 2.1.3 JAVA的特性

抽象类：规定一个或多个抽象方法的类别本身必须定义为abstract，抽象类只是用来派生子类，而不能用它来创建对象。

final类：又称“最终类”，它只能用来创建对象，而不能被继承，与抽象类刚好相反，而且抽象类与最终类不能同时修饰同一个类。

包：Java中的包是相关类和接口的集合，创建包须使用关键字package。

继承：Java作为面向对象编程语言，支持继承这基本概念。但Java只支持单根继承，java.lang.Object是所有其他类的基类。

多态类：在Java中，对象变量是多态的。而Java中不支持多重继承。

接口：Java中的接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方法的特征没有方法的实现，因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为。

通用编程：任何类类型的所有值都可以同Object类型的变量来代替。

封装：把数据和行为结合起在一个包中，并对对象使用者隐藏数据的实现过程，一个对象中的数据叫他的实例字段（instance field）。

重载：当多个方法具有相同的名字而含有不同的参数时，便发生重载。编译器必须挑选出调用哪个方法进行编译。

重写：也可称为方法的“覆盖”。在Java中，子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。值得注意的是，子类在重新定义父类已有的方法时，应保持与父类完全相同的方法头声明。

Class类：Object类中的getClass方法返回Class类型的一个实例，程序启动时包含在main方法的类会被加载，虚拟机要加载他需要的所有类，每一个加载的类都要加载它需要的类。

## 2.2 Android介绍

### 2.2.1 Android简介

Android是一种以Linux为基础的开放源代码操作系统，主要使用于便携设备。Android的系统架构和其它操作系统一样，采用了分层的架构。android分为四个层，从高层到低层分别是应用程序层、应用程序框架层、系统运行库层和linux核心层。

### 2.2.1 四大组件，五大布局以及生命周期

Android四大基本组件分别是

Activity：一个Activity通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应。

Service(服务)：一个Service 是一段长生命周期的，没有用户界面的程序，可以用来开发如监控类程序。

Content Provider(内容提供者)：android平台提供了Content Provider使一个应用程序的指定数据集提供给其他应用程序。这些数据可以存储在文件系统中、在一个SQLite数据库、或以任何其他合理的方式,

其他应用可以通过ContentResolver类(见ContentProviderAccessApp例子)从该内容提供者中获取或存入数据。

BroadcastReceiver(广播接收器)：你的应用可以使用它对外部事件进行过滤只对感兴趣的外部事件(如当电话呼入时，或者数据网络可用时)进行接收并做出响应。广播接收器没有用户界面。

Android的五大布局分别是LinearLayout（线性布局）、FrameLayout（单帧布局）、RelativeLayout（相对布局）、AbsoluteLayout（绝对布局）和TableLayout（表格布局）。

生命周期：

### 2.2.2 Android基本控件

TextView：只读显示控件，可通过getText()获取其android:text属性、setText()设置其android:text属性。

Button：按钮控件，用户通过该控件来提交自己的请求，是用户与应用程序交互的最常用控件之一。

EditText：Android系统提供给用户输入的文本框。

ImageView：显示图片的控件，通过background、src分别设置背景图和前景图。

ProgressBar：进度条控件，可以向用户展示进度。

### 2.2.3 网络通信

#### 2.2.3.1 线程

Android中线程分为2种，主线程(UI线程)和子线程，所有有关UI的操作都必须在主线程中执行，因为若果多个线程都可以直接修改同一个控件的属性，很容易造成混乱，造成读写不一致，所以Android只允许将对控件的修改操作提交给控件所在的线程，由该线程去执行实际的修改动作。而比较耗时的操作如果放在主线程中执行则会造成ANR（Application Not Responding）影响用户体验，所以耗时操作都由子线程执行，最后由Handler机制让主线程更改UI。

thread,handler，looper，messageQueue

简单来讲，主线程创建是会默认创建一个looper，而looper创建是会自动创建一个messageQueue,其他时候可以通过looper.prepare()方法创建。thread通过handler.sendMessage()将message传递给looper，looper再将message放入消息队列中。当Looper对象看到MessageQueue中含有Message，就将其广播出去。对应的handler 对象收到该消息后，调用相应的handler 对象的handleMessage()方法对其进行处理。

#### 2.2.3.2 TCP、HTTP协议

TCP:是一种面向连接的、可靠的、基于字节流的[传输层](http://baike.baidu.com/view/239605.htm)通信协议。

1.TCP的三次握手最主要是防止已过期的连接再次传到被连接的主机。

2.拥塞控制就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。

3.滑动窗口：所谓滑动窗口协议，自己理解有两点：1. “窗口”对应的是一段可以被发送者发送的字节序列，其连续的范围称之为“窗口”；2. “滑动”则是指这段“允许发送的范围”是可以随着发送的过程而变化的，方式就是按顺序“滑动”。

HTTP: 是一个[客户端](http://baike.baidu.com/view/930.htm)和[服务器](http://baike.baidu.com/view/899.htm)端请求和应答的标准（TCP）

Get: 根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的。

Post: 根据HTTP规范，POST表示可能修改变服务器上的资源的请求。

#### 2.2.3.3 JSON

JavaScript Object Notation 是一种轻量级的数据交换格式

JSON 语法是 JavaScript 对象表示语法的子集。数据在键值对中，数据由逗号分隔，花括号保存对象，方括号保存数组。

JSON 值可以是：数字（整数或浮点数），字符串（在双引号中），逻辑值（true 或 false），数组（在方括号中），对象（在花括号中），null。

## 2.3 OSX操作系统下搭建Android开发环境

### 2.3.1 JDK,ADT,Android Studio的下载及安装

JDK:在<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html下载对应自己系统的版本>。

ADT&Android Studio:在<https://developer.android.com/sdk/index.html下载对应自己系统版本的ADT和开发工具>。

安装:将Android Studio.app 拖入Application文件夹，打开Android Studio.app按照默认配置即可安装成功。安装完成后会自动下载ADT。



配置ADB环境变量: 对于android studio而言，默认的adb路径为：

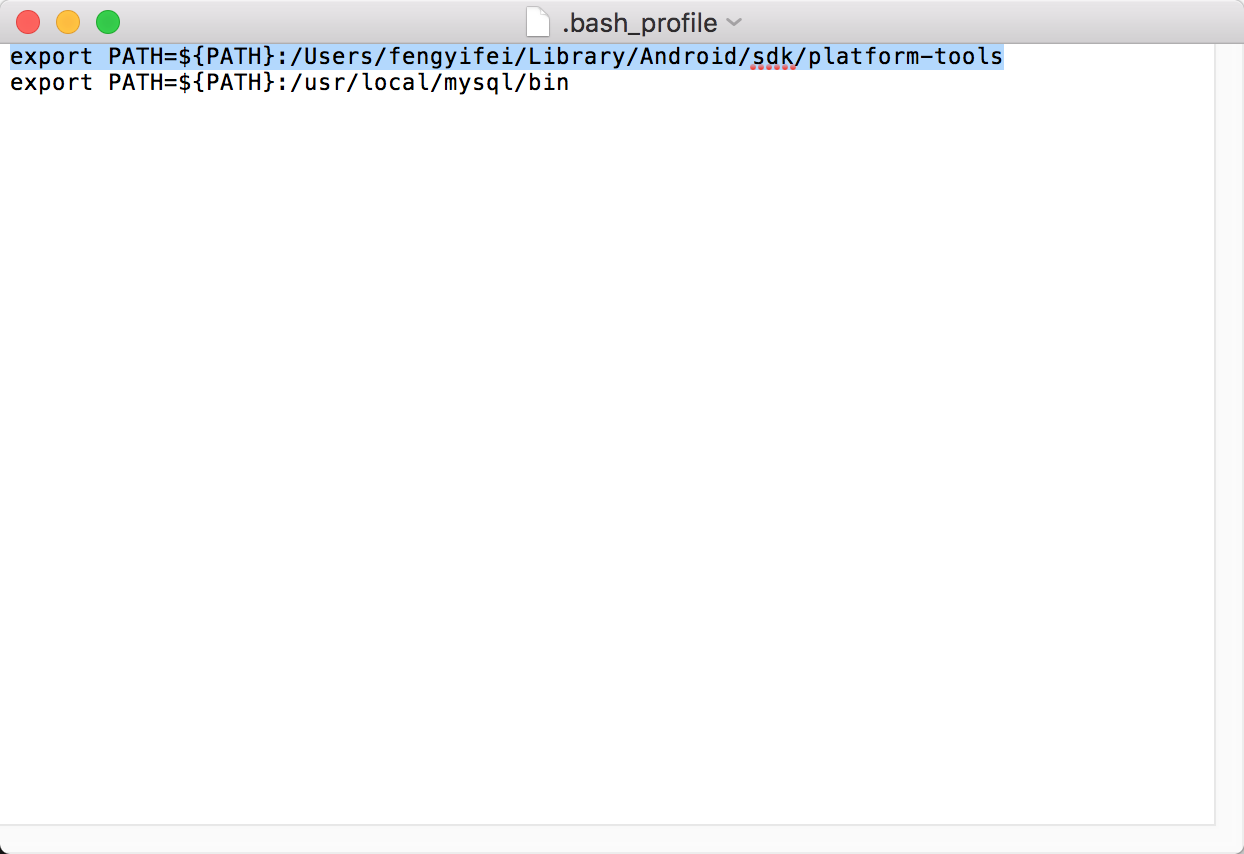
~/Library/Android/sdk/platform-tools

在终端输入以下命令

>touch .bash\_profile

>open -e .bash\_profile

将路径添加到文件



最后更新刚配置的环境变量，命令如下：

source .bash\_profile

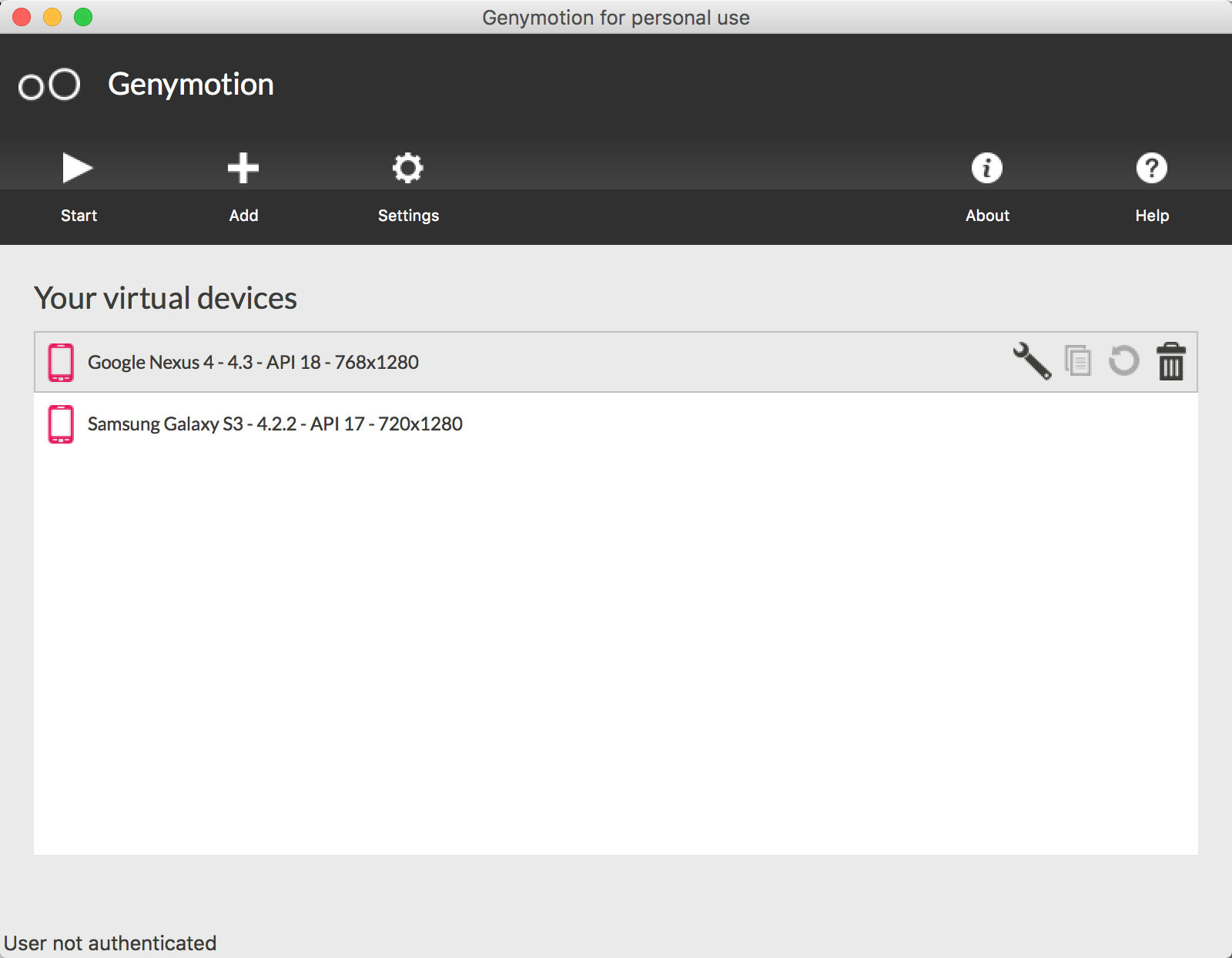
### 2.3.2 Genymotion的下载及创建模拟器

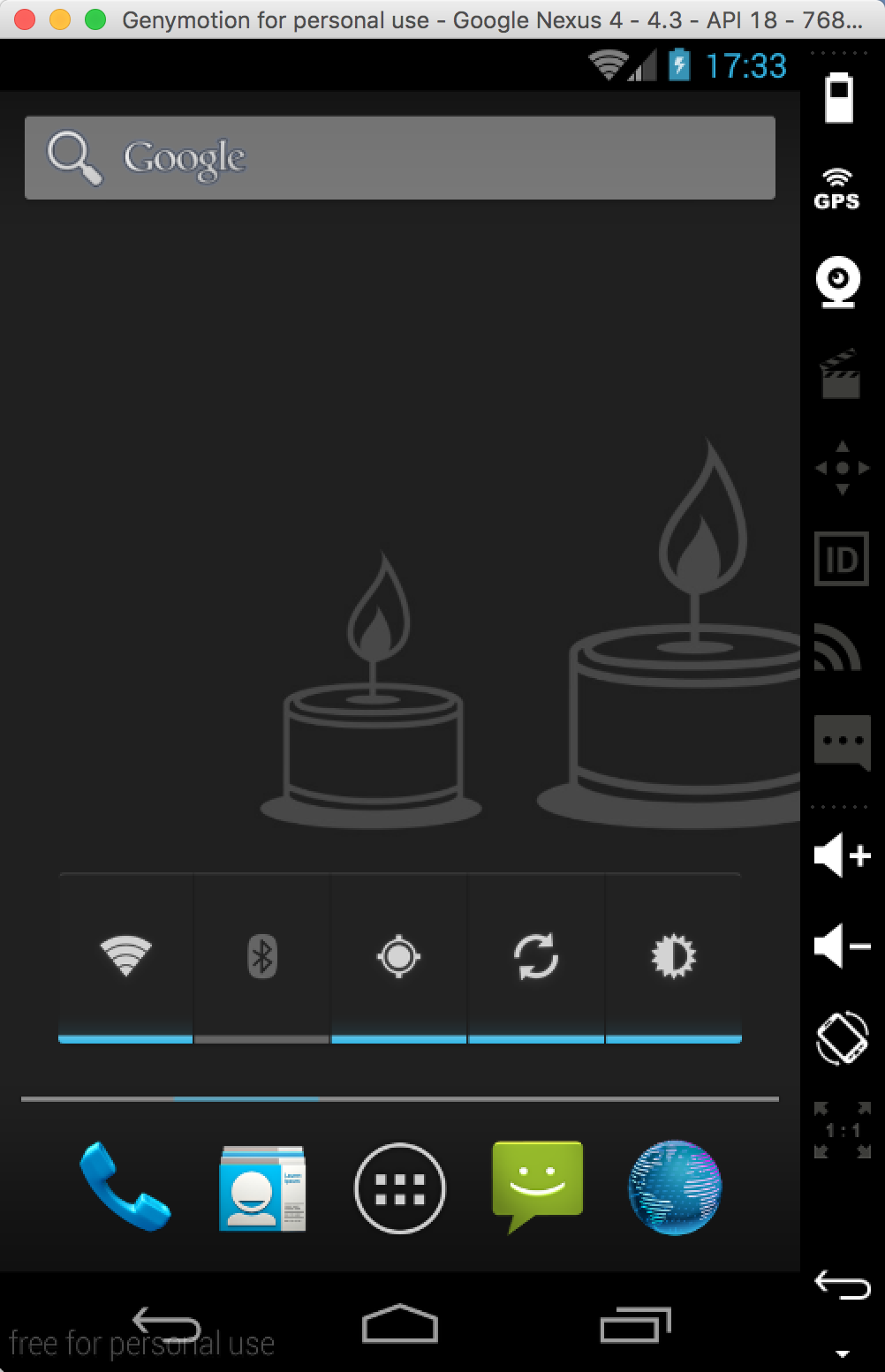
Genymotion是一款Android模拟器，拥有非常流畅的界面和很快的启动速度，用在开发测试比方面原生的Android模拟器好很多。

在<https://www.genymotion.com/>注册一个用户，登陆后即可下载Genymotion，下载时选择个人用户。

安装完成后，在客户端登录刚才所创建的账户，点击Add按钮便可下载各个版本的模拟器。下载完成后选择一模拟器点击Start按钮即可。







## 2.4 Git介绍

Git是一款分布式版本控制系统，可以克隆服务器上完整的Git仓库，在本地根据不同的需要创建分支，commit代码，最后再push到服务器。与svn相比，他的优势在于可以离线工作，即可以在没有网络的情况下commit代码。

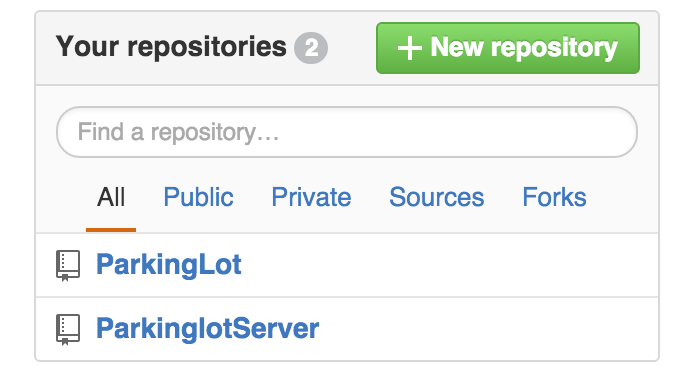
github是一个基于git的项目托管平台，它提供了web界面，你可以在上面创建资源仓库来存放你的项目。

### 2.4.1 什么是版本控制

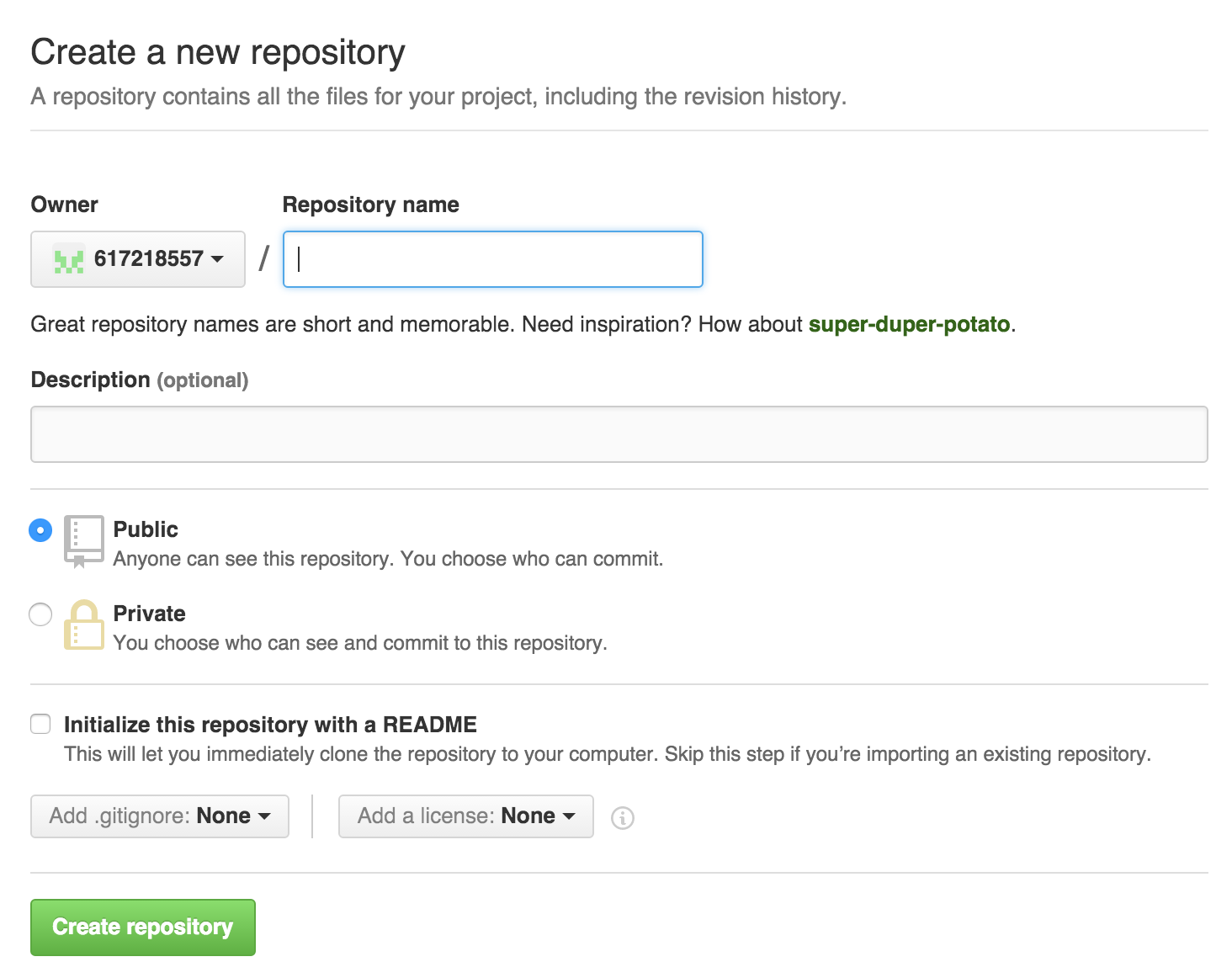
版本控制就是让你得到任何时间修改的版本。版本控制可以管理软件多人协同开发，不会导致软件互相覆盖。每次提交都会有相应的版本号和提交说明，方便回滚或者其他人阅读。

### 2.4.2 在Github上创建仓库

在<https://github.com/>上创建自己的账户，登陆后点击New repository

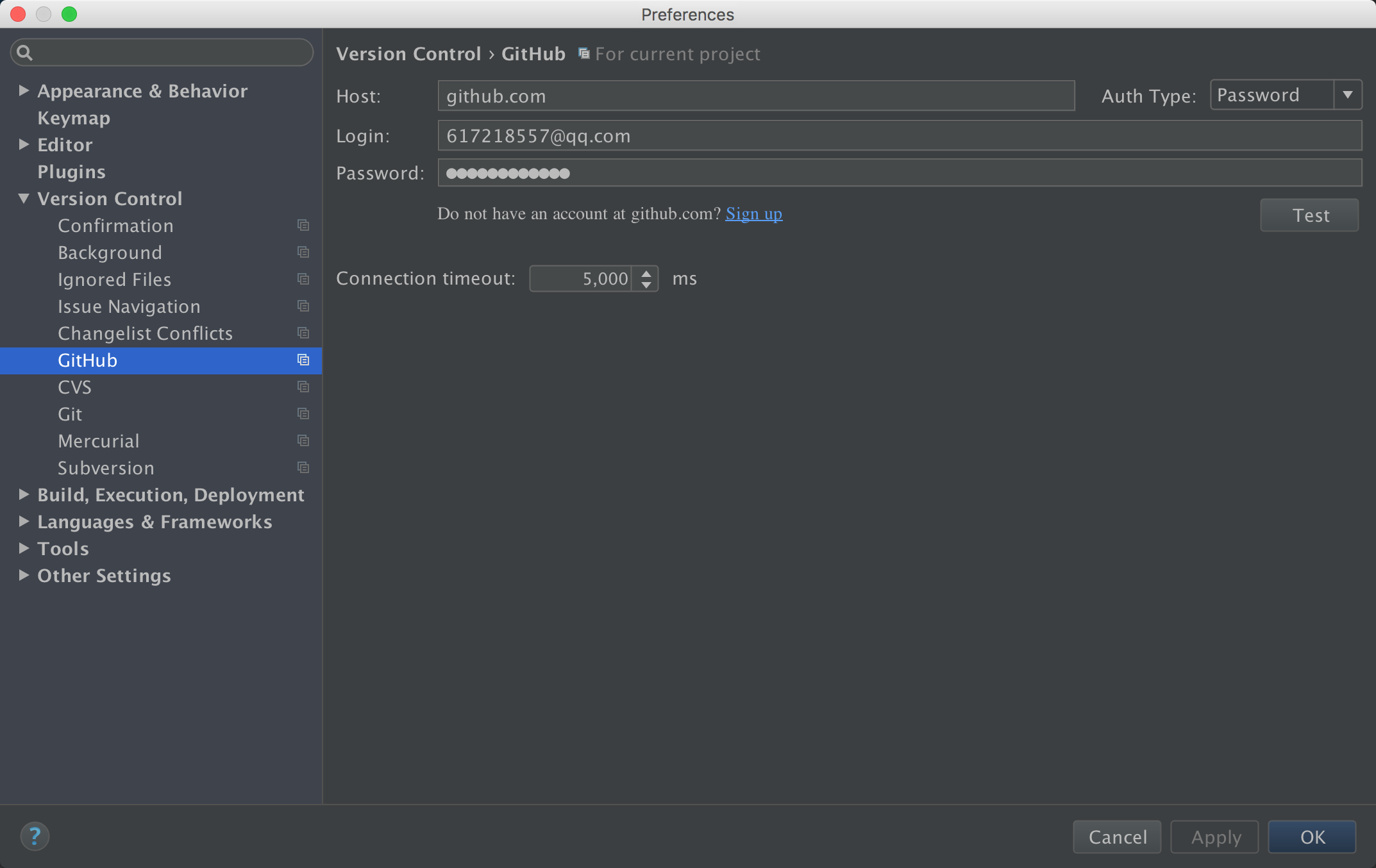


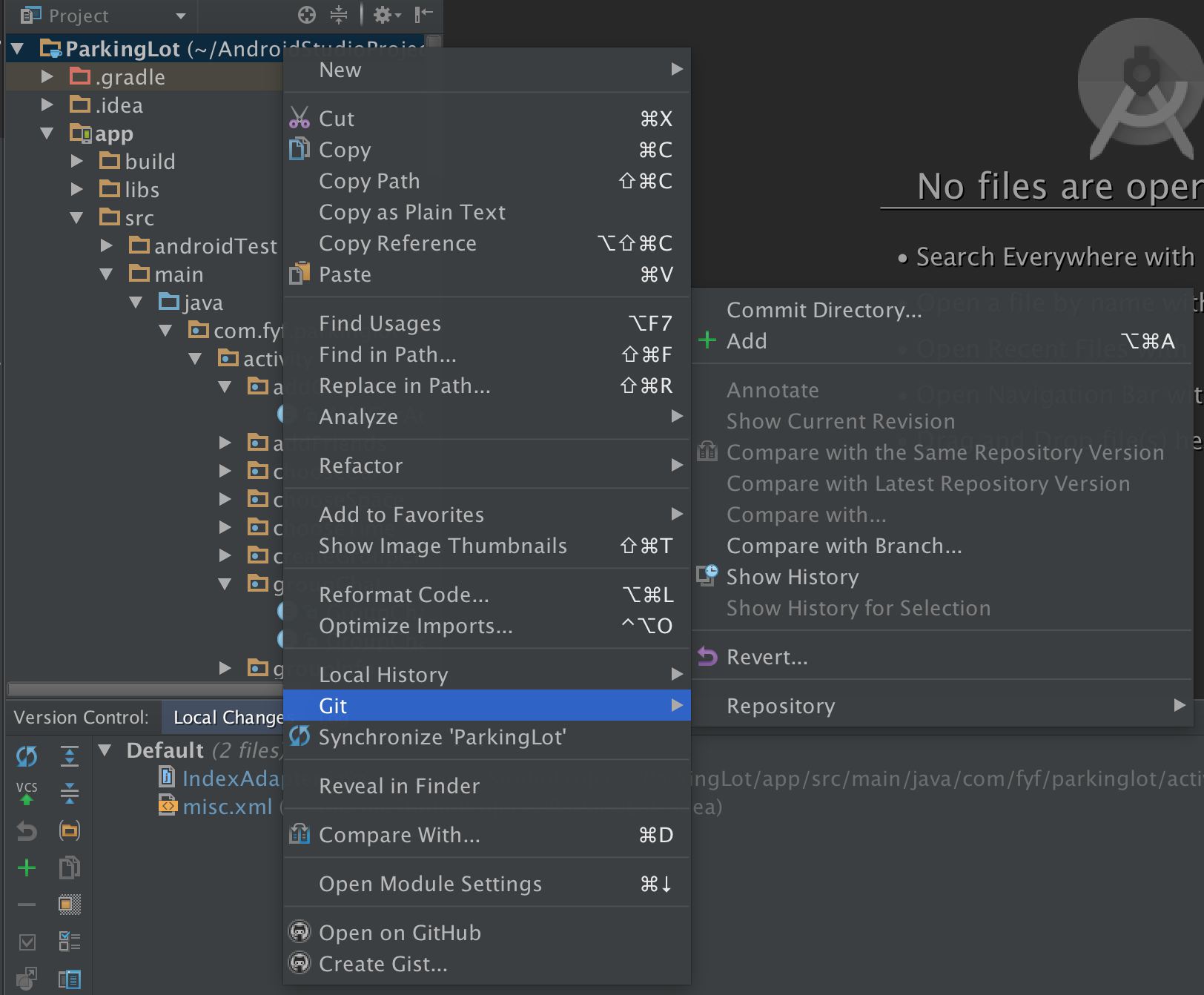
输入项目名称与描述，并选择Public，最后点击Create repository。

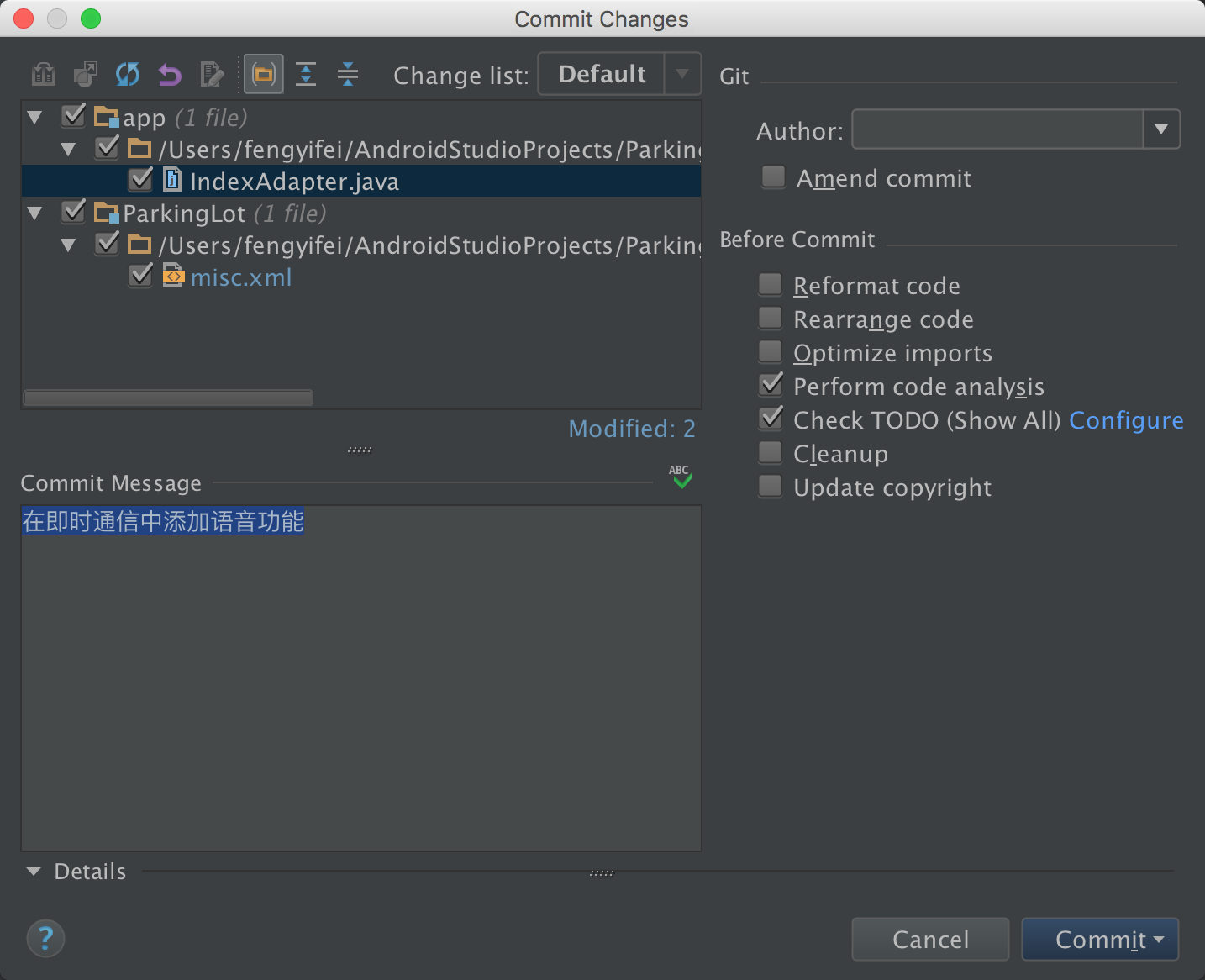


### 2.4.3 使用Android Studio提交代码

打开Android Studio的偏好选项，选择Version Control下的Github,配置自己的Github账号。然后右键项目选择Git下的Commit Directory选项，在弹出的界面中填写Commit Message,点击Commit and Push即可提交代码。







# 3 软件的设计与结构

## 3.1 课题介绍

随着宽带移动通信网络技术的快速发展、智能移动设备的迅速普及及硬件处理能力的不断增强，人们对基于移动终端的信息化管理软件应用需求日益增长。本课题是实心停车场管理的手机端客户软件，要求实现登陆与车位预约功能模块，包括系统架构设计、用户登陆、车位选择、车位预约、相关信息推送、与服务器端的通信等功能。

## 3.2 优秀开源项目

### 3.2.1 Fresco

Fresco是Facebook最新推出的一款用于Android应用中展示图片的强大图片库,可以从网络、本地存储和本地资源中加载图片。其优点有：

1.支持webp格式的图

2.Android5.0以下系统：使用”ashmem”（匿名共享内存）区域存储Bitmap缓存，这样Bitmap对象的创建、释放将永远不会触发GC，关于”ashmem”存储区域，它是一个不在Java堆区的一片存储内存空间，它的管理由Linux内核驱动管理，不必深究，只要知道这块存储区域是别于堆内存之外的一块空间就行了，且这块空间是可以多进程共享的，GC的活动不会影响到它。5.0以上系统，由于内存管理的优化，所以对于5.0以上的系统Fresco将Bitmap缓存直接放到了堆内存中。

3.使用了三级缓存：Bitmap缓存+未解码图片缓存+硬盘缓存。

4.Fresco中的MVC模式

DraweeView——View，DraweeController——Control，DraweeHierarchy——Model，它们之间的关系大致如下：

DraweeHierarchy意为视图的层次结构，用来存储和描述图片的信息，同时也封装了一些图片的显示和视图层级的方法。

DraweeView用来显示顶层视图（getTopLevelDrawable()）。DraweeController控制加载图片的配置、顶层显示哪个视图以及控制事件的分发。

### 3.2.2 OKHttp

OkHttp是一个高效的HTTP库，实现了几乎和java.net.HttpURLConnection一样的API。OKHttp总体设计主要是通过Diapatcher不断从RequestQueue中取出请求（Call），根据是否已缓存调用Cache或 Network这两类数据获取接口之一，从内存缓存或是服务器取得请求的数据。该引擎有同步和异步请求，同步请求通过Call.execute()直接返 回当前的Response，而异步请求会把当前的请求Call.enqueue添加（AsyncCall）到请求队列中，并通过回调（Callback） 的方式来获取最后结果。

### 3.2.3 Gson

GSON是Google开发的Java API，用于转换Java对象和Json对象。在JAVA对象中设置和JSON对象中相同的属性名称（大小写敏感），并且添加get、set方法，最后调用Gson.fromJson(json,Class)便可将json对象转换为java对象;

## 3.2 MVC

### 3.2.1 MVC简介

MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

### 3.2.2 MVC的优点

MVC 分层有助于管理复杂的应用程序，可以在一个时间内专门关注一个方面。例如，可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。

MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

它的优点有：耦合性低、重用性高、生命周期成本低、MVC使开发和维护用户接口的技术含量降低、部署快、可维护性高、分离视图层和业务逻辑层也使得WEB应用更易于维护和修改、有利软件工程化管理

### 3.2.3 MVC在本课题中的使用

本课题的项目架构中应用了类似于MVC的框架模式。M层：做一些业务逻辑处理，比如数据库存取操作，网络操作(HttpUtils)，复杂的算法(有关于时间的Utils)，耗时的任务等都在model层处理。 V层：应用层中处理数据显示的部分，XML布局可以视为V层，显示Model层的数据结果。 C层：在Android中，Activity处理用户交互问题，因此可以认为Activity是控制器，Activity读取V视图层的数据（eg.读取当前EditText控件的数据），控制用户输入（eg.EditText控件数据的输入），并向Model发送数据请求（eg.发起网络请求等）。

## 3.3 功能需求与界面设计

### 3.3.1 所需功能

本课题需要实现用户的登录，注册，预约车位(选择车位，选择车辆，选择时间)，取消预约，结束停车，查询停车记录，用户个人信息的显示与维护（头像，姓名，年龄，性别），修改密码，个人车辆信息的维护（添加，修改（图片，车型，车牌号），删除），退出登录，以及一些widgit和自定义布局。

### 3.3.2 界面展示

**尺寸资源 dimens.xml：**

界面的设计需要统一，所以一些常用的尺寸资源写在这里，使软件的布局统一，如左边距，右边距，文字大小等。

<resources>

<dimen name="activity\_horizontal\_margin">16dp</dimen>

<dimen name="activity\_vertical\_margin">16dp</dimen>

<dimen name="fab\_margin">16dp</dimen>

<dimen name="appbar\_padding\_top">8dp</dimen>

<dimen name="app\_bar\_height">180dp</dimen>

<dimen name="text\_margin">16dp</dimen>

<dimen name="text\_size\_default">16sp</dimen>

<dimen name="text\_size\_big">18sp</dimen>

<dimen name="text\_size\_small">14sp</dimen>

</resources>

**标题栏 layout\_actionbar.xml：**

ActionbBar是Android3.0(API level 11)中引入的新控件，但是由于开发中对于ActionBar的灵活性要求较高，原生的Actionbar无法满足要求，所以采用了自己实现一个ActionBar，同时对Application设置Theme.AppCompat.Light.NoActionBar属性，即去掉系统原生的ActionBar。

自定义ActionBar的背景为主题色黑色，高度为48dp，宽度占满全屏，其中有3个控件，分别是左侧的Button、中间的TextView和右侧的Button。左侧的Button主要用作于返回按钮，中间的title主要用于显示当前界面的标题，右侧的Button主要用于一些自定义事件，每个界面都有其自己的事件，比如添加车辆，删除车辆，预约车位界面中的下一步等等，有些界面没有此类事件，则将Button的visiable属性设置为不可见。

**登录及注册界面 activity\_login.xml：**

本软件将注册于登录放于同一个界面中，采用动画淡入淡出的效果切换。为了防止某些机型的屏幕过小，无法显示全部界面的情况，用ScrollView将整个界面包裹起来，让用户可以上下滑动。界面的EditText用Google在2015年发布的Design Support Library中的 TextInputLayout包裹，此控件的优点是可以动态的提示用户在EditText中输入文本的错误，而不用传统的Toast提示，达到更好的交互效果。软件中所有的确定、提交按钮采用CircularProgressButton，一个可以动态显示当前状态与执行结果的按钮。

**主activity activity\_main.xml：**

主界面采用滑动导航方式，界面中的导航栏采用Design Support Library中的 AppBarLayout包裹TabLayout，展示的页面采用viewpager。

**选择车辆以及列表布局 activity\_choose\_car.xml activity\_choose\_car\_list\_item.xml：**

所有的listView都采用了JazzyListView，一个可以在滑动时以特效的方式显示列表中的每个item。每个item的图片加载则采用facebook发布的图片加载框架SimpleDraweeView。Item中左侧垂直居中显示宽150dp高100dp的图片，右侧以纵向居中显示每个车辆的车辆类型和车牌号。

**选择车位 activity\_choose\_space.xml：**

界面采用ScrollView包裹以适配更多的机型。每个车位都由一个圆角正方形表示，通过自定义的layout根据屏幕尺寸来布局停车场。每个车位都可能有3种情况，分别为已停车、未停车和我的车位，每种情况显示不同的颜色。在下方，纵向显示了3个情况的颜色标注，方便用户识别区分。

**选择时间 activity\_choose\_time.xml：**

这个界面主要由2个部分构成，分别是安卓原生的DatePicker（日期选择器）和TimePicker（时间选择器）。用户可以滑动屏幕改变预约车位的时间。

**个人中心界面 fragment\_mine.xml：**

界面分为2个纵向展示的部分，第一部分的背景使用了自定义的view，动态的显示多个大小、透明度不同的光晕在做直线运动，用户的圆形头像居中展示，用户的个人信息（手机号，年龄，性别）在头像下方水平居中展示。第二部分纵向的显示了我的车辆、停车记录、修改密码、天气预报、退出登录等功能，点击进入相应的界面。

**个人信息的维护 activity\_update\_user\_info.xml：**

界面纵向展示了2个EditText，一个RadioGroup和一个CircularProgressButton，分别是编辑姓名、性别（男女），和提交修改按钮。

**修改密码 activity\_update\_user\_password.xml：**

界面纵向展示了2个EditText和一个CircularProgressButton，分别是输入原密码、新密码，和提交修改按钮。

**个人车辆信息以及列表 activity\_my\_car.xml activity\_my\_car\_list\_item.xml：**

此界面与选择车辆的界面展示基本相同，右上角多了个添加车辆的按钮。

**个人车辆信息的维护 activity\_update\_car.xml：**

界面顶部居中显示了一个SimpleDraweeView用来展示车辆图片。下面用2个水平居中的TextInputLayout以及EditText分别提示用户输入车辆类型和车牌号。最后用一个水平居中的CircularProgressButton来显示提交按钮。

**查询我的预约以及列表 fragment\_order.xml：**

**fragment\_order\_ist\_item.xml:**

fragment以纵向显示，顶部用了一个TextView靠右侧显示“我的预约”标题，紧接着是一条颜色为#979797的分割线，之后只一个占满父控件宽于高的JazzyListView，最后在右下角有一个Design Support Librar中的FloatingActionButton作为预约按钮。

每个item中垂直居中展示了3个TextView，每两个TextView中以一条颜色为#979797的竖线分割，这三个TextViewde 功能为别是展示预约时间，预约车位以及计费速率。

**停车记录以及列表 activity\_record.xml：**

**activity\_record\_list\_item.xml:**

activity中只有一个actionBar和JazzyListView。

每个item左侧居中展示了一个宽150dp高100dp的SimpleDraweeView用来展示车辆图片，右侧垂直居中纵向展示了4个LinearLayout，每个layout中都包含了2个TextView。4个LinearLayout分别表示停车时间，停车费用，车辆类型和车牌号。

## 3.4 项目框架搭建

### 3.4.1 项目结构图

com.fyf.parkinglot(包名)

activity(activity在这个路径下)

addCar

AddCarActivity.java(添加车辆的activity)

chooseCar

ChooseCarActivity.java(选择车辆的Activity)

ChooseCarListViewAdapter.java(选择车辆列表的适配器)

chooseSpace

ChooseSpaceActivity.java(选择车位的activity)

DrawParkinglot.java(布局停车场的layout)

GridViewAdapter,java(每个车位的适配器)

chooseTime

ChooseTimeActivity,java(选择预约时间的Activity)

login

LoginActivity,java(登录注册的Activity)

main

MainActivity.java(首页Activity)

MainPagerAdapter.java(首页Activity的ViewPager适配器)

myCar

MyCarActivity.java(我的车辆的Activity)

CarListViewAdapter.java(我的车辆Activity列表的适配器)

record

RecordActivity.java(停车记录的Activity)

RecordListViewAdapter.java(停车记录Activity列表的适配器)

updateCar

UpdateCarActivity.java(更新用户车辆信息的Activity)

updateUserInfo

UpdateUserInfoActivity.java(更新用户信息的Activity)

updateUserPassword

UpdateUserPassword.java(更新用户密码的Activity)

application(application在这个路径下)

ParkinglotApplication.java

common(一下常量定义在这个路径下)

ContextManager.java(存放Context)

GlobalDefine.java(定义常量)

ScreenInfo.java(手机屏幕信息)

SQLWord.java(服务器数据库的字段信息)

URLAddress.java(网络请求地址)

fragment(fragment在这个路径下)

mine

MineFragment.java(我的fragment)

CropperUserActivity.java(裁剪用户头像的Activity)

order

OrderFragment.java(预约车位的fragment)

OrderAdapter.java(已预约车位列表的适配器)

model(数据模型在这个路径下)

CarInfoBean.java(车辆信息模型)

ParkinglotInfoBean.java(车位信息模型)

ParkingRecordInfoBean.java(停车记录信息模型)

ResultInfoBean.java(返回的json固定格式模型)

UserInfoBean.java(用户信息模型)

UserInfoInCache.java(内存中缓存的用户信息)

push(推送相关功能在这个路径下)

MyPushMessageReceiver.java(处理推送)

Utils.java(推送的工具类)

utils(工具类在这个路径下)

DataOperateUtils.java(操作本地缓存的文件)

HttpUtils.java(封装网络请求)

JsonUtils.java(解析Json的工具类)

ListViewUtils.java(ListView的工具类)

NotifyDataChangeUtils.java(通知状态改变的工具类)

Utils.java(计算日期,解码,正则表达式验证等工具类)

view(自定义view在这个路径下)

CustomPrgressDailog.java(自定义加载动画)

CustomToast.java(自定义Toast)

HaloView.java(自定义光晕背景)

MyGridView.java(自定义GridView)

MyListView(完全展开的listView)

### 3.4.2 封装网络请求

网络请求采用OKHttp框架，在本地再次封装后使用。

package com.fyf.parkinglot.utils;

import android.util.Log;

import com.squareup.okhttp.OkHttpClient;

import com.squareup.okhttp.Request;

import com.squareup.okhttp.RequestBody;

import com.squareup.okhttp.Response;

/\*\*

\* Created by fengyifei on 15/11/23.

\*/

public class HttpUtils {

private static OkHttpClient client = new OkHttpClient();

// post请求

public static String httpPost(String url, RequestBody body) {

try {

Request request = new Request.Builder().url(url).post(body).build();

Response response = client.newCall(request).execute();

if (response.isSuccessful()) {

String result = Utils.base64Decode(response.body().string());

Log.e("HttpResult",result);

return result;

} else {

return response.code() + "";

}

} catch (Exception e) {

return "";

}

}

// get请求

public static String httpGet(String url) {

try {

Request request = new Request.Builder().url(url).build();

Response response = client.newCall(request).execute();

if (response.isSuccessful()) {

String result = Utils.base64Decode(response.body().string());

Log.e("HttpResult",result);

return result;

} else {

return response.code() + "";

}

} catch (Exception e) {

return "";

}

}

// 返回的数据不用进行base64解码的get请求

public static String httpGetWithOutBase64(String url) {

try {

Request request = new Request.Builder().url(url).build();

Response response = client.newCall(request).execute();

if (response.isSuccessful()) {

String result = response.body().string();

Log.e("HttpResult",result);

return result;

} else {

return response.code() + "";

}

} catch (Exception e) {

return "";

}

}

}

### 3.4.3 定义常量

public class GlobalDefine {

public static String APP\_NAME = "parkinglot";

// 七牛

public static String QINIU\_AK = "XPHnXDfTqIHcGg5M1nqnaLXtmupPqSEJfKkB3Czz";

public static String QINIU\_SK = "zFC8HWzqUXVxJIT5kQ5aKO\_aZVipCJa1U3LSUGUU";

// 七牛中默认车辆图片名称

public static String QINIU\_CAR\_DEFAULT = "car\_default.jpg";

// IM即时通信

public static String IM\_APPKEY= "d0044ef8d4baa85de9e393dd";

public static String IM\_USER\_PASSWORD = "parkinglot";

//百度车联网API

public static String CAR\_API\_AK = "fTvBOmDjpfu5bulB06ByOToR";

// 用户表

public static int USER\_GENDER\_FEMALE = 0; // 女性

public static int USER\_GENDER\_MALE = 1; // 男性

// 停车场表

public static int PARK\_NOT\_USED = 0; // 车位未被占用

public static int PARK\_USED = 1; // 车位已被占用

public static double PARK\_FEE = 5.0;// 每小时停车费用

// 用户注册

public static int REGIST\_FAIL = -1; // 注册失败

public static int REGIST\_EXIST = -2;// 用户已存在

// 用户登录

public static int LOGIN\_FAIL = -1;// 登录失败

public static int LOGIN\_NOT\_EXIST = -2;// 用户不存在

public static int LOGIN\_PASSWORD\_ERROR = -3;// 密码错误

// 查找可用车位

public static int PARK\_FIND\_FAIL = -2;// 查找失败

public static int PARK\_FIND\_NO\_NOT\_USED = -2;// 暂无可用车位

// 预约车位

public static int PARK\_ORDER\_FAIL = -1;// 预约失败

public static int PARK\_ORDER\_USED = -2;// 车位已被占用

public static int PARK\_ORDER\_TIME\_NOT\_LEGEL = -3;// 预约时间不合法

// 取消预约车位

public static int PARK\_CANCLE\_FAIL = -1; // 失败

// 停止停靠

public static int PARK\_END\_FAIL = -1;// 失败

// 查询车辆信息

public static int CAR\_FIND\_FAIL = -1;// 失败

// 添加车辆信息

public static int CAR\_ADD\_FAIl = -1;// 失败

// 更改车辆信息

public static int CAR\_UPDATE\_FAIL = -1;// 失败

// 删除车辆信息

public static int CAR\_DELETE\_FAIL = -1;// 失败

// 根据用户id查询车辆停靠记录信息

public static int RECORD\_FIND\_USER\_FAIL = -1;// 失败

// 根据车辆id查询车辆停靠记录信息

public static int RECORD\_FIND\_CAR\_FAIL = -1;// 失败

// SharedPreference 相关信息

public static String SHAREDPRE\_USER\_LOGIN\_INFO = "USER\_LOGIN\_INFO";

// 预约提前时间

public static final int orderForwardHour = 2;

}

public class URLAddress {

// 服务器地址

public static final String publicURL = "";

public static final String localURL = "http://115.28.111.133:8080/Parkinglot";

public static final String URL = localURL;

public static final String QINIU\_URL = "http://www.7xoqdw.com1.z0.glb.clouddn.com";

public static final String loginURL = URL + "/userlogincl";// 登录

public static final String registURL = URL + "/userregistcl"; // 注册

public static final String allSpaceInfoURl = URL + "/findallparkingspacecl"; // 查询所有车位信息

public static final String findUserCarURL = URL + "/findcarinfocl"; // 查询用户车辆

public static final String orderSPaceURl = URL + "/orderparkingspacecl"; // 预约车位

public static final String findTodayOrderURl = URL + "/findtodayorderbyuseridcl";// 查询用户今日预约

public static final String updateUserInfoURL = URL + "/updateuserinfocl";// 更新用户信息

public static final String updateUserPasswordURL = URL + "/updateuserpasswordcl";// 查询用户密码

public static final String findRecordByUserIdURL = URL + "/findparkingrecordbyuseridcl";//根据用户id查询停车记录

public static final String cancleOrderURL = URL + "/cancleorderparkingspacecl"; // 取消预约

public static final String endParkingURL = URL + "/endorderparkingspacecl";// 结束停车

public static final String addCarURL = URL + "/addcarinfocl";// 添加车辆

public static final String updateCarURL = URL + "updatecarinfocl";// 更新车辆

public static final String deleteCarURL = URL + "/deletecarinfocl";// 删除车辆

public static final String findImaccountURl = URL + "/findimaccountcl";// 查找即时通信账户

public static final String weatherForcastURL = "http://api.map.baidu.com/telematics/v3/weather";// 天气预报

}

string.xml

<resources>

<string name="app\_name">ParkingLot</string>

<!— 登陆界面 —>

<string name="activity\_login\_phoneNum">手机号</string>

<string name="activity\_login\_password">密码</string>

<string name="activity\_login\_orRegist">或者,创建账户</string>

<string name="activity\_login\_orLogin">或者,登录</string>

<string name="activity\_login\_login">登录</string>

<string name="activity\_login\_regist">创建账户</string>

<string name="activity\_login\_fail">x</string>

<string name="activity\_login\_success">√</string>

<!— actionBar —>

<string name="actionBar\_order">预约停车</string>

<string name="actionBar\_next">下一步</string>

<string name="actionBar\_finish">完成</string>

<string name="actionBar\_updatePassword">修改密码</string>

<string name="actionBar\_updateUserInfo">修改个人信息</string>

<string name="actionBar\_record">停车记录</string>

<string name="actionBar\_myCar">我的车辆</string>

<string name="actionBar\_add">添加</string>

<string name="actionBar\_del">删除</string>

<string name="actionBar\_update\_car">修改车辆信息</string>

<string name="actionBar\_add\_car">添加车辆</string>

<!— 选择停车位界面—>

<string name="activity\_chooseSpace\_choose">选择停放车位</string>

<string name="activity\_chooseSpace\_used">已停车</string>

<string name="activity\_chooseSpace\_notUsed">未停车</string>

<string name="activity\_chooseSpace\_mine">我的车位</string>

<string name="activity\_chooseSpace\_notChoose">您还未选择车位</string>

<!— 选择车辆界面 —>

<string name="activity\_chooseCar\_choose">选择停放车辆</string>

<string name="activity\_chooseCar\_type">车型:</string>

<string name="activity\_chooseCar\_licenseNum">车牌号:</string>

<!— 选择停放时间界面 —>

<string name="activity\_chooseTime\_choose">设置停车时间</string>

<!— 我的预约界面 —>

<string name="fragment\_order\_mine">我的预约</string>

<!— 个人中心界面 —>

<string name="fragment\_mine\_car">我的车辆</string>

<string name="fragment\_mine\_record">停车记录</string>

<string name="fragment\_mine\_changePassword">修改密码</string>

<string name="fragment\_mine\_weatherForcast">天气预报</string>

<string name="fragment\_mine\_logout">退出登录</string>

<!— 修改个人信息界面 —>

<string name="activity\_changeUserInfo\_change">修改个人信息</string>

<string name="activity\_changeUserInfo\_name">姓名</string>

<string name="activity\_changeUserInfo\_gender">性别</string>

<string name="activity\_changeUserInfo\_age">年龄</string>

<string name="activity\_changeUserInfo\_submit">提交</string>

<string name="activity\_changeUserInfo\_fail">x</string>

<string name="activity\_changeUserInfo\_success">√</string>

<string name="activity\_changeUserInfo\_male">男</string>

<string name="activity\_changeUserInfo\_female">女</string>

<!— 修改密码 —>

<string name="activity\_changeUserPassword\_old">旧密码</string>

<string name="activity\_changeUserPassword\_new">新密码</string>

<string name="activity\_changeUserPassword\_submit">提交</string>

<string name="activity\_changeUserPassword\_fail">x</string>

<string name="activity\_changeUserPassword\_success">√</string>

<!— 停车记录 —>

<string name="activity\_record\_time">时间:</string>

<string name="activity\_record\_fee">费用:</string>

<string name="activity\_record\_type">车型:</string>

<string name="activity\_record\_licenseNum">车牌号:</string>

<!— 我的车辆 —>

<string name="activity\_my\_car\_type">车型:</string>

<string name="activity\_my\_car\_licenseNum">车牌号:</string>

<!— 修改车辆信息 —>

<string name="activity\_update\_car\_type">车型:</string>

<string name="activity\_update\_car\_licenseNum">车牌号:</string>

<string name="activity\_update\_car\_submit">提交</string>

<string name="activity\_update\_car\_fail">x</string>

<string name="activity\_update\_car\_success">√</string>

<!— 添加车辆 —>

<string name="activity\_add\_car\_type">车型:</string>

<string name="activity\_add\_car\_licenseNum">车牌号:</string>

<string name="activity\_add\_car\_success">√</string>

<string name="activity\_add\_car\_submit">添加</string>

<string name="activity\_add\_car\_fail">x</string>

</resources>

# 4 程序实现及结果

## 4.1 登录与注册

登录与注册写在一个界面，采用动画效果来切换注册与注册功能。

淡入动画效果:时长500毫秒，透明度从0变为1，并且在动画后保持动画最后的效果

<set xmlns:android="http://schemas.android.com/apk/res/android">

<alpha

android:duration="500"

android:fillAfter="true"

android:fromAlpha="0.0"

android:toAlpha="1.0" />

</set>

淡出动画效果: 时长500毫秒，透明度从1变为0，并且在动画后保持动画最后的效果

<set xmlns:android="http://schemas.android.com/apk/res/android">

<alpha

android:duration="500"

android:fillAfter="true"

android:fromAlpha="1.0"

android:toAlpha="0.0" />

</set>

对于手机号的验证采用正则表达式

public static boolean isMobileNum(String mobiles) {

Pattern p = Pattern.compile("^((13[0-9])|(15[^4,\\D])|(18[0-9])|177)\\d{8}$");

Matcher m = p.matcher(mobiles);

return m.matches();

}

当点击切换登录注册按钮时，会将当前所有按钮设置为不可点击，执行淡出动画，当变淡动画执行完之后，如果当前状态为登录，则将标志flag设置为注册状态，按钮的文字变为注册。如果当前状态为注册，则将标志flag设置为登录状态，按钮的文字变为登录。然后执行淡入动画，当淡入动画执行完后将所有按钮设置为可点击。

登录和注册采用异步处理

登陆的AsyncTask：在线程执行前，将确定按钮设置为运行状态，获取手机号和密码输入框中的内容。然后开始

class LoginTask extends AsyncTask {

String user\_phoneNum;

String user\_password;

@Override

protected void onPreExecute() {

btn\_login.setProgress(50);

user\_phoneNum = et\_phoneNum.getText().toString();

user\_password = et\_password.getText().toString();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

DataOperateUtils.saveLoginInfo(LoginActivity.this, user\_phoneNum, user\_password);

RequestBody body = new FormEncodingBuilder()

.add("user\_phoneNum", user\_phoneNum)

.add("user\_password", user\_password)

.build();

return HttpUtils.httpPost(URLAddress.loginURL, body);

}

@Override

protected void onPostExecute(Object o) {

setLoginInfo(o.toString());

setClickAbled(true);

super.onPostExecute(o);

}

}

// 注册的task

class RegistTask extends AsyncTask {

String user\_phoneNum;

String user\_password;

@Override

protected void onPreExecute() {

btn\_login.setProgress(50);

user\_phoneNum = et\_phoneNum.getText().toString();

user\_password = et\_password.getText().toString();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

DataOperateUtils.saveLoginInfo(LoginActivity.this, user\_phoneNum, user\_password);

RequestBody body = new FormEncodingBuilder()

.add("user\_phoneNum", user\_phoneNum)

.add("user\_password", user\_password)

.add("user\_name", "")

.add("user\_age", "0")

.add("user\_gender", "1")

.add("user\_img", "")

.build();

return HttpUtils.httpPost(URLAddress.registURL, body);

}

@Override

protected void onPostExecute(Object o) {

setRegistInfo(o.toString());

setClickAbled(true);

super.onPostExecute(o);

}

}

在点击登录或注册按钮时需要对手机号及密码进行验证，若都正确则调用下一步方法：

btn\_login.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

hideKeyboard();

// 检查手机号码合法性

if (!Utils.isMobileNum(et\_phoneNum.getText().toString())) {

warpper\_phoneNum.setError("手机号码不正确");

return;

}

warpper\_phoneNum.setError("");

if (et\_password.getText().toString().equals("")) {

warpper\_password.setError("密码不能为空");

return;

}

warpper\_password.setError("");

if (et\_password.getText().toString().length() < 6) {

warpper\_password.setError("密码长度需大于6位");

return;

}

warpper\_password.setError("");

setClickAbled(false);

if (flag == FLAG\_LOGIN) {

// 登录状态

LoginTask loginTask = new LoginTask();

loginTask.execute();

} else {

// 注册状态

RegistTask registTask = new RegistTask();

registTask.execute();

}

}

});

}

## 首页

首页采用ViewPager的方式，ViewPager的适配器代码如下：

public class MainPagerAdapter extends FragmentPagerAdapter {

public MainPagerAdapter(FragmentManager fm) {

super(fm);

}

@Override

public Fragment getItem(int position) {

// getItem is called to instantiate the fragment for the given page.

// Return a PlaceholderFragment (defined as a static inner class below).

switch(position){

case 0:

return new OrderFragment();

case 1:

return new FriendsFragment();

case 2:

return new MineFragment();

default:

return new OrderFragment();

}

}

@Override

public int getCount() {

// Show 3 total pages.

return 3;

}

@Override

public CharSequence getPageTitle(int position) {

switch (position) {

case 0:

return "预约";

case 1:

return "车友";

case 2:

return "我的";

}

return null;

}

}

在MainActivity中需要进行一些数据的初始化，比如查询个人车辆信息:

class GetCarAsyncTask extends AsyncTask {

CustomPrgressDailog dailog = new CustomPrgressDailog(MainActivity.this, R.style.DialogNormal);

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.USER\_ID, UserInfoInCache.user\_id + "").build();

return HttpUtils.httpPost(URLAddress.findUserCarURL, body);

}

@Override

protected void onPostExecute(Object o) {

handleCarResult(o.toString());

dailog.dismiss();

super.onPostExecute(o);

}

}

获取屏幕信息：

private void setScreenInfo() {

int screenWidth = getWindow().getWindowManager().getDefaultDisplay().getWidth();

int screenHeight = getWindow().getWindowManager().getDefaultDisplay().getHeight();

int statusBarHeight = getStatusHeight(this);

screenHeight = screenHeight - statusBarHeight;

DisplayMetrics dis = new DisplayMetrics();

getWindowManager().getDefaultDisplay().getMetrics(dis);

// 密度

float de = dis.density;

ScreenInfo.setDe(de);

ScreenInfo.setScreenHeight(screenHeight);

ScreenInfo.setScreenWidth(screenWidth);

ScreenInfo.setStatusBarHeight(statusBarHeight);

}

获取状态栏高度

private int getStatusHeight(Activity activity) {

int statusHeight = 0;

Rect localRect = new Rect();

activity.getWindow().getDecorView().getWindowVisibleDisplayFrame(localRect);

statusHeight = localRect.top;

if (0 == statusHeight) {

Class<?> localClass;

try {

localClass = Class.forName("com.android.internal.R$dimen");

Object localObject = localClass.newInstance();

int i5 = Integer.parseInt(localClass.getField("status\_bar\_height").get(localObject).toString());

statusHeight = activity.getResources().getDimensionPixelSize(i5);

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (InstantiationException e) {

e.printStackTrace();

} catch (NumberFormatException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (SecurityException e) {

e.printStackTrace();

} catch (NoSuchFieldException e) {

e.printStackTrace();

}

}

return statusHeight;

}

### 4.2.2 查询预约页

进入该页面，首先会自动查询用户的预约车位的信息:

class GetOrderTask extends AsyncTask {

CustomPrgressDailog dailog = new CustomPrgressDailog(getActivity(), R.style.DialogNormal);

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.USER\_ID, UserInfoInCache.user\_id + "").build();

return HttpUtils.httpPost(URLAddress.findTodayOrderURl, body);

}

@Override

protected void onPostExecute(Object o) {

handleOrderResult(o.toString());

dailog.dismiss();

super.onPostExecute(o);

}

}

同时，在这个页面也可以点击右下角的悬浮按钮进行预约车位:

fa\_fab.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Snackbar.make(view, "是否预约车位?", Snackbar.LENGTH\_SHORT)

.setAction("是", new View.OnClickListener() {

@Override

public void onClick(View v) {

startActivity(new Intent(getActivity(), ChooseSpaceActivity.class));

}

}).setActionTextColor(getResources().getColor

(R.color.white\_normal)).show();

}

}

);

### 4.2.3 个人中心页

个人中心主要功能是用户头像的处理，拍照功能如下:

public void getCamera() {

String SDState = Environment.getExternalStorageState();

if (SDState.equals(Environment.MEDIA\_MOUNTED)) {

new DateFormat();

String name = DateFormat.format("yyyyMMdd\_hhmmss", Calendar.getInstance(Locale.getDefault())) + ".jpg";

String path = Environment.getExternalStorageDirectory().getAbsolutePath();

imgPath = path + "/" + GlobalDefine.APP\_NAME + "/" + name;

File tempFile = new File(Environment.getExternalStorageDirectory(), "/" + GlobalDefine.APP\_NAME + "/" + name);

Intent intent = new Intent("android.media.action.IMAGE\_CAPTURE");

intent.putExtra(MediaStore.EXTRA\_OUTPUT, Uri.fromFile(new File(tempFile.getAbsolutePath())));

intent.putExtra(MediaStore.EXTRA\_VIDEO\_QUALITY, 1);

startActivityForResult(intent, RESULT\_CAMERA);

}

}

从相册中获取图片:

public void getImage() {

Intent intent = new Intent();

intent.setType("image/\*");

intent.setAction(Intent.ACTION\_GET\_CONTENT);

startActivityForResult(intent, RESULT\_PHOTO);

}

得到图片后需要调用图片裁剪的功能, android4.4会出现从相册选取图片查询路径为空的情况,所以需要将路径转换下:

private String uri2filePath(Uri uri, Activity activity) {

String path = "";

if (Build.VERSION.SDK\_INT >= 19 && DocumentsContract.isDocumentUri(activity, uri)) {

String wholeID = DocumentsContract.getDocumentId(uri);

String id = wholeID.split(":")[1];

String[] column = {MediaStore.Images.Media.DATA};

String sel = MediaStore.Images.Media.\_ID + "=?";

Cursor cursor = activity.getContentResolver().query(

MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI, column, sel,

new String[]{id}, null);

int columnIndex = cursor.getColumnIndex(column[0]);

if (cursor.moveToFirst()) {

path = cursor.getString(columnIndex);

}

cursor.close();

} else {

String[] projection = {MediaStore.Images.Media.DATA};

Cursor cursor = activity.getContentResolver().query(uri, projection, null, null, null);

int column\_index = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);

cursor.moveToFirst();

path = cursor.getString(column\_index);

}

return path;

}

个人中心的背景是动态的光晕:

private void creatHalo() {

haloView.loadFlower();

haloView.addRect();

Timer timer = new Timer();

TimerTask timerTask = new TimerTask() {

@Override

public void run() {

Message msg = new Message();

msg.what = 0x999;

handler.sendMessage(msg);

}

};

timer.schedule(timerTask, 0, 85);

}

### 个人信息完善

在此界面可已更新用户的个人信息，点击确定按钮后，本地会进行输入信息合法性的判断，然后向服务器发送请求:

class SubmitTask extends AsyncTask {

private String name, age;

private int gender;

@Override

protected void onPreExecute() {

btn\_submit.setProgress(50);

name = et\_name.getText().toString();

age = et\_age.getText().toString();

gender = rg\_gender.getCheckedRadioButtonId() == R.id.activity\_changeUserInfo\_rb\_female ? 0 : 1;

btn\_submit.setClickable(false);

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.USER\_ID, UserInfoInCache.user\_id + "").add(SQLWord.USER\_NAME, Base64.encode(name))

.add(SQLWord.USER\_AGE, age).add(SQLWord.USER\_GENDER, gender + "").build();

return HttpUtils.httpPost(URLAddress.updateUserInfoURL, body);

}

@Override

protected void onPostExecute(Object o) {

setUpdateInfo(o.toString());

btn\_submit.setClickable(true);

super.onPostExecute(o);

}

}

得到修改结果后，更新界面:

private void setUpdateInfo(String json) {

if (json == null || json.equals("")) {

btn\_submit.setProgress(0);

CustomToast.showToast(getApplicationContext(), "修改失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 修改失败,并显示原因

btn\_submit.setProgress(0);

CustomToast.showToast(getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 修改成功,解析用户信息

btn\_submit.setProgress(100);

NotifyDataChangeUtils.REFRESH\_USER\_INFO = 1;

JsonUtils.getUserInfo(UpdateUserInfoActivity.this, JsonUtils.getResultMsgString(json));

finishDelay();

}

}

## 个人车辆信息的维护

### 查询

在MainActivity中已经查询过一次用户车辆，如果车辆信息有更新的话，只需将新的车辆信息储存到内存缓存中再调用adapter.notifyDataSetChanged()更新界面即可。列表都采用JazzyListView,可以添加动画效果。

### 增加

从本地选取车辆照片或者拍照获取图片的逻辑与用户头像的逻辑相同。完成基本信息的填写之后，添加车辆信息的逻辑处理如下:显示加载Dialog->向服务器发送车辆图片->若不成功Toast提示失败，若成功则进行下一步->向服务器发送车辆的基本信息->接收处理结果->关闭加载Dialog->显示结果。最后再更新内存中的缓存数据。

// 检查表单的完整性

private boolean checkForm() {

if (et\_type.getText().toString().equals("")) {

warpper\_type.setError("车辆类型不能为空");

return false;

}

warpper\_type.setError("");

if (et\_licenseNum.getText().toString().equals("")) {

warpper\_licenseNum.setError("车牌号不能为空");

return false;

}

warpper\_licenseNum.setError("");

return true;

}

//上传车辆图片

private void uploadImage() {

dialog = new CustomPrgressDailog(UpdateCarActivity.this, R.style.DialogNormal);

dialog.show();

if (imgPath.equals(UserInfoInCache.myCarList.get(index).getCar\_img()) || imgPath.equals("")) {

UpdateCarTask updateCarTask = new UpdateCarTask(imgPath);

updateCarTask.execute();

} else {

UploadManager uploadManager = new UploadManager();

File data = new File(imgPath);

String key = imgPath.split("/")[imgPath.split("/").length - 1];

String token = Utils.getUploadToken();

uploadManager.put(data, key, token,

new UpCompletionHandler() {

@Override

public void complete(String key, ResponseInfo info, JSONObject res) {

// res 包含hash、key等信息，具体字段取决于上传策略的设置。

UpdateCarTask updateCarTask = new UpdateCarTask(URLAddress.QINIU\_URL + "/" + key);

updateCarTask.execute();

}

}, null);

}

}

//添加车辆的Task

class AddCarTask extends AsyncTask {

private String car\_type, car\_licenseNum, car\_image;

public AddCarTask(String car\_image) {

this.car\_image = car\_image;

}

@Override

protected void onPreExecute() {

car\_type = et\_type.getText().toString();

car\_licenseNum = et\_licenseNum.getText().toString();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.USER\_ID, UserInfoInCache.user\_id + "")

.add(SQLWord.CAR\_TYPE, Base64.encode(car\_type))

.add(SQLWord.CAR\_LICENSENUM, Base64.encode(car\_licenseNum))

.add(SQLWord.CAR\_IMG, car\_image).build();

return HttpUtils.httpPost(URLAddress.addCarURL, body);

}

@Override

protected void onPostExecute(Object o) {

String json = o.toString();

if (json == null || json.equals("")) {

dialog.dismiss();

CustomToast.showToast(getApplicationContext(), "加载失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 显示失败原因

dialog.dismiss();

CustomToast.showToast(getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 成功时相关处理

GetCarAsyncTask getCarAsyncTask = new GetCarAsyncTask();

getCarAsyncTask.execute();

}

super.onPostExecute(o);

}

}

### 修改

修改车辆信息与添加车辆信息逻辑相同只是与更新车辆信息调用的接口不同，先上传图片再上传车辆信息，根据服务器返回的结果，判断是将新的车辆信息更新到内存缓存中还是通过Toast提示失败。

### 删除

删除车辆成功后，需要在在内存缓存中移除该车辆信息，若未成功，则通过Toast提示删除失败

// 删除车辆的Task

class DeleteCarTask extends AsyncTask {

@Override

protected void onPreExecute() {

dialog = new CustomPrgressDailog(UpdateCarActivity.this, R.style.DialogNormal);

dialog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.CAR\_ID, UserInfoInCache.myCarList.get(index).getCar\_id() + "").build();

return HttpUtils.httpPost(URLAddress.deleteCarURL, body);

}

@Override

protected void onPostExecute(Object o) {

String json = o.toString();

if (json == null || json.equals("")) {

CustomToast.showToast(getApplicationContext(), "加载失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 显示失败原因

CustomToast.showToast(getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 成功时相关处理

CustomToast.showToast(getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

UserInfoInCache.myCarList.remove(index);

finish();

}

dialog.dismiss();

super.onPostExecute(o);

}

}

## 停车记录

查询成功后再界面中显示，若为查询到结果，则通过Toast提示原因。

// 查询停车记录

class GetRecordTask extends AsyncTask {

CustomPrgressDailog dailog = new CustomPrgressDailog(RecordActivity.this, R.style.DialogNormal);

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.USER\_ID, UserInfoInCache.user\_id + "").build();

return HttpUtils.httpPost(URLAddress.findRecordByUserIdURL, body);

}

@Override

protected void onPostExecute(Object o) {

handleRecord(o.toString());

dailog.dismiss();

super.onPostExecute(o);

}

}

## 修改密码

修改密码时需要用户输入原密码作为校验，原密码正确则允许用户修改，在提交新密码新密码时，需要判断位数（不少于6位），根据服务器返回的结果提示用户修改成功或者修改失败的原因。

// 校验表单

private boolean checkForm() {

// 旧密码

if (et\_old.getText().toString().equals("")) {

warpper\_old.setError("密码不能为空");

return false;

}

warpper\_old.setError("");

if (et\_old.getText().toString().length() < 6) {

warpper\_old.setError("密码长度需大于6位");

return false;

}

warpper\_old.setError("");

// 新密码

if (et\_new.getText().toString().equals("")) {

warpper\_new.setError("密码不能为空");

return false;

}

warpper\_new.setError("");

if (et\_new.getText().toString().length() < 6) {

warpper\_new.setError("密码长度需大于6位");

return false;

}

warpper\_new.setError("");

return true;

}

// 更改密码的task

class UpdatePasswordTask extends AsyncTask {

CustomPrgressDailog dailog = new CustomPrgressDailog(UpdateUserPasswordActivity.this, R.style.DialogNormal);

private String old\_password, new\_password;

@Override

protected void onPreExecute() {

old\_password = et\_old.getText().toString();

new\_password = et\_new.getText().toString();

btn\_submit.setClickable(false);

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.USER\_ID, UserInfoInCache.user\_id + "").add("old\_password", old\_password)

.add("new\_password", new\_password).build();

return HttpUtils.httpPost(URLAddress.updateUserPasswordURL, body);

}

@Override

protected void onPostExecute(Object o) {

handleUpdatePassword(o.toString(), new\_password);

btn\_submit.setClickable(true);

dailog.dismiss();

super.onPostExecute(o);

}

}

// 处理服务器返回的结果

private void handleUpdatePassword(String json, String new\_password) {

if (json == null || json.equals("")) {

btn\_submit.setProgress(0);

CustomToast.showToast(getApplicationContext(), "修改失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 修改失败,并显示原因

btn\_submit.setProgress(0);

CustomToast.showToast(getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 修改成功,解析用户信息

btn\_submit.setProgress(100);

UserInfoInCache.user\_password = new\_password;

DataOperateUtils.saveLoginInfo(UpdateUserPasswordActivity.this, UserInfoInCache.user\_phoneNum, new\_password);

finishDelay();

}

}

## 退出登录

退出登录时需要清空本地缓存文件，注销推送tag。

ll\_logout.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

new AlertDialog.Builder(getActivity()).setMessage("退出登录?")

.setPositiveButton("是", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

DataOperateUtils.clearAllData(getActivity());

EMChatManager.getInstance().logout();

MyPushMessageReceiver.unregistPushTag(getActivity(), UserInfoInCache.user\_id + "");

startActivity(new Intent(getActivity(), LoginActivity.class));

getActivity().finish();

}

}).setNegativeButton("否", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

dialog.dismiss();

}

}).create().show();

}

});

## 预约车位

### 选择停车车位

在选择车位时，用到了自定义View，根据屏幕的长宽计算每个车位的应该显示在屏幕上的大小。

private final int column = 7;//定义车位显示列数

private final int padding = 10;//车位间距

private void calculateWidth() {

width = (gv\_gridView.getWidth() - padding \* (column - 1)) / column; // 计算每个小车位的宽度

}

每个车位也是自定义的view，根据不同的状态显示不同的颜色。

public View getView(final int position, View convertView, ViewGroup parent) {

//定义一个ImageView,显示在GridView里

final ImageView iv = new ImageView(context);

iv.setLayoutParams(new android.widget.AbsListView.LayoutParams(width, width));

if (flagList.get(position) == USED) {

// 车位被占用

iv.setImageDrawable(context.getResources().getDrawable(R.drawable.frame\_parking\_space\_grey));

} else if (flagList.get(position) == USER\_USED) {

// 用户占用

iv.setImageDrawable(context.getResources().getDrawable(R.drawable.frame\_parking\_space\_cyan));

} else {

// 车位未被占用

iv.setImageDrawable(context.getResources().getDrawable(R.drawable.frame\_parking\_space\_white));

}

iv.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

if (flagList.get(position) == 0) {

// 该处车位可选择

if (!flagList.contains(USER\_USED)) {

// 用户未选择

flagList.set(position, USER\_USED);

} else {

// 用户已选择

flagList.set(flagList.indexOf(USER\_USED), NOT\_USED);

flagList.set(position, USER\_USED);

}

notifyDataSetChanged();

}

}

});

return iv;

}

在Activity中需要查询所有车位的状态

class AllSpaceInfoTask extends AsyncTask {

CustomPrgressDailog dailog = new CustomPrgressDailog(ChooseSpaceActivity.this, R.style.DialogNormal);

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

return HttpUtils.httpGet(URLAddress.allSpaceInfoURl);

}

@Override

protected void onPostExecute(Object o) {

handleSpaceResult(o.toString());

dailog.dismiss();

super.onPostExecute(o);

}

}

用户选择好停车位后可以进行下一步

btn\_next.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

if (drawParkinglot.getParkId() != -1) {

Intent intent = new Intent(ChooseSpaceActivity.this, ChooseCarActivity.class);

intent.putExtra(SQLWord.PARK\_ID,drawParkinglot.getParkId());

startActivity(intent);

} else {

CustomToast.showToast(getApplicationContext(),

getString(R.string.activity\_chooseSpace\_notChoose), 1000);

}

}

});

### 选择停放车辆

与首页查询车辆的逻辑相同，查询到用户车辆后在listView中显示出来。用户点击某一辆车之后，出发ListView的监听。

lv\_listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

Intent intent = new Intent(ChooseCarActivity.this, ChooseTimeActivity.class);

intent.putExtra(SQLWord.PARK\_ID, park\_id);

intent.putExtra(SQLWord.CAR\_ID, UserInfoInCache.myCarList.get(position).getCar\_id());

startActivity(intent);

}

});

### 选择停车时间

选择停车时间界面采用DatePicker与TimePicker结合，将时间精确到秒，在用户选择完停车时间时候本地会对停车时间做一次校验，满足停车时间必须在当前时间之后以及智能预约2小时以内的车位，避免资源浪费。

// 校验表单

private boolean checkPickerTime() {

String year = dp\_picker.getYear() + "";

String month = (dp\_picker.getMonth() + 1) + "";

String day = dp\_picker.getDayOfMonth() + "";

String hour = tp\_picker.getCurrentHour() + "";

String minute = tp\_picker.getCurrentMinute() + "";

// 调整时间字符串格式

if (dp\_picker.getMonth() < 9) {

month = "0" + (dp\_picker.getMonth() + 1);

}

if (dp\_picker.getDayOfMonth() < 10) {

day = "0" + dp\_picker.getDayOfMonth();

}

if (tp\_picker.getCurrentHour() < 10) {

hour = "0" + tp\_picker.getCurrentHour();

}

if (tp\_picker.getCurrentMinute() < 10) {

minute = "0" + tp\_picker.getCurrentMinute();

}

// 判断是否在当前时间之前

if (Utils.comparePointTime(year + "-" + month + "-" + day

+ " " + hour + ":" + minute + ":00")) {

if (Utils.comparePointTimeInPointHour(year + "-" + month + "-" + day

+ " " + hour + ":" + minute + ":00", GlobalDefine.orderForwardHour)) {

// 选择时间在当前时间2小时之后

CustomToast.showToast(getApplicationContext(), "只能预约2小时以内", 1000);

return false;

} else {

// ok

park\_startTime = Utils.getCurrentTime(year + "-" + month + "-" + day

+ " " + hour + ":" + minute + ":00");

return true;

}

} else {

CustomToast.showToast(getApplicationContext(), "选择的时间不合适", 1000);

return false;

}

}

表单校验无误后，便向服务器提交预约数据。

class OrderTask extends AsyncTask {

CustomPrgressDailog dailog = new CustomPrgressDailog(ChooseTimeActivity.this, R.style.DialogNormal);

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.PARK\_ID,

park\_id + "").add(SQLWord.CAR\_ID, car\_id + "")

.add(SQLWord.PARK\_STARTTIME, park\_startTime).build();

return HttpUtils.httpPost(URLAddress.orderSPaceURl, body);

}

@Override

protected void onPostExecute(Object o) {

handleOrderResult(o.toString());

dailog.dismiss();

super.onPostExecute(o);

}

}

### 预约结果

预约成功后会向用户提示预约结果，并且可以直接返回到主界面。主界面会在onResume()中刷新用户预约情况的数据。

private void handleOrderResult(String json) {

if (json == null || json.equals("")) {

CustomToast.showToast(getApplicationContext(), "加载失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 显示失败原因

CustomToast.showToast(getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 成功时相关处理

NotifyDataChangeUtils.REFRESH\_TODAY\_ORDER = 1;

new AlertDialog.Builder(ChooseTimeActivity.this).setMessage("预约成功")

.setPositiveButton("返回主界面", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

startActivity(new Intent(ChooseTimeActivity.this, MainActivity.class));

}

}).create().show();

}

}

## 取消预约

在主界面中，停车记录是以列表形式现实的，如果用户有预约记录但是还未开始停车，则可以通过长按这条记录来取消预约:

class CancleOrderTask extends AsyncTask {

private CustomPrgressDailog dailog = new CustomPrgressDailog(getActivity(), R.style.DialogNormal);

private int index;

public CancleOrderTask(int index) {

this.index = index;

}

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.PARK\_ID, parkList.get(index).getPark\_id() + "").build();

return HttpUtils.httpPost(URLAddress.cancleOrderURL, body);

}

@Override

protected void onPostExecute(Object o) {

String json = o.toString();

if (json == null || json.equals("")) {

CustomToast.showToast(getActivity().getApplicationContext(), "取消预约失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 显示失败原因

CustomToast.showToast(getActivity().getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 成功时相关处理

parkList.remove(index);

adapter.updateList(parkList);

}

dailog.dismiss();

super.onPostExecute(o);

}

}

## 结束停车

如果用户已经开始停车,则可以通过长按这条记录来结束停车:

class EndParkingTask extends AsyncTask {

private CustomPrgressDailog dailog = new CustomPrgressDailog(getActivity(), R.style.DialogNormal);

private int index;

public EndParkingTask(int index) {

this.index = index;

}

@Override

protected void onPreExecute() {

dailog.show();

super.onPreExecute();

}

@Override

protected Object doInBackground(Object[] params) {

RequestBody body = new FormEncodingBuilder()

.add(SQLWord.PARK\_ID, parkList.get(index).getPark\_id() + "").build();

return HttpUtils.httpPost(URLAddress.endParkingURL, body);

}

@Override

protected void onPostExecute(Object o) {

String json = o.toString();

if (json == null || json.equals("")) {

CustomToast.showToast(getActivity().getApplicationContext(), "结束停车失败", 1000);

return;

}

if (JsonUtils.getResultCode(json) < 1) {

// 显示失败原因

CustomToast.showToast(getActivity().getApplicationContext(), JsonUtils.getResultMsgString(json), 1000);

} else {

// 成功时相关处理

parkList.remove(index);

adapter.updateList(parkList);

}

dailog.dismiss();

super.onPostExecute(o);

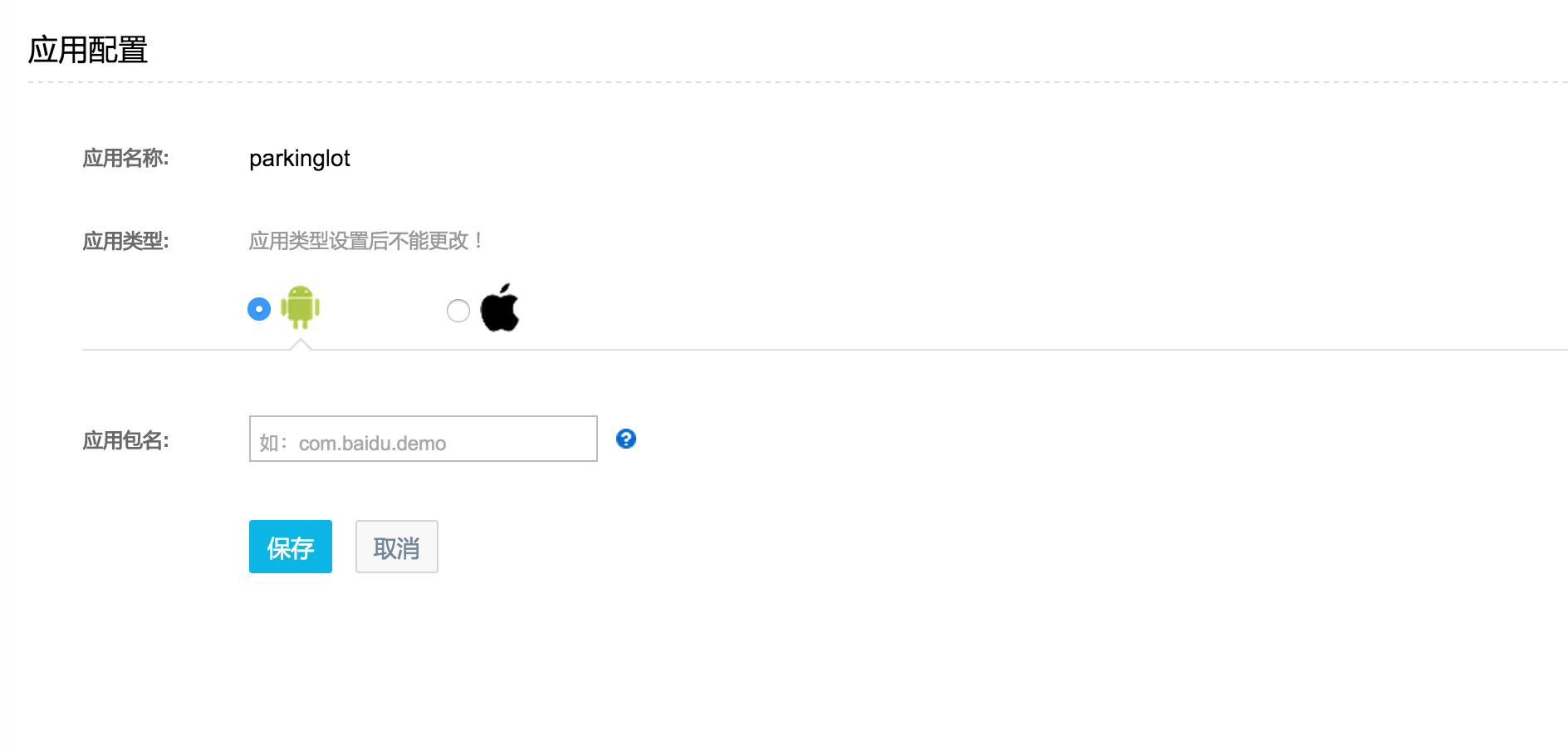
}

}

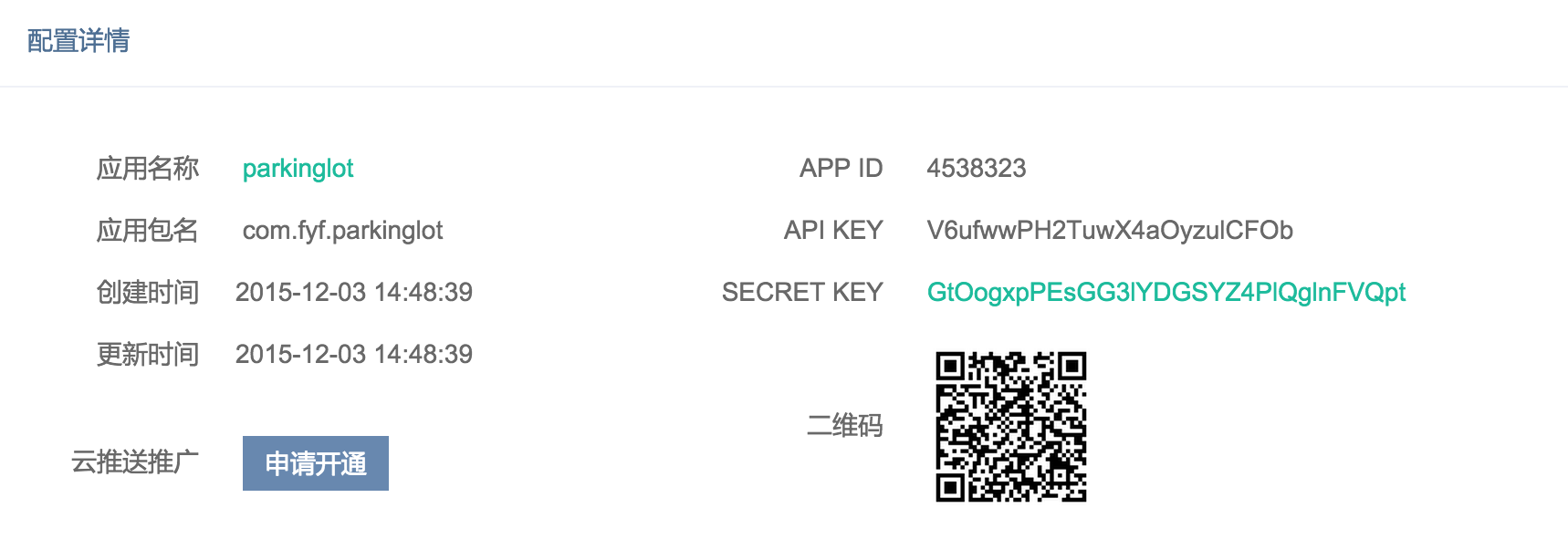
## 4.11 推送

### 4.11.1 在推送平台创建应用

推送采用百度云推送的SDK，首先登陆http://push.baidu.com/ 注册账号，完成开发者身份认证，点击创建应用，完成基本信息的填写。



配置完成后即可获取到应用的APP ID与API KEY，在集成推送SDK是需要用到。

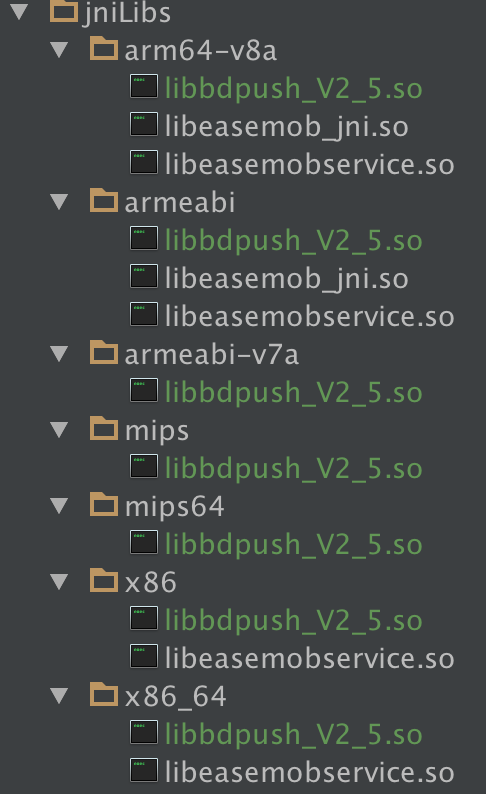


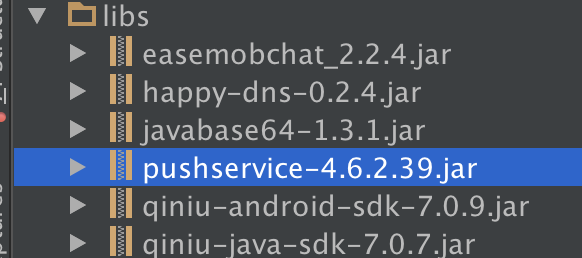
### 4.11.2 集成SDK

在<http://push.baidu.com/sdk/push_client_sdk_for_android> 下载安卓版的推送SDK。



将so文件拷贝至jniLibs下，将jar包拷贝至libs下





在AndroidManifest.xml中声明权限，注册服务及广播。

<!-- Push service 运行需要的权限 -->

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.READ\_PHONE\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" />

<uses-permission android:name="android.permission.WRITE\_SETTINGS" />

<uses-permission android:name="android.permission.VIBRATE" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.DISABLE\_KEYGUARD" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_DOWNLOAD\_MANAGER"/>

<uses-permission android:name="android.permission.DOWNLOAD\_WITHOUT\_NOTIFICATION" />

<uses-permission android:name="android.permission.EXPAND\_STATUS\_BAR" />

<!-- push service start -->

<!-- 用于接收系统消息以保证PushService正常运行 -->

<receiver android:name="com.baidu.android.pushservice.PushServiceReceiver"

android:process=":bdservice\_v1" >

<intent-filter>

<action android:name="android.intent.action.BOOT\_COMPLETED" />

<action android:name="android.net.conn.CONNECTIVITY\_CHANGE" />

<action android:name="com.baidu.android.pushservice.action.notification.SHOW" />

<action android:name="com.baidu.android.pushservice.action.media.CLICK" />

<!-- 以下四项为可选的action声明，可大大提高service存活率和消息到达速度 -->

<action android:name="android.intent.action.MEDIA\_MOUNTED" />

<action android:name="android.intent.action.USER\_PRESENT" />

<action android:name="android.intent.action.ACTION\_POWER\_CONNECTED" />

<action android:name="android.intent.action.ACTION\_POWER\_DISCONNECTED" />

</intent-filter>

</receiver>

<!-- Push服务接收客户端发送的各种请求-->

<receiver android:name="com.baidu.android.pushservice.RegistrationReceiver"

android:process=":bdservice\_v1" >

<intent-filter>

<action android:name="com.baidu.android.pushservice.action.METHOD" />

<action android:name="com.baidu.android.pushservice.action.BIND\_SYNC" />

</intent-filter>

<intent-filter>

<action android:name="android.intent.action.PACKAGE\_REMOVED" />

<data android:scheme="package" />

</intent-filter>

</receiver>

<service android:name="com.baidu.android.pushservice.PushService" android:exported="true"

android:process=":bdservice\_v1" >

<intent-filter >

<action android:name="com.baidu.android.pushservice.action.PUSH\_SERVICE" />

</intent-filter>

</service>

<!-- 4.4版本新增的CommandService声明，提升小米和魅族手机上的实际推送到达率 -->

<service android:name="com.baidu.android.pushservice.CommandService"

android:exported="true" />

<!-- push结束 -->

<!-- 4.6版本新增的Activity声明，提升Push后台存活率 -->

<activity

android:launchMode="singleInstance"

android:name="com.baidu.android.pushservice.PushKeepAlive"

android:theme="@android:style/Theme.Translucent.NoTitleBar"/>

最后在Application中添加执行初始化的代码

private void initPush() {

PushManager.startWork(getApplicationContext(),

PushConstants.LOGIN\_TYPE\_API\_KEY,

Utils.getMetaValue(getApplicationContext(), "BAIDU\_PUSH\_API\_KEY"));

// Push: 如果想基于地理位置推送，可以打开支持地理位置的推送的开关

PushManager.enableLbs(getApplicationContext());

}

### 4.11.3 处理推送消息

在用户成功登陆客户端后需要将该用户的用户id注册到百度云推送，以确保该用户能单独收到的推送消息。在用户退出登录时，需要将该用户的id在百度云推送解绑，以保证该用户不会再收到的推送消息。

// 绑定

public static void registPushTag(Context context, String user\_id) {

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

list.add(user\_id);

try {

PushManager.setTags(context, list);

} catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

}

// 解绑

public static void unregistPushTag(Context context, String user\_id) {

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

list.add(user\_id);

try {

PushManager.delTags(context, list);

} catch (Exception e) {

// TODO: handle exception

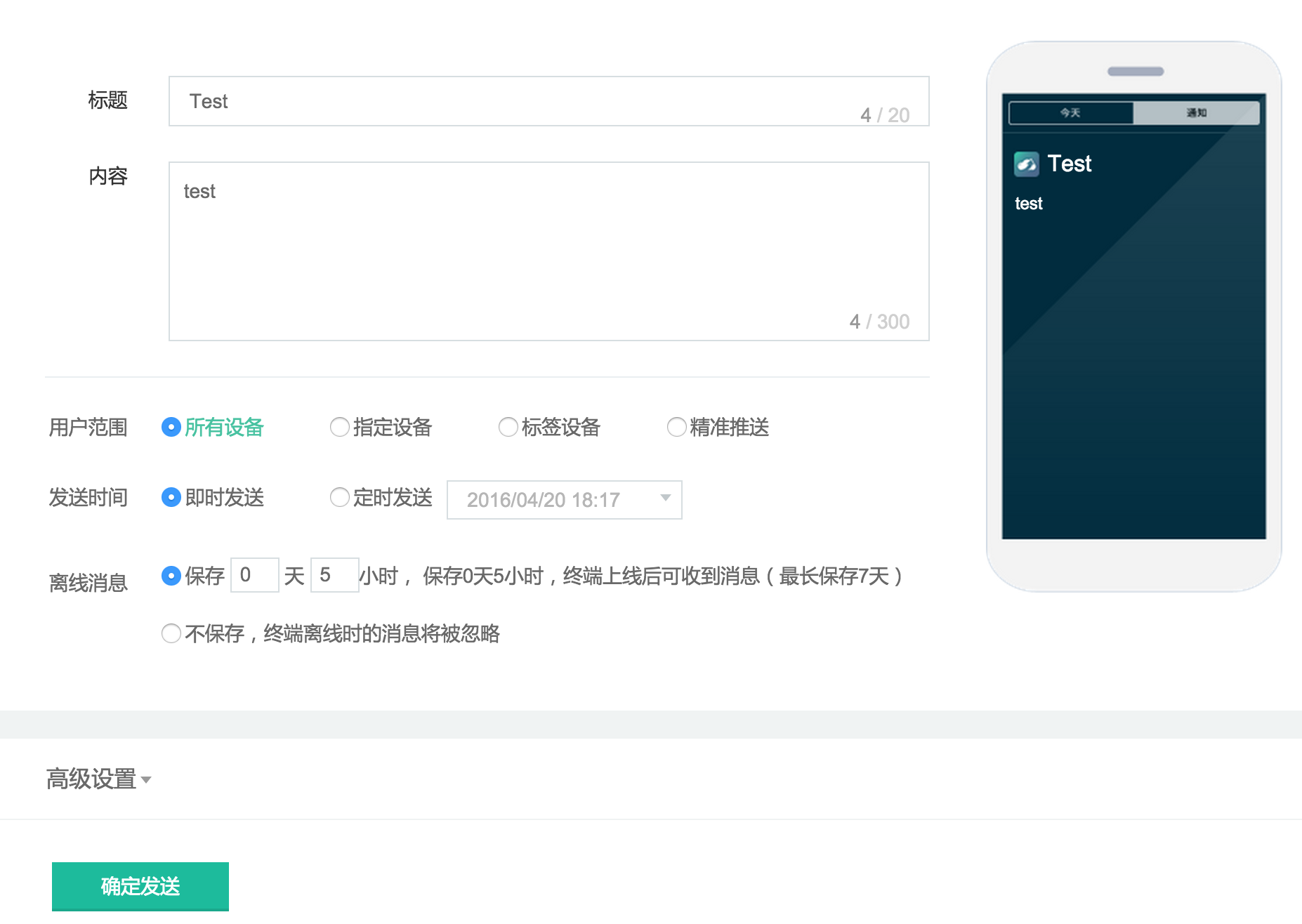
e.printStackTrace();

}

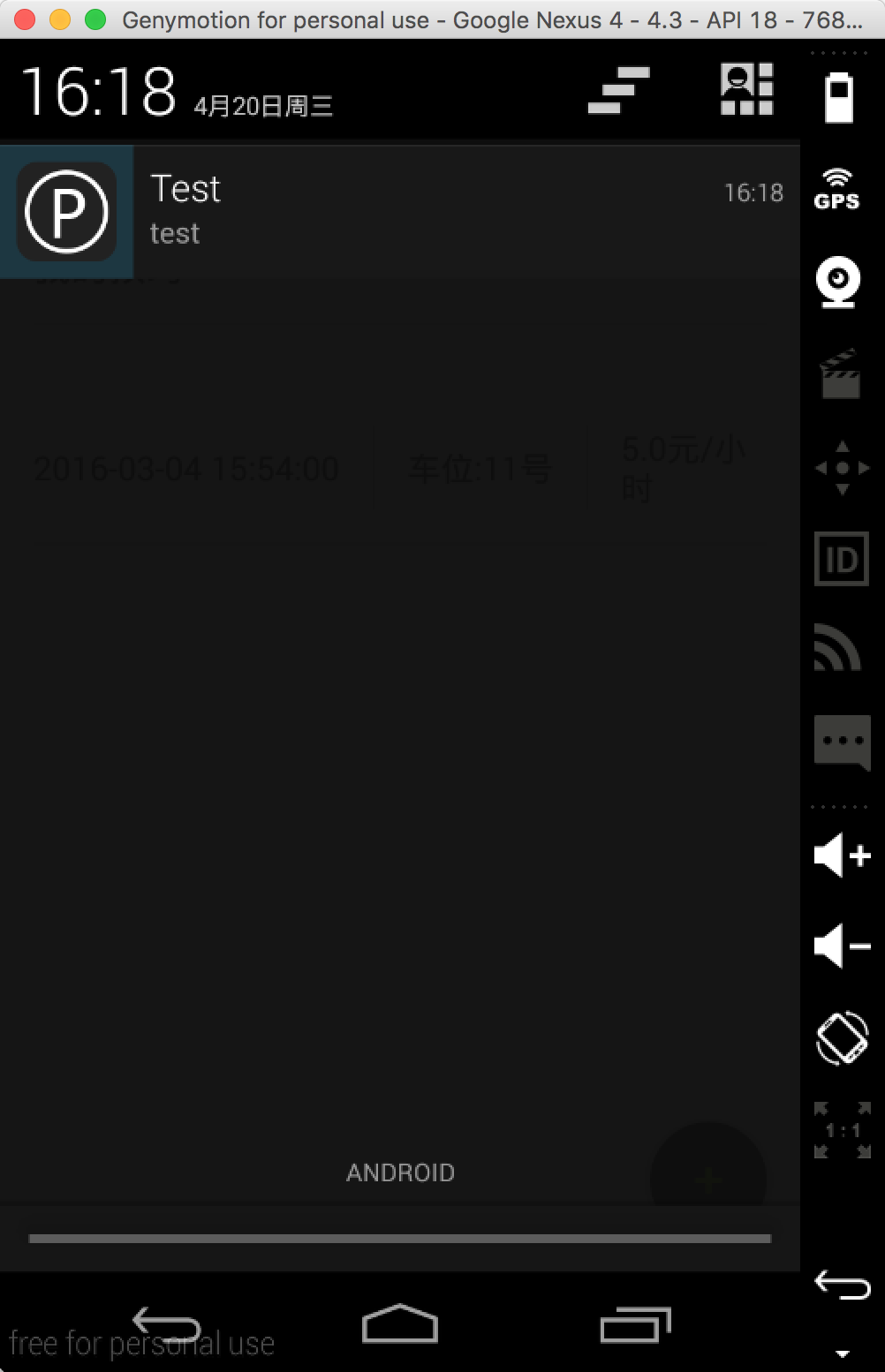
}

### 4.11.4 推送结果

在云推送页面可以创建推送消息，配置完成后点击确定发送即可。



在客户端可以看到推送结果。



## 4.12 build.gradle

项目需要在在build.gradle中配置运行项目所需最低的安卓版本，以及版本号、版本名称，同时因为引入了jar包，所以需要在gradle中配置。

apply plugin: 'com.android.application'

android {

compileSdkVersion 23

buildToolsVersion "23.0.3"

defaultConfig {

applicationId "com.fyf.parkinglot"

minSdkVersion 15

targetSdkVersion 23

versionCode 1

versionName "1.0"

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

packagingOptions {

}

}

dependencies {

compile fileTree(include: ['\*.jar'], dir: 'libs')

testCompile 'junit:junit:4.12'

compile 'com.android.support:appcompat-v7:23.3.0'

compile 'com.android.support:design:23.3.0'

compile 'com.android.support:cardview-v7:23.3.0'

compile 'com.github.dmytrodanylyk.circular-progress-button:library:1.1.3'

compile 'com.google.code.gson:gson:2.4'

compile 'com.sefford:circular-progress-drawable:1.31@aar'

compile 'com.twotoasters.jazzylistview:library:1.2.1'

compile 'com.twotoasters.jazzylistview:library-recyclerview:1.2.1'

compile 'de.hdodenhof:circleimageview:2.0.0'

compile 'com.edmodo:cropper:1.0.1'

compile 'com.squareup.okhttp:okhttp:2.6.0'

compile 'com.loopj.android:android-async-http:1.4.9'

compile 'com.facebook.fresco:fresco:0.9.0+'

compile files('libs/happy-dns-0.2.4.jar')

compile files('libs/qiniu-android-sdk-7.0.9.jar')

compile files('libs/javabase64-1.3.1.jar')

compile files('libs/pushservice-4.6.2.39.jar')

compile files('libs/easemobchat\_2.2.4.jar')

compile 'com.jakewharton:butterknife:5.1.1'

}

# 总结与展望

通过此次的停车场管理手机客户端软件开发，我对java与android有了更深的认识，对于Android Studio这个开发平台的掌握更深。

在整个毕设过程中，我首先仔细阅读了毕设的要求与题目，构建出了软件大致的开发结构，之后与同学一起探讨软件的功能流程、界面的设计与配色，做出了原型图

# 致谢

# 参考文献

文献翻译

Android Application Fundamentals

Android applications are written in the Java programming language. The Android SDK tools compile the code—along with any data and resource files—into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

Once installed on a device, each Android application lives in its own security sandbox:

The Android operating system is a multi-user Linux system in which each application

is a different user.

By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.

Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.

By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

However, there are ways for an application to share data with other applications and for an application to access system services:

It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources,applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate).

An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

Application Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your application's overall behavior.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. Here are the four types of application components:

Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of Activity and you can learn more about it in the Activities developer guide.

Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide. Content providers

A content provider manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the Note Pad sample application uses a content provider to save notes.

A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other applications to perform transactions. For more information, see the Content Providers developer guide.

Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object. For more information, see theBroadcastReceiver class.

A unique aspect of the Android system design is that any application can start another application’s component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application.

When the system starts a component, it starts the process for that application (if it's not already running) and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process.

Therefore, unlike applications on most other systems, Android applications don't have a single entry point (there's no main() function, for example).

Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

Activating Components

Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.

An intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component—an intent can be either explicit or implicit, respectively.

For activities and services, an intent defines the action to perform (for example, to "view" or "send" something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an Intent (for example, you can issue an intent to let the user pick a personal contact and have it returned to you—the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates "battery is low").

The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a ContentResolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the ContentResolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

There are separate methods for activating each type of component:



• You can start an activity (or give it something new to do) by passing an Intent to startActivity() or startActivityForResult() (when you want the activity to return a result).

You can start a service (or give new instructions to an ongoing service) by passing an Intent to startService(). Or you can bind to the service by passing an Intent tobindService().

You can initiate a broadcast by passing an Intent to methods like sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

You can perform a query to a content provider by calling query() on a ContentResolver.

For more information about using intents, see the Intents and Intent Filters document. More information about activating specific components is also provided in the following documents: Activities, Services, BroadcastReceiver and Content Providers.

Declaring components

The primary task of the manifest is to inform the system about the application's components. For example, a manifest file can declare an activity as follows:

In the <application> element, the android:icon attribute points to resources for an icon that identifies the application.

In the <activity> element, the android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attributes specifies a string to use as the user-visible label for the activity.

You must declare all application components this way:

<activity> elements for activities <service> elements for services <receiver> elements for broadcast receivers <provider> elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run.

However, broadcast receivers can be either declared in the manifest or created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling registerReceiver().

Declaring component capabilities

As discussed above, in Activating Components, you can use an Intent to start activities, services, and broadcast receivers. You can do so by explicitly naming the target component (using the component class name) in the intent. However, the real power of intents lies in the concept of intent actions. With intent actions, you simply describe the type of action you want to perform (and optionally, the data upon which you’d like to perform the action) and allow the system to find a component on the device that can perform the action and start it. If there are multiple components that can perform the action described by the intent, then the user selects which one to use.

The way the system identifies the components that can respond to an intent is by comparing the intent received to the intent filters provided in the manifest file of other applications on the device.

When you declare a component in your application's manifest, you can optionally include intent filters that declare the capabilities of the component so it can respond to intents from other applications. You can declare an intent filter for your component by adding an <intent-filter> element as a child of the component's declaration element.

For example, an email application with an activity for composing a new email might declare an intent filter in its manifest entry to respond to "send" intents (in order to send email). An activity in your application can then create an intent with the “send” action

(ACTION\_SEND), which the system matches to the email application’s “send” activity and launches it when you invoke the intent with startActivity().

For more about creating intent filters, see the Intents and Intent Filters document.

Declaring application requirements

There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your application from being installed on devices that lack features needed by your application, it's important that you clearly define a profile for the types of devices your application supports by declaring device and software requirements in your manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read them in order to provide filtering for users when they search for applications from their device.

For example, if your application requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file. That way, devices that do not have a camera and have an Android version lower than 2.1 cannot install your application from Google Play.

However, you can also declare that your application uses the camera, but does not require it. In that case, your application must perform a check at runtime to determine if the device has a camera and disable any features that use the camera if one is not available. Here are some of the important device characteristics that you should consider as you design and develop your application:

Screen size and density

In order to categorize devices by their screen type, Android defines two characteristics for each device: screen size (the physical dimensions of the screen) and screen density (the physical density of the pixels on the screen, or dpi—dots per inch). To simplify all the different types of screen configurations, the Android system generalizes them into select groups that make them easier to target.

The screen sizes are: small, normal, large, and extra large.

The screen densities are: low density, medium density, high density, and extra high density.

By default, your application is compatible with all screen sizes and densities, because the Android system makes the appropriate adjustments to your UI layout and image resources. However, you should create specialized layouts for certain screen sizes and provide specialized images for certain densities, using alternative layout resources, and by declaring in your manifest exactly which screen sizes your application supports with the <supports-screens> element.

For more information, see the Supporting Multiple Screens document.

Input configurations

Many devices provide a different type of user input mechanism, such as a hardware keyboard, a trackball, or a five-way navigation pad. If your application requires a particular kind of input hardware, then you should declare it in your manifest with the <uses-configuration> element. However, it is rare that an application should require a certain input configuration.

Device features

There are many hardware and software features that may or may not exist on a given Android-powered device, such as a camera, a light sensor, bluetooth, a certain version of OpenGL, or the fidelity of the touchscreen. You should never assume that a certain feature is available on all Android-powered devices (other than the availability of the standard Android library), so you should declare any features used by your application with the <uses-feature> element.

Platform Version

Different Android-powered devices often run different versions of the Android platform, such as Android 1.6 or Android 2.3. Each successive version often includes additional APIs not available in the previous version. In order to indicate which set of APIs are available, each platform version specifies an API Level (for example, Android 1.0 is API Level 1 and Android 2.3 is API Level 9). If you use any APIs that were added to the platform after version 1.0, you should declare the minimum API Level in which those APIs were introduced using the <uses-sdk> element. It's important that you declare all such requirements for your application, because, when you distribute your application on Google Play, the store uses these declarations to filter which applications are available on each device. As such, your application should be available only to devices that meet all your application requirements.

For more information about how Google Play filters applications based on these (and other) requirements, see the Filters on Google Play document.

Application Resources

An Android application is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the application. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using application resources makes it easy to update various characteristics of your application without modifying code and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as different languages and screen sizes).

For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your application code or from other resources defined in XML. For example, if your application contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo, which you can use to reference the image and insert it in your user interface.

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then, based on a language qualifier that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI. Android supports many different qualifiers for your alternative resources. The qualifier is a short string that you include in the name of your resource directories in order to define the device configuration for which those resources should be used. As another example, you should often create different layouts for your activities, depending on the device's screen orientation and size. For example, when the device screen is in portrait orientation (tall), you might want a layout with buttons to be vertical, but when the screen is in landscape orientation (wide), the buttons should be aligned horizontally. To change the layout depending on the orientation, you can define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation.

For more about the different kinds of resources you can include in your application and how to create alternative resources for various device configurations, see theApplication Resources developer guide.

安卓应用基础

在Java编程语言编写的Android应用程序的Android的SDK工具编译代码以及与任何数据和到一个Android的包，一个归档文件档案资源的.apk后缀，所有的在一个单一的代码.apk文件被认为是一个应用程序，是Android的文件，供电设备来安装应用程序。 一旦安装在设备上，每个Android应用程序的生命在它自己的安全沙箱：

而Android操作系统是一个多用户Linux系统中，每个应用程序是一个不同的用户。

默认情况下，每个应用程序的系统分配一个唯一的Linux用户ID（该ID仅用于由系统分配的未知应用程序），系统设置所有的应用程序中的文件权限，以便只有用户ID分配给该应用程序可以访问它们。

每个进程都有它自己的虚拟机（VM），因此应用程序的代码的运行与其他应用程序隔离。

默认情况下，每个应用程序运行在它自己的Linux进程。Android的启动过程时，应用程序的任何组件需要被执行，然后关闭该进程时，它不再需要或恢复时，系统必须为其他应用程序的内存。

这样一来，Android系统实现了最小特权原则，也就是说，每个应用程序，默认情况下，只能访问的组件，它需要做的工作，没有更多，这将创建一个非常安全的环境，使应用程序无法访问的，这就是它没有给予许可制度的部分。

但是，有一个应用程序的方法与其他应用程序和应用程序访问系统服务的数据：

这有可能为两个应用程序安排共享相同的Linux用户ID，在这种情况下，它们能够相互访问文件。为了节约使用相同的用户ID系统资源，应用程序还可以安排运行在相同的Linux进程和共享同一个VM（应用也必须使用相同的证书签名）。

应用程序可以请求访问权限，如用户的联系人，短信，可安装存储（SD卡），摄像头，蓝牙等设备的数据，所有应用程序的权限必须由用户在安装时授予。

应用程序组件(Application Components)

Android的核心功能之一就是一个应用程序可以使用其它应用程序的元素（如果那个应用程序允许的话）。比如说，如果你的应用程序需要一个图片卷动列 表，而另一个应用程序已经开发了一个合用的而又允许别人使用的话，你可以直接调用那个卷动列表来完成工作，而不用自己再开发一个。你的应用程序并没有吸纳 或链接其它应用程序的代码，它只是在有需求的时候启动了其它应用程序的那个功能部分。

为达到这个目的，系统必须在一个应用程序的一部分被需要时启动这个应用程序，并将那个部分的Java对象实例化。与在其它系统上的应用程序不同，Android应用程序没有为应用准备一个单独的程序入口（比如说，没有main()方法）， 而是为系统依照需求实例化提供了基本的组件。共有四种组件类型：

活动(Activities) 一个 activity 代表用户界面的一个独立屏幕。例如，一个邮件应用程序应该有一个 activity 用于显示新邮件列表，另一个 activity 用于撰写一封邮件，还有一个 activity 用于读取邮件。尽管所有 activitie 协同工作以构成邮件应用程序的用户体验，但彼此之间相对独立。应次，不同的应用程序能够从任何一个 activity 启动 (只要邮件应用程序允许)。例如，用户需要分享一张照片，一个拍照应用程序能够启动邮件应用程序的 activity 。activity 是一个实现了 Activity 的子类，你可以在 Activities 开发者指导部分了解更多。

服务(Services)

service 是在后台运行，执行长时间操作或者执行远程操作。 service 不提供用户界面。例如，当用户在另一个应用程序时，一个 service 可在后台播放音乐，或者是从网络上获取数据，而不阻断用户与当前 activity 的交互。其他组件，比如一个 activity ，为了与该 service 互动，可以启动或者 绑定它。 service 是一个实现了 Service 的子类，你可以在 Services 开发者指导部分了解更多。

广播接收器(Broadcast receivers)

广播接收器是一个专注于接收广播通知信息，并做出对应处理的组件。很多广播是源自于系统代码的──比如，通知时区改变、电池电量低、拍摄了一张照片或者用户改变了语言选项。应用程序也可以进行广播──比如说，通知其它应用程序一些数据下载完成并处于可用状态。

应用程序可以拥有任意数量的广播接收器以对所有它感兴趣的通知信息予以响应。所有的接收器均继承自BroadcastReceiver基类。

广播接收器没有用户界面。然而，它们可以启动一个activity来响应它们收到的信息，或者用NotificationManager来通知用户。通知可以用很多种方式来吸引用户的注意力──闪动背灯、震动、播放声音等等。一般来说是在状态栏上放一个持久的图标，用户可以打开它并获取消息。

内容提供者(Content providers)

内容提供者将一些特定的应用程序数据供给其它应用程序使用。数据可以存储于文件系统、SQLite数据库或其它方式。内容提供者继承于ContentProvider 基类，为其它应用程序取用和存储它管理的数据实现了一套标准方法。然而，应用程序并不直接调用这些方法，而是使用一个ContentResolver 对象，调用它的方法作为替代。ContentResolver可以与任意内容提供者进行会话，与其合作来对所有相关交互通讯进行管理。

参阅独立的内容提供者Content Providers 章节获得更多关于使用内容提供者的内容。

每当出现一个需要被特定组件处理的请求时，Android会确保那个组件的应用程序进程处于运行状态，或在必要的时候启动它。并确保那个相应组件的实例的存在，必要时会创建那个实例。

Android系统设计的一个独特方面是任何的一个程序都可以启动另一程序的组件。比如，你想让你的程序可以使用照相机拍照，如果已经有了实现这种功能的程序并且你你的程序能使用它（有权限），那么你就没有再要再写一个新的Activity来实现这个功能。你的程序不需要包含或者链接这个拍照程序。相反，你只需要在你的程序中打开这个拍照程序中的实现拍照功能的Activity。当拍完之后，拍好的照片甚至会自动返回给你的程序。者对于用户来说，就好像是想拍照功能的程序就是你的这个程序的一部分一样。

当系统启动一个组件之后，如果这个组件所在的程序之前没有运行的话，系统会自动开始这个程序的进程，并初始化这个组件所需要的相关类。比如，你的程序开启了一个拍照功能程序的Activity，这时系统会启动这个Activity所在的程序，所以这个Activity运行在拍照功能的程序当中，而不是在你的程序中。所以，不像其他操作系统的中的程序一样，Android程序没有一个单独的入口点（比如没有我们常见的main()函数）。 因为系统中的程序运行在自己的独立进程中，并且程序中的文件都有自己的限制其他程序访问的权限，所以，你的程序不能直接激活其他程序中的组件。但是Android系统就可以。具体是这样的实现的，为了激活（activate）其他程序中的组件，你必须向系统发送一个消息来详细说明你要启动其他组件的意图，这样系统才会为你激活这个组件。

激活组件(Activating Components)

四大组件中的三个组件——activities、services和broadcast receiver——是由一种叫intent的异步消息来激活的。这些intents在运行时（runtime）将这些属于你的程序或不同程序的单独的组件绑定在一起（bind），你可以把这些intents看作是需要其他组件的action的messengers。

一个intent就是一个Intent对象，这个intent定义了一种可以激活（activate）某个特定组件或者某种特定类型的组件，这两种情况分别对应两种intent的定义方式或者显示的或者隐式的。

对于activities和services，一个intent定义了要执行的操作（action）(比如，要“view”或者“send”什么)和要操作的数据的URI。比如，一个intent可能会为一个activity传递一个请求来展示一张图片或者打开一个网页。有时，你可以启动一个activity来得到返回的结果，在这个例子中这个activity的返回的结果也是一个Intent（比如，你可以发送一个intent让用户选择一个personal contact并返回给你——这个返回的intent就包含了一个指向用户选择的联系人的URI）。（关于activity和service的启动方式，下面将介绍。） 对于广播接收者来说，intent只是简单的定义了要广播的内容（比如，一个用以表明电池电量很低的广播仅包含了一个表明电池电量很低的字符串）。

最后一种组件类型content provider并不是由intent来激活的（activate）。而是由接收到ContentResolver的请求时激活的。

它们都各自有自己的方法来激活相应的组件：

你可以通过传递一个Intent给startActivity()或startActivityForResult()启动一个activity（或者给他一些新的要做的内容）。使用startActivityForResult()你将得到一个返回结果。

你可以通过传递一个Intent给startService()来start一个service（或者给一个正在运行的service一些新的指令(instructions)）。或者你可以通过把一个Intent传递给bindService()来绑定一个service。

你可以通过传递一个Intent给诸如sendBroadcast()、sendOrderedBroadcast()或者sendStickyBroadcast()等方法来初始化一个广播。

你可以通过调用ContentResolver的query()方法来执行一次content provider的查询操作。

更多的关于intent的内容，可以参看文档中的Intents and Intent Filters。更多的关于激活特定组件的内容可以参看文档中的：Activities、Services、BroadcastReceiver、Content Providers。

关于Manifest文件

在Android系统可以启动一个应用程序组件之前，Android系统必须通过读取这个程序的AndroidManifest.xml（即manifest文件）文件来确定要启动的组件存在。你的程序必须在这个manifest文件声明用到的所有的组件，并且这个manifest文件必须在项目的根目录下。

另外，这个manifest文件还声明一些其他的东西，比如： λ 确定这个程序需要的所有权限，比如Internet访问权限或者读取用户联系人权限。

声明这个运行这个程序所需要的最低API版本，这个可以根据开发该程序所使用的API版本。

声明该程序所需要的硬件或软件特征（features），比如照相机、蓝牙服务或者多点触屏。

声明该程序需要链接（link against）的API库（不是Andorid的framework APIs），比如Google Maps library等等。

组件声明

Manifest文件的首要任务就是通知系统关于程序中要使用的组件。比如，一个manifest文件可以用如下的方式来声明一个activity：



在<application>元素中，android:icon属性用于指定一个用于标示该程序的icon。 在<activity>元素中，android:name属性用于确定这个扩展自Activity的子类的全路径名，android:label属性用于标示这个activity的对于用户可见的label。

你必须要用以下方式来声明你的程序组件：

activities：<activity>标签

services:<service>标签

broadcast receiver：<receiver>标签

content providers：<provider>标签 如果程序中用到activities、services和content providers，你没有在manifest文件中声明，那么这些组件将不会被系统知道，结果就是你的程序不能运行。然而，broadcast receiver既可以在manifest文件中声明也可以在代码中动态创建（BroadcastReceiver），并通过调用registerReceiver()在系统中注册。

更多关于怎样为你的程序构建manifest文件，请参看文档The AndroidManifes.xml文件。

声明组件的能力

正如在上面Activating Components中讨论的那样，你可以使用一个Intent来启动activities、services和broadcast receiver。你可以通过在intent中注明目标组件的名字（使用的是组件的类名）来显示的启动组件。然而，intents真正强大的地方在与关于intent的actions的概念。通过intent的actions，你可以简单的描述你要执行的操作的类型（并且可以有选择的描述你要处理的数据），可以允许系统在device中找到这个组件并启动它。如果有多个组件可以执行intent中描述的action，这时用户就可以选择一个来执行。

系统可以识别能对某intent做出反应的方式是通过将接收到的intent和设备中其他程序的manifest文件的intent filters进行比较实现的。

当你在程序的manifest文件中声明一个组件之后，你可以有选择包含intent filters，这些intent filters表明了组件对接收自其他程序的intent做出反应的能力（capabilities）。你可以通过添加一个<intent-filter>元素作为a child of the component's declaration element来为你的程序声明一个intent filter。

比如，在一个邮件程序中的一个activity可以编写新的邮件，这样的话你就需要在manifest文件中来声明一个intent filter来对“发送”intent响应（为了发送邮件）。这样，在你的程序中，一个activity就可以创建一个发送intent(ACTION\_SEND)，这样当你调用startaActivity()时，系统就会匹配邮件程序中的发送activity并启动它。

更多关于创建intent filters的内容，可以参看Intents and Intent Filter文档。 声明运行程序所需的条件

Andorid系统可以支持很多不同的设备，并且这些设备的性能特征并不相同。为了防止你的程序被安装在不能正常运行你的程序的较低android系统版本上，通过在manifest文件中声明你的程序支持的设备和软件，便变得尤其重要起来。大多数的这些声明仅是一些信息，而系统并不会读取它们，但是其他的服务比如Android Market却会阅读这些声明来帮助通过通过自己的设备搜索软件的用户过滤软件。

比如，你的程序需要照相机，并且使用的Android2.1的APIs，那么你就必须在你的manifest文件中声明这些需要。这样的话，在Android Market上，没有照相机或者Android系统版本低于2.1将不能安装你的程序。

然而，如果你的程序不需要照相机，你仍可以声明你需要照相机。这种情况下，你的程序必须在运行时做一下检查，来检查这个设备是否含有照相机，如果没有照相机可用，则系统将会使使用照相机的相关程序不能用。

下面是一些你在设计和开发你的程序时，必须要考虑的关于设备的一些重要方面： 屏幕大小和分辨率：

为了根据屏幕的类型进行分类，Android定义了两个特征：屏幕大小和分辨率。 屏幕尺寸有： 小，中，大，超大； 屏幕分辨率类型：低分辨率，中分辨率，高分辨率，超高分辨率； 默认情况下，你的程序可以兼容所有的屏幕尺寸和分辨率，因为Android系统对你的程序的UI布局和image资源做了适当的调整。

输入方式：

很多设备有不同类型的输入方式，比如键盘、轨迹球、五位元导航。如果你的程序需要某特定形式的输入方式，则你必须在manifes文件中使用<uses-configuration>标签来声明。不过这种情况是比较少的。

设备配置：

有许多硬件或软件并不全在Android系统的设备上，比如，一个照相机、光线传感器、某个版本的 OpenGL，或者屏幕的保真度（fidelity）。你在任何条件下都不能假定Android设备具备某种特性（feature）（当然得除掉Android标准库的情况），所以如果你的程序使用了某feature，则你必须使用<uses-feature>标签来声明。

不同地 Android 平台设备通常运行不同版本的 Android ，比如 Android 1.6 或者 Android 2.3。每个后续版本通常包含之前版本所不支持的新增 API。 In order to indicate which set of APIs are available, 每个平台版本对应一个例如， Android 1.0 对应于 API Level 1 ， Android 2.3 对应于 API Level 9)。如果你使用任何在 1.0 版之后平台新增的 API，你应该使用 <uses-sdk> 元素声明最低 API Level 是包含这些 API的。 为你的应用程序声明所有这些要求至关重要，因为，当你在 Android Market 上发布你的应用程序时，Market 使用这些声明来过滤该应用程序是否对于每台设备可用。 这样，你的应用程序仅对能够满足你的应用程序要求的设备可用。

应用程序资源

一个应用程序不仅仅由代码组成——它需要区别于源代码的资源，比如图片，音频文件，以及任何与应用程序视觉呈现相关联的内容。例如，你应该使用 XML 文件定义

动画，菜单，风格，颜色，以及 activity 用户界面的布局。 使用应用程序资源文件，可以更容易地更新你的应用程序的特性而无需修改代码，并且 — 通过提供多套可替换资源文件 — 使您能够针对各种设备配置优化你的应用程序(比如不同语言或屏幕大小)。

对于你的安卓工程里面包含的每一项资源， SDK 构建工具定义一个唯一的正整数 ID 标识符，你可以使用该标识符从你的应用程序代码中或者从XML文件中定义的其他资源中特指该资源。例如，如果你的应用程序中包含一个名为 logo.png 图片文件(保存在 res/drawable/ 文件夹里)， SDK 工具会生成一个资源 ID 命名为 R.drawable.logo，你可以使用该 ID 特指这张图片并插入你的用户界面中。

将资源提供工作同你的源代码分隔开来最重要的原因之一是能够使您为不同的设备配置提供可替换的资源文件。例如，在 XML 中定义 UI 字符串，你可以将这些字符串翻译成其他语言并保存在特定的文件夹中。然后，基于语言 qualifier / 修饰词 你添加资源文件夹名称 (比如 res/values-fr/ 对应于法语字符串) 以及用户语言设置， Android 系统会给你的 UI 提供适当的语言字符串。

对于你的可替代资源，Android 支持许多不同的 qualifiers / 修饰符 。修饰符是包含在你的资源文件夹名称中的一个短字符串，以便界定哪些设备配置可使用这些资源。 另一个例子，对于不同的设备屏幕和大小，你应当为你的 activities 创建不同的布局。 例如，当设备屏幕是纵向的 (高)，你可能希望一个按钮垂直排列的布局，但当屏幕是横向的(宽)，按钮应当水平排列。为了根据方向调整布局，你可以定义两个不同的布局文件并给每个布局文件夹提供适当的修饰符。这样，系统会根据特定的设备方向自动为其提供适当的布局。