**河 南 科 技 学 院**

**2018 届本科毕业论文（设计）**

**论文（设计）题目: 租房管理系统设计与实现**

**学生姓名： 黄刚**

**所在学院： 信息工程学院**

**所学专业： 计算机科学与技术**

**导师姓名： 金松林**

**完成时间：** **2018**­­­­‑‑‑**-05-02**

**租房管理系统设计与实现**

**摘 要**

**自互联网发展以来，许多产品由线下走到了线上，这非常符合国家总理李克强提出的“互联网+”理念。促进了互联网的产物与传统的事物进行有机的结合，这不仅节省了时间、空间、物力还有效的利用了人们的业余时间，大大促进了生产力的发展和提高。在此基础上网上租房系统App应运而生，它省去了中介使租客直接与房东进行沟通和交流。**

**针对目前的租房情况展开了对房东和租客的深入调查和分析，确定了租房App的基本功能和内部构造，对此进行划分出了详细的功能模块。前段分为四大模块为：房源信息浏览，发布房源，房源资讯，用户资料。后端采用对数据进行逻辑处理，推送发布房源、房源展示、房源订阅、房源收藏、用户资料收集与处理。采用SQL数据对数据进行处理，使用Objective-C，Swift语言。使用了ReactiveCocoa，ReactiveObjc，Snapkit，ObjectMapper，JSONModel，TZImagePickerController，Masonry第三方类库。涉及使用的设计模式有KVO，KVP，MVC，MVVM，代理，工厂等设计模式。经测试，达到了租房App的预定模块的功能，结构优秀，耦合性弱以扩张。**

**关键词：功能模块，数据处理，设计模式，性能**

**Design AND IMPLEMENTATION OF RENTAL MANAGEMENT SYSTEM**

**Abstract**

**Since the development of the Internet, many products have been online, which is in line with the "Internet plus" concept proposed by premier Li KeQiang. Promote the product of the Internet and the traditional organic combination of things, this not only saves time, space, material and effective use of people's spare time, greatly promoted the development of the productive forces and improving. On this basis, the online rental system App came into being, which eliminated the middleman and allowed the tenant to communicate with the landlord directly.**

**In view of the current rental situation, we have conducted in-depth investigation and analysis of landlords and tenants, and determined the basic functions and internal structure of the App, which has been divided into detailed functional modules. The previous paragraph is divided into four modules: room source information browsing, publishing house source, housing information, user data. The back-end adopts the logical processing of the data, pushing and publishing house source, house source display, room source subscription, house source collection, user data collection and processing. SQL data is used to process the data, using objective-c, the Swift language. Using ReactiveCocoa, ReactiveObjc, Snapkit, ObjectMapper, JSONModel, TZImagePickerController, Masonry third-party library. The design patterns involved are KVO, KVP, MVC, MVVM, agent, factory and other design patterns. After testing, the function of the booking module of renting App was reached, the structure was excellent, and the coupling was weak to expand.**

**Key words: functional module, data processing, design pattern, performance.**

**目 录**

[1 引言 1](#_Toc481425027)

[1.1 课题背景 1](#_Toc481425028)

[1.2 国内外研究现状 1](#_Toc481425029)

[1.3 开发的意义 1](#_Toc481425030)

[2 需求分析 1](#_Toc481425031)

[2.1 功能分析 1](#_Toc481425032)

[2.2 技术可行性分析 2](#_Toc481425033)

[3 概要设计 2](#_Toc481425034)

[3.1 功能模块划分 2](#_Toc481425035)

[3.1.1 前端功能模块 3](#_Toc481425036)

[3.1.2 后端功能模块 4](#_Toc481425037)

[3.2 数据库设计 4](#_Toc481425038)

[3.2.1 概念设计 4](#_Toc481425039)

[3.2.2 逻辑结构设计 5](#_Toc481425040)

[4 详细设计 8](#_Toc481425041)

[4.1 结构设计思想 8](#_Toc481425042)

[4.2 关键功能模块的设计与实现 8](#_Toc481425043)

[4.2.1 登录模块 8](#_Toc481425044)

[4.2.2 注册模块 9](#_Toc481425045)

[4.2.3 个人中心模块 10](#_Toc481425046)

[4.2.4 私教模块 11](#_Toc481425047)

[4.2.5 健吧模块 11](#_Toc481425048)

[4.2.6 订阅私教模块 12](#_Toc481425049)

[4.2.7 定制计划模块 13](#_Toc481425050)

[4.2.8 我的分享模块 14](#_Toc481425051)

[4.2.9 文章推送模块 15](#_Toc481425052)

[4.2.10 教练管理模块 15](#_Toc481425053)

[5 测试 16](#_Toc481425054)

[5.1 前台功能测试 16](#_Toc481425055)

[5.2 后台功能测试 19](#_Toc481425056)

[6 总结 20](#_Toc481425057)

[参考文献 21](#_Toc481425058)

[致 谢 22](#_Toc481425059)

# 引言

## 课题背景

紧随互联网的大潮，互联网带动了各行各业的发展和变化，许多产品从线下走到了线上摆到了人们的面前，增加了人们更多的选择性。

租房App也是在这种背景下产生的，传统的租房是租客通过中介或者传单获得租住房子的信息，绕过了对房子最熟悉的人——房东。通过这款App，租客和房东可以端对端的进行沟通和了解。租客通过这款App可以实时实地的对租住房子进行了解和考察，对房子有更全面的了解增加房源信息的透明度。而房东通过App可以详细介绍自己的房源情况和价格，发布房源以后房东可以自由支配自己的时间，增加了办事效率。

## 国内外研究现状

在国外，自2008美国楼市引起的次贷危机爆发，虽然大受打击，但是这些年正在不断的由寒转暖。民意调查机构皮尤研究中心(Pew Reasearch Center)最新分析发现，当前美国住在租来的房子比1965年的还要多，2006年租房家庭的比例从31%到2016攀升到38%。而且由于买房价格普遍偏高，人们更愿意租房住。

目前在国内租房市场还是挺好的，最近几年我国房价一直在攀升的进行时中，如，一线城市的北京，上海，广州，深圳，高昂的房价，工作的变动，孩子的教育等，使许多准备买房的望而却步。据中国的租赁报告显示，我国目前约有30%左右的90后大学生在毕业后，人均租住在面积不足25平方米的房间，租住的方式主要通过合租、合拼的方式。从增长规模看，去年我国达到1.603亿人租房，预测到2020年将达到2亿左右，2030年大概增长到2.74亿人成为租客。而且这个租赁市场规模也在不断扩大，截止到2020年可达1.67万亿元，2030年可达到4万亿元。

## 开发的意义

根据以上的国内和国外的用户需求，发现租房市场的空间是巨大的。以后可能会有更多的人将自己多余的房间租住出去，而且由于中国城市流动人口很大，会有更多的人租住房子。根据以上的情况设计了这款App，快捷下载到手机。对于租客可以随时快速找到适合自己的住房，免去了中介费用。对于房东，扩大了自己的房间租出的可能性概率，有效的利用了空间和时间。

# 需求分析

## 功能分析

通过对现有租房App的研究和分析，结合用户的需求、习惯和技术方面的支持，确定了一下的逻辑功能。

1. 注册账号。用户选择注册，填入必要信息字段，即可完成注册。
2. 登录用户。由用户输入手机号和密码，进行确认登录。
3. 发布房源。用户需要完成注册或者登录后进行房源信息的发布。
4. 资讯。 用户通过阅读推送的信息，了解最近的房市变化信息。
5. 查看房源。通过发布的房源找到自己感兴趣的房源，进行筛选和比较。
6. 收藏房源。对于自己喜欢的房源，可以进行收藏，以便以后查阅。
7. 订阅房源。用户根据条件选择自己需要订阅的房源信息。
   1. 相关技术分析

2.1 开发语言

Objective-C(以下简称为OC)是C语言的扩展，它的设计借用了有“面向对象编程之母”称呼的Smalltalk语言的思想。主要用于编写iOS操作系统(如，iPhone、iPad、iPod touch 等苹果的终端设备)的应用程序和Mac OS操作系统。

OC 管理内存的方式不同于Java，采用的是ARC(自动引用计数)机制，抛弃了以前使用的MRC(手动引用计数)，提高了开发效率，不用程序员手动释放内存。

OC 管理线程使用NSThread、NSOperation、GCD。NSThread是原始的线程，量级轻，相对简单，但是需要手动管理线程的活动。如，生命周期，同步，睡眠。NSOperation自带周期管理，更注重自己逻辑，它是面向对象的抽象类。GCD是Apple并发多核编程的线程类，高效，可避开并发陷阱。

此外，OC 不支持多继承，它是动态类型的，主要是因为Runtime的存在。Runtime底层是一套纯C(C和汇编)的API，处理静态语言的编译和链接，造就了OC的运行时的机制，可以使类进行动态的绑定、加载，增加了灵活度。因为Runtime的存在，导致了OC根本不存在私有方法。因为通过Runtime可以随时取到OC对象的方法和属性，并对其进行添加和删除，所以OC是不安全的。

所以在2014年的WWDC大会提出了Swift语言，可与Objective-C兼容运行在Mac OS，iOS，watchOS和tvOS设备上。Swift采用了安全的编程模式使得Swift在运行时必须要求类型确定，这是与OC的最大区别。Swift对于初学者来说来学习是非常易学的，它是第一个不仅满足工业标准，还是一个像脚本语言充满表现力和趣味的系统编程语言。

* 1. 开发工具

租房App在开发过程中用到的开发工具包括：Xcode，SourceTree，Datum Free。

Xcode是一款运行在Mac OS操作系统上的集成开发工具，功能强大。可以完成对Mac，tvOS, watchOs，iPhone的软件开发，提供了许多项目创建的模板。自带检测工具Insruments，可以对程序中的漏洞进行检测。

SourceTree是一款项目管理的git图形界面工具，减少终端命令提交代码的复杂难度，实时检测项目的完成度。提高了开发的效率。

Datum Free 是一款SQLite图形界面工具，可以用它进行各种表的操作，使用它可以有效检测数据表单的操作情况。

* 1. 数据管理

数据库的管理使用的是SQLite，它是一款轻型的数据管理库，遵守ACID关系型数据管理库系统。支持现在市场普遍的操作系统。如，Windows、Linux，Unix，macOS等操作系统。

在iOS开发中，对SQLite的调用使用的是iOS原生sqlite3框架。使用时只需要将其框架导入，然后调用方法执行SQL原生语句。

图片等较大文件存放在程序的沙盒目录下，数据库存储其存放地址，减少了数据库的容量和负担。

# 概要设计

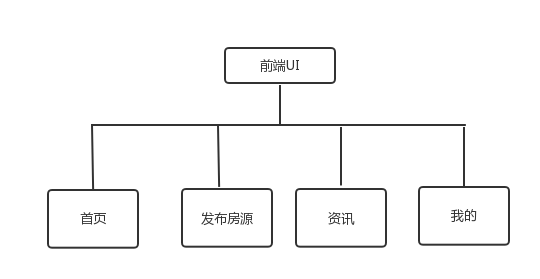
* 1. 硬件支持

该租房App只支持iPhone，系统支持iOS 10.3以上。

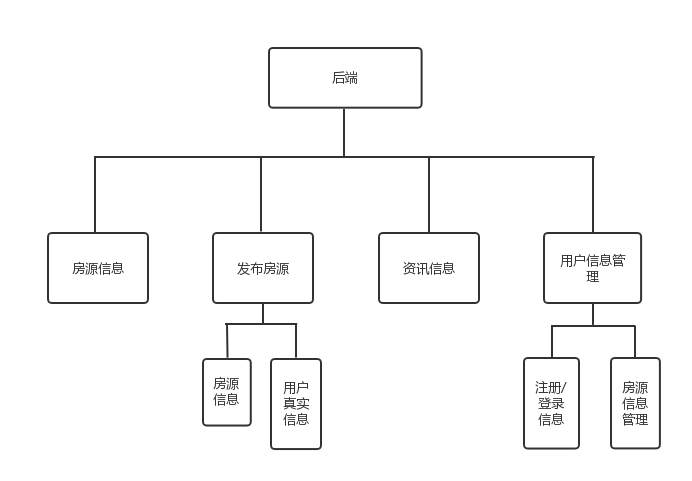
注意：以下对该系统的的说明，统称为“租房App”。

## 功能模块划分

根据用户实际需求和市场需要，租房App分为前端UI和后端数据处理两大模块，前端UI如3-1图，后端如图3-2图：



**图3-1 前端UI功能模块图**

****

**图3-2 后端功能模块图**

### 前端功能模块

1. 首页

用户可以查看他人发布的房源基本信息，呈现形式以列表列出，点击击进去，即可查看房源的详细信息。可以对房源进行收藏、订阅、预约。

1. 发布房源

用户进行登录和注册后方能进行此项功能能，否则不会对用户开放。用户根据自己的房源信息进行填写，然后填写自己的个人真实信息即可发布自己的房源。

1. 资讯

系统对客户端推送最近有关房市信息的资讯，使用户对最近的房市有一定的了解和认知。

1. 我的
2. 登录

用户通过输入手机号和密码登录租房App应用，查看自己使用的痕迹。可以查看自己的基本信息，收藏房源，订阅房源，已发布的房源等信息。

1. 注册

用户通过输入昵称、手机号、密码进行租房App应用的注册，解锁使用该App的功能能。

1. 发布房源

用户注册后，可以发布自己的房源信息。

1. 收藏房源

用户注册后，可以对其他人发布的房源进行订阅。

1. 房源订阅

用户注册后，根据他人发布的房源进行筛选，选择适合自己条件的房源。

### 后端数据处理功能模块

1. 房源信息

从数据库中取出发布信息的数据，将其加入到一个NSMutableArray的数组中，在首页中将其数据通过MVVM设计模式展现在UITableView中。

点击某一房源进入详情页，是通过UITableView的属性NSIndex属性得出数组中的数据元素，再从数组中取出Model，将其展示在UI界面。

1. 发布房源

进入发布房源功能模块，填写房源的基本信息，将这些房源信息字段存放在一个对象的子段属性中，点击发布时还要填写发布房源人的真实信息，将其存放在SQLite数据库中。

1. 资讯信息

由该App获取网上优秀的房市文章，用WKWebView通过网址显示资讯信息。

1. 用户信息管理

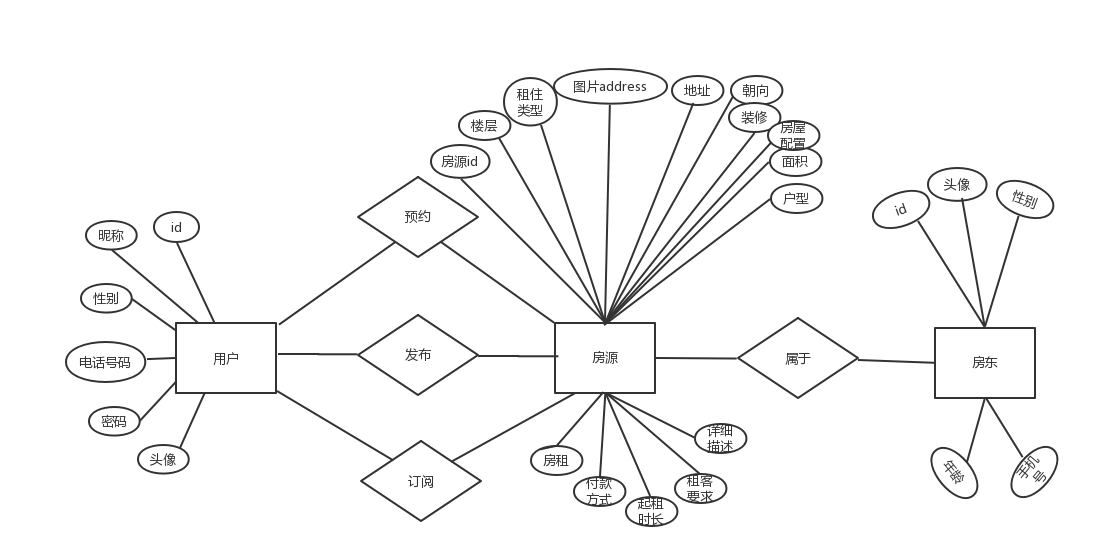
户如果没有注册，需要先注册填写基本的信息，然后才能使用租房App的一些基本功能。为了方便管理用户的信息，将用户的信息封装为一个单例类用归档对其基本信息字段进行存储，同时将其存放在数据库中。

对于收藏房源，预约清单，房源订阅功能，将其放在SQLite数据库中，根据其ID获取其信息。

## 数据库设计

### 概念设计

数据库的E-R图，见图3-3所示。



**图3-3 E-R图**

### 逻辑结构设计

租房App的SQLite数据表是经过市场需求、用户体验，经过不断在实践操作中创建的。在一个完整的项目工程中，数据库的创建和逻辑关系会影响App的用户体验，以及影响以后项目的扩展和优化。所以对于数据库的各个方面都有严格的要求，如，数据库的表明，字段名要做到字如其意(开发者),SQLite的数据库类型严格按照需求，以免造成内存的浪费。

(1) 用户(user)表

用户表（user）用来存储租房用户的基本信息，其表的类型和结构，见下表的3-3-1：

**表3-3-1 用户表(user)**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| userId | Text | Yes | 用户Id |
| nickname | Text | No | 用户名字 |
| phoneNumber | Text | No | 用户密码 |
| password | Text | No | 用户订阅的私教 |
| sex | Text | No | 健身计划序号 |
| avatarPath | Text | No | 用户打卡次数 |

* (2) 房源信息(housingResource)表

房源信息表(housingResource)是租客用户根据自己的房源情况填写房源细节而发布的房源信息，数据字段详细记录了要发的房源信息，见表3-3-2：

**表**3-3-2 **房源信息表(housingResource)**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| houseResourceId | Text | Yes | 房源id |
| userId | Text | No | 用户id |
| rentType | Text | No | 租住类型 |
| housePictures | Text | No | 房源图片 |
| location | Text | No | 地理位置 |
| floor | Text | No | 楼层/总楼层 |
| houseType | Text | No | 户型 |
| orientation | Text | No | 朝向 |
| decorated | Text | No | 装修 |
| area | Text | No | 面积 |
| houseConfiguration | Text | No | 房屋配置 |
| rent | Text | No | 房租 |
| paymentWay | Text | No | 付款方式 |
| onHireDuration | Text | No | 起租时长 |
| renterRequiere | Text | No | 对租客要求 |
| detailDescribe | Text | No | 详细描述 |

* (3) 收藏房源(collectHouseResource)表

收藏房源(collectHouseResource)收藏的是用户喜欢的房源，存储了找到房源的一些基本信息，见表3-3-3：

**表3-3-3 收藏房源表(collectHouseResource)**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| userId | Text | No | 用户Id |
| houseResourceId | Text | Yes | 房源Id |

## (4) 预约房源(precontractHouseResource)

预约房源信息表是用户对预租房源信息的标记，用来保存了解房源的一些必要信息，见表3-3-4所示：

**表3-3-4 预约房源(precontractHouseResource)**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| userId | Text | No | 用户id |
| houseResourceId | Text | Yes | 房源id |
| precontractTime | Integer | No | 预约时间 |
| isPrecontract | Integer | No | 是否预约 |

(5) 房东(landlord)表

存储房源所属户主的信息，存储了房东的一些必要信息，见表3-3-5：

**表3-3-5 房东(landlord)表**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| landlordId | Text | Yes | 房东id |
| avatarPath | Text | No | 房东头像 |
| name | Text | No | 房东名字 |
| sex | Integer | No | 性别 |
| age | Text | No | 年龄 |

(6)资讯(message)表

我的分享表(pushlish)用来存储用户分享的数据，当该用户的打卡记录在前七名中，取出最近一条存储的数据到minduser表中。表结构如表3.6所示。

**表3.6 我的分享表**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| Id | Int(20) | 是 | 励志分享Id |
| userId | Int(20) | 否 | 励志用户Id |
| PushText | Varchar(255) | 否 | 励志用户分享文字 |
| pushImg1 | Varchar(255) | 否 | 励志用户分享Img |
| pushImg2 | Varchar(255) | 否 | 励志用户分享Img |
| pushImg3 | Varchar(255) | 否 | 励志用户分享Img |
| pushImg4 | Varchar(255) | 否 | 励志用户分享Img |

(7) 文章内容表

文章内容表(article)用来存储健康和饮食的文章数据。表结构如表3.7所示。

**表3.7 文章内容表**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| Id | Int(11) | 是 | 文章Id |
| Type | Int(11) | 否 | 文章类型 |
| Cap | char(100) | 否 | 文章标题 |
| Content | text(200) | 否 | 文章内容 |
| Create\_time | Bigint(20) | 否 | 文章发表时间 |

(8) 管理员表

管理员表(admin)用来保存健身APP管理员的基本信息。表结构如表3.8所示。

**表3.8 管理员表**

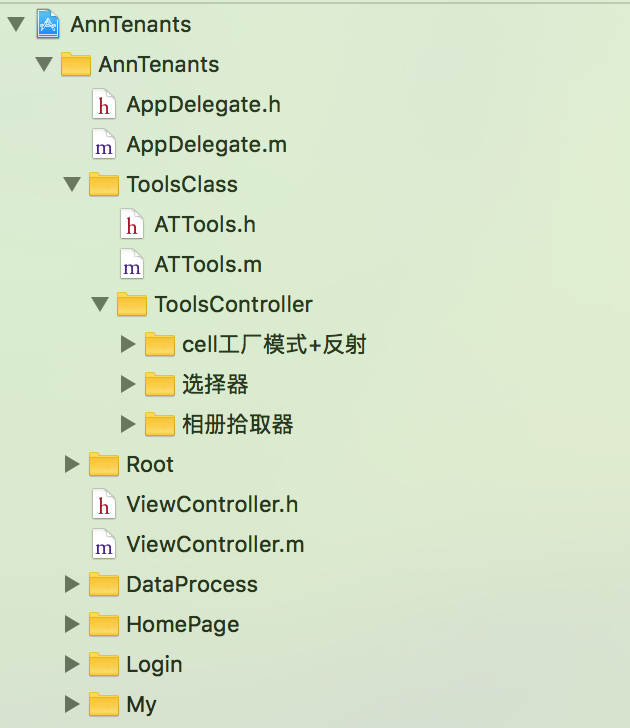
|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否主键 | 说明 |
| Id | Int(11) | 是 | 管理员Id |
| account | char(29) | 否 | 管理员名称 |
| Password | char(60) | 否 | 管理员密码 |

# 详细设计

## 结构设计与实现

租房App是一款运行在iPhone设备上的App，主要以Objective-C开发语言为主，还有少量的Swift开发语言的App。程序主要采用的是MVVM设计模式，MVVM设计模式比MVC更好。它将MVC中的C模块中的逻辑模块抽取出来了，形成VM模块，是C模块不会那么臃肿，降低了耦合度。使节构调理更加清晰，便于以后的功能扩展和优化，以及后来的单元测试。为了达到这种效果，还采用了MVC,单例，代理，依赖注入，工厂模式。

下面是租房App的文件结构图:



借用了第三方类库，如：

ReactiveCocoa，ReactiveObjC 分别是OC版，Swift版的响应式编程库；

Masonry，Snapkit 分别是OC版，Swift版的约束库，是其UI界面兼容各种屏幕尺寸的设备；

ObjectMapper，JSONModel 分别是OC版，Swift版的json数据解析库；

TZImagePickerController OC版的相机，图片选择器。

## 关键功能模块的设计与实现

### 工具类模块

1. 模块简介

开发一款App时，要想提高自己的开发效率，首先要提高代码的利用率，也就是造轮子，使这段代码可以重复的利用。工具采用了很多的类防法，避免了初始化对象时内存的浪费。类方法中封装了一个个具体的功能。

1. 模块功能实现

工具类模块放在ToosClass文件夹中，包括处理数据的ATTools类，生成UI控件的ATRootCell、ATPicker、ATPickerController。ATTools类中，将其功能封装成一个个类方法，省去了初始化对象，直接对传来的数据加工成想要的类型。UI控件类可以直接初始化自己想要的控件，分别生成自己想要的Cell，自定义拾取器控件，图片选择(包含拍照)，大大提高代码的复用率。

1. 核心代码实现
2. 用于生成数据库中的ID号

+ (NSString \*) createUniqueUUID {

CFUUIDRef uuid\_ref = CFUUIDCreate(NULL);

CFStringRef uuid\_string\_ref= CFUUIDCreateString(NULL, uuid\_ref);

NSString \*uuid = [NSString stringWithString:(\_\_bridge NSString \*)uuid\_string\_ref];

CFRelease(uuid\_ref);

CFRelease(uuid\_string\_ref);

return [uuid lowercaseString];

}

1. 对颜色字符串进行过滤处理，生成颜色对象

+ (UIColor \*)colorWithHexString:(NSString \*)color

{

NSString \*cString = [[color stringByTrimmingCharactersInSet:[NSCharacterSet whitespaceAndNewlineCharacterSet]] uppercaseString];

if ([cString length] < 6) {

return [UIColor clearColor];

}

if ([cString hasPrefix:@"0X"])

cString = [cString substringFromIndex:2];

if ([cString hasPrefix:@"#"])

cString = [cString substringFromIndex:1];

if ([cString length] != 6)

return [UIColor clearColor];

NSRange range;

range.location = 0;

range.length = 2;

NSString \*rString = [cString substringWithRange:range];

range.location = 2;

NSString \*gString = [cString substringWithRange:range];

range.location = 4;

NSString \*bString = [cString substringWithRange:range];

unsigned int r, g, b;

[[NSScanner scannerWithString:rString] scanHexInt:&r];

[[NSScanner scannerWithString:gString] scanHexInt:&g];

[[NSScanner scannerWithString:bString] scanHexInt:&b];

return [UIColor colorWithRed:((float) r / 255.0f) green:((float) g / 255.0f) blue:((float) b / 255.0f) alpha:1.0f];

}

1. 创建沙盒目录路径

//创建目录

+ (void) createFileDirectory:(NSString \*) directoryname {

NSArray \*paths =NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,NSUserDomainMask,YES);

NSString \* path = [paths[0]stringByAppendingPathComponent:directoryname];

if (![[NSFileManager defaultManager] fileExistsAtPath:path]){//推断createPath路径目录是否已存在。此处createPath为须要新建的目录的绝对路径

[[NSFileManager defaultManager] createDirectoryAtPath:path withIntermediateDirectories:YES attributes:nil error:nil];//创建目录

}

}

1. 将图片的二进制写入文件中

+ (NSString \*)saveImageWithName:(NSString \*)name withImage:(UIImage \*) currentImage {

NSData \*imageData = UIImageJPEGRepresentation(currentImage, 0.8);

NSString \*imageName = [NSString stringWithFormat:@"/ATPicture\_%0.f.png", [[NSDate date] timeIntervalSince1970]];

NSString \*imagePath = [NSString stringWithFormat:@"%@%@",ATDocumentPath,imageName];

BOOL isWrite = [imageData writeToFile:imagePath atomically:NO];

if (!isWrite) {

ATLog(@"图片未写入成功");

}

return imageName;

}

1. 相机权限的检查

//检查相机权限和相册权限

- (BOOL) checkCameraAuthority {

AVAuthorizationStatus authStatus = [AVCaptureDevice authorizationStatusForMediaType:AVMediaTypeVideo];

//iOS7之前都可以访问相机，iOS7之后访问相机有权限设置

if ((authStatus == AVAuthorizationStatusRestricted || authStatus == AVAuthorizationStatusDenied) && iOS7Later) {

// 无相机权限 做一个友好的提示

if (iOS8Later) {

[self showAlertControllerWithTitle:@"无法使用相机" message:@"请在iPhone的""设置-隐私-相机""中允许访问相机"];

} else {

[self showAlertControllerWithTitle:@"无法使用相机" message:@"请在iPhone的""设置-隐私-相机""中允许访问相机"];

}

} else if (authStatus == AVAuthorizationStatusNotDetermined) {

// fix issue 466, 防止用户首次拍照拒绝授权时相机页黑屏

if (iOS7Later) {

[AVCaptureDevice requestAccessForMediaType:AVMediaTypeVideo completionHandler:^(BOOL granted) {

if (granted) { //点击允许访问时调用

dispatch\_async(dispatch\_get\_main\_queue(), ^{

[self cameraAction];

});

}

}];

} else {

[self cameraAction];

}

// 拍照之前还需要检查相册权限

} else if ([TZImageManager authorizationStatus] == 2) { // 已被拒绝，没有相册权限，将无法保存拍的照片

if (iOS8Later) {

[self showAlertControllerWithTitle:@"无法访问相册" message:@"请在iPhone的""设置-隐私-相册""中允许访问相册"];

} else {

[self showAlertControllerWithTitle:@"无法访问相册" message:@"请在iPhone的""设置-隐私-相册""中允许访问相册"];

}

} else if ([TZImageManager authorizationStatus] == 0) { // 未请求过相册权限

[[TZImageManager manager] requestAuthorizationWithCompletion:^{

[self cameraAction];

}];

} else {

return YES;

}

return FALSE;

}

1. 相机保存图片，返回选择图片数组

#pragma mark - UIImagePickerControllerDelegate

//相机获取图片的操作

- (void)imagePickerController:(UIImagePickerController\*)picker didFinishPickingMediaWithInfo:(NSDictionary \*)info {

[picker dismissViewControllerAnimated:YES completion:nil];

NSString \*type = [info objectForKey:UIImagePickerControllerMediaType]; ////指定用户选择的媒体类型

if ([type isEqualToString:@"public.image"]) {

TZImagePickerController \*tzImagePickerVc = [[TZImagePickerController alloc] initWithMaxImagesCount:1 delegate:self];

tzImagePickerVc.sortAscendingByModificationDate = NO;

[tzImagePickerVc showProgressHUD];

UIImage \*image = [info objectForKey:UIImagePickerControllerOriginalImage]; //获取原始图片

// save photo and get asset / 保存图片，获取到asset

[[TZImageManager manager] savePhotoWithImage:image location:self.location completion:^(NSError \*error){

if (error) {

[tzImagePickerVc hideProgressHUD];

NSLog(@"图片保存失败 %@",error);

} else {

[[TZImageManager manager] getCameraRollAlbum:NO allowPickingImage:YES needFetchAssets:NO completion:^(TZAlbumModel \*model) {

[[TZImageManager manager] getAssetsFromFetchResult:model.result allowPickingVideo:NO allowPickingImage:YES completion:^(NSArray<TZAssetModel \*> \*models) {

[tzImagePickerVc hideProgressHUD];

TZAssetModel \*assetModel = [models firstObject];

if (tzImagePickerVc.sortAscendingByModificationDate) {

assetModel = [models lastObject];

}

// [self.headView getHeadPortraitFromCameraWithHeadPortrait:image completion:^(UIImage\* image) {

//

// }];

[self.headView getHeadPortraitFromCameraWithHeadPortrait:image];

[self dismissViewControllerAnimated:FALSE completion:nil];

}];

}];

}

}];

}

}

### 数据处理模块

1. 模块简介

数据处理模块，采用的设计模式中的单例模式，行使单一类的功能，对传来的数据进行一系列的逻辑处理和判断，得出结果。

1. 模块实现

数据处理模块用来对传来的参数进行处理和加工，遵守着数据输入，数据输出的规则。对于用户，因为只是单一的管理用户信息，为了方便数据处理。我们将其设计为单例类，将其数据进行归档存放在沙盒中。

而对于SQLite数据库的类，我们使用开放封闭的准则。对于新加的功能，在不破坏原有的基础上，我们对其进行OC的类的扩展，增加类的功能。

1. 核心代码实现
2. 单例类初始化对象的创建，遵守Apple单例创建规范

+ (instancetype) sharedInstance {

//只进行一次

static dispatch\_once\_t onceToken;

//第一个参数是检查后面第二个参数所代表的代码块是否被调用的谓词,第二个参数则是在整个应用程序中只会被调用一次的代码块。dispach\_once函数中的代码块只会被执行一次，而且还是线程安全的。

dispatch\_once(&onceToken, ^{

instance = [[[self class] alloc] init];

/\* 基础属性值都写在这里： instance.height = 10; ......\*/

});

return instance;

}

//重写+allocWithZone方法，保证永远都只为单例对象分配一次内存空间

+ (instancetype) allocWithZone:(struct \_NSZone \*)zone {

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

instance = [super allocWithZone:zone];

});

return instance;

}

1. 表的数据插入，只需要将插入的数据转化为字典，传递即可

+ (BOOL) isInsertDataForRegister:(NSDictionary \*)dictionary {

[ATSQLiteManager isOpenDatabaseForSqliteDocument];

char \*errorMesg = NULL;

if ([ATUserManager shareUser].isCreateUserTable) {

NSString \*sql = [NSString stringWithFormat:@"insert into user (userId, nickName, phoneNumber, password) values('%@', '%@', '%@', '%@');", dictionary[@"userId"], dictionary[@"nickName"], dictionary[@"phoneNumber"], dictionary[@"password"]];

int result = sqlite3\_exec(\_db, sql.UTF8String, NULL, NULL, &errorMesg);

// 销毁指针

sqlite3\_free(errorMesg);

[ATSQLiteManager closeDatabase];

return result == SQLITE\_OK;

}else {

ATLog(@"注册User表插入数据失败:%s", errorMesg);

return FALSE;

}

}

1. 数据库的开启和关闭

+(void)closeDatabase{

int result = sqlite3\_close(\_db) ;

[ATSQLiteManager judgeWithResult:result action:@"关闭数据库"];

// pthread\_mutex\_unlock(&mutex); //多线程中对数据库的操作

}

+ (bool)isOpenDatabaseForSqliteDocument {

// pthread\_mutex\_lock(&mutex);

// 获得沙盒中的数据库文件名

NSString \*fileName = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) lastObject];

NSString \*filePath = [fileName stringByAppendingPathComponent: @"AnnTenants.sqlite"];

NSLog(@"数据库路径:%@", filePath);

// 创建(打开)数据库（如果数据库文件不存在，会自动创建）

int result = sqlite3\_open(filePath.UTF8String, &\_db);

return result == SQLITE\_OK;

}

1. 用户单例类代码

//NSUserDefaults 设置默认值,不会覆盖用户设置的值，放在 applicationDidFinishLaunching

- (void) setUserManagerDefaultValue {

NSUserDefaults \*userDefaults = [NSUserDefaults standardUserDefaults];

[userDefaults registerDefaults:@{@"isCreateUserTable": @FALSE}];

[userDefaults synchronize];

}

//删除NSUserDefaults所有记录

- (void) deleteAllUserInformation {

NSUserDefaults \* userDefaults = [NSUserDefaults standardUserDefaults];

NSDictionary \* dict = [userDefaults dictionaryRepresentation];

for (id key in dict) {

[userDefaults removeObjectForKey:key];

}

[userDefaults synchronize];

}

### 视图控制器的UI逻辑模块

1. 模块简介

视图UI在iOS开发中，扮演着非常重要的角色，它将从数据库或者从网上传来的数据进行一系列的处理展示在界面上。虽然说起来，看起来简单，但是这其中还要考虑到用户体验、界面逻辑、屏幕尺寸的适配等等各个方面。

1. 模块实现

为了提高代码未来的可扩展性，降低UI和数据模型、数据逻辑的耦合度，我们抛弃了传统的MVC设计模式，采用更好更优秀的MVVM设计模式。为了使结构清晰，将数据模型放在Model文件夹，将视图控制器，视图放在View文件夹。将视图和数据的处理逻辑放在ViewModel的文件夹中。

1. 核心代码实现
2. 登录中ViewModel判断是不是手机号的逻辑

class func isTelNumber(num:NSString)->Bool {

let mobile = "^1((3[0-9]|4[57]|5[0-35-9]|7[0678]|8[0-9])\\d{8}$)"

let CM = "(^1(3[4-9]|4[7]|5[0-27-9]|7[8]|8[2-478])\\d{8}$)|(^1705\\d{7}$)";

let CU = "(^1(3[0-2]|4[5]|5[56]|7[6]|8[56])\\d{8}$)|(^1709\\d{7}$)";

let CT = "(^1(33|53|77|8[019])\\d{8}$)|(^1700\\d{7}$)";

let regextestmobile = NSPredicate(format: "SELF MATCHES %@",mobile)

let regextestcm = NSPredicate(format: "SELF MATCHES %@",CM )

let regextestcu = NSPredicate(format: "SELF MATCHES %@" ,CU)

let regextestct = NSPredicate(format: "SELF MATCHES %@" ,CT)

if ((regextestmobile.evaluate(with: num) == true) || (regextestcm.evaluate(with: num) == true)

|| (regextestct.evaluate(with: num) == true) || (regextestcu.evaluate(with: num) == true)) {

return true

}else {

return false

}

}

1. 验证注册时，输入的数据是否正确

public class func login(\_ phoneNumber: String, \_ password: String) -> isLogin {

// 判断输入是为空

let isPN = ATLoginViewModel.isStringEmpty(content: phoneNumber)

let isPW = ATLoginViewModel.isStringEmpty(content: password)

if !isPN || !isPW {

return isLogin(isPhoneNumber: isPN, isPassword: isPW)

}

guard let resultPN = ATSQLiteManager.selectData("userId, nickName, phoneNumber, password", fromDatabase: "user", conditions: "phoneNumber = '\(phoneNumber)'") else { return isLogin(isPhoneNumber: false, isPassword: true) }

guard let resultPW = ATSQLiteManager.selectData("userId, nickName, phoneNumber, password", fromDatabase: "user", conditions: "password = '\(password)'") else { return isLogin(isPhoneNumber: true, isPassword: false) }

if resultPN.count > 0 && resultPW.count > 0{

return isLogin(isPhoneNumber: true, isPassword: true)

}else if resultPN.count > 0 && resultPW.count == 0 {

return isLogin(isPhoneNumber: true, isPassword: false)

}else if resultPN.count == 0 && resultPW.count > 0 {

return isLogin(isPhoneNumber: false, isPassword: true)

}else {

return isLogin(isPhoneNumber: false, isPassword: false)

}

}

1. 存储图像，动态获取沙盒路径存储

+ (void) saveHeadPortrait:(UIImage \*)headImage withName:(NSString \*)name {

NSString \*unarchiverImagePath = [ATTools unarchiveImageDirectory:@""];

[ATTools deleteImageWith:unarchiverImagePath];

NSString \*imagePath = [ATTools saveImageWithName:name withImage:headImage];

ATMyHeadModel \*headM = [ATMyHeadModel new];

headM.avatarPath = imagePath;

WZLSERIALIZE\_ARCHIVE(headM, @"ATMyHeadModel", [ATUserManager filePath]);

[ATSQLiteManager updateData:@"avatarPath" values:imagePath fromDatabase: @"user" conditions: [NSString stringWithFormat:@"userId = '%@'", headM.userId]];

}

+ (NSDictionary \*) readHeadPortrait {

ATMyHeadModel \* headM = nil;

NSString \*unPath = [ATUserManager filePath];

WZLSERIALIZE\_UNARCHIVE(headM, @"ATMyHeadModel", unPath);

NSString \*imagePath = [NSString stringWithFormat:@"%@%@", ATDocumentPath, headM.avatarPath];

NSString \*nickName = headM.nickName;

UIImage \*headImage = [UIImage imageWithContentsOfFile: imagePath];

if (headImage == nil) {

headImage = [UIImage imageNamed:@"headPortrait"];

}

if (nickName == nil) {

nickName = @"匿名";

}

return @{@"headImage": headImage, @"nickName": nickName};

}

1. 宏定义定义的用运行时对模型类进行归档

#pragma mark - 宏定义的序列化与反序列化

#define WZLSERIALIZE\_CODER\_DECODER()

1. 用工厂设计模式和运行时生成不同样式的单元格

- (UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath {

ATShareHouseSectionsModel \*shareHouseSectionsM = self.cellInfo[indexPath.section];

ATShareHouseModel \*shareHouseM = shareHouseSectionsM.sectionHousesInfo[indexPath.row];

NSString \*cellClassName = shareHouseM.cellStyle;

static NSString \*cellID = nil;

Class CellClass = NSClassFromString(cellClassName);

id cell = nil;

// 运用工厂模式和运行时

if ([cellClassName isEqualToString:@"ATShareHouseCellView"]) {

cellID = @"ATShareHouseCellView";

cell = [tableView dequeueReusableCellWithIdentifier:cellID];

if (!cell) {

cell = [[CellClass alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellID];

}

[cell bindViewModel:shareHouseM];

} else if ([cellClassName isEqualToString:@"ATShareCellView"]) {

cellID = @"ATShareCellView";

cell = [tableView dequeueReusableCellWithIdentifier:cellID];

if (!cell) {

cell = [[CellClass alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellID];

}

[cell bindViewModel:shareHouseM];

}else if ([cellClassName isEqualToString:@"ATShareWriteCellView"]) {

cellID = @"ATShareWriteCellView";

cell = [tableView dequeueReusableCellWithIdentifier:cellID];

if (!cell) {

cell = [[CellClass alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellID];

}

[cell bindViewModel:shareHouseM];

}

return cell;

}

### 导入文件模块

1. 导入文件模块简介

导入文件模块把要使用的类的头件统统放在这个文件中，减少了需要把类一个一个导入每个文件中。

1. 模块文件的实现

导入文件模块通过使用Objective-C关键字，#import，#define 等关键字，使公有方法达到了共享。

1. 核心代码实现
2. pch文件代码

#ifndef Teannt\_pch

#define Teannt\_pch

#pragma mark - 导入类

#import <sqlite3.h>

#import <SnapKit/SnapKit-Swift.h>

#import <ObjectMapper/ObjectMapper-Swift.h>

. . . . . .

. . . . . .

. . . . . .

#endif /\* Teannt\_pch \*/

1. 自定义打印语句

//自定义打印日志

func ATLog<T>(\_ message: T, fileName: String = #file, methodName: String = #function, lineNumber: Int = #line){

//文件名、方法、行号、打印信息

//print("\(fileName as NSString)\n方法:\(methodName)\n行号:\(lineNumber)\n打印信息\(message)");

print("错误日志:methodName:\(methodName) lineNumbers:\(lineNumber)\n~~~~~~>:\(message)\n");

}

### 第三方文件模块

1. 模块简介

通过终端配置CocoaPod对第三方类库进行管理，使用他人写好的类库，提高项目的开发进度。

1. 模块实现

CocoaPod配置好以后，实现以下步骤即可：

1. 打开终端，输入终端命令 cd 项目路径；
2. 输入 pod init，在项目的目录下会自动创建一个Podfile的文件夹，将要使用的类库名和版本填入即可；
3. 输入 pod install –-verbose 指令，终端会自动给你下载好第三方库文件。
4. 核心代码实现

target 'AnnTenants' do

# Uncomment the next line if you're using Swift or would like to use dynamic frameworks

pod 'ReactiveCocoa', '~> 7.1.0-rc.2'

pod 'ReactiveObjC', '~> 3.1.0'

pod 'SnapKit', '~> 3.2.0' #约束Swift

pod 'ObjectMapper', '~> 3.1.0' #数据解析Swift

pod 'JSONModel', '~> 1.7.0' #数据解析OC

pod 'TZImagePickerController', '~> 2.0.0.9' #图片选择器

pod 'Masonry', '~> 1.1.0' #约束OC

use\_frameworks!

# Pods for AnnTenants

target 'AnnTenantsTests' do

inherit! :search\_paths

# Pods for testing

end

end

# 程序测试

测试程序是为了让程序充分满足用户的需求，使用户在使用程序时，不会出现意外的闪退和体验问题。为用户带来使用上的方便，让他们喜欢上这款App。本租房App包含的测试有前台的UI测试，后关于逻辑的单元测试。

5.1 UI测试

1. UI测试：注册

测试方法：打开租房App，点击底部按钮的“我的”TabBar按钮，然后点击“登录/注册”，即可打开“注册”页面，根据提示填写必填信息后，点击“注册”按钮即可完成。见图 5-1：



**图5-1 用户登录**

**测试完成后，可以通过查询数据库和“我的”界面进行查看。**

1. UI测试：发布房源

测试方法：在底部的UITabBar选择“发布房源”，进入发布房源界面，按照提示，填写自己的房源情况，然后点击“发布”，然后进入填写房东信息的界面，最后进行发布。见图5-2所示。



**图5-2 用户发布房源**

**测试完成后，可以在“首页”看到刚才自己发布的房源信息。**

**5.2 单元测试**

**单元严格遵守一个准则，一个单元测试，只能测试一个功能。当然，若是经过特殊处理也可以多测试一些方法。单元测试是对一些功能进行测试，看这个功能是否正常。经常对一些复杂的逻辑功能和网络接口进行调试，看输出的结果与自己的料想是否正确，这里通过一些断言来进行判断。**

**通过单元测试对数据库的处理和逻辑运算的测试，得出代码的健壮性和准确性。**

**租房App经过测试，界面流畅稳定，没有任何其他的问题。测试结果可能是因为测试条件不够严苛或者一些其他情况没有考虑，后期会对其进行改进，增加用户体验效果。**

# 总结

为了使来到异地的用户，可以找到一个拥有家一样的地方，特意设计这个款租房的App。这款App界面简洁大方，用户可以通过界面可快速准确找到自己想要的信息。而且，此外用户若有房源也可以通过这款App进行发布，使空间资源得到合理的应用。

这款App运行流畅，内部架构层次明确。总体使用了MVVM架构，通过程序的文件夹可以一目了然的看到。在这个架构的基础上使用了MVC，代理，工厂设计模式。使App的后期的功能扩展和单元测试降低了难度。对于数据库的操作，感觉自己还有点欠缺，以后家要加大这方面的练习和操作。

经过差不多一年的实习，非常感谢公司同事向我提出建议和耐心的给我解决问题以及对我的严格要求，使我将学校的理论应用到项目中。同时也非常感谢大学三年教过我的老师，对我的严格要求，使我在大学三年中没有白白度过，过的很充实。

参考文献

[1] 葛布林大帝(博客)，@numbbbbb(博客). Swift3.0官方教程中文版[J/OL]. <http://www.cocoachina.com/special/swift/> .2014-06-04.

[2] 关东升. iOS 开发指南 从零基础到App Store上架(第1版) [M].人民有点出版社出版发行.2013.

[3] 程杰. 大话设计模式[M].北京：清华大学出版社.2007.

[4] 三创iOS和PHP开发. iOS应用程序用户信息操作之用户头像图片存入沙盒及从沙盒中读取图片并显示在头像上.[J/OL].https://www.jianshu.com/p/d419b715e704.2017-06-24.

[5] 崔江涛.iOS开发之UITableView全面解析[J/OL]. http://www.cocoachina.com/ios/20140922/9710.html.2014-09-22.

[6] Kazuki Sakamoto，Tomohiko Furumoto.Objecive-C高级编程 iOS与OS X多线程和内存管理[M].黎华，译.北京：人民邮电出版社，2013.

[7] 沙梓社，吴航.iOS应用逆向工程，北京：机械工业出版社，2015.

[8] Matt Galloway. EffectiveObjective-C[M].爱飞翔，译.北京：机械工业出版社，2014.

[9] 王巍，Swifter 100个Swift 必备tips(第二版) [M]，北京：电子工业出版社，2015.

[10] Steve McConnell.Code Complete 代码大全[M].天奥，译.北京：学苑出版社，1993.

[11] 管蕾.iOS 9开发指南[M].北京：人民邮电出版社.2015.

[12] 王珊、萨师煊.数据库系统概论(第5版)[M].北京：高等教育出版社.2014.

[13] 郭晓亮.iOS UI开发捷径[M].北京：中国工信出版社.2017.

[14]张子怡.自己动手做iOS App[M].北京：电子工业出版社.

[15] 向治洪.React Native移动开发实战[M].北京：人民邮电出版社.2018.

[16] 张益辉.Swift从入门到精通[M].北京：清华大学出版社.2017.

[17] 刘一道.编写高质量代码 改善Objective-C程序的61个建议[M].北京：机械工业出版社.2015.

[18] 丁如敏，王琳.腾讯iOS测试实践[M].北京：机械工业出版社.2017.

[19] Stephen G.Kochan.Objective-C程序设计[M].林奕，范俊，朱弈欣.译.北京：电子工业出版社.2014.

# 致 谢

在毕业设计完成过程中非常感谢郑颖老师，金松林老师。在毕业设计定稿时，郑颖老师给了我许多的指导和建议，而且时时的在聊天群里了解我们的毕业设计的进度情况，向我提供一些建议。对于我们遇到的问题，即使在百忙中也会立即给我们回复消息。因为后来郑老师有其他的事，把我们交托给了金老师。金老师非常负责，对于学校发布的关于毕业生的事总是能即使告知我们，给我提供非常有用的信息和建议，尤其感谢金老师在百忙中为我把未完善的协议书填写完整。

同时感谢在实习找工作时，我的小伙伴们给我提供的面试经验，使我成功找到了一份工作。在实习过程中，带我的主管和同事对于我的问题给予了积极的回答，并指导我学习的方向，修正了许多开发中的不好习惯，让我把学校的理论和实践有机的结合起来。还有感谢在校的同学，在公司实习过程中，积极的帮助我们处理学校的事宜，免去了我们在公司和学校两地的来回奔波。

最后感谢大学三年来老师对我的教导，使我的大学三年过的很充实，使我在工作后把我以前的理论立马用到了实践上，让我再次深深体会到大学生活的弥足珍贵。

对于这次参加审阅论文和毕业答辩的老师，感谢你们在百忙中抽出时间对我们这一年的实习进行检测，你们的指导和建议我们一定会铭记在心！谢谢！