



南京大学

本科毕业设计

院 系 软件学院
专 业 软件工程
题 目 基于 iOS 平台的集点卡应用的设计与实现
年 级 2012 学 号 121250151
学 生 姓 名 王琨
指 导 老 师 刘钦 职 称 讲师
论 文 提 交 期 间 2016 年 5 月 25 日

南京大学本科生毕业论文（设计）中文摘要

毕业论文题目：基于 iOS 平台的集点卡应用的设计与实现

软件学院 院系 软件工程 专业 2012 级本科生姓名：王琨

指导教师（姓名、职称）：刘钦（讲师）

摘要：

为了吸引更多的客户，店铺会发起各式各样的促销活动，其中“集点卡”是一个非常常用的促销方式。然而，传统的集点卡对于商家来说浪费纸张、成本颇高，对于消费者来说容易丢失、不易保存。本应用提供了一套完整的针对这个问题的解决方案，既为商家提供了发布活动信息的平台、省去了印制集点卡、集点贴纸的成本，又为消费者提供了获取优惠信息的渠道以及在线集点的功能。

本文主要讲述了“基于 iOS 平台的集点卡应用”的设计与实现，第一章概述了项目的背景以及该项目于国内外已有项目的比较，第二章简要介绍了在开发过程中使用的一些技术，第三章讲述了该项目的需求分析和概要设计，第四章选取项目中的典型模块“活动”重点介绍了其详细设计和实现。

本项目使用了苹果公司在近年才发布的新的编程语言 Swift 并使用 Swift 2.2 后所推崇的面向协议的编程思想进行代码的设计与编写，使用 CocoaPods 完成第三方库依赖的集成，使用 LeanCloud 的数据存储服务替代传统的服务器，使用了高德地图的第三方服务完成了兴趣点搜索、地点定位的功能，使用 Git 完成版本控制。

本项目从产品需求的分析到原型以及 UI 的设计，再到代码的实现以及调试，均由我个人完成。实现了从商家创建店铺、发布活动，到消费者发现店铺、找到店铺、完成集点操作并兑换商品、线上评论的整个流程以及一些用户相关的必要功能。

关键词：iOS 开发、集点、Swift、POP（面向协议编程）、LeanCloud

南京大学本科生毕业论文（设计）英文摘要

THESIS: The Design And Implementation Of The Application Of Loyalty Card
Based On iOS System

DEPARTMENT: Software Institute

SPECIALIZATION: Software Engineering

UNDERGRADUATE: 2012

MENTOR: QIN LIU

ABSTRACT:

To attract more clients, the stores would organize various sales promotion events and ‘loyalty card’ is a common form among these events. However, traditional loyalty card wastes paper and has a high cost for the merchants, meanwhile, it is easy to lose and hard to conserve for the consumers. This application provides a complete solution for this problem mentioned above which provides a platform for merchants to announce the information of events and saves the costs of producing loyalty cards or loyalty tags, at the same time, this application offers a channel for consumers to achieve the sales promotion information and provides a way of collecting scores online.

This paper mainly narrates the design and implementation of an application of loyalty card based on iOS system. The first chapter summarizes the background of this project and the comparison of projects already existed both nationally and internationally. The second chapter briefly introduces some techniques used during the process of developing. The third chapter narrates the requirement analysis and outline design of this project. The fourth chapter emphatically introduces its detailed designs and implementation by selecting typical modules ‘activity’ of the project.

This project used a new programming language of Swift which was released in recent years, and used Protocol Oriented Programming which was recommended after Swift 2.1 to design and program. CocoaPods was utilized to integrate the third-party libraries dependency. The data storage server of LeanCloud was used instead of traditional servers. The AutoNavi Map Service was used to realize the function of

searching for the interest points and locating the points. Git was used to control the version of this project.

This project was accomplished all by myself, including the analysis of requirements, the design of prototype and UI, the programming and debugging of codes. Till now, this project has realized some necessary functions as follows: creating stores and announcing events by merchants, discovering and locating the stores by consumers, collecting scores and exchanging goods, and a complete procedure of commenting online and some other functions relating the users.

KEY WORDS: iOS Development、Loyalty、Swift、POP(Protocol Oriented Programming)、LeanCloud

目 录

图目录	VI
表目录	VIII
第一章 引言	1
1.1 项目背景	1
1.2 国内外相关产品的比较	1
1.3 论文的主要工作和组织结构	2
第二章 技术概要	3
2.1 项目设计	3
2.1.1 Sketch 简介	3
2.2 开发语言	3
2.2.1 Swift 简介	4
2.2.2 AutoLayout 简介	4
2.2.3 Swift 与 Objective-C 的混编	5
2.3 第三方服务	5
2.3.1 LeanCloud 数据存储服务	6
2.3.2 高德地图开放平台简介	6
2.4 项目集成	6
2.4.1 Xcode 简介	7
2.4.2 CocoaPods	7
2.5 本章小结	8
第三章 系统需求分析与概要设计	9
3.1 项目整体概述	9
3.2 系统的需求分析	9
3.2.1 业务需求	9
3.2.2 用户需求	10
3.2.4 系统级需求	17
3.3 系统的概要设计	21
3.3.1 基于 POP (面向协议编程) 的 MVVM 设计	21
3.3.2 接口概述	23
3.3.3 模块单元结构图	24
3.3.4 持久化机制	24
3.3.5 安全机制	25
3.3.6 数据库概要设计	25
3.4 本章小结	26

第四章 活动模块的详细设计与实现.....	27
4.1 基于 iOS 平台的集点卡应用活动模块的概述.....	27
4.2 基于 iOS 平台的集点卡应用的详细设计.....	27
4.2.1 活动模块类设计	27
4.2.2 数据库详细设计	30
4.2.3 界面原型设计	30
4.3 基于 iOS 平台的集点卡应用活动模块的实现	36
4.3.1 View 实现	36
4.3.2 Model 实现	40
4.3.3 Controller 实现	44
4.3.4 通过协议扩展来减少重复代码与解耦.....	45
4.3.5 基于 POP 的 MVVM 的设计模式的详细实现	47
4.4 本章小结	51
第五章 总结与展望.....	52
5.1 总结.....	52
5.2 展望.....	53
参考文献	54
致谢	55

图目录

图 2.1 Sketch 使用界面	3
图 2.2 Podfile	7
图 3.1 用例图	11
图 3.2 系统顺序图：集点	18
图 3.3 系统顺序图：兑换	18
图 3.4 活动图：集点	19
图 3.5 活动图：兑换	20
图 3.6 iOS 传统 MVC 架构	21
图 3.7 传统的 MVVM 架构图	22
图 3.8 本项目中使用的基于 POP 的 MVVM 架构图	22
图 3.9 本项目完整框架图	23
图 3.10 模块单元结构图	24
图 4.1 活动列表类图	28
图 4.2 活动详情类图	29
图 4.3 数据库 ER 图	30
图 4.4、4.5 界面原型：活动列表，界面原型：活动详情	31
图 4.6 界面原型：店铺详情	32
图 4.7、4.8 界面原型：集点卡，界面原型：消息列表	33
图 4.9 界面原型：个人中心	34
图 4.10 界面原型：兑换	34
图 4.11 界面原型：店铺编辑	35
图 4.12 界面原型：活动创建	35
图 4.13 Storyboard：活动模块	36
图 4.14 Storyboard：Segue 界面跳转	37
图 4.15 利用 Segue 进行视图跳转的代码实现	37
图 4.16 使用 Xib 组件关联的实现	38
图 4.17 Autolayout 约束制定的实现	39
图 4.18 ActivityToolBar 组件配置的代码实现	40
图 4.19 ActivityModel 的代码实现	41
图 4.20 Activity API 扩展的代码实现	43
图 4.21 ActivityListViewController 视图控制器的代码实现	44
图 4.22 IBAction 事件绑定的实现	45

图 4.23 通过协议扩展抽离重复逻辑的代码实现	47
图 4.24 基于 POP 的 MVVM 类关系图	47
图 4.25 ActivitySimpleViewModel 的代码实现	48
图 4.26 Presentable 协议以及协议扩展的代码实现	49
图 4.27 ActivityListDataSource 的代码实现	50

表目录

表格 3.1 用例列表	12
表格 3.2 用例描述：查看活动	13
表格 3.3 用例描述：查看店铺	13
表格 3.4 用例描述：店铺评论	14
表格 3.5 用例描述：提供集点码	15
表格 3.6 用例描述：集点	15
表格 3.7 用例描述：兑换商品	16
表格 3.8 用例描述：验证兑换码	17
表格 3.9 数据库通用字段	26

第一章 引言

1.1 项目背景

现如今，物质生活日渐丰富，大街小巷奶茶甜品店层出不穷，竞争激烈。各商家往往会发起各式各样的促销活动来吸引更多的客户，其中集点卡就是一个非常普遍的方式，商家通过发放给消费者集点卡，每完成一次消费集一次点，集满特定次数可以兑换特定商品，对于客户来说，免费的商品总是最具有吸引力的，对于商家来说，这种方式则能更好地增加回头客。然而，对于商家来说，活动的宣传往往仅仅局限于店面的海报以及传单，印制集点卡和贴纸需要不小的成本；对于消费者来说，尽管乐意得到这样的优惠，但是并不乐意自己的钱包因为各类集点卡而变得鼓鼓囊囊，而且集点卡容易丢失，贴贴纸、保存集点卡会让人觉得麻烦。

而今，移动互联网发展迅速，智能手机已经帮助人们解决了生活中大大小小的问题，尤其对于年轻人群来说，各色各样的 App 大大简化了人们的衣食住行，尤其在吃方面起到了相当大的引导作用，国内虽然已经有类似“美团”、“大众点评”能够帮助这些人们解决部分问题，但是一方面这些应用是基于店铺推荐而非基于活动，针对性不强，另外并没有一款应用具备完整的集点功能。针对上述的需求，我们可以通过定义一套完整的针对这些奶茶甜品店集点服务的方案，且将受众定义为年轻人——这些店铺的主要顾客，设计一款功能齐全且具有良好交互的应用，来帮助商家以及消费者解决这个问题。

1.2 国内外相关产品的比较

在 1.1 中有提到“美团”、“大众点评”这些人们耳熟能详的应用，在早期，它们的主要功能是“团购”，即通过网上团购的方式来让用户获得更大的利润，而今，随着信息化的日益发展，更多的用户使用他们来获取店铺资讯，通过同为消费者的其他用户的评论和推荐来决定吃哪家店、点什么菜。当然，通过在线支付或者购买团购券来获取优惠依然是他们的一个核心功能。然而，它们已经占领了市场上的大部分份额，如果做一个跟它们类似的应用显然没有竞争优势，所以可以在特定领域上寻找突破口。

本项目的灵感来源于一款国外的应用——**barback**。这款应用在设计上相当简约，但是功能完善。消费者可以通过它找到周边正在搞促销活动或者集点卡活

动的店铺，除了具备同类应用都具备的大部分功能外，还集成了集点卡的功能。在国外，该应用已经具备了一定的商家以及用户量，然而缺点也是非常明显，可能是开发成本的问题，该应用很明显使用了 **Hybrid** 技术进行开发，在用户体验方面存在较大的问题。

同市场上已有的那些美食资讯或者团购类应用不同的是，本应用的活动由店铺自行发布，且更具时效性以及定制性，同时，本应用将商家的受众范围集中于奶茶甜品，主打集点服务，能够更快地吸引特定用户，与原有的应用并不够成明显的竞争关系。

通过在 **AppStore** 以及网页搜索发现，国内并没有类似的专业应用于店铺集点服务的产品，所以通过分析需求以及国内外竞品，同时避开垄断行业，得出结论：这样的一款产品，如果设计和推广方式合理，可以收获一定的用户，获得成功。

1.3 论文的主要工作和组织结构

本文主要介绍了“基于 **iOS** 平台的集点卡应用”的设计与实现，概要介绍了项目的背景和前景，简单描述了在项目开发过程中使用的主要技术，详细介绍了项目的设计以及实现。项目从最初的设计到最后的编码以及论文的编写均由本人独立完成，最终设计并实现了将集点过程实现在移动端的一整套方案。

第一章：引言部分，主要介绍了项目背景、国内外相关产品的比较，并描述了论文的主要工作。

第二章：技术概要，主要介绍了在产品的设计和开发过程中使用到的技术概述。

第三章：需求分析和概要设计，详细地描述了基于 **iOS** 平台的集点卡应用的需求分析以及概要设计。

第四章：详细设计与实现，主要抽取“活动”这一典型的模块介绍了系统的详细设计和实现。

第五章：总结和展望，总结项目目前的进展以及在完成过程中的收获，探讨项目目前的缺点和不足，指出可能的扩展和发展方向。

第二章 技术概要

2.1 项目设计

本项目“基于 iOS 平台的集点卡应用”使用了 Sketch 3.0 来完成原型和 UI 的设计。

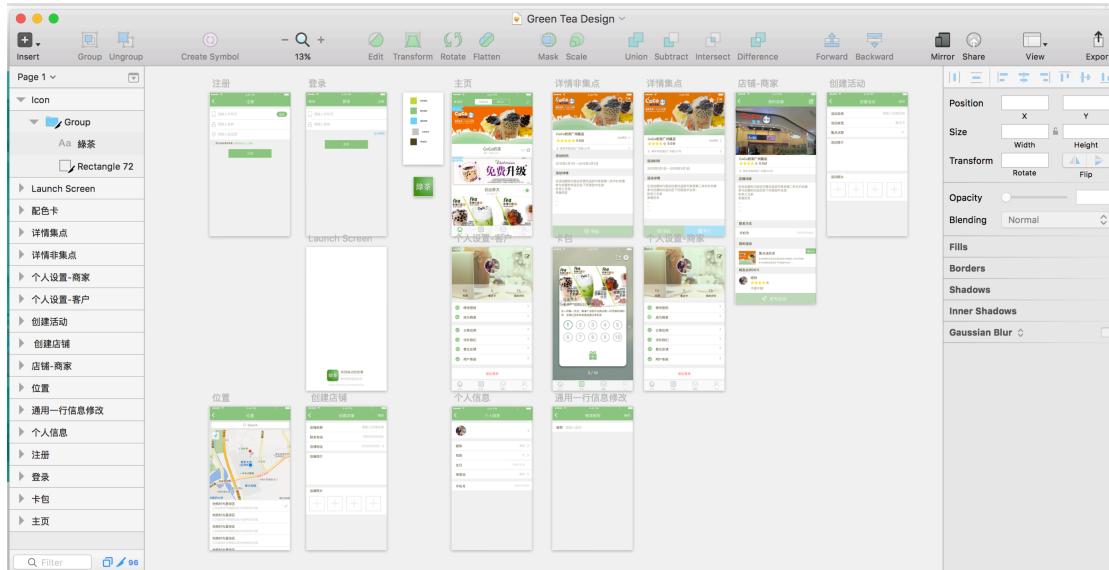


图 2.1 Sketch 使用界面

2.1.1 Sketch 简介

Sketch 是一款基于 Mac OS X 系统的轻量、易用的矢量设计工具，同 Photoshop 不同，它并没有强大的图像处理功能，但是却更适合于进行 UI 设计，是大部分 UI 设计师的首选。

在对产品的功能模块以及界面构成有了初步的想法后，我便使用 Sketch 制作了各个界面的原型，而后精化，最后完成了初步的 UI 设计。

2.2 开发语言

本项目——“基于 iOS 平台的集点卡应用”基于 Swift 2.1 语言进行开发，开发环境使用的是 Xcode7.2，在苹果更新了 Xcode7.3 并更新了 Swift 2.2 后，项目随后升级到了 Swift 2.2，并使用了一部分新特性。

在界面搭建方面，使用的苹果自行开发的“Autolayout”——自动布局技术进

行开发，同时使用了 **Xcode** 客户端所提供的“**Xib**”和“**Storyboard**”完成了大部分 **UI** 的快速搭建。

同时，该项目也使用了一些基于“**Objective-C 2.0**”编写的第三方库，应用通过动态链接库的方式实现了“**Swift** 与 **Objective-C** 的混编”。

2.2.1 Swift 简介

Swift 是由苹果公司于 **WWDC 2014** 上推出的一门专门用于 **iOS** 以及 **OS X** 开发的编程语言，一出世便万众瞩目，很快成为关注度最高的语言。**Swift** 以 **LLVM** 编译，其设计者 **Chris Lattner** 参考了 **Objective-C**、**Rust**、**Rubby**、**Python** 等众多优秀语言的特点，最终形成了 **Swift** 这一门独特的语言的语法特性。

与多年前就发布的 **Objective-C** 不同，**Swift** 是一门类型安全的静态语言，使用起来更加安全。然而它却又具备了许多动态语言的语法特性和交互方式，同时也支持包括闭包、面向对象、泛型、元组、集合、多返回值、可选变量等高级语言特性。

除了集百家之长以外，**Swift** 还具备一些非常诱人的特性，“可空类型”让编程变得更加安全，极大地提高了程序的稳定性，独有的强大的“**Playground**”功能更是突破了传统的交互式编程，还能显示图像、跟踪变量、控制台输出等等。

更重要的是，**Swift** 是一门快速发展的语言，从 **2014** 年发布，到 **2015** 年 **12** 月份发布开源，再到最近 **Swift 2.2** 的推出，很快 **Swift 3.0** 即将横空出世，**Swift** 已经不单单是一门用于苹果产品开发的语言，开源后它已经支持了 **Linux** 平台，并与全球开发者一起共同见证其进步。

在 **WWDC 2014** 上，苹果公司对 **Swift** 做出了重大更新，推出了 **Swift 2.0**，而本项目就是基于 **Swift 2.0** 编写并使用了 **Swift 2.0** 中的许多特性做了一些新鲜尝试。在 **Swift 2.1** 中，苹果首次确认了面向协议编程的思想方向，更是推出了“**Protocol Extension**”——协议扩展这一新功能。

尽管目前为止，**Swift** 仍然无法取代已经被使用了几十年的 **Objective-C**，大部分的应用以及公司仍然在使用 **Objective-C** 进行开发和维护，但是不可否认，**Swift** 就语言而言已经超越了 **Objective-C**，并且成为了新的开发趋势。

2.2.2 Autolayout 简介

为了能够兼容不同屏幕尺寸，苹果从 **iOS 6.0** 推出了新的自动布局方式——

Autolayout, 通过制定约束替代绝对位置来完成界面的布局, 界面元素会在界面尺寸、设备方向、容器的尺寸和方向等发生改变的时候自动对界面进行重新布局, 使用 **Autolayout** 进行界面设计和布局, 大大简化了对不同屏幕尺寸的适配。

然而, **Autolayout** 是一门学习成本比较高的技术, 使用传统的编码进行设置不仅上手困难, 代码也晦涩难懂, 针对这些问题, 一方面, 开发者开发出了像“**masonry**”这样优秀的第三方开源工具完成了原有 **API** 的封装, 使得 **Autolayout** 使用起来更加方便, 代码可读性也得到增强。而苹果官方则推出了所见即所得, 上手更加容易的“**Xib**”和“**Storyboard**”来帮助开发者进行布局方面的开发, 从 **iOS 6.0** 到现在的 **iOS 9.3**, 苹果在这方面始终都有更新, 技术也日渐进步, 无论是功能还是可用性都得到了很大的提高。

本项目的界面搭建使用了“**Storyboard**”结合“**Xib**”的方式, 对于一些只在固定位置出现的且没有复杂跳转的界面直接使用“**Storyboard**”进行构建并实现跳转。对于一些工具性的经常被使用的界面, 直接使用“**Xib**”使用搭建, 通过代码手动进行页面跳转。对于一些界面变化较大需要手动配置的界面, 则使用代码进行界面编写, 同时使用了“**Cartography**”这一个基于 **Swift** 编写的 **Autolayout** 库进行界面约束的设置。

2.2.3 Swift 与 Objective-C 的混编

由于 **Swift** 依旧处于发展的阶段, 一个完整项目的开发很多时候还必须使用一些完善的基于 **Objective-C** 编写的第三方库, 这个时候就涉及到了两种语言之间的混编。

针对这个问题, 苹果官方早已想到了最简单的方式。如果想要使用 **Swift** 来调用 **Objective-C** 编写的代码, 需要创建一个“桥接头文件——**Bridging Header**”, 并且在项目配置中指定项目的桥接头文件目录位置, 只需要在头文件中引入需要使用的 **Objective-C** 头文件, **Swift** 便可以轻松地调用 **Objective-C** 代码。

2.3 第三方服务

由于开发人力、开发时间和开发成本的限制, 同时也是基于项目本身比较轻量级的原因, 本项目“基于 **iOS** 平台的集点卡应用”并没有设计并编写独立的服务器进行数据服务, 而是使用了 **LeanCloud** 的数据存储服务替代了服务器的开发。

作为一款典型的“O2O”项目，本项目集成了“高德地图 SDK”完成项目的“POI（兴趣点）搜索”、地图显示、导航等服务，在项目中，商家可以通过定位以及搜索来设置店铺位置，顾客可以通过定位来搜索到与自己距离相近的店铺，并在地图上查看店铺的具体位置并且导航。

2.3.1 LeanCloud 数据存储服务

LeanCloud 提供了一站式的后端云服务，为移动开发省去了租用和维护服务器、编写后端代码的时间。

本项目集成了 LeanCloud 的 iOS SDK，通过阅读相关文档顺利地完成了项目的集成，虽然在集成过程中遇到了类似“混编”、“证书错误”的一系列问题，同时因为第三方数据存储服务的功能限制，必然导致整个项目的设计偏向“胖客户端”，但是依旧是省去了大量的工作量，同时也做到了数据服务的文档和高效。

所有的 Model 只需要继承 LeanCloud 所提供的 `AVObject` 基类，并且遵循 `AVSubClassing` 的协议，即可方便快捷地实现数据的查询、存储、删除、修改。同时可以在控制台查询数据、编写对应的脚本进行更进一步的工作。

2.3.2 高德地图开放平台简介

高德地图 iOS SDK 是一套基于 iOS 6.0 及以上版本的地图应用程序开发接口，提供了地图的显示与操作、POI 检索、地理编码、导航等功能，本项目集成了起 SDK，在应用内使用了其地图服务。

基于高德地图的 iOS，本项目封装了一个用于定位用户、自动搜索周围兴趣点、手动搜索兴趣点的工具类，在商家创建、编辑店铺的时候，用户查看店铺位置、导航的时候均使用了其服务。

2.4 项目集成

本项目“基于 iOS 平台的集点卡应用”使用 Xcode 7.3 进行开发并集成，同时使用了 CocoaPods 完成了第三方库依赖的集成。

2.4.1 Xcode 简介

Xcode 是由苹果公司开发的基于 Mac OS X 的集成开发工具，是开发 OS X 和 iOS 应用程序的最快捷的方式。Xcode 具有统一的用户界面设计，编码、测试、调试都在一个简单的窗口内完成。

2.4.2 CocoaPods

CocoaPods 是 iOS 应用的开源的依赖管理工具，只需要一个简单的“Podfile”文件便可以轻易地实现第三方库的依赖管理。

本项目使用到的大部分第三方开源库都使用 CocoaPods 集成。



```

platform :ios, '8.0'
use_frameworks!

target 'Loyalty' do
pod 'AVOSCloudDynamic'
pod 'AVOSCloudCrashReportingDynamic'
pod 'XCGLogger', '~> 3.1.1'
pod 'PKHUD'
pod 'Cartography'
pod 'Koloda'
pod 'ActionSheetPicker-3.0', :git => 'https://github.com/skywinder/
ActionSheetPicker-3.0.git'
pod 'Kingfisher'
pod 'AMapSearch'
pod 'Eureka'
end

post_install do |installer|
`find Pods -regex 'Pods/pop.*\\.h' -print0 | xargs -0 sed -i '' 's/\\(<\\)pop\\(/\\(.*
\\)\\(>\\)/\\\"\\2\\\"/'`
end

```

图 2.2 Podfile

上图为本项目的 Podfile 文件，其中指定了项目的最低版本要求：8.0，由于使用 Swift 开发，必须加入 use_frameworks! 标识符，加入该标识符后会在项目打包的时候将这些第三方库以来分别打包成 framework 而非使用静态库。

当我们有需要使用的第三方库的时候，只需要加入 pod ‘库名’ 即可完成配置，如图，我们可以配置所需要库的版本号、仓库地址等等，也可以通过一些额外命令来对集成后的库进行一些处理来解决一些基本操作所无法解决的问题。

2.5 本章小结

本章简要介绍了本项目——一款基于 iOS 平台的集点卡应用从设计、到项目搭建最后到完成所涉及到的技术。同样，这其中的大部分内容也是 iOS 开发者所使用也必须需要掌握的。

作为一名开发者，掌握最基本的设计技能以及了解做基本的交互支持也是比较重要的，在设计阶段，本项目使用了 Sketch 完成原型和最终 UI 的设计，开发过程中直接对照着 Sketch 源文件进行界面搭建和 UI 元素的切图，为开发带了不少方便。

本项目并没有选择传统且成熟的 Objective-C 来完成项目的编写，而是使用了新兴的正在发展中的 Swift 来完成，一方面是由于 Swift 这门语言的确存在很大的优势和新特性，另一方面也算是一种尝试，希望能够利用 Swift 的这些新特性新思想来优化代码质量以及提高开发体验。

任何一个具有一定规模的 iOS 项目都会使用一些第三方库，本项目使用 CocoaPods 完成第三方库的依赖配置和管理，方便快捷。对于一些使用 Objective-C 编写的第三方库，CocoaPods 会自动生成桥接头文件来供 Swift 来调用，对于一些手动集成的 Objective-C 第三方库，本项目通过手动创建统一的桥接头文件来完成混编的需求。

在第三方服务上，本项目使用了 LeanCloud 的数据存储服务来代替传统的服务端服务，这就意味着本项目使用的是“胖客户端”的设计。同时作为一个典型的“O2O”餐饮业项目，本项目使用了高德地图开放平台来集成地图和兴趣点搜索等相关服务来实现对应需求。

第三章 系统需求分析与概要设计

3.1 项目整体概述

本项目的主要目标是提供一个基于移动端的集点卡解决方案，对于商家来说，能够有一个平台发布活动的宣传，并且完成集点的这个流程，省去了印制集点卡的成本；对于客户来说，一方面能够更方便快捷地获取到周边奶茶甜品店的活动信息，一方面也解决了集点卡往往只集一次点便容易丢失不容易保存的问题，为客户带来方便。

这是一个典型的“O2O”项目，在同一个客户端上允许商家和客户进行操作。线上，商家注册店铺、发布活动，客户根据所在位置获取系统推荐的店铺活动，查看活动介绍、店铺介绍。线下，客户根据应用所提供的位置来到指定店铺进行消费并集点，在集点过程中，客户输入商家当面提供的集点码即可完成一次集点后，当集点数目达到指定要求的数目后，便可生成兑换对应商品的二维码，商家扫描对应的二维码即可完成消费。消费完毕，客户可以对活动以及店铺进行评论，商家可以对评论进行回复。

3.2 系统的需求分析

3.2.1 业务需求

在移动互联网飞速发展的现在，消费者们已经习惯于通过手机 App 来引导自己进行饮食消费，而本应用正是针对奶茶甜品领域而设计的一套完整的解决方案。

而如今，大街小巷四处都是奶茶店、甜品店，以年轻人为代表的都市人群成为了这些店铺的主要消费者，而这些人群的共同特点便是习惯于利用手机来完成生活中很大一部分的事物。众所周知，这些店铺每年几乎都会从不间断地推出各类促销活动来吸引客户，与同行竞争，而“集点卡”又是一个普遍的、典型的营销手段。本项目对于商家来说，一方面是为了帮助商家更好地宣传自己的店铺以及店铺所进行的活动，为商家提供一个同粉丝消费人群交流的平台，同时其核心业务“集点卡”更是帮助商家节约了印制卡片和贴纸的成本，大大地减少了开销。对于消费者尤其是那些热爱奶茶甜品的年轻人来说，本项目为消费者提供了周边各奶茶甜品店的活动信息，同时电子的集点卡解决了大部分年轻人容易丢失集点卡的问题，帮助消费者维护自身的权益。

总的来说，本项目为了满足以上的目标，需要具备以下的特性：1. 允许商家

注册自己的店铺并根据自身需求发布制定的活动信息；**2.**根据消费者所在的位置提供其周边的活动及店铺信息，引导消费者前往消费；**3.**为消费者和商家提供最方便快捷安全的在线集点及兑换服务；**4.**提供消费者反馈的入口，并允许商家与消费者进行一定程度的交流，从而改进服务质量

3.2.2 用户需求

本项目的核心功能是提供活动信息、完成在线集点，主要用户是广大消费者，当然商家也是其必不可少的用户群体。所以用户需求是商家和消费者对这个系统能够完成的具体任务的期望，描述了系统能够帮助消费者以及商家做什么，将业务需求转化为用户需求，如下所示：

- UR 1.** 商家和消费者可以使用系统来查看周边的活动
- UR 2.** 商家和消费者可以使用系统来搜索指定的活动
- UR 3.** 商家和消费者可以使用系统对指定活动进行收藏、取消分享以及分享操作
- UR 4.** 商家和消费者可以使用系统查看店铺的详情
- UR 5.** 商家和消费者可以使用系统针对店铺进行评论以及回复指定评论
- UR 6.** 商家可以使用系统对自己的店铺信息进行编辑
- UR 7.** 商家可以使用系统发布新的活动信息或编辑旧的活动信息
- UR 8.** 商家可以在消费者索要集点码的时候使用系统获取合法的集点码
- UR 9.** 商家可以在消费者出示兑换码的时候使用系统验证兑换码
- UR 10.** 消费者可以使用系统查看自己的集点卡以及集点进度
- UR 11.** 消费者可以使用系统完成店铺的集点活动
- UR 12.** 消费者可以删除已有的集点卡
- UR 13.** 消费者可以使用系统的导航功能找到指定店铺
- UR 14.** 消费者可以使用系统来完成集点的操作
- UR 15.** 消费者可以使用系统来兑换已经集点完成的集点卡

如图 3.1 中所示由于商家和消费者有众多重合的需求，本应用将商家功能与客户功能集中在一个客户端，并根据用户的身份来区分权限。在用例图中，中间部分的用例代表商家和客户均可以使用的功能，左侧部分即用户权限，这里需要说明的是：普通用户即客户可以通过创建店铺成为商家，从而对自己的店铺以及发布的活动具备了商家权限，而作为商家依然对于非自己所发布的活动和店铺享有普通用户权限。也就是说即使成为了商家也是可以集点和消费，仅仅是针对自己所发布的活动来说是商家这个角色。

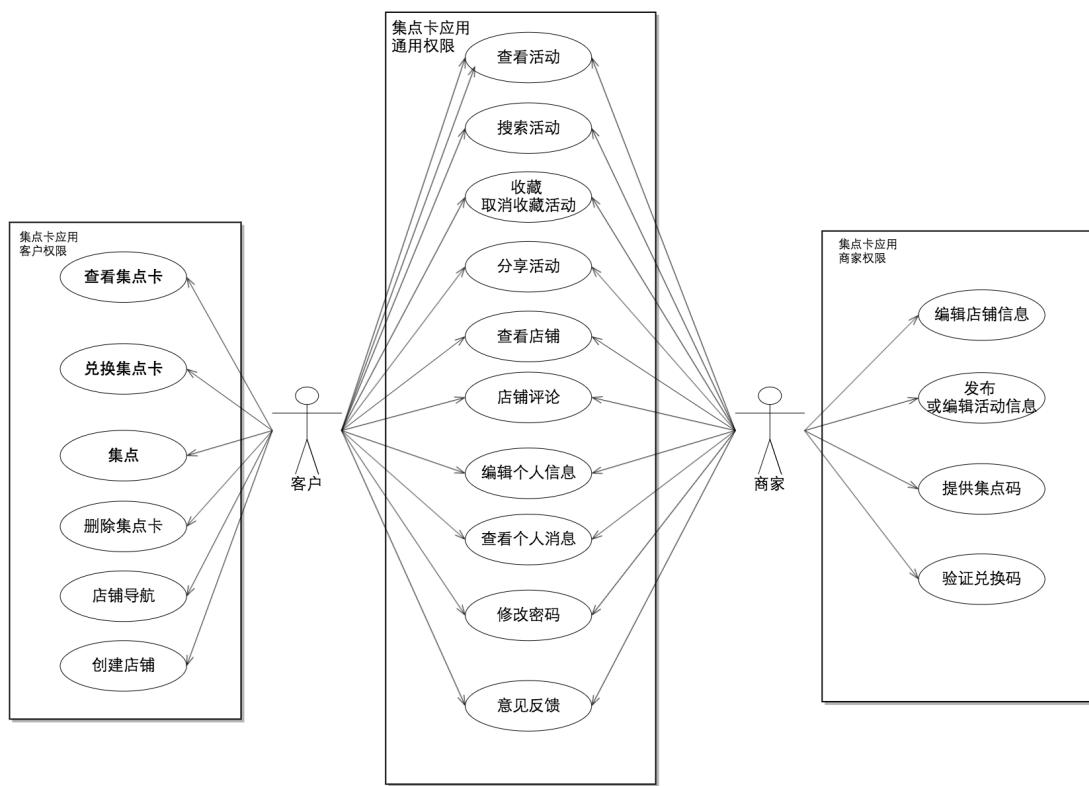


图 3.1 用例图

参与者	用例
用户（即包括商家和客户）	1. 查看活动
	2. 搜索活动
	3. 收藏、取消收藏活动
	4. 分享活动
	5. 查看店铺
	6. 店铺评论
	7. 编辑个人信息
	8. 修改密码
	9. 意见反馈
商家	10. 编辑店铺信息
	11. 发布或编辑活动信息
	12. 提供集点码
	13. 验证兑换码
客户	14. 查看集点卡
	15. 兑换商品

	16.集点
	17.删除集点卡
	18.店铺导航
	19.创建店铺、成为商家

表格 3.1 用例列表

由于用例众多，本小节只列数了覆盖系统核心功能的用例，从消费者角度，其流程覆盖到消费者从找到一个活动，到达店铺，完成集点，在集点到达一定要求后，完成兑换的流程；从商家角度，覆盖了在消费者集点和兑换过程中商家方面所需要进行的操作。

用例 1：查看活动

ID	1	名称	查看活动
创建者	王琨	最后一次更新	王琨
创建日期	2016年1月25日	最后一次更新	2016年5月5日
参与者	消费者或商家		
触发条件	用户需要查看活动信息		
前置条件	无		
后置条件	系统显示活动信息（列表形式、详情形式）		
优先级	高		
正常流程	1. 消费者进入应用首页。 2. 系统默认选择“集点卡”选项卡，加载并显示集点卡活动列表，列表内容包括活动宣传图、活动名称、店铺地址、收藏数、用户是否收藏。 3. 用户选择选项卡（选项卡内容包括集点卡和促销活动）。 4. 系统加载并显示对应选项下的活动列表。 5. 用户选择指定活动 6. 系统显示活动详情，包括除列表项显示内容外，显示店铺名称、店铺评分、活动时间、店铺评论数、活动详情。		
扩展流程	3.1 用户上拉列表 4.1 系统重新加载最新的活动列表 3.2 用户下拉列表 4.2 系统加载更多活动列表，若没有更多活动，则提示用户已经加载完毕		

特殊需求	1. 系统能够根据消费者所在位置来提供消费者距离自己最近的活动列表
------	-----------------------------------

表格 3.2 用例描述：查看活动

用例 2：查看店铺

ID	2	名称	查看店铺
创建者	王琨	最后一次更新	王琨
创建日期	2016 年 1 月 25 日	最后一次更新	2016 年 5 月 5 日
参与者	消费者或商家		
触发条件	用户需要查看店铺具体信息		
前置条件	用户正在查看活动详情，且用户并非店铺创建者		
后置条件	系统显示店铺详情		
优先级	高		
正常流程	1. 用户选择进入店铺查看详情 2. 系统显示店铺详细信息，包括店铺名称、店铺图片、店铺地址、店铺评分、网友点评		
扩展流程	3.1 用户选择联系店铺 4.1 应用调用手机系统拨打功能拨打店铺电话 3.2 用户选择店铺地址 4.2 系统进入地图页面，在地图上显示店铺所在地址 3.3 用户下拉评论列表 4.3 系统加载更多评论内容		
特殊需求	1. 若进入店铺详情的用户为店铺创建者，不具备扩展流程 3.1, 3.2 2. 若进入店铺详情的用户为店铺创建者，除了正常流程 2 中所提到的信息外，还显示店铺所正在进行的活动列表，并允许通过列表查看活动详情		

表格 3.3 用例描述：查看店铺

用例 6：店铺评论

ID	6	名称	店铺评论
创建者	王琨	最后一次更新	王琨
创建日期	2016 年 1 月 25 日	最后一次更新	2016 年 5 月 5 日

参与者	消费者或商家
触发条件	用户需要对店铺进行评价或者回复已有的评论
前置条件	用户进入店铺详情，用户并非店铺主人
后置条件	系统能显示最新的评论内容，并且更新店铺评分。
优先级	中
正常流程	<ol style="list-style-type: none"> 1. 客户选择发布评论 2. 客户选择评分等级 3. 客户输入评价内容 4. 客户选择确认发布 5. 系统验证内容合法，成功发布，并刷新店铺详情界面 6. 客户可以看到最新的店铺评论和店铺信息
扩展流程	<p>1.1 客户选择已有评论进行评论</p> <p>1.2 评价界面不显示评分等级选项，其后流程同正常流程</p> <p>5.1 系统检验并未进行评分，成功发布，并刷新评论列表，不影响店铺评分</p> <p>5.2 系统检验并没有输入评价内容，成功发布，并刷新店铺列表，不影响评论列表</p> <p>5.3 系统检验没有评分且没有评价内容，提醒用户至少完成其一，发布失败</p>
特殊需求	<ol style="list-style-type: none"> 1. 支持多级回复 2. 店铺主人允许回复已有回复但是不允许自行进行店铺评价

表格 3.4 用例描述：店铺评论

用例 12：提供集点码

ID	12	名称	提供集点码
创建者	王琨	最后一次更新	王琨
创建日期	2016 年 1 月 25 日	最后一次更新	2016 年 5 月 5 日
参与者	商家		
触发条件	消费者完成消费，需要集点，向店家索要集点码		
前置条件	商家进入自己所创建的对应活动页面		
后置条件	系统提供给商家有效的集点码		
优先级	高		

正常流程	1. 商家选择生成集点码 2. 系统将集点码显示给用户 3. 商家选择完成集点 4. 系统验证该集点码已经被成功使用，并提醒商家已使用
扩展流程	4.1 系统验证该集点码未被使用，提醒商家该集点码为被使用，询问商家是否销毁该集点码 5.1 用户选择销毁，则集点码不再生效。用户选择否，则集点码始终有效
特殊需求	1. 正常流程 3 中如果商家不选择完成集点，系统也会定时检验集点码是否已经被使用，并返回到先前页面 2. 一个集点码在系统中存在时间不会超过 24 小时

表格 3.5 用例描述：提供集点码

用例 16：集点

ID	16	名称	集点
创建者	王琨	最后一次更新	王琨
创建日期	2016 年 1 月 25 日	最后一次更新	2016 年 5 月 5 日
参与者	消费者		
触发条件	消费者完成消费，需要进行集点		
前置条件	消费者完成查看卡包或者查看活动用例		
后置条件	消费者在对应活动的集点卡上集点数+1		
优先级	高		
正常流程	1. 消费者选择集点 2. 系统判断消费者是否已有该活动的为完成的集点卡，若没有，则自动帮其领取一张。 3. 系统显示该活动的集点卡，并弹出输入框，提醒用户输入集点码 4. 消费者输入集点码 5. 系统验证集点码成功，集点卡集点数+1 6. 若集点数满足集点要求，则提醒用户已达到兑换条件		
扩展流程	5.1 系统验证集点码失败，提醒用户输入正确的集点码，集点数目不变		
特殊需求	1. 每个集点码只允许验证一次，防止恶意刷点		

表格 3.6 用例描述：集点

用例 15：兑换商品

ID	15	名称	兑换商品
创建者	王琨	最后一次更新	王琨
创建日期	2016年1月25日	最后一次更新	2016年5月5日
参与者	消费者		
触发条件	消费者对应活动的集点卡已经到达数量要求		
前置条件	消费者进入对应集点卡页面		
后置条件	消费者成功完成兑换，该集点卡销毁		
优先级	高		
正常流程	1. 消费者选择兑换按钮 2. 系统判断是否满足兑换条件，若满足条件，则显示兑换二维码 3. 用户选择已经兑换完成 4. 系统判断兑换成功，销毁集点卡		
扩展流程	2.1 系统判定并没有满足兑换条件，提醒用户需要继续集点 4.1 系统判定并没有兑换成功，提醒用户并没有成功消费		
特殊需求	1. 用户即使没有选择兑换完成，系统也会在二维码存在期间定时验证是否成功兑换，若成功兑换，自动给予提示 2. 即使已生成二维码，但是没有成功兑换，不影响下次兑换		

表格 3.7 用例描述：兑换商品

用例 13：验证兑换码

ID	13	名称	验证兑换码
创建者	王琨	最后一次更新	王琨
创建日期	2016年1月25日	最后一次更新	2016年5月5日
参与者	商家		
触发条件	消费者提供兑换码，需要兑换		
前置条件	商家进入兑换码对应的活动界面		
后置条件	系统成功完成商品兑换		
优先级	高		
正常流程	1. 商家选择兑换商品 2. 系统显示二维码扫描界面，提醒用户将摄像头对准二维码 3. 商家将摄像头对准消费者所提供的二维码		

	4. 系统成功扫描二维码并完成兑换，提示商家兑换成功 5. 商家提供对应商品，完成交易
扩展流程	4.1 系统扫描到的二维码非法，系统提示并没有对应的集点卡，提示商家不能进行兑换 5.1 交易失败
特殊需求	1. 一个兑换二维码只允许兑换一次，一旦兑换成功则失效

表格 3.8 用例描述：验证兑换码

3.2.4 系统级需求

系统级需求是用户对系统行为的期望，相比用户需求更为具体详细，而一系列的系统行为联系在一起就可以帮助用户完成各种任务，最后满足了业务需求。系统需求可以直接映射为系统行为，定义了系统中需要实现的功能，描述了开发人员需要实现什么。

下文将以系统的核心需求：集点与兑换为例，通过系统顺序图以及活动图来描述本应用的系统行为，描述消费者以及商家是如何使用系统来完成这样一个过程的。

图 3.2 为集点过程的顺序图，虽然是同样的客户端，但是为了区分，图中的集点卡应用为两个实体，一个为消费者权限，一个为商家权限，消费者允许集点卡应用获取自身位置，集点卡应用通过位置返回活动信息，消费者选择特定活动，应用返回活动详情，消费者通过导航来到店铺，完成消费，向商家索取集点码，商家向客户端请求集点码，客户端生成集点码并返回，商家告知消费者后，消费者向应用输入集点码，应用完成集点码的验证并告知用户结果完成集点。

图 3.3 为兑换过程的顺序图，当集点完成后，消费者向应用请求兑换集点码，应用生成对应的二维码，消费者要求商家扫描二维码完成兑换，商家通过使用自己的商家权限的客户端扫描对应二维码并且验证是否合法，若合法，则提供指定商品。

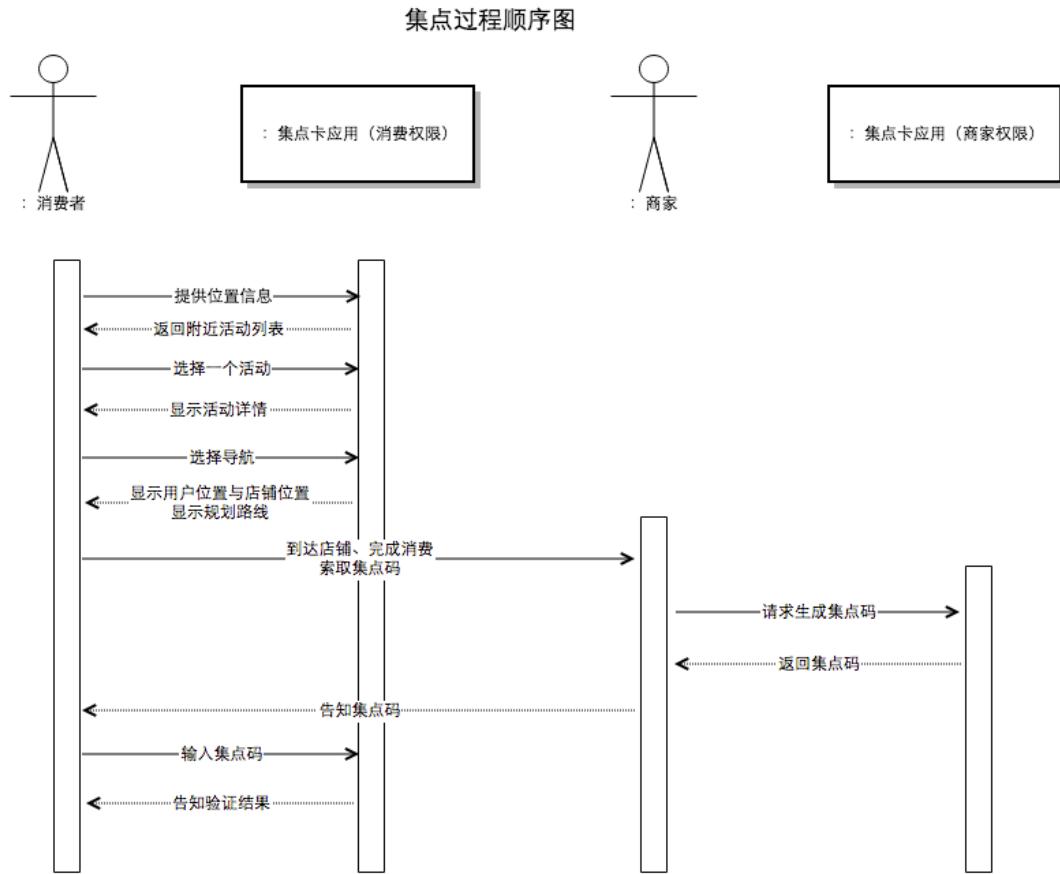


图 3.2 系统顺序图：集点

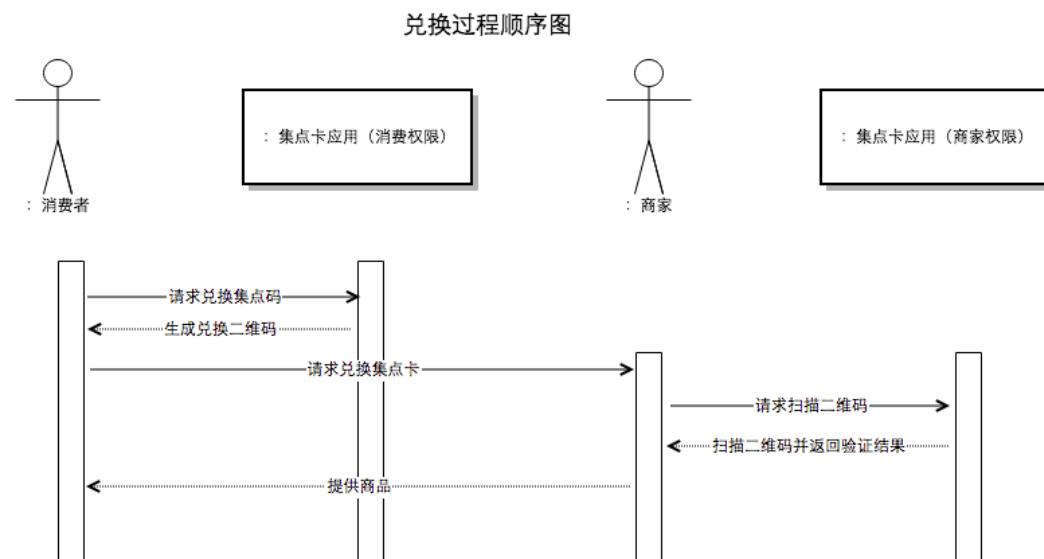


图 3.3 系统顺序图：兑换

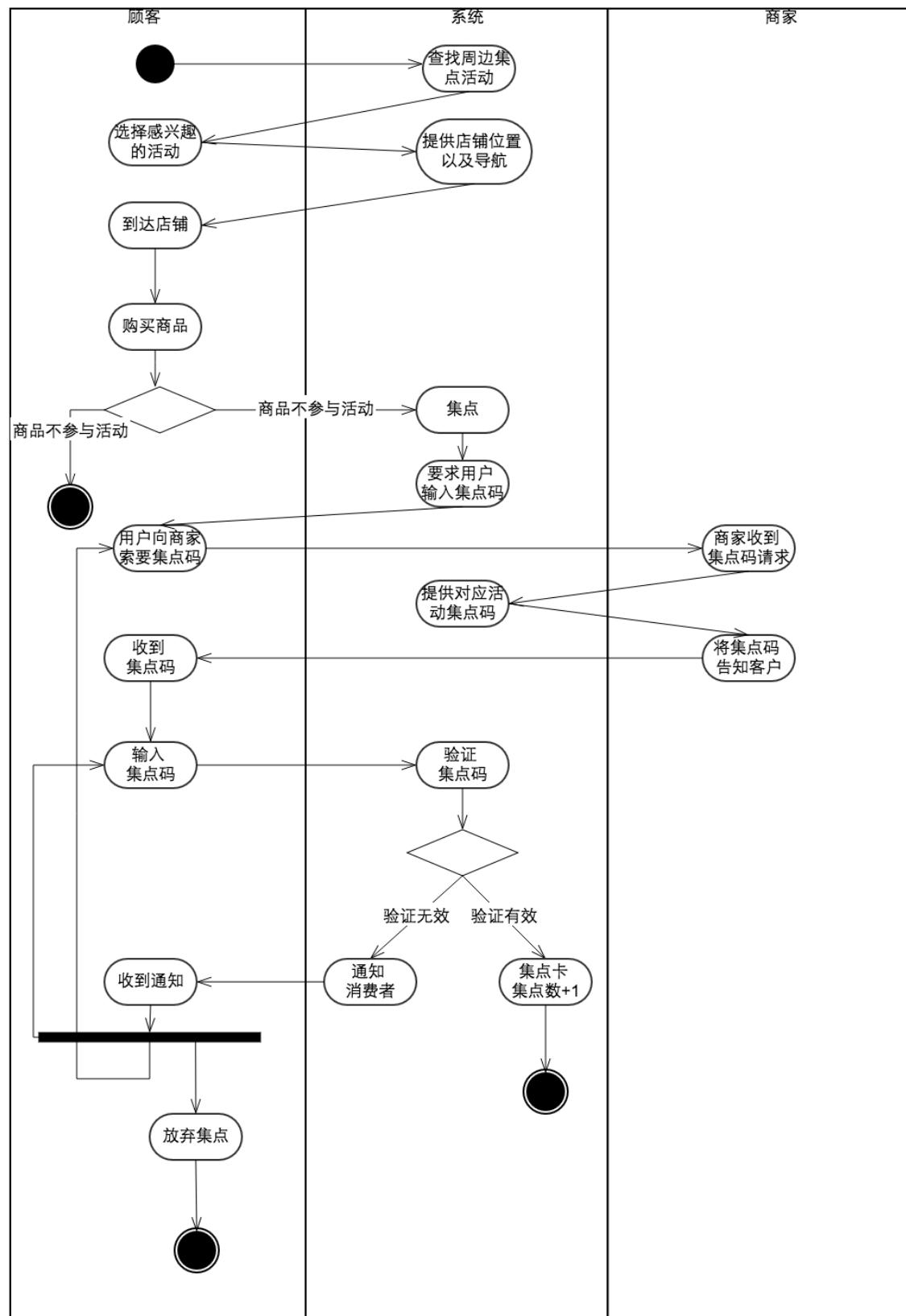


图 3.4 活动图：集点

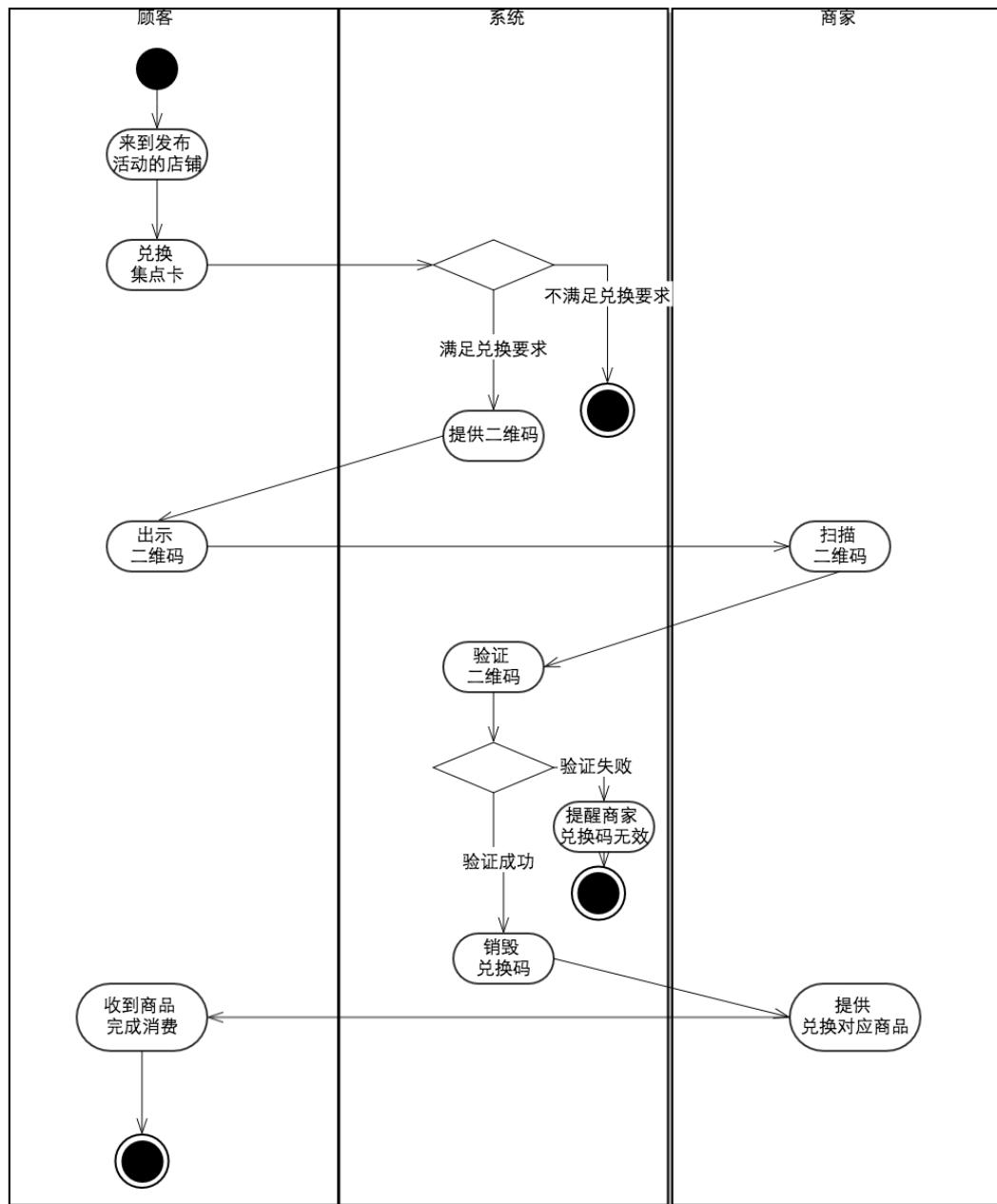


图 3.5 活动图：兑换

图 3.4 以活动图的形式描述了整个集点的过程。从消费者角度，从用户产生消费的需求，并通过系统找到活动中的店铺、来到店铺完成消费。向商家获取对应活动的集点码并输入至系统中，系统验证集点码合法性，并根据对应结果做出对应处理。从商家角度，商家只需要向系统索要集点码并提供给客户即可。

图中省略了商家创建店铺、发布活动的流程，由于其流程为一次性流程，且相对简单，在此略过。

图 3.5 以活动图的形式描述了整个兑换的过程，兑换的前置条件是集点，

所以此处省去了应用引导用户到达店铺的过程，开始于消费者已经来到店铺，消费者使用系统在卡包中找到可以兑换的集点卡选择兑换生成二维码，商家使用系统扫描二维码并完成消费。

3.2.5 系统的非功能性需求

安全性：

1. 系统只允许商家对自己的店铺以及活动进行操作，包括编辑、获取集点码以及兑换。
2. 每个集点码只能使用一次，一旦使用成功立刻销毁。
3. 每个兑换码能且只能使用一次，既要维护消费者的权益也要维护商家的利益。

可维护性：

1. 当应用需要扩展新的功能升级时，不影响老版本的客户数据。
2. 如果存在严重的数据错误，管理员可以快速有效地对突发情况进行处理可靠性。
1. 在客户端与服务端通信的时候，如果遇到网络故障问题，系统不能崩溃。
2. 在遇到网络较差的情况，数据即使存在丢失也必须提供找回途径。
3. 项目尽可能排除内存泄露，崩溃率低于 1/1000。

约束

1. 本系统基于 iOS 平台开发，适用于使用 iOS 8 以上系统的任何 iOS 设备。

3.3 系统的概要设计

3.3.1 基于 POP（面向协议编程）的 MVVM 设计

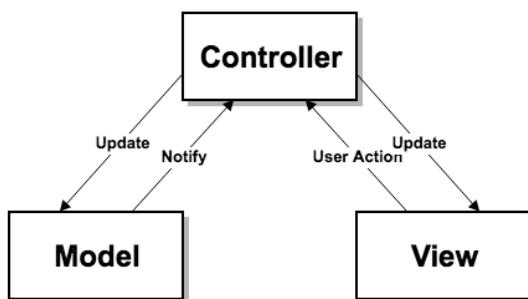


图 3.6 iOS 传统 MVC 架构

图 3.6 为苹果官方所推荐的也是 iOS 开发最传统的设计模式 MVC，在 iOS 的 MVC 中 Model 层若发生变化，通过 Notification 或者 KVO 的方式告知 Controller 从而更新 View，Controller 直接持有 Model，View 收到用户操作通过代理的方式告知 Controller，从而对 Model 进行操作，在 iOS 的 MVC 中 View 与 Model 是解耦的，而 Controller 层则会承担大部分的业务，导致其庞大臃肿。

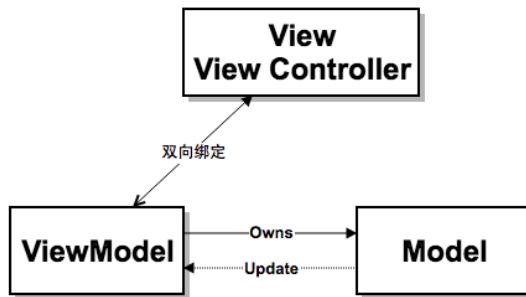


图 3.7 传统的 MVVM 架构图

上图为 iOS 中使用的 MVVM 模式，由于 iOS 中的 Controller 全称为 UIViewController 即视图控制器，在 MVVM 中，Controller 只需要控制视图，其余的职责则委托给了 ViewModel 进行，ViewModel 来负责处理数据请求、业务逻辑这些事情。然而在传统的 MVVM 模式中，View 和 ViewModel 之间往往使用双向绑定的技术，View 的变化会自动反映在 ViewModel 中而反之亦然。

虽然 MVVM 很大程度解放了 Controller，但是为了实现这种双向绑定，在 iOS 开发中往往会使用一些比较重的类似 Reactive Cocoa 的框架，带来了可读性和可调试性上的一些新问题，同时如果使用不当 ViewModel 一样也会臃肿不堪。

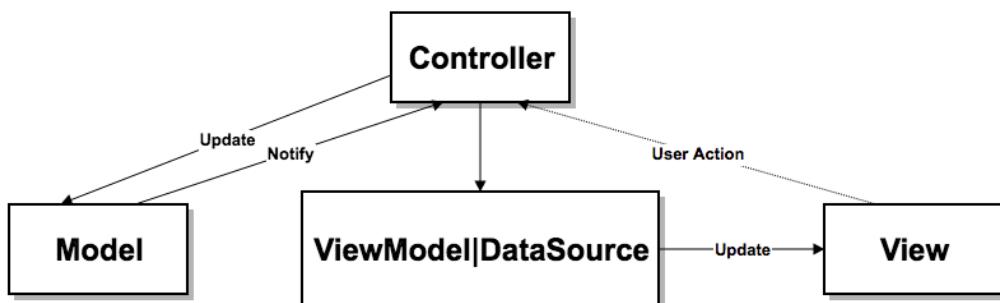


图 3.8 本项目中使用的基于 POP 的 MVVM 架构图

本项目并没有打破 iOS 开发原有的 MVC 设计模式，但是从一定程度上汲取了 MVVM 模式的优势，通过 ViewModel 或 DataSource 来抽离 Controller 的部分职责，

由于 iOS 开发中 API 的限制——每个界面必须由一个 `UIViewController` 所控制，所以项目中并没有也没有办法完全抛弃 `UIViewController`，而是如图所示，`UIViewController` 依旧控制视图的显示与跳转，处理界面响应，处理数据请求与获取，而数据的处理与绘制与重绘交由 `ViewModel` 或者 `DataSource` 来执。

其次这里的 `ViewModel` 是基于 POP 的即 Protocol Oriented Programming ——面向协议编程，`ViewModel` 不直接依赖于 `Model` 而是依赖于协议的组合，通过 Swift 2.1 的新特性——协议扩展来完成协议默认实现的配置，最大程度地减少绘制以及渲染界面所需要的代码量，增加代码的复用性以及代码的扁平度。具体将在第四章详细设计中举例说明。

由于应用使用了 `LeanCloud` 的数据存储服务来替代服务端及数据库，所有的 `Model` 都继承了 `AVObject`，并通过 `LeanCloud` 数据存储的 SDK 完成数据请求、存储、修改以及删除，所以整体应用的架构如下图所示：

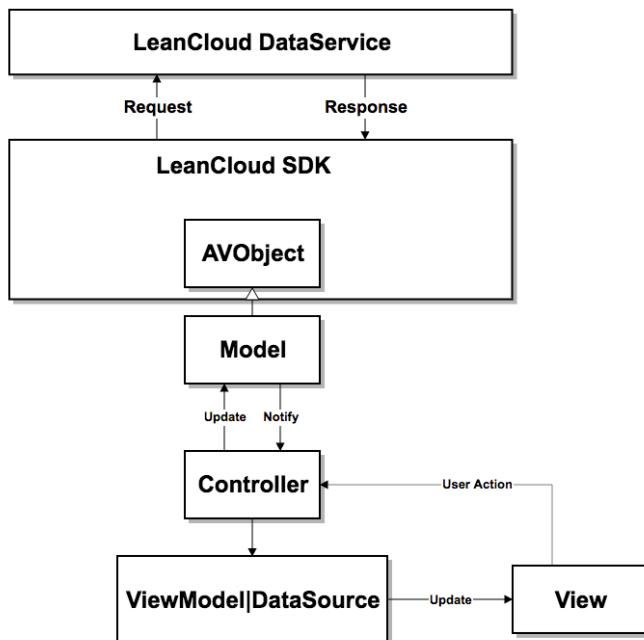


图 3.9 本项目完整框架图

3.3.2 接口概述

虽然使用了 `LeanCloud` 的数据服务替代服务器，但是整个系统依然算是一个 C/S 架构，所有的数据接口均由 `LeanCloud` 的 SDK 进行封装，只留下最简单的增删查改接口供开发者调用，项目通过为 `Model` 写扩展来完成数据操作的进一层封装，从而将其与客户端逻辑想隔离，方便日后维护。

3.3.3 模块单元结构图

如图 3.10 所示，本系统——基于 iOS 的集点卡应用根据系统需求将系统主要分为 5 个模块，每个模块中均使用了 3.3.1 中所说的 MVVM 架构，首页活动列表为例：ActivityListViewController 作为 Controller，View 通过 ActivityStoryboard.storyboard 来进行配置，ActivityListDataSource 来抽离列表渲染逻辑，Activity 作为 Model，ActivitySimpleViewModel 作为 Model 的抽象负责渲染每个列表项。除了这五个功能模块，另有一个模块专门用于通用的工具类，包括图片选择、地图导航、兴趣点搜索、持久化数据存储、弹框封装。

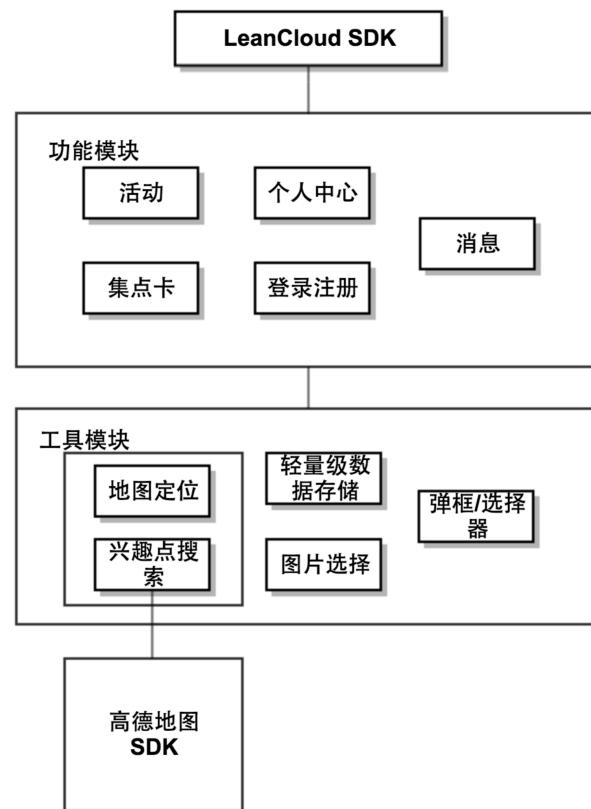


图 3.10 模块单元结构图

3.3.4 持久化机制

由于项目需求决定了项目数据的多变性，在客户端，项目并没有使用数据库来进行数据的存储，每次启动直接根据用户所在地点或预设地点拉取对应数据，并根据用户实际操作拉取实时数据。对于一些轻量级的变化较少的数据，系统通过 iOS 中提供的 API——**NSUserDefaults** 进行存储，并且通过一个工具类将原

有 API 进行封装，方便项目使用。

项目使用 **LeanCloud** 的数据存储服务替代服务器，所有数据均存在于 **LeanCloud** 对应应用的数据库中，并通过 **LeanCloud** 的 SDK 来实现客户端和数据库的交互。项目中所涉及到的图片资源通过 **LeanCloud** 的 **AVFile** 来封装并上传至 **LeanCloud** 服务器并保存，每一张图片在数据库中均有字段保存。

3.3.5 安全机制

该项目在业务层面的安全要求主要包括：

1. 用户认证，在用户认证方面，系统通过子类化 **LeanCloud** 的 **AVUser** 类，并通过其注册、登录、自动保存等机制来完成用户的认证。用户通过手机短信验证码来完成注册，**LeanCloud** 会自动在客户端将用户的密码进行加密并保存，每次启动会自动登录。
2. 用户权限，本系统在 **AVUser** 的子类中新增 **userType** 字段，每次进入活动以及店铺，均会对用户类型以及活动、店铺主人进行判定，从而保证为对应的用户执行对应的行为。
3. 兑换券、集点码防止重复使用，每次的兑换码和集点码在正常使用后均会自动销毁，防止重复使用。每次的验证必须经过服务器，从而确保数据的实时性。**LeanCloud** 作为一家用众多用户的公司，能够有效处理本项目规模情况下的多用户访问。同时，**LeanCloud** 提供的数据存储服务允许后台脚本的执行，系统会定期执行脚本，删除那些已经生成但是超过 24 小时没有使用的兑换券与集点码。

3.3.6 数据库概要设计

本项目数据库表的设计尽可能地满足了第三范式，基于性能与开发便捷考虑允许一定的数据冗余，数据表包括：**User**、**Activity**、**Card**、**Like**、**Comment**、**File**、**Shop**。由于使用了 **LeanCloud** 的数据存储服务，每一张表都包含以下标准字段。

字段名	字段说明
objectId	STRING （表唯一 ID）
ACL	ACL （特殊格式）用于管理用户权限

createAt	DATE (数据创建时间)
updateAt	DATE (数据更新时间)

表格 3.9 数据库通用字段

3.4 本章小结

本章节介绍了基于 iOS 的集点卡应用的需求分析和系统概要设计。

在需求分析这一小节，本章先是对需求进行一个概要的描述，针对功能性需求，先分析了系统的业务需求，再通过用例图、用例描述的方式分析了系统的用户需求，最后通过顺序图和活动图的方式描述了系统的系统需求；针对非功能性需求，本章从安全性、可维护性、可靠性、约束这四个角度来描述了系统的非功能性需求。

在概要设计这一小节，本章使用的是基于面向协议编程的 MVVM 架构，先简要介绍了 iOS 开发传统的 MVC 模式以及近年来流行的 MVVM 模式，从而引入本项目基于 MVC 模式，吸收了 MVVM 模式中的一些思想，同时利用了 Swift 2.1 的新特性尝试的一种通过 ViewModel 或者 DataSource 抽离 Controller 部分业务的新的模式，在详细设计章节会对该模式做一个更加详细的介绍。

在概要设计这一小节，本章还提供了项目的接口概述、模块单元图、持久化机制、安全机制、数据库概要设计。

第四章 活动模块的详细设计与实现

4.1 基于 iOS 平台的集点卡应用活动模块的概述

本应用基于 iOS 8.0 及以上系统设计并实现，并使用 Swift 2.2 实现，核心功能即为提供一个商家、客户进行集点活动的平台，其衍生的必要功能包括商家创建店铺发布活动，用户获取活动到达店铺以及双方的互动评论等。

本应用是一个典型的“O2O”应用，其设计和开发过程也遵循一个典型的移动应用开发流程，属于敏捷开发。其在设计和实现流程中也存在一些基于新语言新特性而诞生的新的尝试。在下面两小节将分别对该系统的设计和实现进行具体介绍。

由于本应用规模较大，本章节并不会对所有的模块的详细设计与实现都做详细介绍，而是挑选了“活动”这一个比较典型的模块。活动模块的首页是一个具有一个选项卡的列表页，用户通过选项卡来选择查看“集点卡应用”或者不包括集点事件的“促销活动”，通过选择列表进入活动详情，在活动详情中，用户可以查看活动的详情并进行针对该活动的相关操作——包括集点以及导航，集点操作会直接跳转到集点模块，而导航则会使用到第三方的高德地图服务。在活动详情页面，用户还可以进入活动对应的店铺详情页面，查看店铺详情针对店铺进行评论的操作。除了这些主要功能之外，在活动的首页还集成了定位的模块，一方面实现了自动定位一方面也提供了用户手动选择所在位置的入口

4.2 基于 iOS 平台的集点卡应用的详细设计

4.2.1 活动模块类设计

图 4.1 描述了活动模块中活动列表部分的类关系图，其中省去了各类与 UIKit 即 iOS 原生 API 组件类的关系，其中 `ActivityRootViewController`、`ActivityListViewController` 均继承了 `UIViewController`（视图控制器），`ActivityRootViewController` 因为存在一个选项卡（SegmentControl）而遵循了 `SegmentControlDelegate` 协议来处理选项卡选择的回调。`ActivityListViewController` 由于是一个列表包含一个 `UITableView`（iOS 中的列表组件），遵循了 `UITableViewDelegate` 协议，用于处理 `UITableView` 的事件回调，而其持有的 `ActivityListDataSource` 则作为 `UITableView` 的数据提供者，遵循了 `UITableViewDataSource` 协议，用于提供列表项的数据以及页面的渲染。

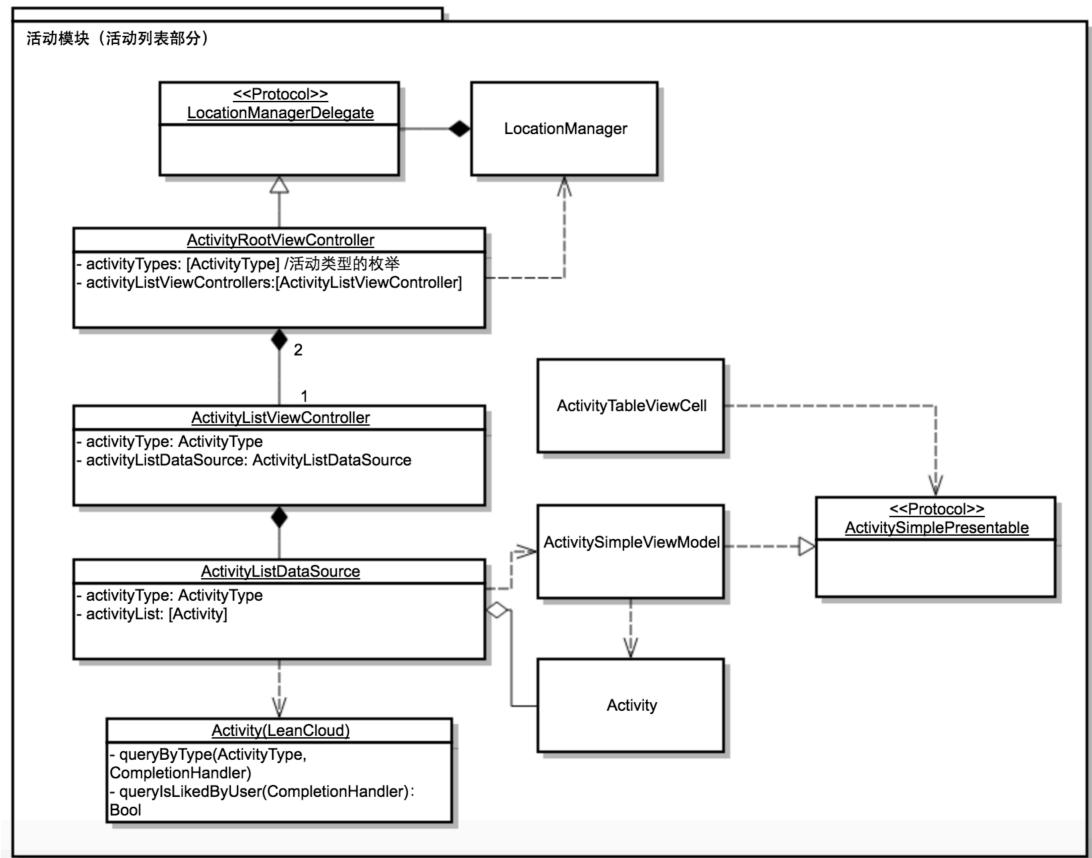


图 4.1 活动列表类图

注：Protocol 即协议，为 Objective-C 和 Swift 中的表示，类似于 UML 图中的接口，所以在图中用接口来表示。而 **Activity(LeanCloud)** 在 Swift 中代表针对 **Activity** 这个类写的扩展，在 UML 2.0 中并没有针对该关系的表示，所以这里直接使用另外一个实体来表示。

ActivityRootViewController 中持有了一个活动类型枚举的数组用于控制选项卡，并持有了一个 **ActivityListViewController** 的数组用于根据选项卡来控制显示哪个列表，可以说 **ActivityRootViewController** 是一个容器视图。**ActivityRootViewController** 另外依赖了 **LocationManager**（一个用于定位的工具），并实现了 **LocationManagerDelegate** 的协议用于处理地点回调，从而显示当前位置，进而根据位置进行查询。

ActivityListViewController 中将数据查询的工作委托给了 **ActivityDataSource**，**ActivityDataSource** 通过参数（类别以及地点）调用 **Activity** 类的扩展 **Activity(LeanCloud)** 来查询对应的活动并针对各个活动查询用户是否点赞，进而刷新页面。

ActivityTableViewCell 继承了 **UITableViewCell**，即每个列表项，然而 **ActivityTableViewCell** 并没有直接依赖于 **Activity**，甚至没有依赖于

`ActivityViewModel`, 而是依赖于协议 `ActivitySimplePresentable`, 从字面意思上来说就是可以有从来表示简单活动信息的对象, 然而协议 `ActivitySimplePresentable` 实际上并不是一个实际的协议, 而是可以理解为一个用宏表示的多个协议的组合, 具体实现方案会在 4.3.4 中具体描述。

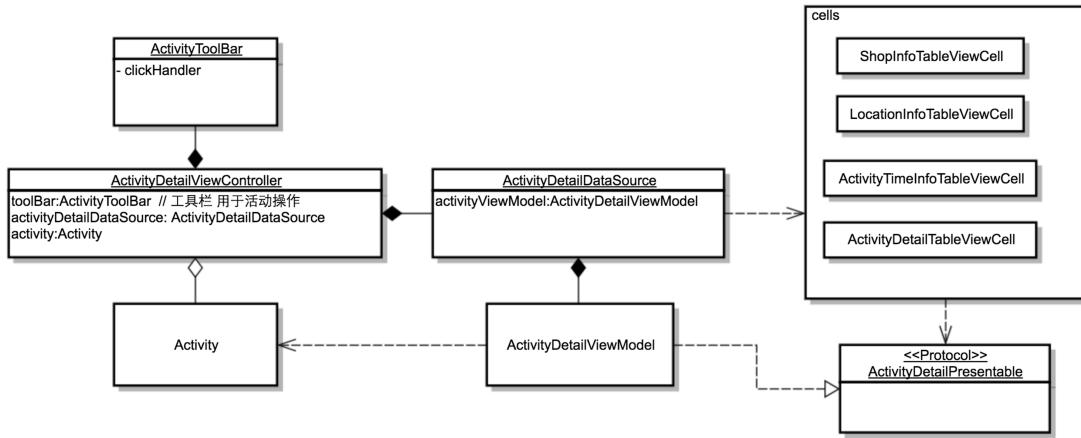


图 4.2 活动详情类图

注：图中所有的组合和聚合均为一对一的关系

图 4.2 描述了活动详情页面的类关系图，活动列表页依赖于活动详情页，然而活动列表页到活动详情页的跳转通过 Storyboard 的 segue 来实现，具体见 4.3.1。所需要做的只是在跳转前将对应的 Activity 传递给详情页。

`ActivityDetailViewModel` 相比于 `ActivitySimpleViewModel` 来说信息更加完整，均通过 `Activity` 初始化。通过将 `Model` 转换成 `ViewModel` 的过程中对数据进行了处理，直接转换成可以在页面上显示的数据。

活动详情页面也是一个可以滚动的 `TableView`, 其设计方式与列表页统一，均使用了一个 `DataSource` 来抽离数据上的处理和界面的渲染，在详情页需要多个类型的 `cell`, 但是这些 `cell` 同样不依赖于 `Activity` 和 `ActivityDetailViewModel`, 而是依赖于一个协议的组合 `ActivityDetailPresentable`。

由于活动详情页存在一个工具栏，而工具栏并非静态固定的，会根据用户权限的不同而产生变化，所以将工具栏抽离成一个特定的 `View` 并通过代码而非 `Xib` 文件来配置 `UI` 元素，从而更加灵活。详情页通过对 `ToolBar` 传入 `Handler` 的闭包，当按钮被点击，`ToolBar` 根据被点击按钮的类别来执行闭包，来完成对用户点击事件的处理。

商店页面同活动详情页面设计大同小异，在此不做赘述。

4.2.2 数据库详细设计

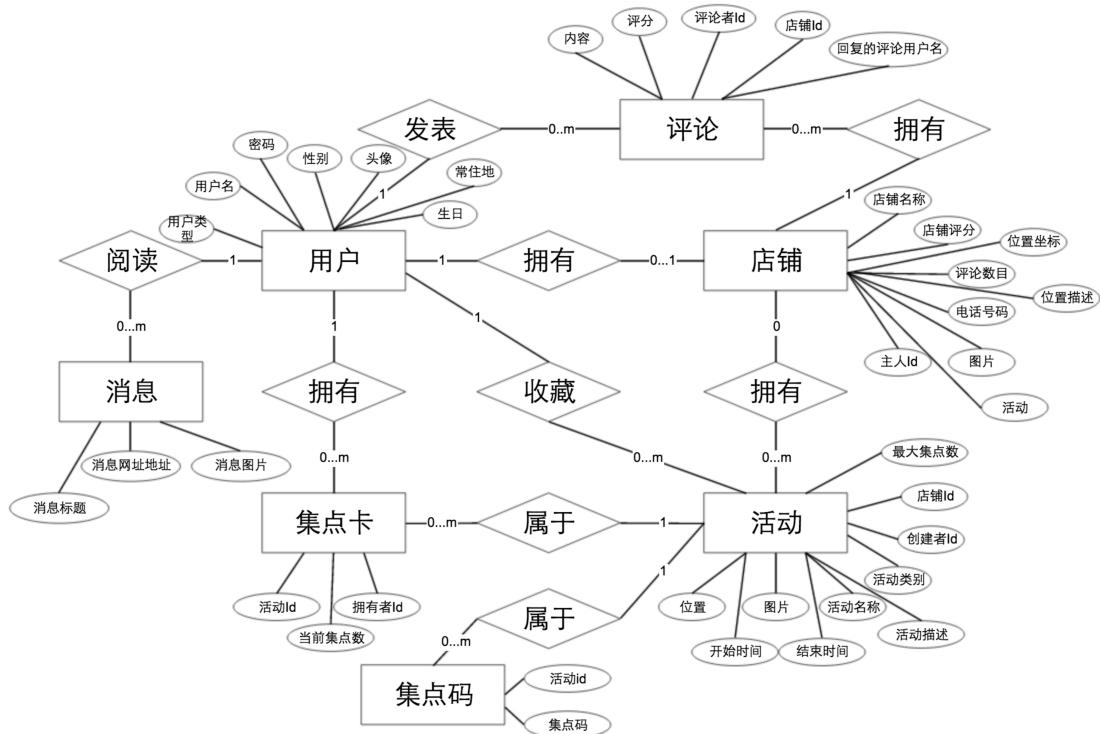


图 4.3 数据库 ER 图

整个应用的数据库设计遵循上图，其中一共涉及到 9 张表，分别为用户表 User、店铺表 Shop、活动表 Activity、评论表 Comment、集点卡表 Card、消息表 Message、消息阅读记录表 Read、收藏表 Like，集点码表 Code，另有一张 File 表存储的是文件信息并没有在图中体现，一些通用的字段已经在 3.3.6 中列出也未体现。

由于使用 LeanCloud 的数据存储服务，其数据库已经在基本的数据库上做了更上一层的封装，比如允许存储列表、自动进行外键绑定等，为数据库设计和开发过程提供了不少方便。但是在开发过程中，由于没有服务端的参与，在获取一些数据的同时还需要获取对应关联数据中的内容，会影响请求的效率和请求此处，所以添加了一些数据冗余，比如在活动表中虽然可以通过活动所属的店铺查询到店铺的地址名称和位置，但是为了方便直接根据位置查询活动同时在请求数据时能够直接返回活动的地址，所以这里增加了两个冗余数据。

4.2.3 界面原型设计

本项目的 UI 全部使用 Sketch 进行设计，并使用了大量 iOS 8 的原生组件。界面分为：活动列表界面、活动详情界面、店铺详情界面、集点卡界面、个人设置界面、店铺创建界面、位置选择界面、兑换界面。



图 4.4、4.5 界面原型：活动列表，界面原型：活动详情

整个界面构成由 4 个 TabBar 构成，图 4.4 为活动列表，属于第一个 Tab，导航栏设置了选择地点的入口、搜索的入口以及活动列表选项卡。每一个列表项包含活动的宣传照、活动名称、地址、收藏数目。选择列表项即可进入活动详情。

图 4.5 为活动详情页，用户可以看到店铺的更多信息，同时允许进行一系列的操作，包括收藏、分享、导航、查看店铺。如果是集点活动，则可以集点，如果是商家发布的活动，则可以进行兑换和生成集点码的服务。

图 4.6 为商家视角的店铺，包含店铺的各种信息，并允许商家进行编辑和发布活动。在用户视角，下方的按钮变为评论，同时没有编辑的按钮，联系方式隐去，取而代之的是在店铺旁边添加拨打按钮，允许用户直接联系店铺。



图 4.6 界面原型：店铺详情



图 4.7、4.8 界面原型：集点卡，界面原型：消息列表

图 4.7 为卡包界面，为第二个 Tab，允许用户快速地查看自己已经集点的、正在集点的卡片，并更加快捷地进行集点操作，卡片间的切换使用左右滑动的操作。

图 4.8 为消息界面，为第三个 Tab，用于显示一些新消息以及，点击每一个选项会进入一个 WebView，该消息由后台推送，暂时不支持商家发布。

图 4.9 为个人设置界面，显示用户的个人信息，并提供编辑的入口，另外列表项提供一些其他的非核心的功能，非商家通过这里注册成为商家，商家通过这里进入自己的店铺。点击右上角的编辑按钮

图 4.10 描述的是兑换界面的设计风格，在集点卡如果收集完成可以点击下方的兑换按钮，系统会生成一张二维码，商家通过扫描二维码完成兑换。

图 4.11 为编辑店铺以及选择店铺位置的界面设计，在选择店铺位置的界面，系统首先自动定位用户位置，并搜索附近商家，默认选取第一家，用户可以在待选店铺中选择自己的店铺或者通过搜索的形式搜索自己的店铺。

图 4.12 为编辑活动的界面，该列表会根据活动类型的变化隐去或显示集点数这一栏，从而实现不同类别的活动的创建或者编辑。



图 4.9 界面原型：个人中心



图 4.10 界面原型：兑换



图 4.11 界面原型：店铺编辑



图 4.12 界面原型：活动创建

4.3 基于 iOS 平台的集点卡应用活动模块的实现

4.3.1 View 实现

本系统的 View 层主要使用三种方式进行实现，包括 Storyboard、Xib 以及代码构建。

1. 使用 Storyboard 构建相互间有静态关联的视图。以活动模块为例，对于活动模块的所有视图，我们采用 Storyboard 构建所有视图并通过 Storyboard 中的 Segue 来完成界面之间的切换。

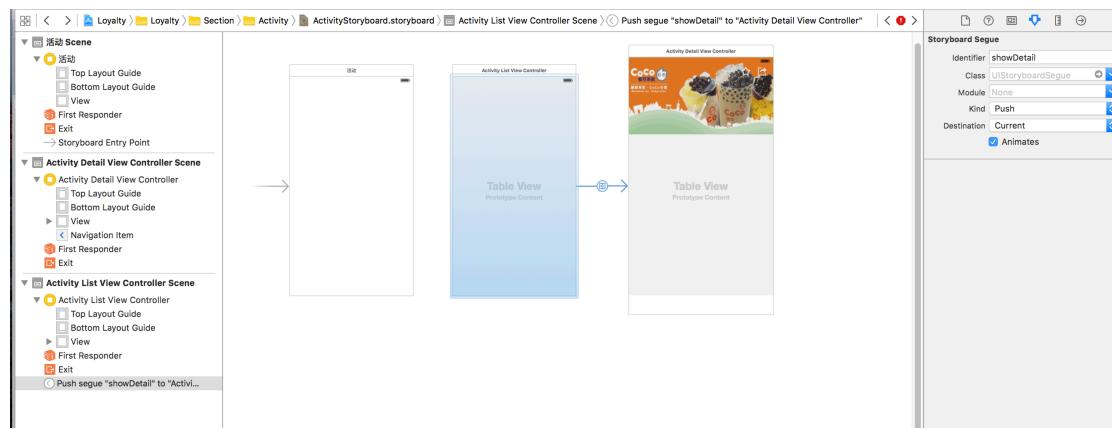


图 4.13 Storyboard: 活动模块

图 4.13 是活动模块的 storyboard——ActivityStoryboard.storyboard，这是一个非常简单的 Storyboard，只有 3 个视图，分别对应着为 4.2.1 中提到的 ActivityRootViewController 、 ActivityListViewController 以及 ActivityDetailViewController ，其中 ActivityRootViewController 被设置为 Storyboard 的入口。

图 4.14 中选中了两个视图中间的连线，我们称之为 Segue，对于使用 Storyboard 搭建的视图来说，可以通过 Segue 来直接实现界面的跳转，如图我们给 Segue 定义了 Identifier——唯一标识符为 showDetail，且切换方式定位 Push。我们可以通过编写图 4.15 中的代码来利用 segue 完成界面跳转。

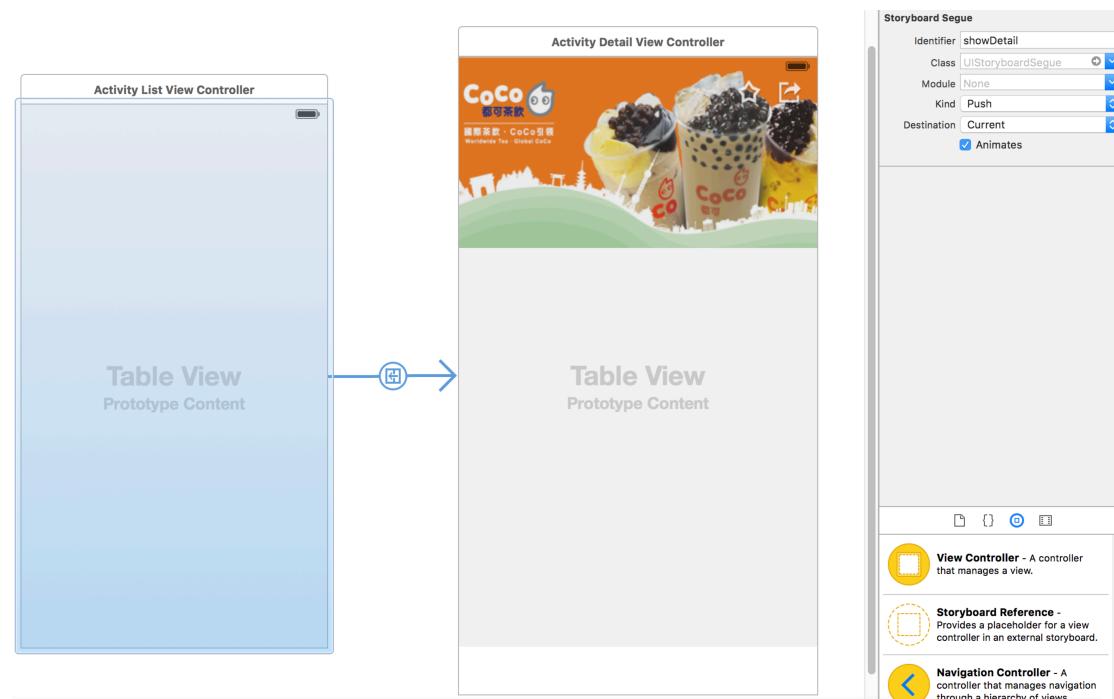


图 4.14 Storyboard: Segue 界面跳转

在代码中我们这样实现：

```
//当列表项被选择时候 Delegate 收到的回调
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    .....//做一些操作
    self.performSegueWithIdentifier("showDetail", sender: indexPath)
}

// 在使用 segue 进行跳转前会执行的方法
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    //判断 segue 的标识符是否为 showDetail
    if segue.identifier == "showDetail" {
        //获取目标视图并传入参数 Activity
        let nextViewController = segue.destinationViewController as? ActivityDetailViewController
        if let indexPath = sender as? NSIndexPath {
            nextViewController?.activity = self.activityListModel?.activityList?[indexPath.row]
        }
    }
}
```

图 4.15 利用 Segue 进行视图跳转的代码实现

在 `ActivityListViewController` 中，在列表项被选择的回调中执行 `performSegue` 方法即可完成跳转，如果需要传入参数，可以在 `prepareForSegue` 中进行设置，这是一个从父类 `UIViewController` 中继承下来的方法。

2. 使用 **Xib** 构建那些独立的会被多处调用的视图组。针对一些在多个地方出现的界面，我们并不使用 **Storyboard** 而是采用 **Xib** 进行构建，比如店铺详情界面，在系统中，我们可以通过活动详情界面进入店铺详情，也可以通过设置界面中“我的店铺”进入店铺详情，所以项目使用 **Xib** 来进行店铺详情界面的配置。

Xib 同 **Storyboard** 最大的区别之一在于没有办法配置视图与视图之间的关系，所以界面的跳转需要通过代码手动来书写。而视图组件的配置。

在项目中，**Xib** 使用最多的是单个 **UI** 组件的配置，以 `UITableViewCell` 及列表项居多。

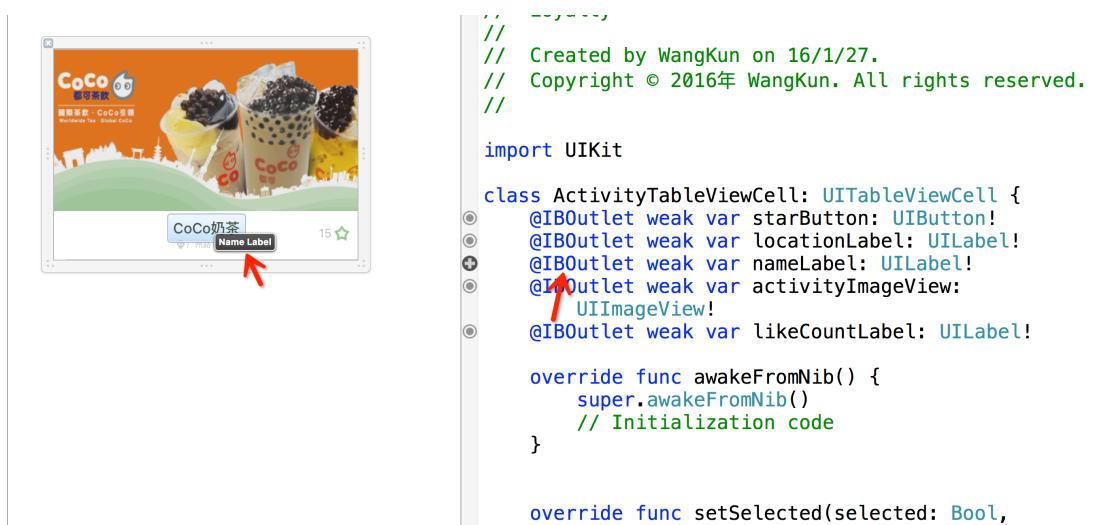


图 4.16 使用 **Xib** 组件关联的实现

图 4.16 为一个简单的 **Cell** 的 **Xib** 实现，很容易看出，这是首页活动列表的 **Cell**，每一个 **Xib** 文件存在一个 **Owner** 即视图的拥有者，对于这个 **Cell**，它所对应的类 `ActivityTableViewCell` 即为视图的拥有者，我们可以将这个 **Xib** 上的 **UI** 组件通过 **IBOutlet** 的形式拖拽并映射到 `ActivityTableViewCell`。

图中，将鼠标移动至 `nameLabel` 处，**Xib** 中的标签“CoCo”奶茶高亮显示，即说明两者完成了绑定。



图 4.17 Autolayout 约束制定的实现

在使用 Xib 或者 Storyboard 的过程中，项目均使用 Autolayout 来配置组件的位置，以图中这张图片为例，它需要置顶于列表项并且左右无空隙，便设置与 SuperView 的 Trailing Space, Leading Space, Top Space 均为 0，但是这三个属性只能控制其宽度和父容器一样没有办法决定其高度，这里通过设置长宽比为 15:8 来限定其高度，不适用固定高度是因为首页列表项的高度会根据屏幕尺寸的差别来计算大小，因此适用固定长宽比的方式最为合理。

看右侧的约束，最下方还有两个，这两个约束对于图片来说是可有可无的，然而这两个约束表示的是整个容器中的其他组件与其的位置关系。

当然除了这些简单的位置关系外，我们还可以设置组件水平或垂直居中，设置组件与组件之间高度宽度比，更进阶的可以通过设置不同的弹性系数来设置约束的优先级用于处理那些可能会存在冲突的约束。

3. 使用代码编写无法静态配置的视图组件。针对一些无法静态配置的视图组件，比如活动详情界面底部的工具栏，会根据权限以及活动类别的不同而需要显示不同的内容，所以必须使用代码来完成 UI 的绘制。

```
class ActivityToolBar: UIView {
    var clickHandlers:[()->Void] = []
    //传入按钮颜色的数组（提供默认参数）、图片数组、按钮标题数组以及回调数组
    func updateView(colorArray:[UIColor] = [UIColor.globalLightGreenColor(),UIColor.globalLightBlueColor()],imageArray:[String]!,titleArray:[String]!, clickHandlers:[()->Void]) {
        var buttonArray:[UIButton] = []
        let buttonWidth = self.frame.size.width / CGFloat(titleArray.count)
        for i in 0...imageArray.count-1 {
            let button = UIButton(type: UIButtonType.Custom)
            button.setImage(UIImage(named: imageArray[i]), forState: UIControlState.Normal)
            button.setTitle(titleArray[i], forState: UIControlState.Normal)
            button.addTarget(self, action: #selector(buttonClicked), forControlEvents: .TouchUpInside)
            buttonArray.append(button)
        }
        self.addSubview(buttonArray[0])
        self.addSubview(buttonArray[1])
        self.addSubview(buttonArray[2])
    }
}
```

```

button.setTitle(" \(titleArray[i])", forState: UIControlState.Normal)
button.backgroundColor = colorArray[i]
self.addSubview(button)
button.tag = i
button.addTarget(self, action: #selector(ActivityToolBar.onButtonClicked(_:)), forControlEvents: UIControlEvents.TouchUpInside)
buttonArray.append(button)
constrain(button, self) {
    button, view in
    button.left == view.left + buttonWidth * CGFloat(i)
    button.top == view.top
    button.height == view.height
    button.width == buttonWidth
}
}
self.clickHandlers = clickHandlers
}

func onButtonClicked(sender: UIButton) {
    let tag = sender.tag
    clickHandlers[tag]()
}
}

```

图 4.18 ActivityToolBar 组件配置的代码实现

有了第三方库的支持，图 4.18 中的代码比较简单易懂，但是却比较冗长，可以看出使用代码配置 UI 虽然可以达到跟 Storyboard 和 Xib 一样的效果，但是在更多情况下，UI 组件的配置还是使用 Storyboard 或者 Xib 配置起来更为方便，且所见即所得。

4.3.2 Model 实现

项目中所有的 Model 都继承了 AVObject 父类，AVObject 是 LeanCloud 数据存储 SDK 中定义的需要存储在数据库中的内容都需要继承的父类，提供了增删查改等一系列数据处理方法，同时 Model 必须实现 AVSubClassing 接口，从而告知 LeanCloud SDK 该 Model 在数据库中对应哪张表。

```

class Activity: AVObject, AVSubclassing {
    @NSManaged var activityTypeString: String?
    @NSManaged var avatar: AVFile
}

```

```

@NSManaged var location:AVGeoPoint
@NSManaged var locationName:String
@NSManaged var shopId:String
@NSManaged var creatorId:String
@NSManaged var loyaltyCoinMaxCount:Int
@NSManaged var likeCount:Int
@NSManaged var name:String
@NSManaged var startTime:NSDate
@NSManaged var endTime:NSDate
@NSManaged var activityDescription:String?

var shopInfo:Shop?
var like:Like?
var isLikedByMySelf:Bool?
convenience
init(shop:Shop,name:String,startTime:NSDate,endTime:NSDate,description:String,avatar
:AVFile,activityType:String,loyaltyCoinMaxCount:Int) {
    self.init()
    self.activityTypeString = activityType
    self.shopId = shop.objectId
    self.creatorId = shop.userId ?? ""
    self.name = name
    self.startTime = startTime
    self.endTime = endTime
    self.loyaltyCoinMaxCount = loyaltyCoinMaxCount
    self.activityDescription = description
    self.avatar = avatar
    self.location = shop.location
    self.locationName = shop.locationName
}

var activityType:ActivityType? {
    get {
        guard let activityTypeString = self.activityTypeString else { return nil }
        return ActivityType(rawValue: activityTypeString)
    }
}

// AVSubClassing 协议方法，返回对应的表名
static func parseClassName() -> String! {
    return "Activity"
}
}

```

图 4.19 ActivityModel 的代码实现

图 4.19 中代码为 Activity 这个 Model 的代码，它的属性分 3 类，一类为标记了 @NSManaged 的属性，在 Swift 中，标记了 NSManaged 标签的属性会在编译时候自动加上 @objc 标记，LeanCloud 的 SDK 在将从网上获取到的数据转换成 Model 的时候使用了 Objective-C 的 Runtime 特性，为了能够在 Swift 中正常完成数据的填充，就必须加上这个标记，代表这些属性的内容会被动态生成。

而没有加上 @NSManaged 标签的属性即不会被动态填充，这里 shopInfo、like、isLiked 都需要在进一步的请求后才能被填充。

第三种属性为动态属性，它并不会被存储于对象中，每次调用均会执行对应方法获取返回，比如数据库中无法存储自定义的枚举，而使用过程中需要用到枚举类型来确定活动是什么类型，所以定义了动态属性 activityType，该属性根据 activityTypeString 来返回枚举类型。

在 Swift 中，有三种构造器标记，分别为 Designated、Convenience 和 Required，Designated 的初始化方法的地位是不可撼动且必须调用的，继承了 AVObject 的 Model 无法重写 Designated 的构造器，否则会影响到其数据的自动填充，所以这里通过编写 Convenience 构造器来允许在项目中手动创建 Activity。加入了 Required 标签的初始化方法则确保子类对其进行实现。

```
extension Activity {
    func queryIsLikedBySelf(completionHandler:(like:Like?)>Void){
        if self.isLikedByMyself != nil {
            completionHandler(like: self.like)
            return
        }
        guard let userId = UserInfoManager.sharedManager.currentUser?.objectId else
        { return }
        let query = Like.query()
        query.whereKey("userId", equalTo: userId)
        query.whereKey("activityId", equalTo: self.objectId)
        query.getFirstObjectInBackgroundWithBlock { (object, error) in
            self.like = object as? Like
            self.isLikedByMyself = (self.like != nil)
            completionHandler(like: self.like)
        }
    }
    func queryShopInfo(completion:Shop->Void) {
        //获取 shop 的查询对象
        let query = Shop.query()
        //根据 shopId 查询 shop
    }
}
```

```

query.getObjectInBackgroundWithId(self.shopId) { (shop, error) in
    if let shop = shop as? Shop {
        //回调
        self.shopInfo = shop
        completion(shop)
    }
}

static func query(nearGeoPoint:CGPoint?, type:ActivityType, completion:[Activity] -> Void) {
    let query = Activity.query()
    query.limit = 10
    if let nearGeoPoint = nearGeoPoint {
        query.whereKey("location", nearGeoPoint: AVGeoPoint(latitude: Double(nearGeoPoint.x), longitude: Double(nearGeoPoint.y)))
    }
    query.whereKey("activityTypeString", equalTo: type.rawValue)
    query.findObjectsInBackgroundWithBlock { (activityObject, error) in
        var activityArray:[Activity] = []
        for object in activityObject {
            if let activity = object as? Activity {
                activityArray.append(activity)
            }
        }
        completion(activityArray)
    }
}
}

```

图 4.20 Activity API 扩展的代码实现

在 3.3.1 的框架图中,可以看出 Model 直接与 LeanCloud 的 SDK 进行交互,同 LeanCloud 的 SDK 共同构成了项目的网络层,然而这些操作数据的方法并没有编写在 Model 中,一方面是为了方便维护,另一方面是为了抽离职责。

在 Swift 和 Objective-C 中,有一种特性称之为扩展,在 Objective-C 中叫做 Category,而在 Swift 中称为 Extension,后者的名称显得更为确切一点,即针对原先的类进行扩展,它类似于继承但是又不是新的子类。

我们将数据方面的操作均写在对应 Model 的 Extension 中,比如在获取活动列表的时候我们并不需要知道其店铺信息,然而进入详情后,我们需要获得它的店铺信息,这个时候便调用扩展中的 queryShopInfo 方法,填充活动所在的店铺的具体信息。系统中的所有 Model 都使用本体+Extension 的方式来抽离固有职责以及扩展功能。

4.3.3 Controller 实现

由于使用了 `ViewModel` 或者 `DataSource` 抽离绘制和数据处理的逻辑，`Controller` 的代码变得相对比较简洁。

```
class ActivityListViewController: UIViewController {
    var activityType:ActivityType?
    var selectedImageView: UIImageView?
    @IBOutlet weak var tableView: UITableView!
    var activityListDataSource:ActivityListDataSource?
    //视图被加载 只执行一次
    override func viewDidLoad() {
        super.viewDidLoad()
        if let activityType = self.activityType {
            self.activityListDataSource =
                ActivityListDataSource(tableView:self.tableView, activityType: activityType)
        }
        // Do any additional setup after loading the view.
    }
    //视图显示 每次显示在屏幕均会执行一次
    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)
        self.activityListDataSource?.loadData()
    }
}
extension ActivityListViewController:UITableViewDelegate {
    func tableView(tableView: UITableView, heightForRowAtIndexPath indexPath: NSIndexPath) -> CGFloat {
        return self.activityListDataSource?.estimateHeightForEachRow() ?? 0
    }
}
```

图 4.21 `ActivityListViewController` 视图控制器的代码实现

图 4.21 的代码加上图 4.15 中的代码即为 `ActivityListViewController` 的全部代码，在界面被加载即 `viewDidLoad` 方法中，创建 `activityListDataSource` 并注入 `tableView` 和 `activityType`，将渲染的工作交给 `activityListDataSource`，即完成了所有工作，只需要在界面重新显示在屏幕中的时候委托 `activityListDataSource` 刷新数据即可。

表中还出现了 `ActivityListViewController` 的一个扩展，在 Swift 中，扩展也可以遵循协议，在开发过程中，通过扩展+协议的方式分割每个类的逻辑，可以使得代码更为清晰，也更容易抽离逻辑，在这个扩展中，实现了

`UITableViewDelegate` 的协议，控制了列表项每行的高度以及点击事件的处理，而 `activityListDataSource` 负责的才是渲染，所以这里高度当然也是从 `activityListDataSource` 中获取。

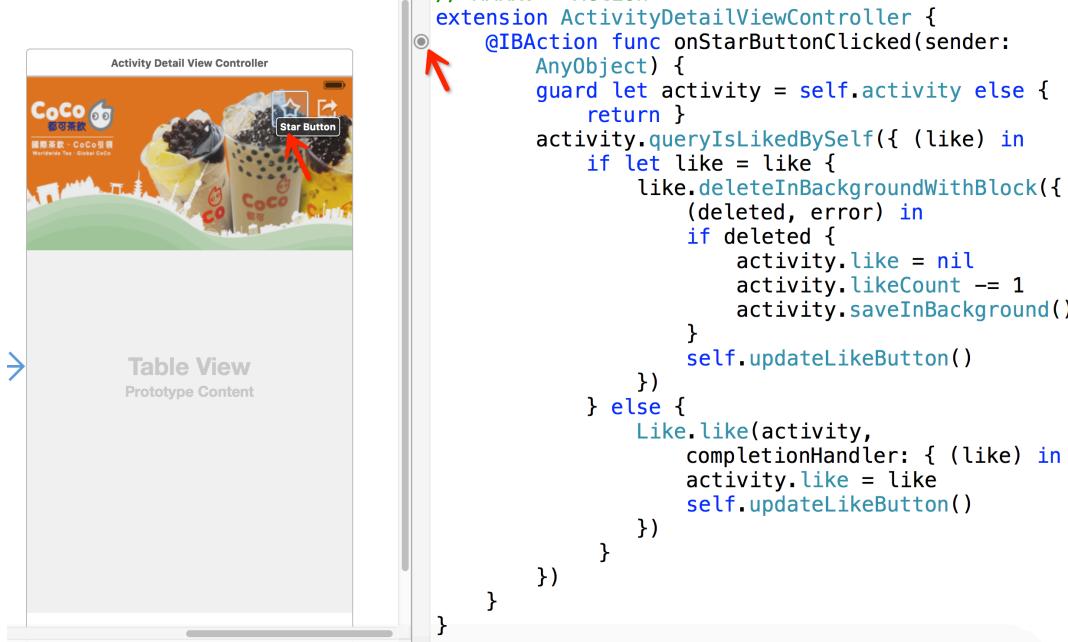


图 4.22 IBAction 事件绑定的实现

需要强调的是，`Controller` 虽然被抽离了绘制的逻辑，但是它依然需要处理 UI 事件，在使用 `Xib` 或者 `Storyboard` 开发的时候，我们不单单可以通过 `IBOutlet` 来建立 `Controller` 和 `View` 之间引用的联系，还可以通过拖拽 `IBAction` 来建立 UI 组件事件与 `Controller` 之间的联系，如果，点赞按钮的点击事件与 `ActivityDetailViewController` 中的 `onStarButtonClicked` 方法所关联，一旦按钮被点击，即执行该方法。这里同样通过 `extension` 的方式抽离了事件响应的逻辑。

4.3.4 通过协议扩展来减少重复代码与解耦

在 iOS 应用开发中，常常会编写一个功能强大的 `Controller` 基类，所有的 `Controller` 集成这个基类，在基类中完成一些基本的配置，并加入一些经常使用的方法供子类来调用。然而久而久之，这会使得项目变得越来越难以维护，一方面并不是所有 `Controller` 都需要这些方法，这就意味着必须重载父类的方法才可以去除这些逻辑。

举一个简单的例子，在应用中存在许多的界面都有一个共同的需求，他们通

过 Present 而非 Push 的形式展示给用户，左上角有一个取消按钮，并且点击取消按钮可以让该页面消失回到原先的页面，那么应该使用何种方式来实现这个需求呢？

继承或许是个不错的方法，写一个父类来完成所有的配置，将所有的这些界面均继承这个父类。听起来不错，然而如果其中的某几个界面又有共同的需求，是不是意味着又要再加一层继承关系？

组合是一个更好的方法，可以写一个导航栏，让这些 Controller 持有这个自定义的导航栏，所有的共同逻辑在这个自定义组件中完成，然而不得不承认，每个 Controller 需要持有这些组件，并且需要管理这些事例的创建和释放，在使用过程中通过间接变量，多了一层的结构。

针对这样的情况，本项目使用协议扩展的方式解决了这个问题，相比使用组合，并没有增加代码，但是却更容易复用。面向协议编程即接口而非具体实现，充分解耦。下面是具体实现。

```
//定义协议
protocol ViewControllerPresentable {
    func configureNavigationItem()
}

//协议扩展且限定遵循协议的是一个 UIViewController
extension ViewControllerPresentable where Self: UIViewController{
    func configureNavigationItem(){
        let leftButton = CancelButton.init(frame:CGRectMake(0, 0, 100, 40))
        leftButton.setTitle("取消", forState: UIControlState.Normal)
        leftButton.setTitleColor(UIColor.whiteColor(), forState: UIControlState.Normal)
        leftButton.addTarget(leftButton, action: #selector(CancelButton.onCancelButtonClicked), forControlEvents: UIControlEvents.TouchUpInside)
        leftButton.titleLabel?.font = UIFont.systemFontOfSize(UIFont.systemFontSize())
        leftButton.contentHorizontalAlignment = UIControlContentHorizontalAlignment.Left
        let leftBarBarItem = UIBarButtonItem.init(customView: leftButton)
        self.navigationItem.leftBarButtonItems = [leftBarBarItem]
    }
}

class CancelButton:UIControl {
    func onCancelButtonClicked(){
        self.viewController()?.dismissViewControllerAnimated(true, completion: nil)
    }
}
```

```

}

//编辑个人信息的界面，即从当前页面直接弹出，左上角有一个取消按钮
class EditUserInfoTableViewController: UITableViewController, ViewControllerPresentable {
    override func viewDidLoad() {
        super.viewDidLoad()
        self.configureNavigationItem()
    }
}

```

图 4.23 通过协议扩展抽离重复逻辑的代码实现

在图 4.23 中的代码中，定义了一个 `ViewControllerPresentable` 的协议，这个协议中定义了方法 `configureNavigationItem()`，然而通过协议扩展的方式给这个协议添加了一个默认实现，这个默认实现仅在遵循协议的类为 `UIViewController` 或者是其子类中生效，加入了这个限制，在这个默认实现中就可以调用 `UIViewController` 的方法了。在使用过程中，只需要让 `EditUserInfoTableViewController` 遵循这个协议，并且调用方法，不用写任何实现即可以完成指定的功能。

在本项目中大量使用了这种方式来消除重复代码做到重复解耦。

4.3.5 基于 POP 的 MVVM 的设计模式的详细实现

在图 3.16 中描述了本应用中使用的基于 POP 的 MVVM 的设计模式，为了更方便地解释这一模式的具体实现，我们将 Controller 移除，那么图应该是这样的。

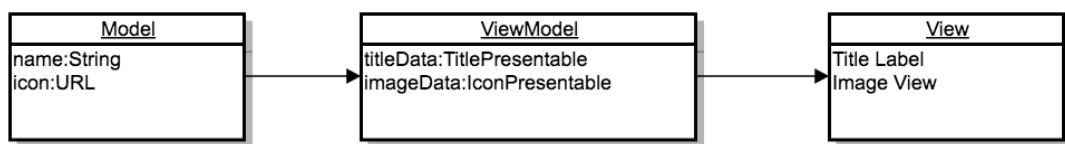


图 4.24 基于 POP 的 MVVM 类关系图

在 `Model` 中，存在一和 `name` 和 `icon` 的属性，`ViewModel` 通过注入 `Model` 来生成对应的 `titleData` 和 `imageData` 分别对应 `Model` 的 `name` 和 `icon`，只不过它们已经被处理成了可以直接渲染在 `View` 上的数据，而处理的方法便是通过协议。下面看具体代码实现，以 `ActivitySimpleViewModel` 为例：

```
//typealias 相当于一个宏定义即别名，将 ActivitySimplePresentable 定义为这些协议的组合，
```

即一个活动的简单表示包括标题、一个来自网页的图片、可以勾选、位置以及小标题

```
typealias ActivitySimplePresentable = protocol<TitlePresentable, WebIconPresentable, Checkable, LocationPresentable, SubTitlePresentable>

// 使用 struct 而非 class 是因为 struct 没有引用所带来的副作用，ViewModel 是一个稳定的对象，struct 中一旦有数据发生改变整个就被重新替换，这样可以通过持有者的 didSet 方法来刷新页面

struct ActivitySimpleViewModel: ActivitySimplePresentable {
    // 来自 WebIconPresentable
    var iconName: String
    // 来自 TitlePresentable
    var title: String
    // 来自 LocationPresentable
    var location: String
    // 来自 SubTitlePresentable
    var subTitle: String
    // 来自 Checkable
    var isChecked: Bool?
    // 注入 activity 来初始化
    init(activity: Activity) {
        self.iconName = activity.avatar.url
        self.title = activity.name
        self.location = activity.locationName
        self.subTitle = "\(activity.likeCount)"
        self.isChecked = activity.isLikedByMySelf
    }
}
```

图 4.25 ActivitySimpleViewModel 的代码实现

ActivitySimpleViewModel 的所有属性都定义与其遵循的协议，且本身没有任何实现，不用编写任何多余代码。下面是这些协议的代码：

```
protocol TitlePresentable {
    var title: String { get }
    var titleColor: UIColor { get }
```

```

func updateTitleLabel(label:UILabel)
}

extension TitlePresentable {
    var titleColor: UIColor {
        return UIColor.globalTitleBrownColor()
    }

    func updateTitleLabel(label:UILabel) {
        label.text = self.title
        label.textColor = self.titleColor
    }
}

protocol WebIconPresentable {
    var iconName: String { get }
    func updateImageView(imageView: UIImageView)
}

extension WebIconPresentable {
    func updateImageView(imageView: UIImageView) {
        imageView.clipsToBounds = true
        imageView.contentMode = UIViewContentMode.ScaleAspectFill
        imageView.setImageWithURLString(iconName)
    }
}

.....

```

图 4.26 Presentable 协议以及协议扩展的代码实现

这里只贴出了 `TitlePresentable` 和 `WebIconPresentable` 的代码，`TitlePresentable` 中定义了 `title` 这个变量的 `get` 方法，意味着只要遵循了 `TitlePresentable` 的协议必须实现 `title` 这个属性，同时定义了 `updateTitleLabel` (`label:UILabel`) 的方法，并提供了默认实现，在这个协议中，我们可以定义更多的属性即默认实现来控制 `Title` 的字体、颜色、背景等等，将这些处理工作全部封装在了协议扩展中，而 `ViewModel` 正常情况下不需要实现一行代码，当需要有个性化定制的时候只需要重写对应方法即可。

针对使用协议化定制的 `ViewModel`，我们使用图 4.26 中的方式进行调用。

```

extension ActivityListDataSource: UITableViewDataSource {
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let tableViewCell = tableView.dequeueReusableCell(indexPath: indexPath) as ActivityTableViewCell
        if let activity = self.activityList?[indexPath.row] {
            //创建 activityViewModel
            let activityViewModel = ActivitySimpleViewModel(activity: activity)
            //渲染 cell
            tableViewCell.render(activityViewModel)
        }
        return tableViewCell
    }
}

class ActivityTableViewCell: UITableViewCell {
    @IBOutlet weak var starButton: UIButton!
    @IBOutlet weak var locationLabel: UILabel!
    @IBOutlet weak var nameLabel: UILabel!
    @IBOutlet weak var activityImageView: UIImageView!
    @IBOutlet weak var likeCountLabel: UILabel!
    func render(activityPresentable:ActivitySimplePresentable) {
        activityPresentable.updateImageView(activityImageView)
        activityPresentable.updateTitleLabel(nameLabel)
        activityPresentable.updateLocationLabel(locationLabel)
        activityPresentable.updateButton(starButton)
        activityPresentable.updateSubTitleLabel(likeCountLabel)
    }
}

```

图 4.27 ActivityListDataSource 的代码实现

ActivityTableViewCell 甚至不需要依赖于 ActivityViewModel 或者 Activity，它仅仅依赖于 ActivitySimplePresentable，并且通过传入 UI 组件的方法来完成对页面的渲染。

4.4 本章小结

本章节重点介绍了本项目——基于 iOS 平台的集点卡应用中详细设计和实现，在类设计和实现上主要针对了活动模块这一个比较典型的模块进行阐述。

在 View 层的实现上，分别针对不同情况用 Storyboard、Xib 以及代码配置三种方式进行视图的配置；在 Model 层的实现上，通过继承 AVObject 父类实现 AVSubClassing 的协议来完成 LeanCloud 数据与本地 Model 数据的映射，通过 Extension 的方式注入与 LeanCloud 数据存储服务交互的职责；在 Controller 层的实现上，通过 IBOutlet 和 IBAction 的方式将 View 的组件和事件与 Controller 相关联，通过 DataSource 和 ViewModel 来抽离渲染逻辑。

本章还详细介绍了在项目中使用协议扩展的方式来抽离重复逻辑，做到依赖最小化，最后以一个活动列表的渲染作为案例详细描述了本项目中是如何使用面向协议编程的思想完成视图内容填充的。

第五章 总结与展望

5.1 总结

本人从大二便开始学习 iOS 开发，从 2015 年年底开始学习 Swift，在 2016 年 1 月份参与了第一届全国 Swift 开发者大会，在大会上深刻地感受到了这门新语言的魅力，也学习到了许多有用的思想，决定在毕业设计中使用 Swift 独立完成一个应用，学以致用，在项目完成过程中也的确学习到了许多新的知识。

本项目——基于 iOS 平台的集点卡应用由我一个人独立完成，从设计到开发大约花费了 2 个月时间，项目规模中等，但是麻雀虽小，五脏俱全，已经具备了一个“O2O”项目所具备的大部分功能，并且能够同时为商家和消费者进行服务。

从功能方面，到目前为止，项目已经完成了所有的高优先级需求，商家能够通过客户端进行店铺、活动的创建，能够与消费者进行评论的回复，能够帮助消费者集点以及兑换商品，消费者能够通过客户端找到附近正在搞活动的店铺、完成集点兑换活动、并且对店铺进行评价。除此之外，还包括部分用户级别的操作，包括账户信息的更改、个人信息的修改、活动的收藏等。

从开发过程方面，项目始终按照最初设计好的框架进行构建，始终遵循着基本的代码原则，保持代码的高质量以及可维护性。同时，在完成功能的同时也考虑到了一系列的性能问题以及用户体验问题。

项目的版本控制通过 Git 来实现，所有的项目配置文件都通过 Git 来完成版本管理，项目托管于 Github，地址为：<https://github.com/luckymore0520/GreenTea>，在日后也会进行进一步的更新。

当然，要想让项目完成可以上线并推广的程度，还有很长一段路要走。到目前为止，项目仅仅完成了最核心的功能，相比一些已经成熟了的竞品应用，还缺乏用户和用户之间进行交流的途径、缺乏商家与用户之间交流的途径，仅仅的评论机制并没有办法完全解决这些问题。同时，对于商家来说，活动的可制定程度还不够高，所有的促销活动、集点活动依旧是按照系统给定的模板进行创建，缺乏提供个体差异的空间。

更重要的是，项目缺乏一个完备的后台管理，缺乏对于应急情况的处理方案。除却技术层面的问题之外，从运营的角度，本系统还缺少完整的推广策略盈利方案。

5.2 展望

目前，项目已经完成了大部分的核心功能，完成了大约 80% 的基本用例，为了能够将它打造成一个完整的可以使用的项目，还需要进一步地针对一些细节进行开发。

作为一款移动应用，必须纪录用户行为、统计用户数据，也必须能够定期地上报线上错误从而完成修复，该项目到目前为止并没有集成这些功能，而是将主要的工作集中于功能和流程的完善，在日后，可以集成类似“友盟”这样的数据统计服务以及类似“Fabric”这样的崩溃与错误上报服务。

我已经将该项目开源于 **Github** 上，一方面可以为初学者提供一些思路和借鉴，一方面希望能有人参与进来一起对项目进行完善无论是在功能扩展方面还是在项目代码质量方面。

最后，作为一个移动互联网项目，一个 App 仅仅是一个开始，更关键的是资源，虽然目前国内并没有发现任何一个有名的类似于该应用的集点卡应用竞品，在国外也有不少类似应用的成功例子，但是要想让这样的一套解决方案真的能够被推广并且能够被一部分商家和消费者接受还有很长一段路要走。

参考文献

- [1] 王巍,《Swifter 10 个 Swift 2 开发必备 Tip》(第 2 版),电子工业出版社,2015
- [2] 李洁信《Pop in Swift》,第一届全国 Swift 开发者
- [3] LeanCloud 数据存储服务官方文档 , https://leancloud.cn/docs/leanstorage_guide-ios.html
- [4] 唐巧,《被误解的 MVC 和被神化的 MVVM》, <http://blog.devta.com/2015/11/02/mvc-and-mvvm/>
- [5] 骆斌, 丁二玉,《需求工程—软件建模与分析》, 高等教育出版社, 2009。
- [6] 高德地图官方文档, <http://lbs.amap.com/api/ios-sdk/guide/introduction/>
- [7] Apple, iOS Develop Library, 《Writing Swift Classes with Objective-C Behavior》,
<https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/WritingSwiftClassesWithObjective-CBehavior.html>
- [8] 骆斌、丁二玉、刘钦,《软件开发的技术基础-软件工程与计算-(卷二)》, 机械工业出版社, 2012
- [9] 维基百科《Xcode》, <https://zh.wikipedia.org/wiki/Xcode>
- [10] Apple,iOS Develop Library, 《The Swift Programming Language (Swift 2.2)》, https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/index.html
- [11] hbooooooob,《 Swift 与 Objective-C 混编(一)》, <http://www.jianshu.com/p/084f2ca45007>
- [12] Apple, iOS Develop Library, 《Auto Layout Guide》, <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/>
- [13] Apple, iOS Develop Library, 《Storyboard Help》, https://developer.apple.com/library/ios/recipes/xcode_help-IB_storyboard/Chapters/AboutStoryboards.html#/apple_ref/doc/uid/TP40014225-CH41-SW1

致谢

光阴似箭，岁月匆匆，大学四年生活接近尾声，随着本次论文的完成，将划上一个完美的句号。

本文在刘钦老师的指导和帮助下完成，从核心功能的定位到项目的开发以及论文的编写，刘老师都向我提供了许多宝贵的意见和富有指导性的建议。所以首先我想要对刘老师的指导和帮助表示深深的感谢！

同时，我还要向所有参考书籍和文章的作者表示感谢。向第一届@Swift 大会的举办者和嘉宾们表示感谢，本项目的许多开发思路都是受到了大会上各位嘉宾们的启发。

在南京大学软件学院的这四年里，我不仅学习到了许许多多宝贵的专业知识、提高了实践能力，更加锻炼了自主学习、独立创新的能力。最后，我需要感谢南大软院的所有老师，感谢四年里一直关心我们、帮助我们的辅导员陈老师和曹老师，也感谢一直陪伴在我身边共同进步的同学们。