南京大学

本 科 毕 业 设 计

院　系 软件学院

专　业 软件工程

题　目 Kikbug-iOS 平台众测软件的实现

年　级 2012　学　号 121250089

学生姓名 刘威廷

指导教师 陈振宇　职　称 副教授

论文提交日期 2015 年 5 月 15 日

南京大学本科生毕业论文（设计）中文摘要

毕业论文题目： Kikbug-iOS 平台众测软件的实现

软件学院 院系 软件工程 专业 2012 级本科生姓名： 刘威廷

指导教师（姓名、职称）： 陈振宇 副教授

摘要：

随着移动终端的不断发展，手机 APP 不断涌现。截止到 2015 年 12 月，国内在网活跃移动智能设备数量达到 8.99 亿，其中苹果设备与安卓设备比例为 3：7，也就是说有许多应用安装在将近 3 亿的 iOS 设备上。因此，我们需要在 iOS 平台上进行众测任务的发布和管理，便于众包测试的发展。

项目开发分为两个部分:管理终端和测试 SDK。iOS 系统属于类 Unix 的商业操作系统。截止到 2016 年 4 月，iOS 系统已经更新到了 9.3.1 版本。我们使用了 Objective-C 语言（以下简称 OC）进行开发，并且在项目中引用 Coccoapods 来管理第三方资源的引用。OC 是扩充了 C 的面向对象的编程语言，其动态的特性使得编程更加的灵活。同时，在项目中我们的基础架构为 MVC，在一些稍微复杂的页面中使用 MVVM 来简化 Controller 的逻辑，并且在代码中灵活运用 OC 动态的特点，减少编码工作。另外，项目中还使用了 URL Navigator 来进行解耦，最大程度上减少类与类之间的耦合程度。

Kikbug 的 iOS 端帮助用户在 iOS 端的接收任务，查看任务，执行测试任务，并且自动化收集测试信息，硬件信息等。帮助测试人员发现 bug，提供更多信息以便于开发者 debug 工作。本人在项目中负责管理终端的开发。主要负责架构设计，从界面到业务逻辑开发，以及网络通信的开发。Kikbug 的管理终端使用 Kikbug 后台提供的 API，通过 http 请求拉取数据。

关键词：众测平台，iOS 开发，软件测试，OC 的动态性，解耦。

南京大学本科生毕业论文（设计）英文摘要

THESIS： Kikbug - Implement iOS Public Testing Platform

DEPARTMENT： Software Institute

SPECIALIZATION: Software Engineering

UNDERGRADUATE: Weiting Liu

MENTOR: Zhenyu Chen

ABSTRACT：

With the continuous development of mobile terminals, mobile phone APP emerging. As of December 2015, in the domestic network of active mobile smart devices reach the number of 899 million, of which Apple devices and Android devices ratio of 3: 7, that is to say there are many applications installed on nearly 300 million iOS devices. Therefore, we need to publish and manage all testing tasks on iOS platform to facilitate the development of crowdsourced testing.

Project development is divided into two parts: the management terminal and test SDK. The iOS system is Unix-like commercial operating system. As of April 2016, iOS system has been updated to version 9.3.1. We use the Objective-C language (hereinafter referred to as OC) to develop and Coccoapods to manage third-party reference to resources in the project. OC is an object-oriented programming language and extension of C, its dynamic nature makes programming more flexible. At the same time, our infrastructure is based on MVC, MVVM is used in the project in some slightly more complex pages to simplify Controller logic. The flexible use of the OC dynamic characteristics reduces coding. In addition, the project also used URL Navigator to decouple, reduce the degree of coupling between classes.

Kikbug of iOS help users receive tasks, view tasks, test tasks, automatically collect test information, hardware information. It Helps testers discovered bug, provide more information for developers to do debug work. I am responsible for managing the development of the terminal in the project. Responsible for architecture design, development of the interface and business logic, and the development of network communications. APIs Kikbug background provided are used to pull data via http requests.

KEY WORDS: Public Test, iOS development, Software Test, OC dynamics , Decoupling

目 录

图目录

表目录

第一章 引言

1.1 项目背景

随着智能设备和移动网络的不断发展，越来越多的移动应用不断出现。移动应用成为了人们生活中不可缺少的一部分。截止到 2016 年 1 月，根据苹果最近发布的 2016 财年第一财季财报[1]中宣布其激活在用的设备总数达到了 10 亿部，同时，有超过 150 万个移动应用发布在 App Store 上。根据 2015 年 7 月的公开数据，应用下载次数突破 1000 亿次，每秒有 850 个应用贝下载，每位 iOS 用户平均下载 119 个应用。如此数量庞大的移动应用和用户给软件的质量和测试工作提出了更高的要求。

传统的软件测试依靠开发人员和软件测试人员在产品上线之前进行测试用例的编写和手工操作以发现在运行过程中产生的错误或者应用崩溃。当发现问题之后，再将 Bug 提供给开发人员，由开发人员根据 Bug 的表现和开发工具来定位 Bug 然后解决问题，之后再由测试人员复查。往往，很多疑难杂症是软件测试工作人员难以通过个人的操作发现的，因为往往显而易见的 Bug 都在编码过程中被发现并解决掉了。但是由于开发人员和真实用户之间的差异，导致开发人员由于思维定式，无法像真实用户一样，从真实用户的视角去看待移动应用的行为。因此无法发现真实用户能够发现的问题。

众包，也叫群众外包，是互联网带来的新的生产组织形式。用于描述一种新的商业模式，即企业利用互联网将工作分配出去、发现创意或者解决技术问题。因此，众包测试就是把软件的测试工作通过互联网分发出去。在生产中，众包测试的实际意义在于，将软件测试的工作放到真实的生产环境中，由接近于真实用户的人们来进行测试。众包测试将测试需求切分为不同的测试任务，用户针对某一个测试任务来执行测试操作，由平台收集测试过程的数据，包括用户的操作步骤以及用户可选的 Bug 报告。

目前在我院的《软件测试》课程中，学生在学习移动应用测试时已经在使用 Android 应用测试了。但是问题在于，很多同学没有 Android 设备的同学在完成课程练习上遇到了困难，需要借用同学或学院的设备。另外，iOS 平台[2]的工作与 Android 平台还有着权限上的差异。由于 Android 是开源系统，系统权限的获取比 iOS 平台更加容易，而 iOS 平台为了其生态系统和安全性考虑使用沙盒机制[1]，是一个封闭的系统，导致很多工作不能照搬 Android 平台的解决方案。在沙盒机制中，每个应用程序只能在为该程序创建的文件系统中读取文件，无法访问其他程序的文件，应用程序请求的数据都要经过权限检测。为了实现在 iOS 平台上的众包[3]测试工作，同时为了能够在软件测试教育当中，使得软件测试不仅仅局限于 Android 应用，我们需要一套新的解决方案，方便众包工作的公开、众包人员执行众包工作、以及众包人员的组织。

[1] 关于 Apple 公布的财报的详细信息，参见 http://www.apple.com/cn/pr/library/2016/01/26Apple-Reports-Record-First-Quarter-Results.html
[2] 关于 iOS 的详细信息，参见：https://zh.wikipedia.org/wiki/IOS
[3] 关于众包的详细信息，参见：https://en.wikipedia.org/wiki/Crowdsourcing

1.2 众包测试工具以及现状

1.2.1 众包测试工具

目前市场上使用较多，为人熟知的众包测试有 Utest（优测），班墨云测试等。下面分别对这两种众包测试工具进行大致的功能介绍以及界面展示。

1） Utest（优测）[2]

这是一个由腾讯开发的，关于 Android 机型的众包测试工具。其测试项主要包括：安装/启动/登录/执行/卸载、核心场景遍历、问题机型复现、运行截图、log 日志下载、错误定位、安装时间/启动时间/CPU/内存、在线报告。

图 1.1 UTest（优测） Step1 截图

图 1.2 UTest（优测） Step2 选择机型页面

2） 班墨云测试[2]

提供兼容测试、专家测试、手游测试、性能测试、登记测试。其在兼容测试方面，主要提供了对于多打 600 种 Android 机型的兼容性测试服务和 iOS 测试。其云测试主要是使用 Monkey。

图 1.3 班墨云测试众测页面

图 1.4 班墨云测试云测部分

1.2.2 众包测试现状

从软件众包平台目前的情况看来，软件众包测试大多集中于 Android 平台的兼容性测试，使用 logcat 和 monkey 进行测试，收集 log 日志和 crash 信息。并且这些测试平台都使用 web 端的形式，由用户上传.apk 文件。Web 端的测试平台要求用户通过选择上传测试文件，由测试人员通过 web 来下载之后手动安装，再进行测试。我们希望能够通过用户在手机上能够直接查看测试任务，使众包测试与平台（Android 和 iOS）的结合更加紧密。

1.3 项目功能概述

Kikbug 客户端是一个在 iOS 上运行的 App，面向众测人员。
项目具有以下功能：

1. 注册并登陆/登出
2. 查看众测任务列表
3. 查看群组详情
4. 搜索群组
5. 通过口令加入群组
6. 查看群组任务
7. 接受任务
8. 查看我的任务
9. 查看任务详情
10. 查看/修改个人信息
11. 查看任务的报告列表
12. 查看报告的 Bug 列表
13. 增加 Bug 报告
14. 与集成在被测试 App 中的测试 SDK 通信。
15. 跳转到 App Store 安装被测 App

1.4 论文的主要工作和组织结构

本人在该毕业设计项目中承担 Kikbug iOS 平台上的需求获取、体系结构设计、项目开发、和论文的撰写工作。其中项目初版的界面设计与使用的基础配色也由本人提出，并由专业人士提出修改方案加以改进。

本文第一章是论文的引文，第二章是项目所用到的技术概述，第三章描述了系统的需求分析和概要设计,主要参考了《软件开发的技术基础-软件工程与计算（卷二）》 (丁二玉，刘钦, 2012)[4]，第四章是详细设计与实现，第五章是总结与展望。

第二章 技术概述

项目采用 OC 作为开发语言是考虑到：首先，OC 作为一个发展成熟、广为使用的语言，其稳定性与丰富的三方库使项目开发避免重复造轮子的过程；其次，Swift 由于其目前刚刚推出 2.0 版本，并且实现了开源，可以预见的是，在将来 Swift 会逐渐替代 OC 在 iOS 应用开发的地位,但这一过程还将需要很久的时间。因此我采用更为稳定，技术上更为熟悉的 OC 语言。

项目的架构上采用了最为简单的 MVC[5]，这也是苹果在 OC 中推荐的设计模式。在一些 Controller 中使用了初步的 MVVM[6]思想来降低 Controller 的复杂程度[7]。

2.1 OC 动态特性[4]

1. 动态类型

对象的类型可以在运行的时候得到确定，在 OC 中，一切对象的基类都是 NSObject 类的子类。在 NSObject 类中，有-isKindOfClass 方法，在对类的类型进行强制转换之前确定对象的类型，确保安全性。

2. 动态绑定

类的实现可以在运行时绑定新的实例方法或者类方法,动态绑定也适用于属性（成员变量）。OC 基于动态类型，在 OC 中的方法并不是在编译时期就把参数信息和函数实现打包,而是采用了“消息”的机制。一个类的实例在收到消息后，从自身的实现中找能够响应消息的方法。而在这个找寻能够响应消息的方法的过程中，我们可以动态的增加新的方法。

3. 动态加载

在 iOS 开发中，图片资源有两种不同的大小。在提供图片资源时，除了原本大小的原图之外（作为@1x，即一倍大小），还要提供@2x 大小的高清图片适用于 Retina 设备。

另外，OC 中大量的使用了策略模式来降低耦合程度，这个模式通过定义协议和代理的方式，将类与类之间协作的耦合程度降低。OC 的 Runtime 是 OC 的运行时库，它是一个主要使用 C 和汇编编写的库，OC 的 Runtime 实现了 OC 的面向对象特性[8]。

2.2 页面自动适配

在 iOS6 中，Apple 提出了自动布局的技术。界面的布局不再通过计算 frame 的大小的方式来确定组件在屏幕上的位置,而是通过其相对父组件的位置或者同

[4] 关于运行时的官方文档，详情请见
https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/ocrHistory.html

级组件之间的相对位置来确定。其好处是显而易见的，自动布局解决了其在不同尺寸屏幕上的适配问题。随着硬件的发展，iPhone 的屏幕尺寸经过了若干次变化。目前，其主要尺寸有：3.5 寸、4 寸、4.7 寸和 5.5 寸。其中所有 3.5 寸的设备屏幕比例均为 4:3，而其余设备的比例为 16：9[9]。

设备	宽 (inch)	高 (inch)	对角线 (inch)	逻辑分辨率	Scale	设备实际分辨率	PPI
3GS	2.4	4.5	3.5	320*480	@1x	320*480	163
4/4s	2.31	4.5	3.5	320*480	@2x	640*960	326
5c	2.33	4.9	4	320*568	@2x	640*1136	326
5/5s	2.31	4.87	4	320*568	@2x	640*1136	326
6	2.64	5.44	4.7	375*667	@2x	750*1334	326
6 plus	3.06	6.22	5.5	414*736	@3x	1080*1920	401

表 2.1 iPhone 设备屏幕尺寸与分辨率

2.3 MVC 中对于 C 的清理

MVC 也被成为 Massive View Controller 模式。在开发中，MVC 模式容易产生的问题是：随着页面的复杂程度越来越高，MVC 中的 C 变得非常臃肿。过多的逻辑处理部分都放在 C 中，使得 Controller 难以维护。因此，我在项目中将 MVC 做出了一点改进，新建 ViewModel，并将 Controller 中的复杂部分移动到 ViewModel 中，使得 Controller 不再依赖 View，而是依赖 ViewModel。而 View 也不再从与 Controller 直接交互。

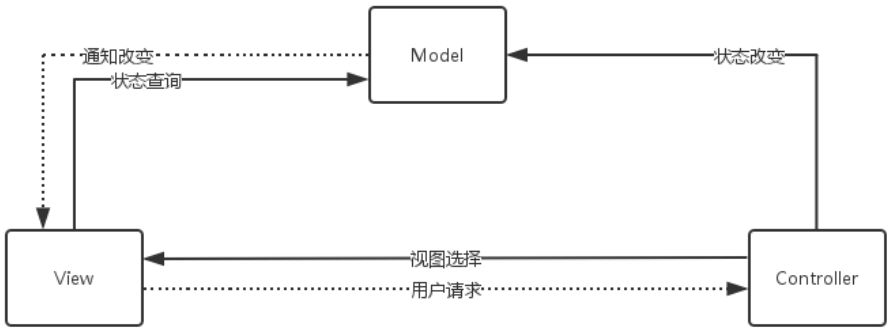


图 2.1 MVC 模块之间的通信

通过把页面的组织方式抽象为根据 ViewModel 决定，Controller 所需要做的事情是根据 ViewModel 的内容，将对应的 View 显示在对应的位置上。而 View 的内容则由 ViewModel 决定。通过把数据转移到 ViewModel 中，和把 View 的组织逻辑放到 ViewModel 中，Controller 的压力得到了减轻。

第三章 系统需求分析与概要设计

3.1 Kikbug for iOS 项目整体概述

众包测试的手机端主要负责众包测试任务的发布，便于用户查看测试内容，寻找群组，根据群组接受任务，执行测试任务，并且在任务结束之后报告发现的Bug。测试过程由测试人员手动执行，其后台的数据收集依靠相应的 Kikbug 后台SDK 来完成。Bug 的提交过程由用户决定。

众包测试是互联网的发展带来的新机遇，在众包测试中，测试人员可以是任何对手机基本操作有一定了解的个体。随着移动 APP 的发展，其在人们生活中扮演的角色也从一开始的娱乐休闲而逐渐向着更加贴近服务人们生活靠近。现如今，很多 APP 在我们的生活承担起了衣食住行的责任。因此，移动 APP 的测试工作才愈发显得不可缺少。

本项目分为两个部分：管理终端和测试 SDK。本人负责管理终端的开发。管理终端的开发即独立 APP 的开发，而测试 SDK 主要负责在测试过程中收集测试执行信息和 Bug 信息或者堆栈信息。以下将对这两个模块进行概述。

3.1.1 管理终端概述

管理终端是一个信息发布和提交的平台。用户在管理终端上接收众包测试的任务，查看众包测试的具体任务描述，并到被测试 APP 中去执行指定的测试任务。在任务执行结束之后，如果用户发现了 Bug，那么用户可以选择以图片和文字的形式来描述遇到的 Bug。管理终端可以发布到 AppStore 中以供下载，或者让用户通过扫描二维码的形式下载并安装。

管理终端所使用的数据均来自 Kikbug 后台实时提供的数据。本地仅存储少量的用户个人数据。管理终端和 Kikbug 的 web 端保持同步，考虑到移动设备在文字输入上的短板，用户不愿意在手机端输入过多文字内容，一些文字编辑的功能放到 web 端去执行。例如，用户在 iOS 管理终端上提交的 Bug 报告，可以在创建阶段只选择图片描述，而忽略文字描述。之后，用户可以自主到 web 端完善这次 bug 报告。

管理终端的开发使用 XCode 结合 OC。是一个纯 Native 的 APP，保证了运行上的稳定性和安全性，带来更好的用户体验。

3.1.2 测试 SDK 概述

测试 SDK 的职责是，如果开发者将我们提供的静态库添加到其 APP 中，并且通过简单的启动语句启动 SDK，那么当管理终端发出开始执行任务的请求时，被测试 APP 就可以解析开始测试请求，并出现相应的测试工具。

在测试完成之后，用户可以在被测试 APP 中选择“结束测试”并返回管理终端。测试 SDK 主要应用了面向切面编程，结合 OC 的动态特性。同时 SDK 也继承

了开源的 Crash 信息收集工具，将可能的 Crash 信息收集成 log，并上传到云存储。

3.2 系统的需求分析

3.2.1 用例图

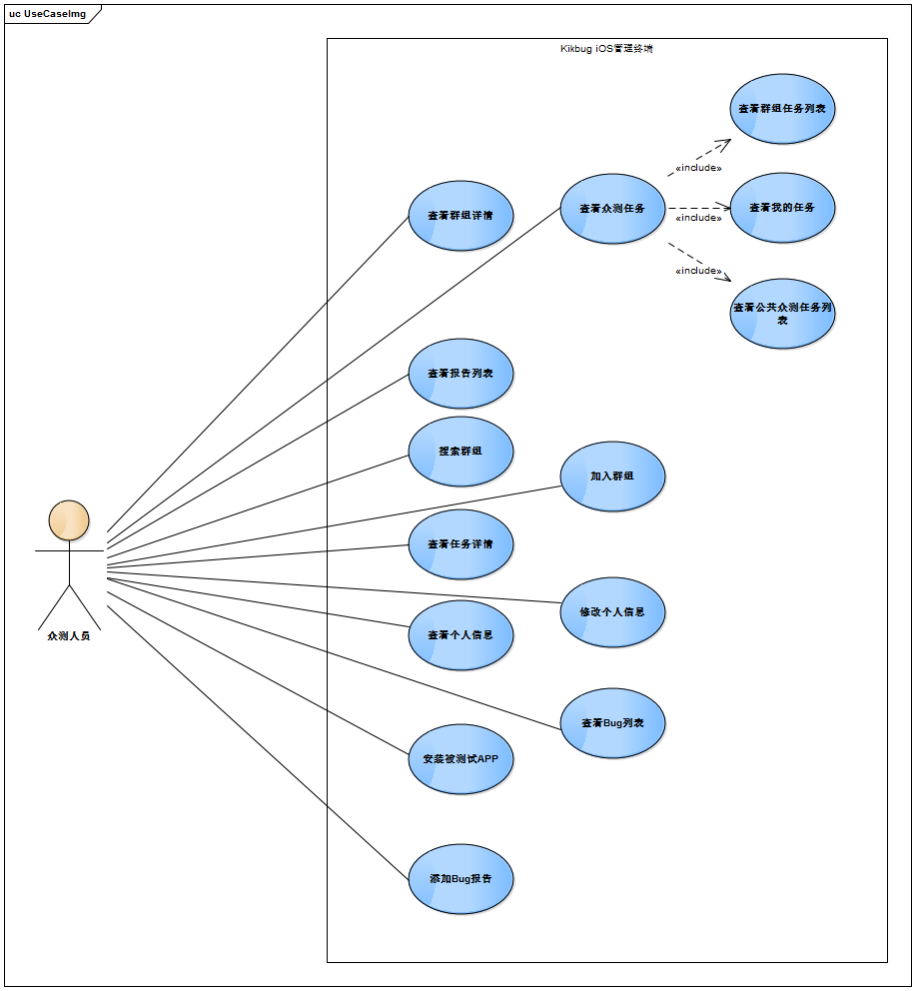


图 3.1 用例图

3.2.2 用例描述

1）用例 01 查看任务广场的众测任务

用例编号	UC001	编制时间	2016/3/27
前置条件	用户已登录(注:之后的用例描述均默认为用户已经登录,否则应用扩展事件流中的处理)		
基本事件流描述	1. 用户打开 APP		

	2. APP 显示加载界面 3. 完成加载后，APP 显示任务广场的众测任务
扩展事件流	1a、用户未登录 1a1、系统显示登录界面
后置条件	无
特殊要求	无
系统的初始状态	用户已登录
系统的初始数据	无
用例说明	任务广场的测试任务是所有人可见的测试任务集合。用户可以自由选择在这个页面查看任务，添加任务。

表 3.1 用例 01

2）用例 02 查看所有来自群组的众测任务

用例编号	UC002	编制时间	2016/3/27
前置条件	无		
基本事件流描述	1. 用户打开 APP 2. 用户在任务广场的 Tab 栏选择群组任务 Tab		
扩展事件流	2a、用户没有加入一个群组 2a1、没有群组任务。		
后置条件	无		
特殊要求	无		
系统的初始状态	用户已登录		
系统的初始数据	无		
用例说明	在这个 Tab 中，用户可以看到自己加入了的群组的所有测试任务。举例：假如用户加入了群组 A 和群组 B，那么在这个 Tab 下，用户应能够看到所有来自群组 A 和群组 B 的测试任务。		

表 3.2 用例 02

3）用例 03 查看群组的众测任务

用例编号	UC003	编制时间	2016/4/13
前置条件	用户已加入群组		
基本事件流描述	1. 用户打开我的群组 Tab 2. 用户点击某一个群组项		
扩展事件流	1a、用户没有加入过群组 1a1、显示群组空页面		
后置条件	无		
特殊要求	用户已经加入群组		
系统的初始状态	无		
系统的初始数据	无		
用例说明	系统通过 API 获取指定群组下的测试任务列表。		

表 3.3 用例 03

4）用例 04 查看我的测试任务

用例编号	UC004	编制时间	2016/4/13
前置条件	用户已经登录		
基本事件流描述	1. 用户打开 APP 2. 用户选择“我的任务”Tab		
扩展事件流	2a、用户没有添加过任务 2a1、显示我的任务页面为空。		
后置条件	无		
特殊要求	无		
系统的初始状态	无		
系统的初始数据	无		
用例说明	用户查看自己添加过的所有任务，包括来自群组和来自任务广场的任务。		

表 3. 4 用例 04

5）用例 05 查看我的群组

用例编号	UC005	编制时间	2016/4/13
前置条件	用户已登录		
基本事件流描述	1. 用户打开 APP 2. 用户选择 Tab 中的“我的群组”		
扩展事件流	2a、用户没有添加过群组 2a1、显示用户群组的空页面。		
后置条件	无		
特殊要求	无		
系统的初始状态	初始时用户的群组均为空		
系统的初始数据	无		
用例说明	用户查看自己加入过的群组。		

表 3. 5 用例 05

6）用例 06 添加任务

用例编号	UC006	编制时间	2016/4/13
前置条件	用户已经登录		
基本事件流描述	1. 用户打开 APP 2. 用户选择一个任务项并点击 3. 用户在任务详情页面点击接受任务按钮		
扩展事件流	3a、用户已经接受过该任务 3a1、系统提示任务已经接受过，无法重复添加。		
后置条件	新增任务被添加到“我的任务”中。		
特殊要求	无		

系统的初始状态	无
系统的初始数据	无
用例说明	无

表 3.6 用例 06

7）用例 07 加入群组

用例编号	UC007	编制时间	2016/4/13
前置条件	无		
基本事件流描述	1. 用户打开 APP 2. 用户选择“我的群组”Tab 3. 用户点击添加群组按钮 4. 用户输入群组名称关键字 5. 系统返回群组搜索结果列表 6. 用户点击搜索结果 7. 用户在群组详情页面点击加入群组按钮 8. 用户输入口令并点击确定		
扩展事件流	7a、用户输入的口令不正确 7a1、系统提示口令错误		
后置条件	无		
特殊要求	无		
系统的初始状态	无		
系统的初始数据	无		
用例说明	无		

表 3.7 用例 07

8）用例 08 搜索群组

用例编号	UC008	编制时间	2016/4/13
前置条件	无		
基本事件流描述	1. 用户打开 APP 2. 用户选择“我的群组”Tab 3. 用户点击添加群组按钮 4. 用户输入群组名称关键字		
扩展事件流	4a、用户在输入搜索关键字之后没有搜索到匹配的群组 4a1、系统显示空的搜索结果页面		
后置条件	无		
特殊要求	当用户输入的关键字为空时，显示所有的可用群组。		
系统的初始状态	无		
系统的初始数据	无		

表 3.8 用例 08

9）用例 09 查看群组详情

用例编号	UC009	编制时间	2016/4/13
前置条件	用户已经搜索到该群组		
基本事件流描述	1. 用户打开 APP 2. 用户选择“我的群组”Tab 3. 用户点击添加群组按钮 4. 用户输入群组名称关键字 5. 用户点击群组搜索的结果列表		
扩展事件流	5a、用户在输入群组关键字之后没有搜到结果 5a1、提示用户搜索结果为空，重新输入关键词。		
后置条件	无		
特殊要求	无		
系统的初始状态	无		
系统的初始数据	无		

表 3. 9 用例 09

10） 用例 10 查看个人信息

用例编号	UC010	编制时间	2016/4/13
前置条件	用户已登录		
基本事件流描述	1. 用户打开 APP 2. 用户选择个人中心 Tab		
扩展事件流	2a、用户点击个人头像 2a1、用户选择使用摄像头/本地图片 2a2、用户对已选图片进行编辑 2a3、用户确认选择 2b、用户点击个人昵称 2b1、用户输入新的昵称 2b2、用户返回个人信息页面		
后置条件	用户的昵称/头像为最后一次编辑的内容		
特殊要求	无		
系统的初始状态	无		
系统的初始数据	用户的个人信息中，头像默认为空，昵称默认为空。		

表 3. 10 用例 10

11） 用例 11 更改个人信息

用例编号	UC011	编制时间	2016/4/13
前置条件	用户已登录		
基本事件流描述	1. 用户打开 APP 2. 用户选择个人信息 Tab 3. 用户点击自己的头像		

	4. 用户选择使用摄像头或者选择本地图片 5. 用户确认已选图片
扩展事件流	3a、用户点击自己的昵称 3a1、用户输入新的昵称然后返回
后置条件	更新在线用户个人信息
特殊要求	新的用户名不能为空
系统的初始状态	无
系统的初始数据	空的头像和空昵称，用户积分为 0。

表 3. 11 用例 11

12） 用例 12 查看任务详情

用例编号	UC012	编制时间	2016/4/13
前置条件	无		
基本事件流描述	1. 用户打开 APP 2. 用户选择某一任务列表（任务广场/群组/单一群组） 3. 用户点击列表项		
扩展事件流	无		
后置条件	无		
特殊要求	无		
系统的初始状态	无		
系统的初始数据	无		
用例说明	任务详情主要有任务的分类，到期时间，积分，任务描述等。		

表 3. 12 用例 12

13） 用例 13 查看任务的测试报告列表

用例编号	UC013	编制时间	2016/4/13
前置条件	用户已经接受过任务，并且点击过开始测试按钮，产生了跳转。		
基本事件流描述	1. 用户打开 APP 2. 选择某个接受过的任务 3. 点击任务详情页中的查看测试报告按钮		
扩展事件流	无		
后置条件	无		
特殊要求	测试报告有两种生成方式，1.由 Kikbug 测试 SDK 自动生成；2.被测试 APP 选择不集成 SDK 的话，生成一份默认的测试报告（内容为预定义的默认值）。		
系统的初始状态	无		
系统的初始数据	无测试报告		

表 3. 13 用例 13

14） 用例 14 查看测试报告中的 Bug 列表

用例编号	UC014	编制时间	2016/4/13
前置条件	用户已经创建了测试报告		
基本事件流描述	1. 用户打开 APP，选择某一个已经接受过的任务，进入任务详情页。 2. 用户进入查看测试报告页面。 3. 用户点击某一测试报告项。		
扩展事件流	3a、用户还未在指定任务下生成测试报告。 3a1、提示空页面		
后置条件	无		
特殊要求	无		
系统的初始状态	无		
系统的初始数据	无		
用例说明	Bug 报告为用户可选内容。用户在测试时如果发现了 Bug，那么以截图和文字描述的方式来创建 Bug 报告。关于 Bug 报告的内容，参见用例 15.		

表 3. 14 **用例** 14

15） 用例 15 增加 Bug 报告

用例编号	UC015	编制时间	2016/4/13
前置条件	用户已经接受任务，用户已经在指定任务中生成测试报告。		
基本事件流描述	1. 用户打开 APP，进入查看 Bug 列表页面。 2. 用户点击“添加 Bug 报告”按钮。 3. 用户选择图片。 4. 用户填写文字描述（可选）。 5. 用户点击确定，上传 Bug 报告。		
扩展事件流	无		
后置条件	Bug 报告上传到 Kikbug 后台存储，用户可以在 web 端自行查看或修改报告内容。		
特殊要求	图片部分不得多于 9 张。		
系统的初始状态	无		
系统的初始数据	用户的文字描述默认为空。		

表 3. 15 **用例** 15

16） 用例 16 开始测试

用例编号	UC016	编制时间	2016/4/13
前置条件	用户已接受任务		
基本事件流描述	1. 用户进入任务详情页 2. 用户点击开始测试按钮，跳转到被测试 APP		
扩展事件流	2a、被测试 APP 还未安装 2a1、无法跳转，提示用户安装		
后置条件	打开被测试 APP，如果被测试 APP 集成了测试 SDK，那么显示测		

	试组件（按钮）。
特殊要求	用户已安装了被测试 APP
系统的初始状态	无
系统的初始数据	无

表 3. 16 **用例** 16

3.2.3 系统顺序图

1) 系统概要顺序图

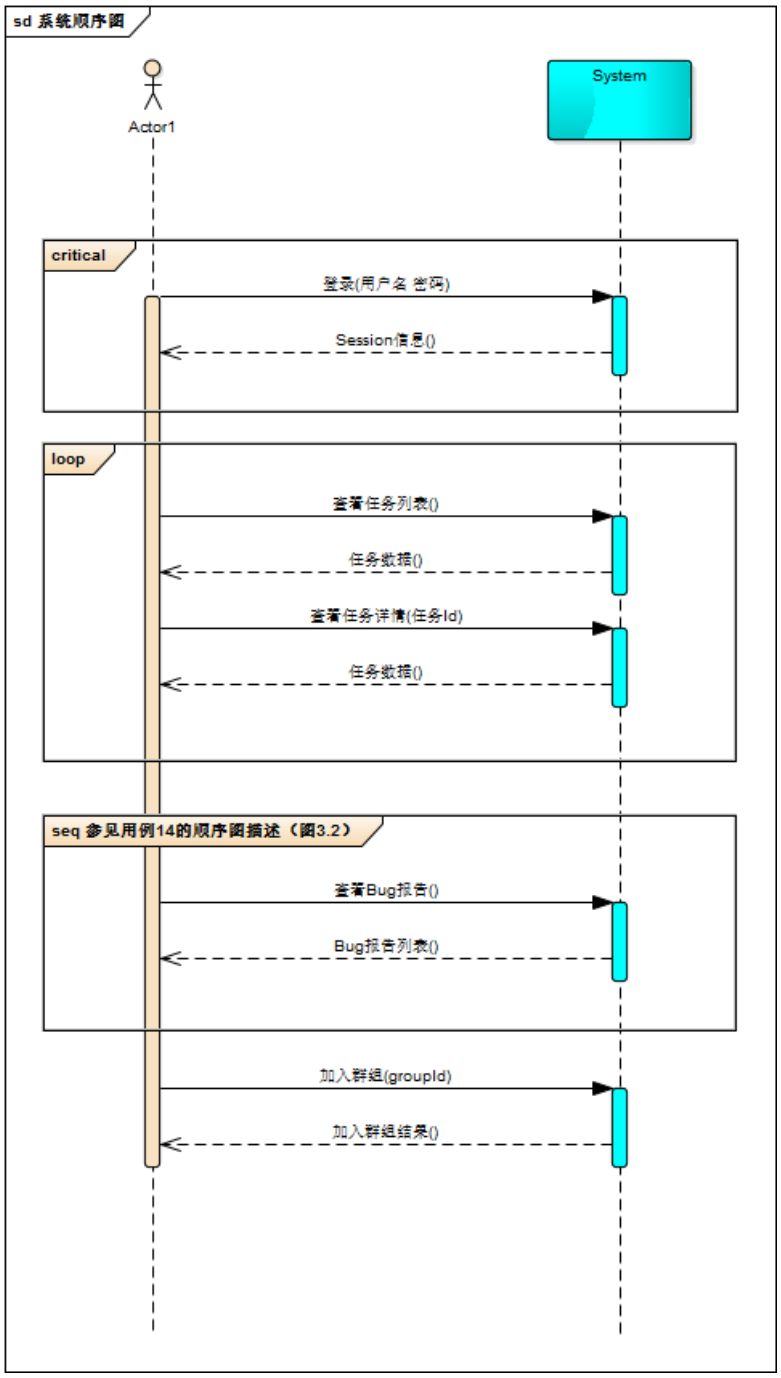


图 3.2 系统概要顺序图

2) 用例 14 的顺序图

因查看报告用例涉及到过多容易混淆的名词，因此附用例 14 的顺序图。同时作为系统概要顺序图中，在查看 Bug 报告部分的补充说明。

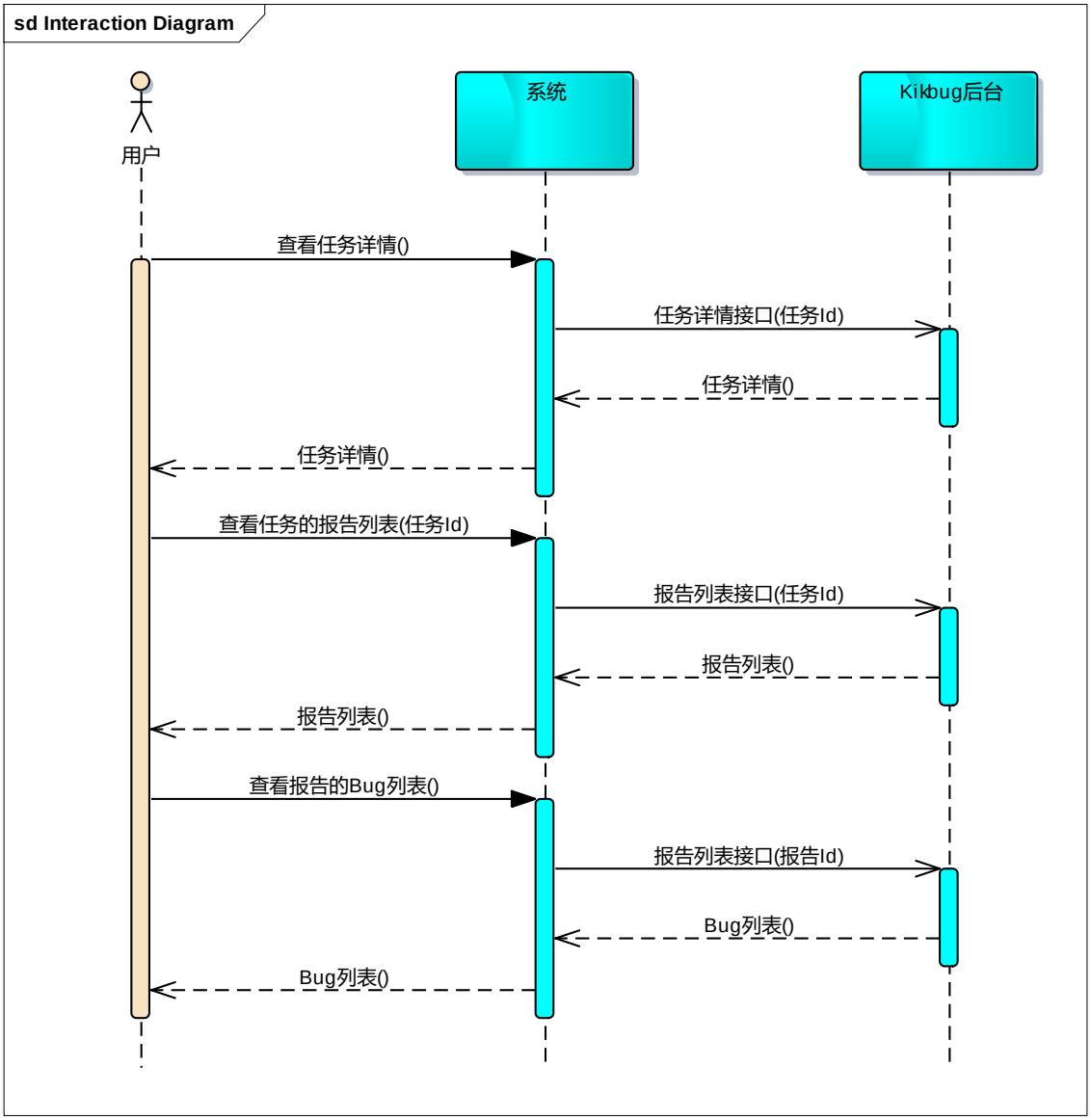
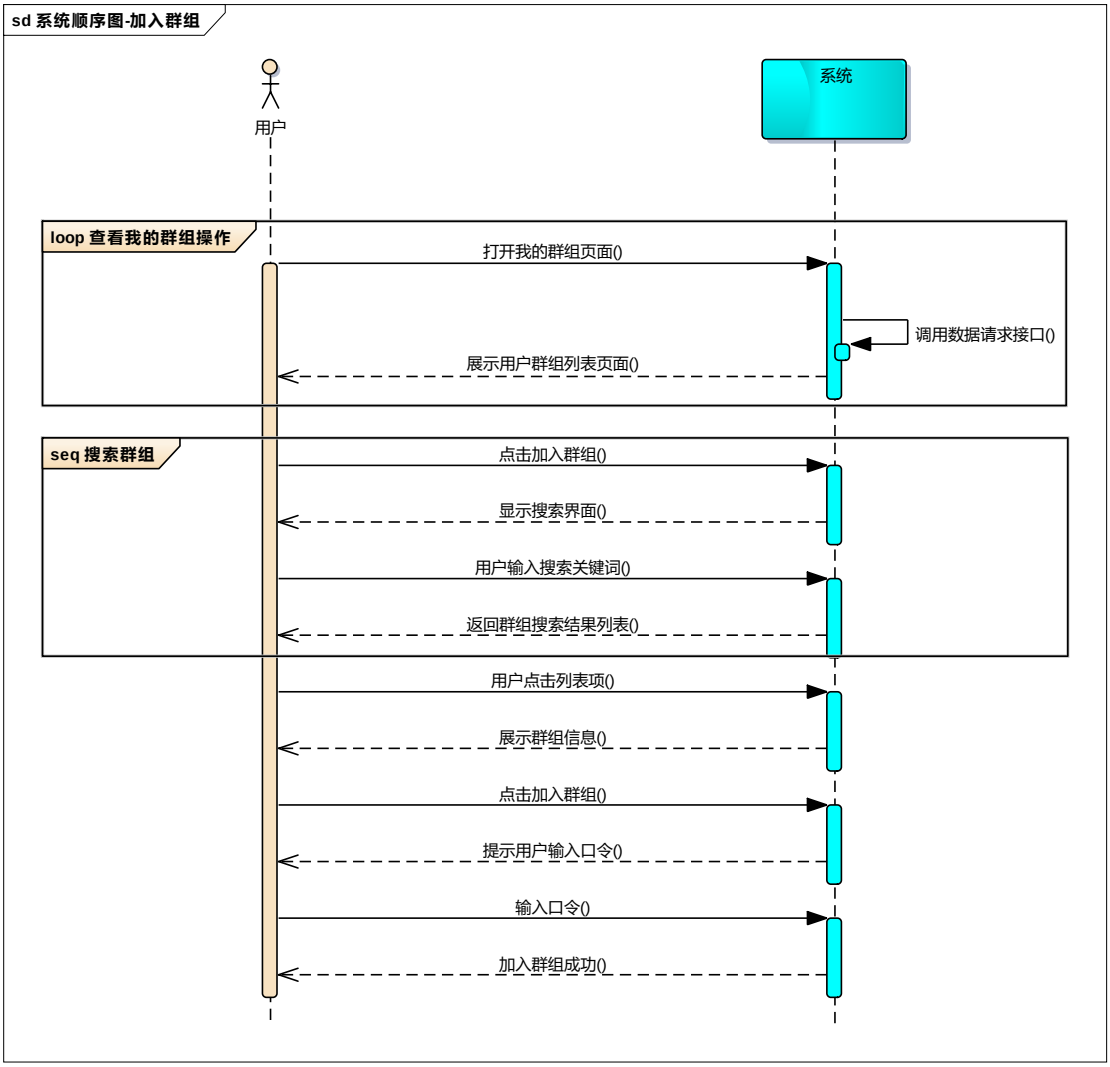


图 3. 3 用例 14 的顺序图描述

3) 用例 07 的顺序图

加入群组的过程涉及到搜索，口令的确认。用户交互较为复杂，并且功能的优先级也较高，因此在此说明加入群组过程的顺序图描述。



3.2.4 非功能需求描述

本小节描述系统的非功能需求和环境约束。

系统的肺功能需求如表 3.x，要求系统具有良好的可靠性、易用性、可维护性和可移植性，同时，系统要求在较短时间内对用户操作做出响应。

名称	内容	非功能需求类别
适配多种屏幕尺寸	系统应能够适配从 iPhone4 到 iPhone6s plus 的所有机型。	兼容性
最低运行系统	系统应能够运行在所有 iOS8 及以上的 iOS 设备上。	兼容性
易于修改维护	系统设计应尽可能合理，容易维护和扩展新模块。	可维护性/可扩展性
用户交互良好	对于所有用户操作，系统应及时作出响应，响应时	效率

	间应在 0.5 秒之内。在需要用户长时间等待的页面，应有清晰易懂的进度条提示。	
低出错率	系统的人机交互人性化，不容易出错或崩溃。	鲁棒性
用户交互界面设计	UI 的设计应该容易操作，容易理解。用户可以快速的学习对于 Kikbug iOS 系统的操作。	易用性

表 3.17 非功能需求表

系统所需要的环境约束见表 3.18.系统需要在有网络的环境下运行，且 iOS 的版本不低于 iOS 8.0。

约束类别	需求内容
iOS 系统版本	运行 APP 的设备的 iOS 版本在 8.0 及以上。
网络环境约束	本 APP 的运行需要网络连通环境。
用户约束	本 APP 的用户必须要有 Kikbug 的账号才能使用其他功能，所有用户操作均要求用户在已登录情况下才能进行。

表 3.18 约束限制表

3.3 系统概要设计

3.3.1 概念类图

概念类是对逻辑类的概要说明。在分析概念类时，从用例文本描述入手，识别出的候选类为：用户，任务广场，群组任务列表，我的任务，我的群组，群组搜索结果列表，任务，群组，任务报告，Bug 报告。从用例描述中看出，概念类之间的关联大致分为列表与列表项，用户与用户数据等关系。根据领域知识，大致为概念类添加重要属性。

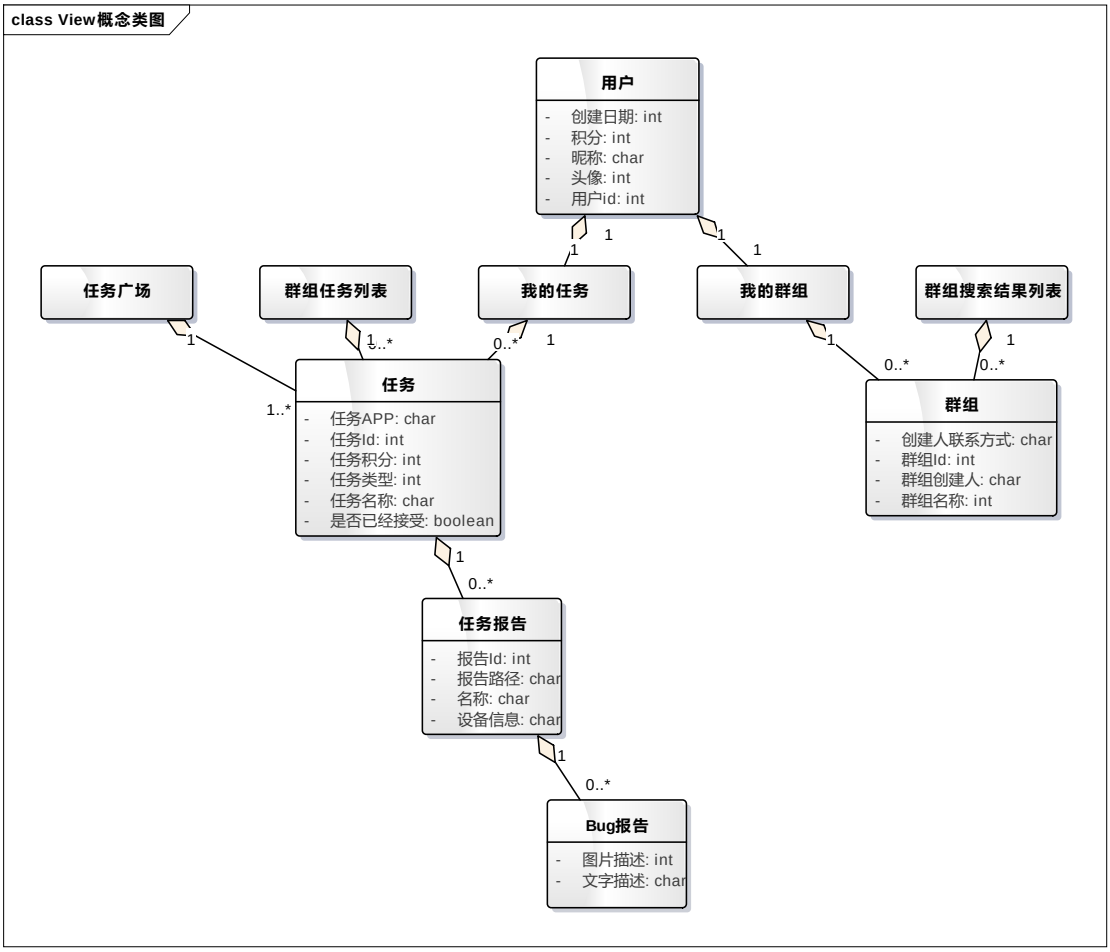


图 3.4 概念类图

3.3.2 体系结构设计-物理视图（Physical View）

以下是 Kikbug for iOS 的部署视图。本系统最终将部署在 App Store 上，目前 App Store 上有超过 150 万移动应用。由于 iOS 系统的生态系统为封闭性的，部署到 App Store 可以让所有用户通过简单的搜索，点击安装的方式完成本系统的安装。本系统运行的基础为 OC 中的 Cocoa Touch 库[5]，这是 Apple 为移动应用封装的一套 API，包含了 UIKit、MapKit、Game Kit 等等子库。UIKit 主要包括了一些 UI 组件，容器等。Cocoa Touch 运行于 Core Service 基础之上，Core Service 库包含了 Foundation 库、Core Data 等内容。Core Service 的下层即 Core OS 系统层。

[5] 关于 Cocoa Touch 的详细内容，请见 https://en.wikipedia.org/wiki/Cocoa_Touch

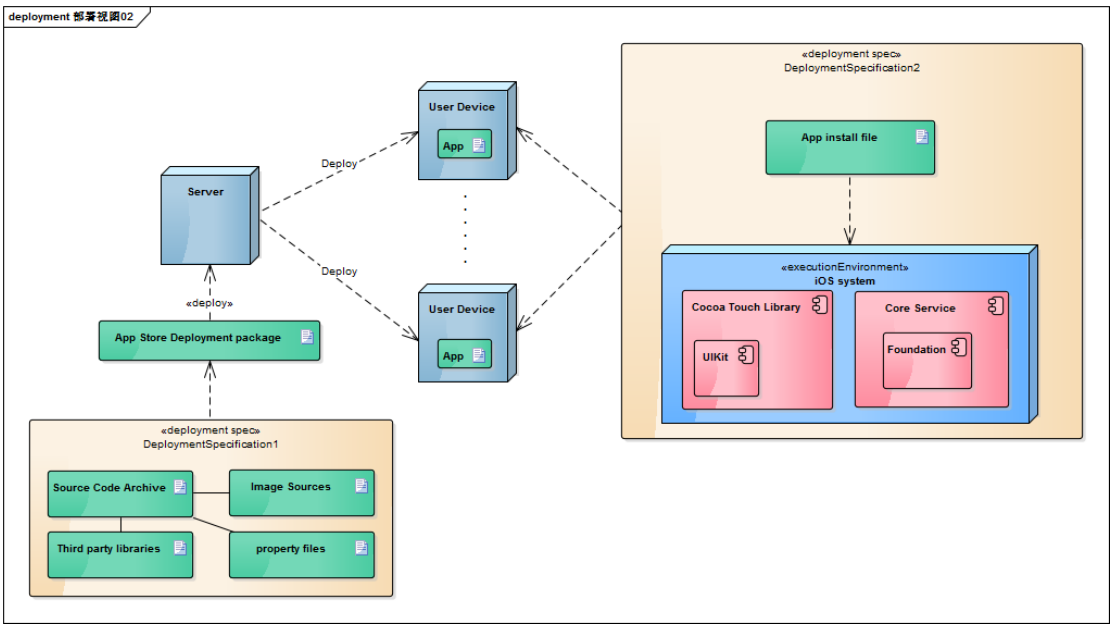


图 3.5 Kikbug iOS 系统部署视图

3.3.3 体系结构设计-开发视图（Development View）

图 3.6 描述了在 Kikbug 系统开发中的静态视图。图中重点描述了 Controller 模块和 Model 模块的开发包设计。

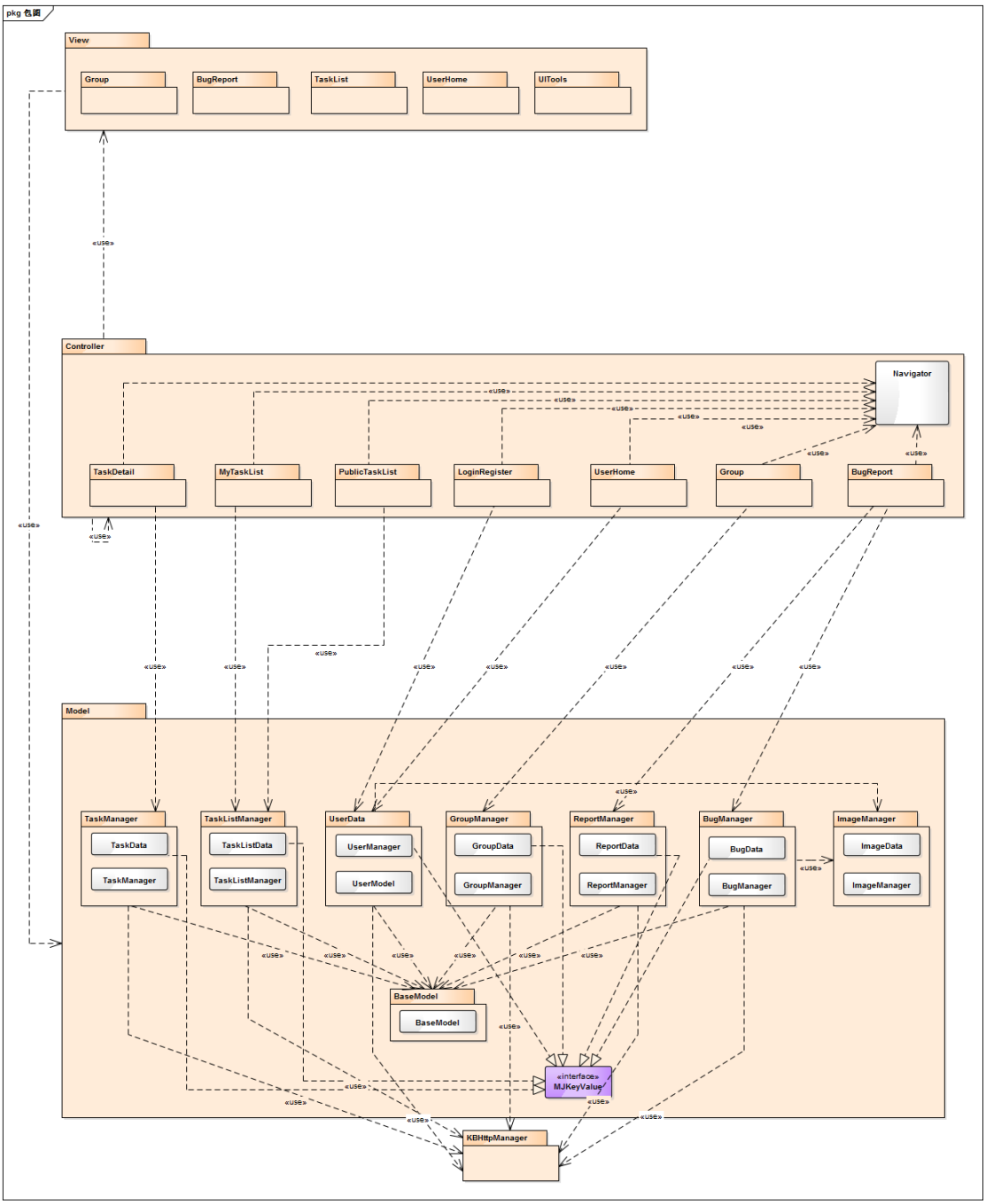


图 3.6 开发视图

3.4 本章小结

本章分为三大部分，分别为 Kikbug for iOS 项目整体概述、需求分析部分和系统概要设计部分。第一部分描述了 Kikbug 在 iOS 系统上的整体解决方案，以及与本文有关的主要内容。第二部分从用例、系统顺序图、非功能性需求的角度描述了 Kikbug 系统的需求。第三部分从整体上描述了 Kikbug 系统的体系结构设计，其中，开发视图描述了 Kikbug 系统的开发包设计。

第四章 Kikbug for iOS 详细设计与实现

4.1 项目子模块概述

由于本项目主要分为 Model，View，Controller 和 Public Tools 四个模块。因此接下来分别对这三个模块进行概要描述。从整体上描述各个模块的功能，职责以及划分的依据。关于各个模块的详细设计与思考将会在下个章节 4.2 中进行具体描述。

4.1.1 Model 模块

Model 模块主要是所有客户端在和服务器交互的过程中用到的数据模型，定义为 Objective-C 实体类，使用 MJExtension[10]进行转换。MJ 是一个第三方库，可以把 OC 的实体类与 JSON 数据进行互相转换。同时，该库提供了对于一些保留字的支持。例如，在 OC 语言中，“id”作为保留字是不可以定义为属性名，可以使用 MJExtension 提供的

```
+ (NSDictionary *)mj_replacedKeyFromPropertyName
```

方法进行转换。该类方法返回的字典中，键对应了定义在 OC 代码中的属性，而值则是该属性在 JSON 数据中的实际 key。解析 JSON 时，MJExtension 在遇到一个 key 时，如果发现这个 key 在上述方法返回的字典中，则将值赋到 OC 的属性中。

同时 MJExtension 还支持嵌套的 Model 定义。如果某个 Model 中包含着自定义的类作为属性，可以在

```
+ (NSDictionary*)mj_objectClassInArray
```

方法中，通过返回的字典指明如何解析自定义类。返回的字典中，key 代表了 OC 中需要进一步解析的属性名，而 value 则指明了用于解析该属性的类名。Value 的值类型可以为字符串或者 Class。

4.1.2 View 模块

View 模块包括各个页面的组件，由 Controller 创建和管理。View 模块依赖于 Model 模块的数据。View 模块的成员仅负责视图上的展示工作，不涉及任何数据调用和接口的调用。也不负责事件的处理。所有的事件处理，事件响应都在 Controller 中完成。当 View 中的一些封装好的部分需要通知其拥有者发生了用户操作时，一般采用代理（delegate[11]）的方式，实现 View 和 Controller 之间的解耦合，同时提高 View 的通用性移植性，便于复用。

4.1.3 Controller 模块

Controller 为客户端开发中的主要部分，对于 View 的使用和对于网络接口的请求在 Controller 中执行。部分过于复杂的界面会使得 Controller 的内容非常臃肿，因此，如何对 Controller 和 View 之间进行解耦以及 Controller 之间的互相调用和传递参数成为了 Controller 模块在设计与实现时的重点[12]。

在开发中，Controller 部分均继承自 KBViewController 类。在这个类中，我集成了一些通用的功能，包括：进度提示、事件提示、方法调用、以及架构上的便利性设计等等。关于这一部分的详细内容，将在 4.5 部分详细说明。

4.1.4 Public Tools 模块

这一模块中主要包含了为模块与模块之间或者模块内部之间提供公用性功能的辅助类，同时也包括整个架构上最为重要导航部分（Navigator）。首先，这个模块中包含了一些 Category 文件。Category 即分类，是 OC 中提供的在不改变原有代码的情况下为类动态的增加新的实例方法或者类方法和属性的一种机制[13]。其结合 Runtime 特性，可以在程序中提供很多便利。这个模块中的 Category 文件封装了对于 String 类型、Image、View 等的易用性改进。然后，CoreData 的辅助类封装了 CoreData 的工厂方法，CoreData 是 OC 提供的本地化对象存储的机制，其使用方法类似于 Hibernate 的形式。Public Tools 模块中最重要的是 Navigator 部分。这个部分把原本分散在程序各个部分的 push 和 present 操作集中到一起管理，提高了整体架构上的稳定性。在 OC 中，一个 Controller 对于另一个 Controller 的展示大体上分为两种：1、Push 2、present。在详细说明中，也会对这两种方式的区别以及其底层原理进行简单的解释。

4.2 Model 子模块的详细设计

4.2.1 数据模型的定义与解析

整个 Model 模块的结构按照业务逻辑分为 Task、TaskList、UserData、Group、Report、Bug、Image 几个部分。其中 Image 部分由于是单独与 UPYUN[6]交互的部分，所以独立出一个与具体业务逻辑无关的 Model。其余的 Model 均是按照业务上的大类来划分。所有的具体模型类均继承自父类 NSObject，并且实现了 MJKeyValue 协议。协议中定义的可选实现方法均由各个具体 Model 类来决定是否要实现，以实现对不同 JSON 数据的解析。

Model 部分的设计重点在于 KBBaseModel 类，KBBaseModel 的设计考虑了尽可能的减少重复代码。由于 Kikbug 后台返回的数据格式固定为{status：int，message：string，data：JSON}。因此 KBBaseModel 中定义的属性如下所示。

[6] UPYUN 为国内的 CDN 服务提供商，详情请见 https://www.upyun.com/zh/index.html

```
#define JSONSTIRNG @property (copy, nonatomic, readwrite) NSString*
#define JSONINT    @property (assign, nonatomic) NSInteger
#define JSONARRAY  @property (strong, nonatomic) NSArray*

@interface KBBaseModel : NSObject<MJKeyValue>
JSONSTIRNG message;
JSONINT status;
@property (strong, nonatomic) NSString* data;
@end
```

数据请求接口通过异步调用拿到数据之后，首先通过 KBBaseModel 对拿到的 JSON 数据进行第一次解析。即，将 JSON 数据 Map 到 KBBaseModel 实例中。然后再将 KBBaseModel 实例的 data 字段单独取出，用于具体 Model 的解析。这样设计的好处有两个：

1、避免重复

如果没有 KBBaseModel 类，那么每一个具体的 Model 类都需要有 message 字段和 status 字段。因此，这样避免了冗余的代码。

2、便于扩展

网络请求模块的实现由统一的接口封装，在网络接口的封装里，业务层的网络请求拿到的返回值直接是数据的 data 字段。对于 message 和 status 这两个字段的处理均收束到底层封装中去。对于以后可能加入的对于其他错误情况的统一处理等扩展提供了可能。

4.2.2 数据请求接口的调用

Model 模块中的每一个部分均包含所有的数据请求类以及对应的数据模型。例如，BugManager 中，既包含了 BugModel，也包含了请求 Bug 相关数据的接口，接口方法以类方法的形式组织在 BugManager 中。

Model 中请求数据的方法均使用异步调用，并且对于返回值的处理以 Block 的形式由调用者决定。OC 中，block 用于实现异步的消息返回或事件处理极为方便有效。Block 的本质是方法指针，与变量的概念相似。不同的是，block 在使用时可以定义参数以及返回值，与标准函数一致。

```
+ (void)fetchTaskDetailInfoWithTaskId:(NSString*)taskId
                        completion:(void (^)(KBTaskDetailModel*, NSError*))block
{
    NSString* url = GETURL_V2(@"GetTaskDetail");
    url = [url stringByReplacingOccurrencesOfString:@"{taskId}" withString:NSSTRI
NG_NOT_NIL(taskId)];
    [KBHttpManager sendGetHttpReqeustWithUrl:url
                                      Params:nil
                                    CallBack:^(id responseObject, NSError* error)
   {
      if (!error) {
          KBTaskDetailModel* model = [KBTaskDetailModel mj_objectWithKeyValues:re
sponseObject];
          block(model, nil);
      } else {
           block(nil, error);
      }
  }];
}
```

在上述代码中，当 Controller 调用了 fetchTaskDetailInfoWithTaskId 之后，传入参数为 taskId。代码通过 GETURL_V2 定义的宏，从配置文件中取出 http 请求的地址，然后设置参数，之后调用 KBHttpManager 发出 Get 请求。Get 请求是异步的，当获得返回值时，通过解析 responseObject 取得 model 的内容。然后调用 block 方法，使得调用者获得返回值进行界面的展示。而在上述整个过程中，调用者只需要知道传入的参数，以及获取到返回值之后需要怎样使用返回值。使用返回值的逻辑完全由调用者提供。

4.3 View 子模块的详细设计

4.3.1 对于UITableView以及UITableViewCell在架构上的思考以及优化

在 iOS 的移动应用开发中，UITableView 是一个非常重要的组件。在这个组件的设计中，效率是非常重要的课题。一般来说，很多地方都可以使用到 UITableView，尤其是各种列表的展示。首先，UITableView 在使用时定义了两个接口：1、UITableViewDataSource 2、UITableViewDelegate。

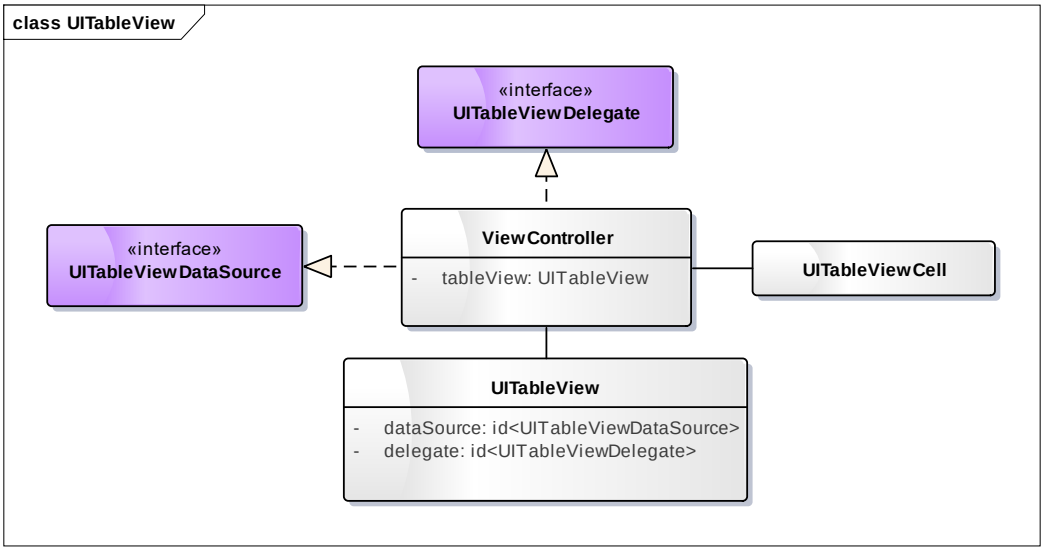


图 4. 1 Controller 与 UITableView 的交互

首先，Controller 拥有一个 UITableView 的实例。在初始化阶段，设置实例的 dataSource 字段和 delegate 字段为 Controller 自身。在接口的定义中，UITableViewDelegate 定义了以下方法。

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(N
SIndexPath *)indexPath;
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSectio
n:(NSInteger)section;
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:
(NSInteger)section;
- (NSString *)tableView:(UITableView *)tableView titleForFooterInSectio
n:(NSInteger)section;
（省略其他不常用方法）
```

其中，常用的为第一个方法 heightForRowAtIndexPath。这个方法根据不同的行数来返回对应行的行高。在代码中，可以设置固定的 cell 高度，也可以设置根据每一个 cell 不同的内容变化的高度。

在另外一个接口 UITableViewDataSource 的定义中，有以下几个方法。

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexP
ath:(NSIndexPath *)indexPath;
// Default is 1 if not implemented
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSI
nteger)section;
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:
(NSInteger)section;
（省略其他不常用方法）
```

在方法 cellForRowAtIndexPath 中，返回一个已经在初始化 tableView 时注册过的 cell 类型。在初始化 TableView 时，可以通过 identifier 注册 cell。由于一个 TableView 可能使用不止一种 cell，因此 identifier 可以有多种。而当需要使用某种 identifier 指定的 cell 时，可以通过 tableView 的方法：

```
dequeueReusableCellWithIdentifier:(NSString*)identifier
```

来获取一个可用的 cell 实例。OC 代码基于性能的考虑，由于在列表中可能有非常多的列表项，那么如果为每一个列表项分别创建一个 cell 实例，那么可能导致内存占用过高甚至不够使用的情况。考虑到实际上在显示时，一次只能显示一各屏幕的 cell 实例。那么 TableView 在一个 cell 滚动出了当前屏幕显示范围之后，就把这个实例放入到“可复用 cell”队列。这样，使用有限的 cell 实例，便可以展示所有的列表内容了。

而在实际开发中，随意的指定 identifier 导致在代码中出现多余的变量以及分散在程序各个部分的具体的变量。因此，我从 UITableViewCell 继承出一个用在本程序中的父组件 KBBaseTableViewCell，并且提供了方法：

```
+ (NSString *)cellIdentifier {
    return NSStringFromClass([self class]);
}
```

由于在 OC 中，类名必须是唯一的，所以这样定义 identifier 保证了其唯一性，也省去了单独定义时的随意与混乱。

4.3.2 View 模块 Cell 部分的类关系图

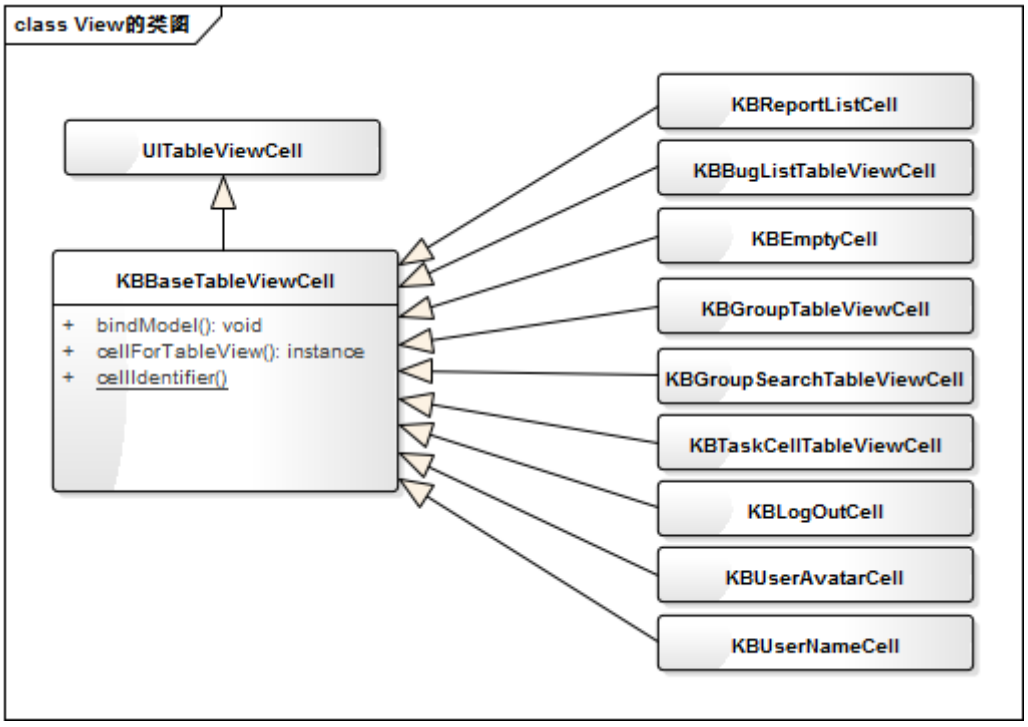


图 4.2 View 模块中 Cell 的继承关系

在代码中，KBBaseTableViewCell 封装了创建实例等公共代码，子类可以通过复写的方式自定义外观以及绑定数据源的逻辑。KBBaseTableViewCell 的设计借鉴了模板方法的思想。

4.4 Controller 子模块的详细设计

Controller 部分是代码中最为复杂的部分。往往，Controller 要负责数据接口的调用，数据模型的展示。因此控制器的设计在很大程度上影响了项目整体的耦合程度以及代码简洁程度。图 4.3 给出了 Kikbug 系统控制器的类图。我对系统原生的 UIViewController 基类进行了扩充。在接下来的第一部分将着重对 KBViewController 进行说明。

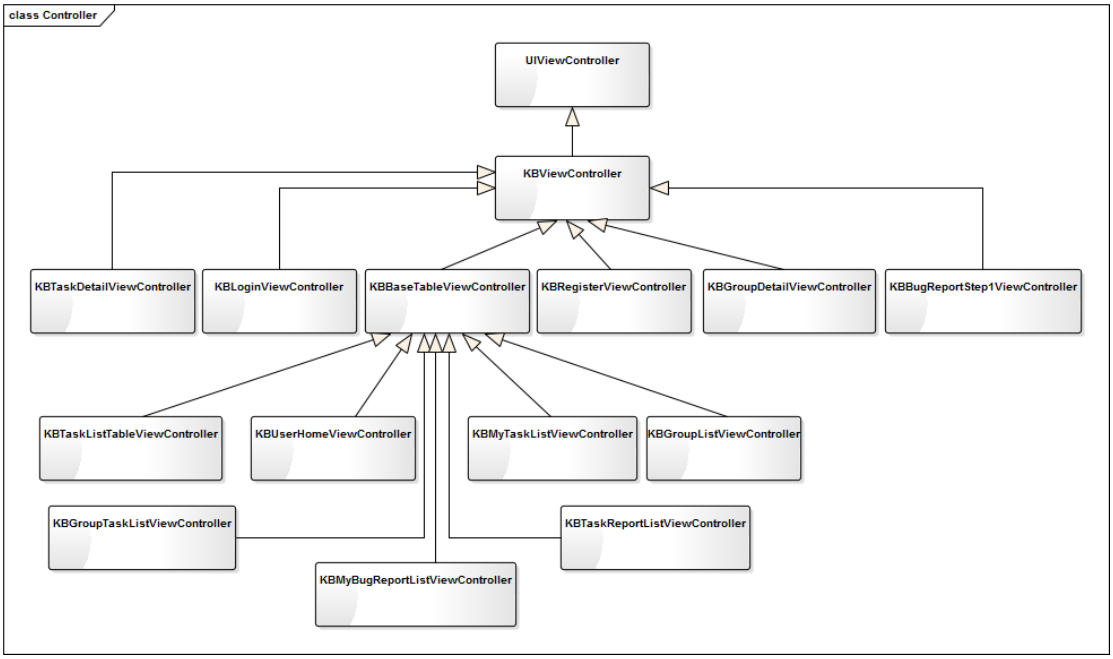


图 4. 3 Controller 类图

(一) KBViewController 的设计与部分实现

如图 4.3 所示，程序中使用自定义的 KBViewController 作为所有 Controller 的基类。在 KBViewController 类中，提供了以下方法：

```
@interface KBViewController : UIViewController
// 加载数据,只需要覆盖,不需要调用!
- (void)loadData;
//  设置界面约束,只需要覆盖,不需要调用!
- (void)configConstrains;

- (void)showLoadingView;
- (void)showLoadingViewWithText:(NSString*)text;
- (void)showLoadingViewWithText:(NSString*)text withDuration:(CGFloat)duration;
- (void)hideLoadingView;

- (void)showHudViewWithText:(NSString*)text;
- (void)showHudViewWithText:(NSString *)text inView:(UIView *)view;

- (void)showAlertViewWithText:(NSString*)text;
@end
```

对于 loadData 方法的调用放在- (void)viewWillAppear:(BOOL)animated 方法中。viewWillAppear 方法会在整个 ViewController 即将显示为顶层视图时被系统调用。因此子类只需要覆盖这个方法，替换为子类自己的 load 逻辑即可。同样的，项目使用约束来配置界面元素的适配。子类只需提供- (void)configConstrains 的具体实现即可，而不用关心其何时被调用。其他的几个 show&hide 方法（除了最后一个）均为功能性方法，用于提示用户成功或者失败，或者向用户展示非模态的信息。而 showAlertViewWithText 方法则是用于向用户展示模态信息框。这几个方法在各个具体的 Controller 中经常被用到，把实现集中到基类中既减少了代码量也便于日后的功能扩充或升级，可以保证各个页面在提示等节目上样式的统一。在架构上，为子类的开发提供了非常多的便利。

(二)　KBBaseTableViewController 的设计与部分实现

这个类在继承树中属于 KBViewController 的子类，与其他平级子类不同的是，这个类不是具体的视图类。其在作用上与 KBViewController 相同，为其他具体的 xxxTableViewController 提供基础功能。其接口如下所示。

```
@interface KBBaseTableViewController : KBViewController<UITableViewDataS
ource,UITableViewDelegate>
- (void)configTableView;
+ (void)configHeaderStyle:(UITableView *)tableView;
- (void)endRefreshing;

- (void)showEmptyView;
- (void)showEmptyViewWithText:(NSString *)text;
- (void)showErrorView;
- (void)removeEmptyView;
@end
```

同样的，为了最大程度的减少冗余代码，KBBaseTableViewController 的设计采用了模板模式。将 configTableView 方法提供给子类进行重写，子类只需要重写这个方法，即可自定义表视图的样式。实际上，在大多数情况下子类都不需要修改默认的实现。这样，对于表视图的开发大幅简化为对于表示图中的 cell 部分的开发。在实际开发中，表视图的开发需要新增的代码非常少，减少了 MVC 中，C 过于臃肿的问题。

(三)　用户个人中心的设计与实现

从概念上来看，个人中心与列表似乎没有联系。但是实际上考虑到 OC 的 UITableView 有 section 的概念。而对于每一个 section 又可以注册不同的 cell。因此，个人中心的每一个项均可以设计成 TableView 的一个 cell。整个个人中心的界面组件由不同种类的 cell 组成。

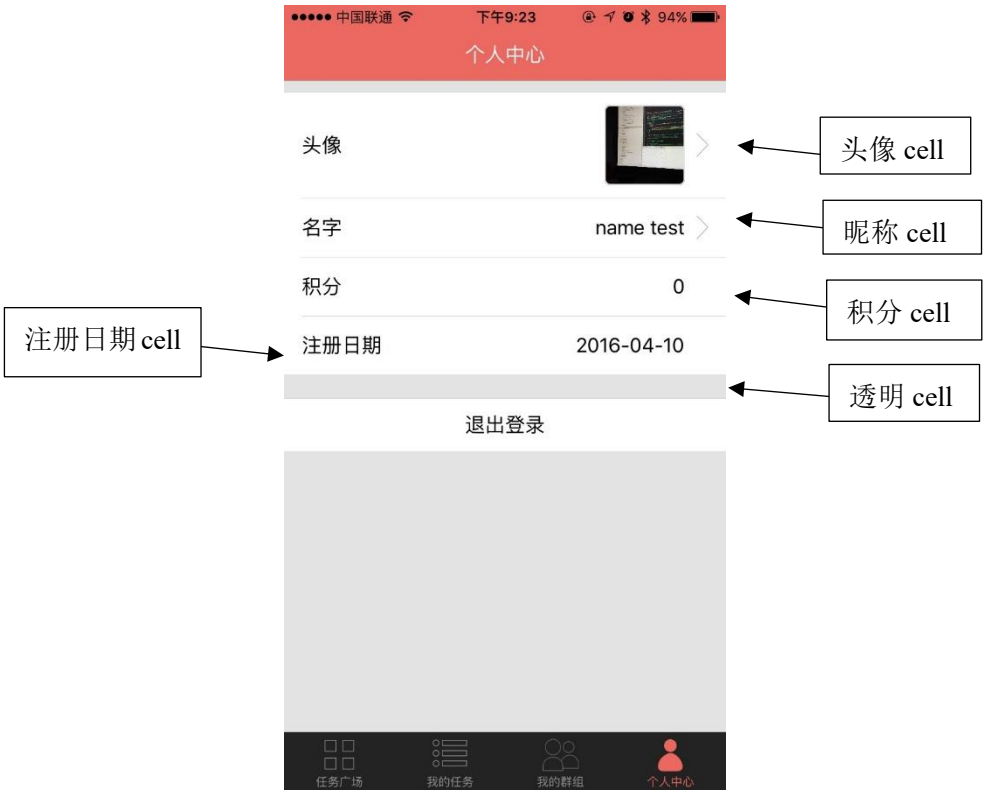


图 4.4 个人中心的 UI 说明

如图 4.4 所示，界面上的每一项元素都是一个列表项。不同的是，这个页面上的列表项均为不重复的。考虑到，如果直接将这 6 种不同的 cell 都由 Controller 管理那么 Controller 的复杂度就会过高，Controller 应该面向抽象而不是具体的 cell 对象。在这里，引入类 KBUserHomeCellModel。下面给出 KBUserHomeCellModel 的具体实现。

```
@interface KBUserHomeCellModel : NSObject
@property (strong, nonatomic) id model;
@property (strong, nonatomic) Class cellClass;
@property (assign, nonatomic) NSInteger cellHeight;
- (instancetype)initWithClass:(Class)className cellHeight:(CGFloat)cellHeight mod
el:(id)model;
+ (instancetype)emptyCellWithHeight:(CGFloat)height;
@end

@implementation KBUserHomeCellModel
- (instancetype)initWithClass:(Class)className cellHeight:(CGFloat)cellHeight mod
el:(id)model
{
    if (self = [super init]) {
        self.cellClass = className;
        self.cellHeight = cellHeight;
        self.model = model;
    }
    return self;
}

+ (instancetype)emptyCellWithHeight:(CGFloat)height
{
    KBUserHomeCellModel* model = [KBUserHomeCellModel new];
    if (model) {
        model.cellClass = [KBEmptyCell class];
        model.cellHeight = height;
        model.model = nil;
    }
    return model;
}

@end
```

ViewModel 对 cell 以及 model 同时进行封装。在创建实例时，指定 cell 的类型、cell 的高度以及 cell 对应的数据模型。Controller 从具体持有不同的 cell 对象转换为仅依赖 ViewModel，降低了 cell 和 Controller 之间的耦合。其类图如下：

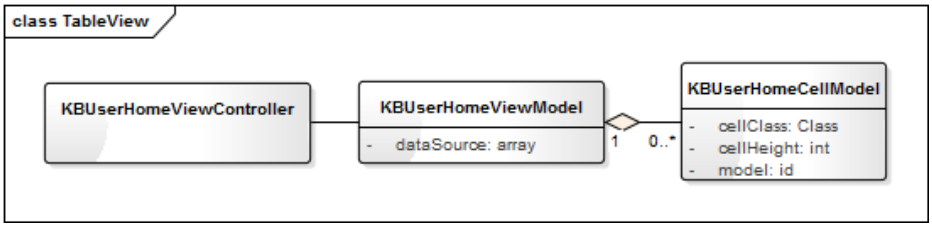


图 4.5 优化后的 Controller 依赖

从图 4.5 中可以看出，KBUserHomeViewController 不再通过具体的 cell 类型来创建 cell，而是通过 KBUserHomeCellModel 中的 cellClass 属性来创建 cell 实例。KBUserHomeViewController 并不关心其创建的 cell 是什么类型以及 model 是什么内容。它只需要知道通过指定的 class 去创建实例，然后用给定的 model 去填充 cell 实例就可以了。其关键代码如下：

```
- (UITableViewCell*)tableView:(UITableView*)tableView cellForRowAtIndexPath:
(NSIndexPath*)indexPath
{
    Class cls = self.dataSource[indexPath.row].cellClass;//(1)
    id cell = [cls cellForTableView:tableView];
    if ([cell respondsToSelector:@selector(bindModel:)]) {
        [cell performSelector:@selector(bindModel:) withObject:self.model];
    }
    return cell;
}
```

在 4.3.2 小节中，我具体的介绍了 Cell 部分的类结构设计。由于项目中所有的 Cell 由同一个父类 KBBaseTableViewCell 继承而来。而 KBBaseTableViewCell 中定义了 cellForTableView 方法和 bindModel 方法（见图 4.2）。在上述方法中，dataSource 为数组，其元素的类型为上文提到的 KBUserHomeCellModel。因此(1)语句从 cellModel 中取得要创建的 cell 类型，然后调用基类的静态方法 cellForTableView 获得实例对象，最后执行绑定模型操作。经过这样的操作之后，用户个人中心中显示的所有内容即为对应的 ViewModel 中创建的数组的内容，既便于修改扩充也实现了对 Controller 和 Cell 的解耦操作。

4.5 Public Tools 子模块的详细设计

公共工具模块由以下五部分组成：1、Category 2、CoreDataTools 3、HttpRequest 4、Navigation 5、Property。在这个小结，我主要想对 Navigation 的设计与实现进行说明。

在 OC 中，页面与页面之间的转换方式有两种，分别为 push 和 present。在页面的静态组织中，页面组件之间是树状结构。而在不同页面之间，OC 采用了堆栈的概念。当页面 1 push 了页面 2 时，那么相当于在页面 1 所处的堆栈中压入了页面 2。当页面 2 需要返回页面 1 时，对应的采用 pop 操作；如果是页面 A

present 了页面 B，那么和 push 不同的是，present 操作会新建一个堆栈。并且以页面 B 为最先入栈的元。之后，如果页面 B 要返回页面 A，那么采用的是不同于 pop，而是 dismiss 的方式。从界面的动画效果上来看，push 操作是从页面右侧推入一个新的界面在层级的最上层，而 present 操作则是展示一个模态的页面在最高层级上。

系统的堆栈操作问题在于，分散在程序各处的 push 和 present 操作均需要各个页面的堆栈实例。我在项目设计阶段，把所有对页面的 push 和 present 操作集中到 KBNavigator 中去执行。KBNavigator 采用了单例模式。另外，对于 push 操作时需要两个页面控制器的交互也做了架构上的优化，利用 url 的方式对具体的页面控制器实现抽象。这样，页面 A 在于页面 B 进行交互时，不需要知道页面 B 的类型而只需要知道页面 B 的 url 是什么即可。

综合上文中提到的两种改进，便是接下来论述的两个重点部分。

(一) 对于 Push 和 Present 操作的简化

如前文提到的，iOS 在界面与界面之间采用堆栈式的概念。这就要求，对于一个根视图而言（一个 App 必须拥有一个根视图），它必须持有堆栈的引用。当根视图 push 了新的页面之后，新的页面也拥有同一个堆栈的引用，以此类推。另外，当根视图 present 一个页面时，根视图会创建一个新的堆栈，并且把新页面作为这个堆栈的根视图。

在 OC 中，对于堆栈命名为 UINavigationController。下面介绍 UINavigationController 的 3 个属性：

1、presentedViewController：the current modal presented on screen

2、topViewController：view controller on top of the navigation stack

3、visibleViewController：the currently displayed view controller on screen (can be either a controller, a modal, a UINavigationController, a UITabbarController, etc)

属性 topViewController 保存着当前堆栈的栈顶视图，正是这个属性，使得页面与页面之间的 push 和 present 操作能够脱离各个视图，集中到 KBNavigator 统一操作。同时，为了能够解决 present 操作创建的新的堆栈问题，KBNavigator 中持有一个变量 presentingContainerVCNav，来保存新创建的视图堆栈。下面给出具体实现：

```
- (UINavigationController*)currentNavigationController
{
    UIViewController *presentedViewController = self.tabBarController.presented
ViewController;
    if ((presentedViewController && presentedViewController == self.presentingC
ontainerVCNav)) {
        return self.presentingContainerVCNav;
    } else if ([self.tabBarController selectedViewController] && !self.presenti
ngContainerVCNav){
        return (UINavigationController*)[self.tabBarController selectedViewCont
roller];
    } else {
        return self.presentingContainerVCNav;
    }
}

- (KBViewController*)topViewController
{
    UIViewController* topViewController = [self currentNavigationController].to
pViewController;
    if (![topViewController isKindOfClass:[KBViewController class]]) {
        return nil;
    }
    return (KBViewController*)topViewController;
}
```

由于只有当模态的 present 了一个视图时，才会对于 presentingContainer VCNav 这个字段赋值。所以，在 currentNavigationController 的实现中可以看出，在 if 语句的第一个分支即判断：如果存在着一个模态的视图创建出的堆栈，并且这个堆栈是当前显示在屏幕最顶层的。那么取出的堆栈即为 presentingCon tainerVCNav 所保存的。

除了第一种情况之外，当没有模态视图时，那么程序处于最简单的情况下，页面由 TabBarController 管理。由于 TabBarController 有四个 ViewControlle r，用户当前处于哪一个标签下，那么 selectedViewController 属性即为对应标签的堆栈。

最后一种情况是为登录页面单独考虑的。由于本系统要求用户必须在登录状态下才能操作，而登录状态下既不属于情况 1，也不属于情况 2。因此，对这种情形做特殊处理。

(二) 页面与页面之间的解耦合

程序使用 HHRouter[14]这个第三方库来进行具体视图与 url 的对应工作。首先，在程序启动时，需要注册所有视图到对应的 url。代码如下：

```
+ (void)registerLocalUrls
{
    [[HHRouter shared] map:LOGIN_PAGE_NAME toControllerClass:[KBLoginViewControl
ler class]];
    //以下省略
}
```

其中，LOGIN_PAGE_NAME 为宏定义的字符串，其实现思路为使用 HashMap 数据结构，将字符串与 Class 对应起来。当需要创建视图实例时，通过以下方法来快速获得视图对象，这与工厂方法的好处相符合：代码应面向抽象而不是具体实现。下面给出创建视图控制器类的具体代码示例：

```
UIViewController *vc = [[HHRouter shared] matchController:LOGIN_PAGE_NAME];
```

应用在本项目中，当用户点击了列表页的某个列表项时，需要 push 一个新的视图。原本，当前视图控制器需要知道被 push 的视图的具体信息如：类名、创建需要调用哪个方法、如何传参等。而应用了 HHRouter 之后，只需要知道接下来要展示的视图对应的宏名称即可，参数的传递可以使用架构提供的设置方法统一实现参数设置。以下给出从任务列表到任务详情页的具体操作：

```
- (void)tableView:(UITableView*)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    UIViewController* detailVC = [[HHRouter shared] matchController:TASK_DETAI
L];
    [detailVC setParams:@{@"taskId":self.dataSource[indexPath.row].taskId}];
    [UIManager showViewController:detailVC withShowType:KBUIManagerShowTypePus
h];
    [self hideLoadingView];
    [tableView deselectRowAtIndexPath:indexPath animated:NO];
}
```

在上述代码中，在取得 detailVC 实例之后有一步 setParams 的操作。这个操作为 HHRouter 通过 Category 机制为 UIViewController 新增的实例方法。任务详情视图可以通过重写的方式，实现传参，其参数使用字典的方式传递。

这样的设计使得各个视图类之间不需要相互 import，在编译时大大缩短了编译所需的时间，消除了视图控制器之间的互相依赖。

(三) 单例模式的应用

在代码中，UIManager 为宏定义的[KBNavigator sharedNavigator]，这是一个线程安全的单例实现。在定义单例时，采用了 GCD（Grand Central Dispatch），

确保在创建时，单例不受多线程的影响。GCD 是 Apple 开发的一个应用于多核编程的解决办法。

单例宏的具体实现代码如下：

```
#define SINGLETON_INTERFACE(className, singletonName) +(className*)singletonNam
e;

#define SINGLETON_IMPLEMENTION(className, singletonName)                    \
                                                                            \
    static className* _##singletonName = nil;                               \
                                                                            \
    +(className*)singletonName                                              \
    {                                                                       \
        static dispatch_once_t onceToken;                                   \
        dispatch_once(&onceToken, ^{                                        \
            _##singletonName = [[super allocWithZone:NULL] init];           \
        });                                                                 \
        return _##singletonName;                                            \
    }                                                                       \
                                                                            \
    +(id)allocWithZone : (struct _NSZone*)zone                              \
    {                                                                       \
        return [self singletonName];                                        \
    }                                                                       \
                                                                            \
    +(id)copyWithZone : (struct _NSZone*)zone                               \
    {                                                                       \
        return [self singletonName];                                        \
    }
```

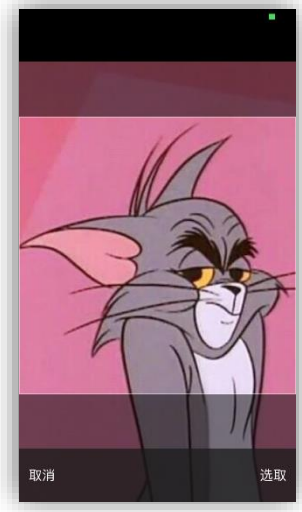
在本项目中，所有用到了单例模式的类，均使用宏定义的方式来实现。为项目的开发大大减少了冗余代码数量。

4.6 客户端运行截图

4.6.1 注册、登录页面

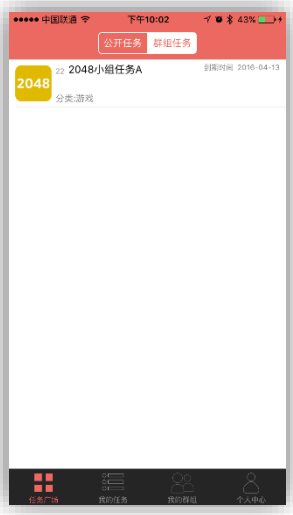
KikBug 客户端的登录可以使用手机号或者慕测邮箱登录，但是注册时只能使用手机号注册。

4.6.2 个人信息及其管理页面



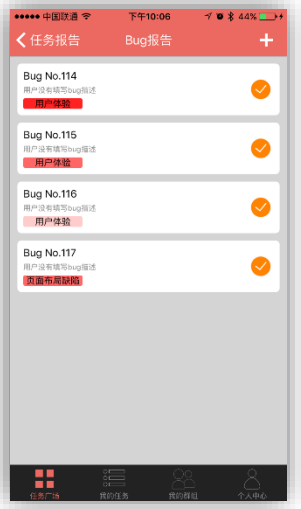
个人中心部分，用户可以自定义名称和头像。头像可以选择使用照相机或者本地相簿。

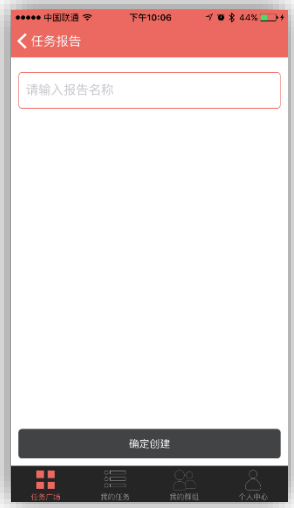
4.6.3 任务列表页面



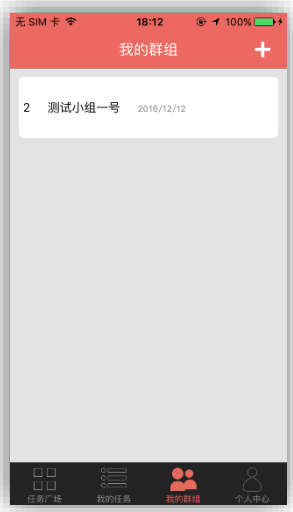
任务列表主要包括：任务广场任务列表、群组任务列表、我的任务任务列表。

4.6.4 任务详情页面以及任务报告和 Bug 报告页面



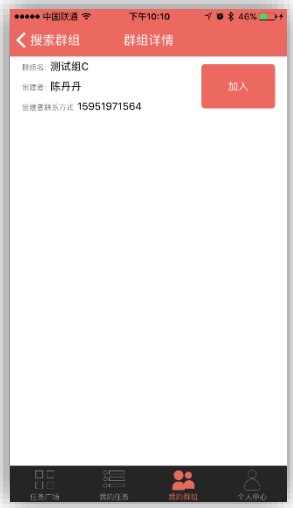


4.6.5 群组相关页面

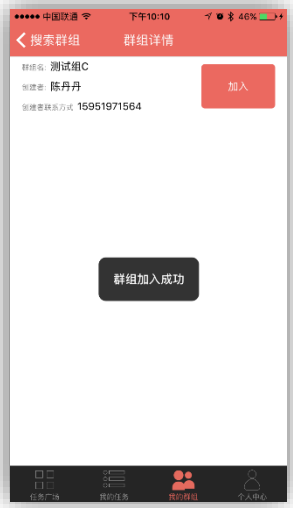

我的群组
2 测试小组一号 2016/12/12
任务广场
我的任务
我的群组
个人中心


我的群组
群组任务
这个群组还没有发布任务
重新加载

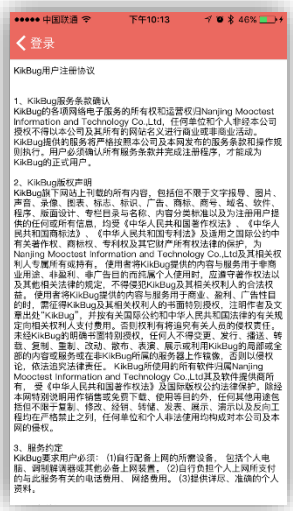

我的群组
搜索群组
输入小组关键字
任务广场
测试小组一号
群组B
测试组C
搜索


搜索群组
群组详情
测试组C
陈丹丹
15951971564
加入


搜索群组
群组详情
群组B
陈丹丹
加入
输入口令
取消
确定
空格
换行


搜索群组
群组详情
测试组C
陈丹丹
15951971564
加入
群组加入成功

4.6.6 用户协议页面


登录
KikBug用户注册协议

4.7 本章小结

本章首先从整体上概述系统的模块划分，之后重点对于各个模块进行局部的详细描述。本章介绍了 Model 模块的类继承关系与 Model 部分使用的主要技术，如何对 JSON 数据进行解析。将 Model 放在最开始描述也是考虑到了在实际开发中，Model 总是最先着手的地方。之后介绍了 View 模块的详细设计，View 模块是与 Model 交互最为紧密的部分。整个 View 模块的设计重点在于 Cell 部分，这个部分是项目架构中最为关键的设计。介绍完 Model 和 View 之后，Controller 部分主要是介绍了我在项目中定义的两个基类 KBViewController 和 KBBaseTableViewController，并且选取了用户个人中心 KBUserHomeViewController 作为介绍重点，给出了其实现代码。最后，本章描述了公共工具模块的设计与实现，作为解耦的重点，这部分着重介绍了 KBNavigator。在结尾处，本章附上了客户端的运行截图与对于截图的描述，进行了界面的 UI 展示。

第五章 总结与展望

5.1 总结

项目的开发过程是一个从无到有，从 0 到 1 的过程。这个项目也是我独立开发的第一个完整项目，项目代码完全开源在 Github 上。从刚开始自学接触 OC 这门开发语言，到后来在实习中不断学习，我感受到这门语言的独特魅力。在实习中，我学习到了很多在架构上和设计上需要特别注意的地方以及如何控制代码的质量，尽量减少 Crash。在 KikBug 项目的开发中，我也一直坚持以工业界的标准要求自己，这也在编码阶段给我带来很多便利。

代码编写过程中，在应用已有知识的基础上，我不断接触新的技能如 CoreData 与 AutoLayout，同时也不断思考如何把本科阶段学习到的软件工程思想和技术应用到项目中来。在项目编码阶段，应用了很多丁二玉老师传授的关于高内聚低耦合和设计模式的思想。这些知识，在体系结构设计方面和具体的代码结构上给了我很大帮助。

最终项目的目标是上线，能够真正为用户服务。我也在项目的上线阶段遇到了很多问题，这些是实习所接触不到的知识。

然而，项目由于受限于平台因素，无法突破沙盒的限制，对于测试数据的收集有一定的影响。如果脱离了吴迪同学开发的 SDK，则无法收集到用户测试时的点击操作等事件，只能收集到基本的设备信息。另外，由于项目的规模限制，在界面人机交互方面还有很多可以提高的地方。在体系结构的设计上，还有一些模块的划分可以进一步思考和改进。

MVC 是一个常用的结构，Apple 在 OC 中也非常推荐使用这样的结构。在我的实习过程中，接触过的两款用户数量在千万级的在线应用里，也采用了这样的结构。作为一个最为常用的架构选择，MVC 自然有其独有的优势，当然随着开发技术的发展，也不断的有更好的技术涌现，例如 MVP 和时下大热的 MVVM 等。

5.2 展望

首先，我想提出对于本项目在界面上的展望。在以后的维护升级中，界面需要更多的注重人机交互。由于项目的开发过程中没有美工的介入，我本身也不具备UI设计和图片编辑的技能,导致界面的UI设计非常粗糙。直到最后收尾阶段，才由李舒颖学姐联系了一名新传的学姐进行界面的设计。希望在日后的维护中，能够一直有专业的界面设计师提供UI支持和图片素材的提供。

目前，后台提供的数据都是暂时的假数据，希望能够尽快有真实数据可以用于测试场景中。希望能够尽快通过App Store的审核，早日上线。

在下一版本的更新中，我希望能够把控制器进行更深一步的清理，进一步解耦控制和View，提高View 的可复用性。目前项目代码中，控制器依然承担了过多的职责。在总结中，我提到了MVC和其他新技术，希望在日后的维护中，这个项目能够不断改进，引进新的技术。

在项目的语言选择上，我选择 OC 的原因一方面是因为其稳定成熟，两一方面也是考虑到自身技术上更熟悉 OC。对于 Swift，这门语言一定会成为取代 OC 的存在，希望当Swift 足够稳定之后，本项目可以逐渐向纯Swift转化。

项目开发中，我采用了 AutoLayout 进行页面的自适应布局。本来考虑的是能够兼容多平台：iPhone 和 iPad 等。在上线阶段，考虑到时间和工时因素，仅选择了 iPhone 平台进行发布。希望，在未来的版本中，能够真的实现全平台覆盖。

最后，作为一款测试其他App的App，本身的稳定性没有被监督。希望在新版本中,能够加入第三方的数据统计服务,或者引入自己的统计功能,在这方面，我在架构上留出了扩展的余地。

参考文献

[1] 智能手机应用安全关键技术探讨， Luo Xuan,Liang Baiqing,Pan Junbiao,Huang Yue，电信科学 2013.5.
[2] UTest, http://utest.qq.com/
[3] 班墨云测试，http://www.yunceshi.com/
[4] 《软件开发的技术基础-软件工程与计算（卷二）》,骆斌，丁二玉，刘钦，2012，机械工业出版社
[5] 《Head First 设计模式》 ，Eric Freeman，Elisabeth Freeman，2012，中国电力出版社
[6] MVVM 模式分析与应用，刘立，上海交通大学
[7] Don't let your UIViewController think for itself ,http://blog.ios-developers.io/dont-let-your-uiviewcontroller-think-for-itself/
[8] Objective-C Runtime Reference, https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/
[9] iPhone 屏幕尺寸、分辨率及适配 http://blog.csdn.net/phunxm/article/details/42174937/
[10] MJExtension，https://github.com/CoderMJLee/MJExtension
[11] Delegation, https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html#//apple_ref/doc/uid/TP40008195-CH14-SW1
[12] Model-View-Controller https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195-CH32-SW1
[13] Category, https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Category.html#//apple_ref/doc/uid/TP40008195-CH5-SW1
[14] HHRouter, https://github.com/Huohua/HHRouter

致谢

在论文完成之际，我想对在毕业设计阶段想我提供过帮助的所有同学，老师表达最诚挚的感谢。在毕业设计阶段，是陈振宇老师、刘钦老师在每周的项目演示时提出了宝贵的意见和建议。也感谢刘嘉老师，在项目启动阶段和我们一起讨论项目的初步需求，为项目的顺利完成奠定基础。另外，还有邵栋老师也在项目中期演示时参与了项目评审。是各位老师的意见，促使本项目不断改进。

在这里，我想特别提出感谢的是我的导师陈振宇老师，在本科毕业设计阶段提供了指导和帮助。项目所使用到的开发设备与测试设备均由陈振宇老师提供支持，感谢陈老师一直以来的监督和鞭策。早在大三时，我便向陈振宇老师提出了自己关于本科阶段学习的一些疑问以及毕业设计的选择，是陈老师的建议和想法使我确立了毕设的构思与大体想法。

其次，我想感谢 iSE 实验室的房春荣学长。是房春荣学长在项目过程中不断提供多方面的支持。孙一学长在项目过程中，也不断协调和监督 kikBug 项目的进展。由于 Kikbug 项目兼容 Mooc 系统的用户登录，感谢刘子聪学长对于 KikBug 作出的兼容性改进。另外，非常感谢 KikBug 另外一个小组中的张玄同学、陈丹丹同学，是他们为 KikBug 客户端提供了后台的接口，在调试阶段解决我在接口调用时出现的各种各样的问题。

最后要感谢的是南京大学软件学院对我四年来的培养和肯定，感谢陈琳老师和曹日喜老师在生活和学习上的监督与关照，感谢软件学院所有老师传授的专业知识和技能。是你们的付出和教育，让我能够顺利完成项目的毕业设计。在毕业论文的编写以及项目的开发阶段，我不断的感受到四年来学习到的专业知识在实际开发中的应用，这也会在将来的学习和工作中得到一次又一次的证明。