南京大学

本 科 毕 业 设 计

院　系 软件学院

专　业 软件工程

题　目 iOS 应用数据的自动收集与分析

年级 2012 级　学　号 121250164

学生姓名 吴迪

指导教师 陈振宇　职　称 副教授

论文提交日期

批注 [CF1]: 格式

南京大学本科生毕业论文（设计）中文摘要

毕业论文题目： iOS 应用数据的自动收集与分析

软件学院 院系 软件工程 专业 2012 级本科生姓名： 吴迪

指导教师（姓名、职称）： 陈振宇副教授

摘要：

幕测平台的搭建，方便了学生的测试课程。为了更好的操作移动应用测试，收集更详细的数据，为此开发了 Kikbug 工具。不仅扩展了幕测平台的测试覆盖性，更有利于自动化测试的实现。

技术简介：项目中使用了面向切面编程和方法调配技术，涉及iOS Runtime机制。采用 Category模式将编写的代码以最小化改动原有代码的形式加入。作为第三方导入进源代码的框架，采用Category模式能够最小程度的减小对源代码结构的影响。

项目组整体完成了Kikbug iOS端的实现，与幕测平台（mooctest）的对接。本人在项目中承担被测用户手机信息的收集，第三方库的植入，报告信息的上传与用户操作的监测。

关键字：类别，幕测平台，移动应用测试，面向切面向切面编程

批注 [CF2]: 不行就加个回车

批注 [CF3]: 控制你的描述范围，你的工作是 kikbug，iOS 端的信息收集和 crash 监控。整个项目跟慕测没关系，你的部分跟 kikbug.net 关系不大，主要是跟 iOS 端的交互。

批注 [CF4]: 慕测去掉，改成测试，crash 监控跟踪，可以加一点众包，写最相关的

批注 [CF5]: 这两部分合起来，介绍一下用什么技术解决了什么问题。

批注 [CF6]: 写英文就行

批注 [CF7]: 跟慕测没关系

批注 [CF8]: 重复了。最好是摘要里出现的字。

南京大学本科生毕业论文（设计）英文摘要

THESIS：The iOS Application data automatically Collecting and Analysis

DEPARTMENT： Software Institute

SPECIALIZATION: Software Engineering

UNDERGRADUATE: Di Wu

MENTOR: Zhenyu Chen

ABSTRACT：

Mooctest functions for the convenience of the students registering for experimental courses. Kikbug instruments have been developed to test the operations of mobile applications and also for better data collection, so that Mooctest could cover a wider range, which promotes the realization of automatic course experiments. This project has employed the technologies of AOP and Method Swizzling, and involves the mechanism of iOS Runtime. It uses the Category mode whereby adding new codes to original codes and minimizing changes to the latter. As the means of complementing original codes with new codes, the Category mode could reduce to the utmost the influences on previous structures.

The team responsible for this project has developed the Kikbug iOS system whereby joining it to Mooctest. Within this team, I was in charge of collecting information from the tested cell phones, uploading accumulated data and monitoring users' operations.

KEY WORDS: category,iOS,mooctest,AOP

目 录

图目录

表目录

第一章 引言

1.1 项目背景

随着时代发展，软件越来越贴近人们的生活，软件的质量也越来越越关于人们的生活质量。因此，软件测试的重要性更加突出。而专业的测试人员任务也愈加艰巨。幕测平台充分了利用了广大群众这一软件的最终用户，来测试一些无需专业知识即可完成的需求。在这一系列测试需求中，移动应用测试更符合众测用户，也更容易达成测试目标。为了更有利于组织测试人员，发布测试任务和回收测试结果，Kikbug移动端的开发显得十分必要。

Kikbug iOS端由于沙盒目录[1]（sandbox）的限制，使得自身无法收集和分析被测APP用户信息和测试信息，第三方库的加入十分必要。Kikbug可以在APP运行过程中就收集数据，记录用户行为，并在测试结束后上传至数据库保存，提供给数据分析人员研究。如果测试人员发现了Bug，可以通过查看用户行为记录数据来重现测试人员的操作步骤。通过查看记录的App运行时电量、网络、屏幕状态，可以发现App在特定状态下才能触发的Bug。

Kikbug iOS端提供给用户引导功能，当用户需要完成特定任务时，即跳转到待测App。跳转成功App启动后，数据收集用户行为监测功能就开始运行，当测试任务完成，用户通过按钮即可快速返回Kikbug iOS端，并返回收集的数据和用户行为。

批注 [CF9]: 去掉，只要写众测

批注 [CF10]: 换种描述方式，这里是介绍背景，应该写开发者调试都需要哪些信息。这段的内容放到本文的工作里面作为总述

批注 [CF11]: 同上

1.2 应用数据收集和分析现状

1.2.1 应用数据统计工具

友盟+

2016 年初由友盟、CNZZ、缔元信.网络数据这三家国内顶尖的大数据分析公司合并而成。是目前国内最大的第三方数据统计工具。友盟的移动数据服务包括应用统计分析、游戏统计分析、用户行为分析和错误分析等。它可以帮助移动

应用开发者分析和统计流量来源、内容使用、用户属性和行为数据。分析 App 的用户人群，最主要的机型。友盟+的统计图表如下图 1.1 所示。

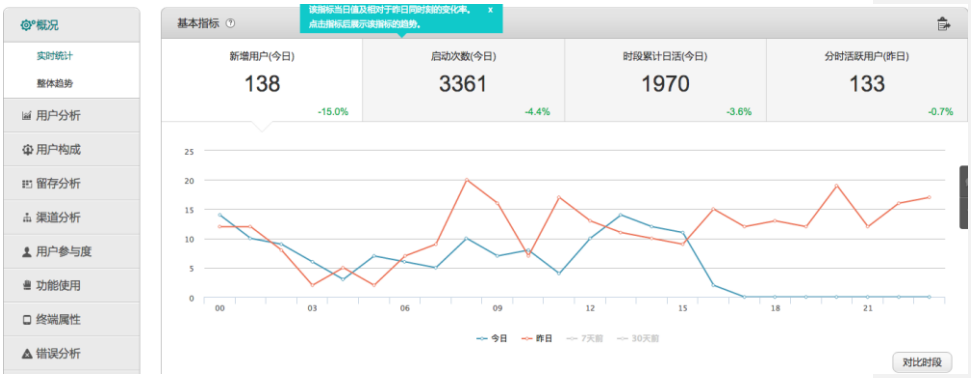

图 1.1 友盟+图

Flurry

Flurry成立于2005年，由企业家Sean Byrnes创立。是作为移动应用统计分析领域里的标杆平台，提供了全面的数据分析功能。它允许用户通过数据观察分析消费者行为。为不同级别的用户提供了不同的功能模块。Flurry的部分统计图表如下图1.2所示。

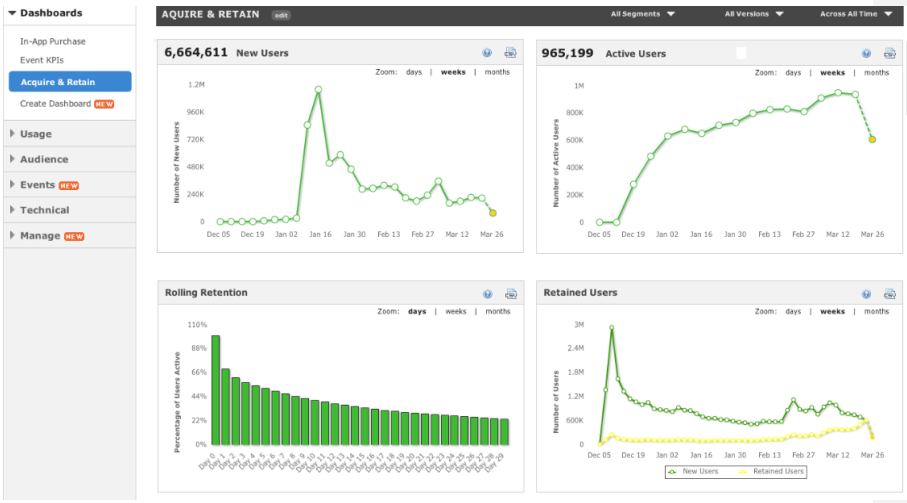

图 1.2 Flurry 图

1.2.2 Bug 与 crash 管理工具

批注 [CF12]: 项目背景要介绍 crash 相关的，不然这边太突兀

Bugtags

2014年10月，Bugtags 原型系统诞生。

2015年8月，Bugtags 1.0初版上线。

Bugtags目标是成为移动时代首选Bug管理系统，它可以在应用中所见即所得提交Bug，自动上传截图，操作步骤，控制台等数据。你提交的数据能够在网页中通过仪表盘的形式展现出来。提交问题的同时，它也收集了手机信息，应用运行时的数据。当应用闪退时Bugtags 也能自动提交至云平台。由于其目标人群是企业，所以目前大部分功能都需购买其企业版才能享受。Bugtags发现bug后的部分统计数据如下图1.3所示。

图 1.3 Bugtags 图

1.3 项目功能概述

本项目是一个集成于被测App的第三方库，当通过Kikbug客户端启动被测App时，此第三方库启动。插件能够收集被测App所在手机的信息，用户操作的行为，应用产生bug时的日志。这些信息能够为研究人员提供信息，从而自动化还原用户操作步骤，并模拟App产生Bug的特定条件。在测试结束后，本项目将产生的信息文件上传至云端，将文件地址返回给Kikbug，供其上传至服务器并保存。

对于不同的涉众，本项目有以下功能：

对于众测人员：

1.生成一个按钮，以便于返回Kikbug客户端。

2.生成一个按钮，以记录用户是否发现Bug。

3.提交操作步骤至服务器。

4.提交Bug日志至服务器。

5.提交手机和应用信息至服务器。

对于研究人员：

1.查看用户操作步骤。

2.查看 Bug 日志，方便统计手机、应用信息与 Bug 信息。

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第三方库的开发、与kikbug客户端的交互和论文撰写工作。 本论文：

第一章是论文的引文和前言主要介绍了项目背景，当前数据统计和移动应用Bug管理工具，并论述本论文的主要内容。

第二章是项目所用到的技术概述，主要介绍实现Kikbug第三方库所用到的相关理论知识和技术研究。

第三章是项目的概述和需求分析。描述项目的整体框架。

第四章是项目的不同模块的详细设计和实现。

第五章是 总结与展望。支出项目的缺点和不足支出，并陈述该项目未来可扩展的方向。

批注 [CF13]: 1）

批注 [CF14]: 本文就是你写的，这个不用提。

批注 [CF15]: 写个一句话就行

批注 [CF16]: 简单的几行，格式一致，要简洁，拒绝错别字

第二章 Kikbug iOS 第三方类库项目技术概述

移动应用测试是针对移动应用而进行的测试，包括人工测试和自动化测试。而目前移动应用开发进度过慢的主要问题就是人工测试效率低下。开发公司将测试任务外包给Mooctest，Mooctest通过Kikbug分发给众测人员。这样可以节省时间和金钱。本章将从理论角度，讲述如何在众测人员跳转到被测App后记录数据。

批注 [CF17]: 跟 mooctest 没任何关系

2.1 iOS 系统架构

对于原版iOS，苹果官方没有提供任何入口得意访问iOS文件系统。因此本章节将对iOS中的沙盒和权限以及文件目录做介绍。

批注 [CF18]:

2.1.1 iOS 目录结构

iOS的文件目录与Mac OS X有血脉关系，Mac OS X则是基于UNIX操作系统。在Mac OS X中常见的目录结构[2]为：

/根目录

/bin:存放二进制文件

/boot:存放使系统能成功启动的文件。iOS中为空

/dev:存放设备文件。/sbin

/etc：在iOS中指向/private/etc

/lib:存放系统库文件。iOS中为空　/mnt:iOS中为空

/private:/tmp:临时目录。iOS中指向/private/tmp

/usr:存放大多数用户工具和程序。

/var:存放会变更的文件。例如：日志，临时文件，用户数据等。

iOS所对应的大多数功能都来自于iOS的特有目录：

/Applications:存放系统App和越狱App，但不包含App Store下载的App。

/Developer：/Library:存放系统App的数据。

/User:用户目录,存放大量的用户数据。

/var/mobile/Media/DCIM:存放照片

/var/mobile/Library/SMS:存放短信

/var/mobile/Library/Mail:存放邮件

/var/wireless/Library/CallHistory:通话记录

/var/mobile/Applications:存放通过App store下载的App。

通常App使用的文件目录为

<Application_Home>/Document:存取用户生成的文件数据等，这些文件无法重新创建，可以通过iCloud进行备份。

<Application_Home>/Library:存取可以重新生成的数据，例如：新闻，地图的缓存文件。

<Application_Home>/tmp：存取临时生成的数据，用户在使用完这些数据后应该及时删除，以防占用存储空间。

kikbug第三方类库的文件存储目录如下图2.1所示：

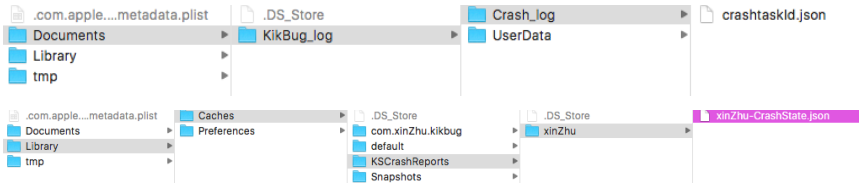

图 2.1 kikbug 项目目录结构

2.1.2 iOS 文件权限

iOS 是一个多用户操作系统，不同用户的所用权和使用权不同，mobile 无法通过调用 reboot 命令重启 iOS 系统。iOS 通过从高到低的 3 位来分别表示读权限，写权限和执行权限。同时，用户有三证可能：1.属主用户。2.在属主组里，但不是属主用户。3.不是属主用户，也不在属主组里。因此通常使用 9 位来代表一个文件的权限。如 111100100 就代表 rwxr—r—即该文件的属主用户拥有 rwx 权限，而其他用户只有 r 权限。通常会将 9bit 转化成十六进制 744.

2.2Bugsnag-cocoa

Bugsnag-cocoa[3]是一个开源的crash分析项目，它主要分为两个模块，KSCrash模块和Bugsnag模块。KSCrash主要负责监听crash事件，并将crash信息保存到本地。bugsnag负责优化crash信息，组织数据并将文件上传至服务器。

2.2.1 KSCrash 模块主要功能

KSCrash[4]主要可以处理以下异常类型：1.mach 内核异常（iOS系统自带的 Apple’s Crash Reporter 记录在设备中的Crash日志，Exception Type项通常会包含两个元素： Mach异常 和 Unix信号。）2.Fatal signals 3.C++ exceptions 4.Objective-C exceptions 5.Main thread deadlock (experimental)6.Custom crashes (e.g. from scripting languages)。KSCrash自己支持以下上传方式Hockey、QuincyKit、Victory、Email。

KSCrash还包含一些高级功能1.on-device 符号化 2.高功能性 3.可定制的用户数据（KSCrash.h）4.zombie对象跟踪（指针被释放，然而对象还在内存中） 5.死循环探测（还不稳定，会直接shut down app，提供主线程栈信息，app在主线程启动，会冲突）6.内存检查 7.定制奔溃处理代码。

2.2.2 KSCrash 部分源代码分析

批注 [CF19]: 只列关键的信息，不要列大段代码

KSCrash.h

1.报告任何crash信息。crash报告目录 $APP_HOME/Library/Caches/KSCrashReports

2.UserInfo

3.KSCDeleteBehavior//是否删除所有报告

4.KSCrashType//crash类型

5.zombieCacheSize//很少用到了

6.deadlockWatchdogInterval//规定主线程可以无响应运行的最长时间

7.searchThreadNames//是否获取线程名

8.introspectMemory//是否检查内存

9.doNotIntrospectClasses//数组类型，规定了一系列类（不能被检查的，有效保证信息安全）

10.定制用户自己的异常信息

KSCrashAdvanced.h

1.activeDurationSinceLastCrash//间隔上次奔溃总共活跃时间

2.backgroundDurationSinceLastCrash//间隔上次奔溃运行时间

3.launchesSinceLastCrash

4.sessionsSinceLastCrash

5.activeDurationSinceLaunch

6.backgroundDurationSinceLaunch

7.sessionsSinceLaunch

8.crashedLastLaunch //上次运行是否产生crash信息

9.reportCount

10.allReports//NSArray

11.crashReportStore //property: path and reportCount

12.sink

13.onCrash//异步处理crash报告

14.logFilePath//日志保存所在目录

15.printTraceToStdout//是否打印栈信息

16.redirectConsoleLogsToDefaultFile//设置logFilePath

KSSystemInfo.h//通过文件目录，UIDevice,NSTimeZone获得

1.app_start_time //启动时间

2.app_uuid //唯一标识符

3.boot_time //引导时间

4.cpu

5.kernel_version //内核版本

6.machine //机器型号

7.memory //内存使用率

8.model //型号

9.os_version //os版本

10.process // 进程

12.system_name

13.system_version

14.time_zone//时区

KSCrashSentry.h

1.handlingCrash//是否处理crash

2.crashedDuringCrashHandling

3.registersAreValid

4.isStackOverflow//探测是否栈溢出

5.crashType

6.crashReason// char*

7.stackTraceLength

8.NSException//struct{ name};

9.CPPEcxeption//struct{ name};

10.signal//struct{userContext;signalInfo};

11.userException//{name;lineOfCode;customStackTrace;customStackTraceLength}

2.2.3 Bugsnag 配置信息

[Bugsnag configuration].apiKey = @“YOUR-API-KEY”];//配置API Key，通过使用API Key可以在bugsnag官网上查看自己的App信息

[Bugsnag configuration].appVersion = @“5.3.55”;//配置App版本

[Bugsnag configuration].autoNotify = NO; //配置是否自动不高

BugsnagConfiguration *config = [[BugsnagConfiguration alloc] init];

config.notifyURL = [NSURL URLWithString:@“YOUR_ADDRESS”];//配置上传的服务器URL，如果不声明此行，内部文件有声明默认地址代码

2.2.4Bugsnag 部分文件信息

bugsnag.h

1.configuration

2.apikey

2.BugsnagConfiguration.h

1.apikey

2.notifyURL

3.releaseStage

4.notifyReleaseStages

5.context

6.appVersion

BugsnagCrashReport.h //readonly

1.releaseStage

2.notifyReleaseStages

3.context

4.appVersion

5.binaryImages//NSArray

6.threads//NSArray

7.error//NSDictionary

8.severity

9.dsymUUID

10.deviceAppHash

11.depth

12.metaData

13.appStats;// These shouldnt be exposed

14.system;

15.state;

BugsnagIosNotifier.h

1.电池信息

2.方向//landscape or portrait

3.lowMemoryWarning

BugsnagMetaData.h

1.metaData

2.tabInfo//显示在网页上有用

3.tabAttribute

BugsnagSink.h

1.filterReports//

2.HTTP//将各种数据整合，上传

2.3 Runntime 技术

批注 [CF20]:

2.3.1 消息传递

Objective-C是一门动态语言，它将许多在编译和链接时做的事放到运行时来处理。C语言使用“静态绑定”，编译期就能确定运行时所确定调用的函数。函数地址是硬编码在指令之中。而OC所调用的函数直到运行期才能确定，这就是“动态绑定”[5]。给对象发送消息为[someObj messageName:param],其中someObj为“接收者”,messageName 叫做“选择子”[6]，也就是OC中常用的“Selector”，选择子与参数合起来为“消息”也就是message。编译器看到此消息，就把它转化为一条标准的C语言调用函数，这一函数是消息传递乃至Runntime的核心objc_msgSend。objc_msgSenf 的原型为 void objc_msgSend(id self, SEL cmd,…)，此函数参数个数可变，第一个参数代表接受者，第二个代表选择子（SEL即为选择子selector的类型），后续参数是消息中所传递的参数。[someObj messageName:param]通过编译转化为objc_msgSend(someObj,@selector(messageName:),param)，objc_msgSend会在

接受者所属类的中搜寻方法列表，如果没有找到，就会沿着继承体系向上继续查找，直至NSObject类，如果还没找到，就会执行消息转发（message forwarding）。

2.3.2 消息转发

对于理解某条消息，程序代码中必须实现其对应的方法。但是在编译期间不会对发送无法解读的消息报错，因为在运行期间可以继续向类中添加方法，所以编译器是无法确定某个类中的方法是否实现。当对象接收到无法解读的消息是，会启动消息转发，程序员可由此过程来告诉对象如何处理没有找到的消息。消息转发很常见，只是很多人没有留意或者并不知道。但你传递一个无法解读的消息或空值。控制台常常会报异常“unrecognized selector sent to instance”，这段消息就是由NSObject所抛出的异常。

2.3.3 Method Swizzling

本项目中需要对用户的行为进行追踪，简单的说，就是将用户跳转到某个View或者点击到某个Button的时候，将用户的这个事件结合时间戳记录下来。有三种方法可以实现：1.手动添加，及在每个ViewController的viewDidAppear中添加代码，记录用户跳转到这个页面。在Button的点击方法中，添加代码，记录用户点击了这个button。这种方法有一个显著的缺点，那就是用户添加过多的代码，破坏了原有项目的整洁性。原有项目的逻辑会被破坏，随着项目的增大，要找到之前的代码会变得困难，用户也经常会忘记添加代码。2.继承或者类别，通过继承或类别，能够将记录代码从主逻辑中剥离，但同时，你需要继承UiViewController，UiBarViewcontroller等所有的Viewcontroller，不仅如此，每个button点击事件的方法也不会相同。使用类别，你也无法控制类别中的方法是否会被调用。3.方法调配（Method swizzling）。

每个类都有一个方法列表，将选择子映射到相关的方法实现之上，“动态消息派发系统”就依次找到应该调用的方法。这些方法都以函数指针（IMP）的形式来表示。如下图2.2所示：每一个方法都有与其相对应的函数指针。

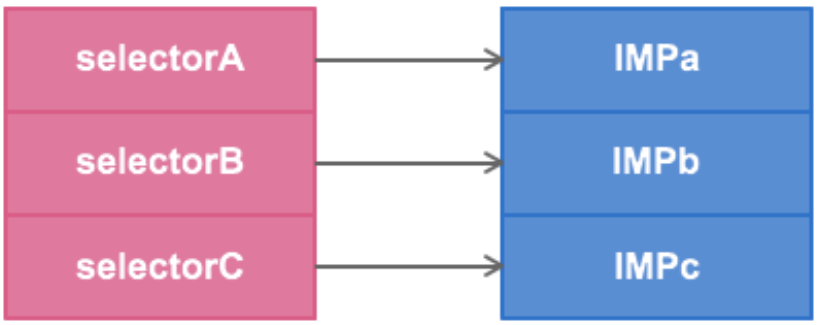

图 2.2 IMP 图

因为 OC 对象在收到消息之后，被调用方法需要在运行期间才能解析出来。因此，我们可以在运行期改变选择子所对应的函数指针，从而达到交换方法的目的。甚至，我们可以新增方法实现，与原有的方法进行交换。如下图 2.3 所示：selectorN 是新增的方法，将它的函数指针与原有方法的函数指针进行调换过后，就可以为已经存在的方法实现新的功能。

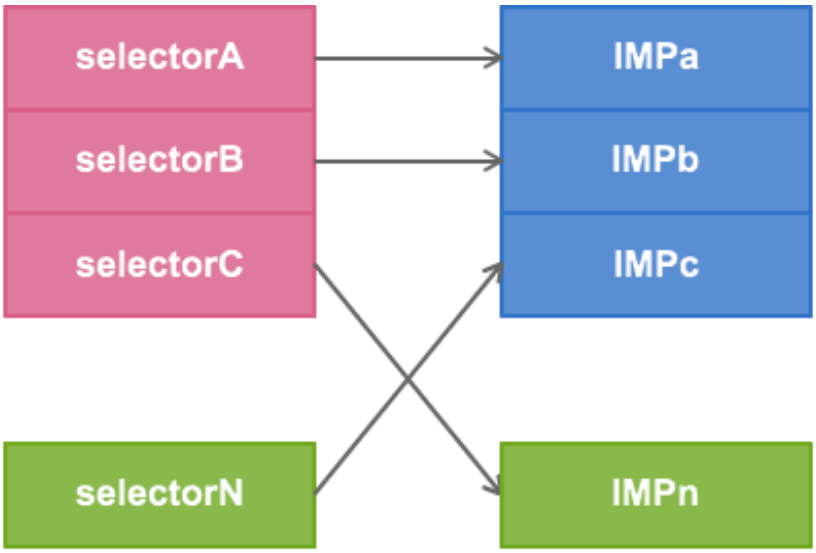

图 2.3 方法调换图

因此可以自己新写一个 new_viewDidAppear 方法，在其中新增记录代码，最后再调用原有的 viewDidAppear 方法。现在当系统执行 viewDidAppear 时，会调用 new_viewDidAppear,并通过调用 new_viewDidAppear 里面的 viewDidAppear 方法来实现原有的方法。

2.4 委托与分类

2.4.1 委托与协议

OC语言中有一项特性叫做协议（protocol），它与java的接口（interface）相似。和java一样，OC并不支持多重继承，因此，我们把某个类应该实现发的方法放在协议中。使用协议能够把代码变得易于维护。而委托模式（delegate）就是协议最常见的用途。

对象之间经常需要通信，委托模式就是一种常用的通信方式。该模式的主旨是：定义一套接口，某个对象想接受另一个对象的委托则必须先实现此接口。另一个对象可以给其委托对象回传一些信息，也可以在某个时间发生时通知委托对象。比如：A类中声明了一个协议，协议中有getInfo的方法。A类中还有一个Button，当点击这个Button就调用协议中的getInfo方法。B类实现了委托对象的方法，即声明自己遵从委托协议，然后把协议中的getInfo方法实现。这样当点击了A类中的方法，B类中实现的getInfo方法就会被调用。因此，我们可以通过getInfo方法传值，也可以通过getInfo方法传递Button被点击了这个事件。

2.4.2 分类与关联对象

分类（category）[7]也是OC中一项重要的语言特性。利用分类，我们可以不用继承子类，直接为当前类添加方法。正如上节所说，因为OC是动态语言，所以才能实现分类特性。一个类有时会有许多方法，这些方法的所有代码全部写在一个巨大的实现文件中。有时这么做是合理的，有时可以使用分类机制，将类方法按逻辑分到几个分区中。

在分类中声明属性，编译期间会给出警告信息，说无法合成属性有关的实例变量，所以开发者需要在分类中为该属性的实现提供存取方法。关联对象是通过

键来管理存值。通过 void objc_setAssociatedObject(id object ,void *key,id value,objc_AssociationPolicy policy)可以以给定的键和策略为某对象设置关联对象值，其中 policy 是关联类型，它与@property 属性相对应。通过 id objc_getAssociatedObject(id object ,void *key)则可以取出关联对象值。

2.5 UIView 的触摸事件

在本项目中，通过kikbug跳转到待测App会出现一个悬浮按钮，这个悬浮按钮的实现则涉及到了iOS的UIResponder的介绍。如下图2.4所示：下图展示了NSObject的继承关系：

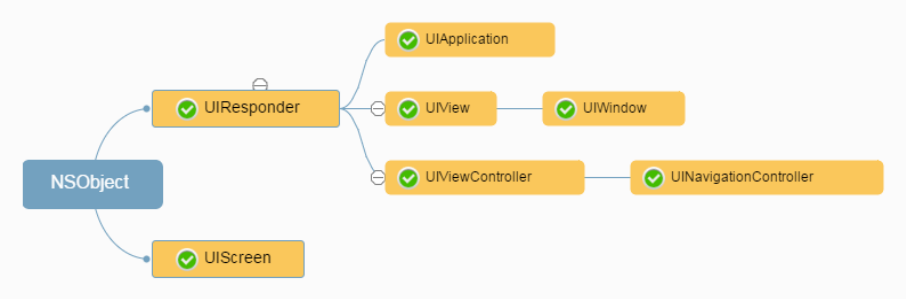

图 2.4 NSObject 继承图

UIApplication 和 UIview，UIViewController 都继承于 UIResponder，UIWindow 则继承与 UIVIew，UIScreen 则直接继承于 NSObject。UIScreen 通常用来获取屏幕尺寸。一个 UIApplication 则代表了一个应用程序，因此 UIApplication 是一个单例对象可以通过 UIApplication.sharedApplication 获得。UIView 的实例就是一块区域中的视图，它可以包含嵌套的 view 和其他 UIController 组件。UIView 处理该区域的绘制和触屏事件。UIWindow 继承于 UIView，一个应用程序通常也只有一个 UIWindow，即使存在多个 UIWindow，也只能有一个 UIWindow 可以接受用户的触屏事件。UIViewController 负责处理 UIView 实例的生命周期，资源的加载与释放。也处理页面由于设备旋转而导致的界面绘制和复杂界面的交互。

如下图所示：图 2.5 中是 kikbug 第三方项目的 UI 组件图，描述了 UI 控件的包含关系，图 2.6 是项目的层次结构图，其中悬浮按钮位于最顶层。

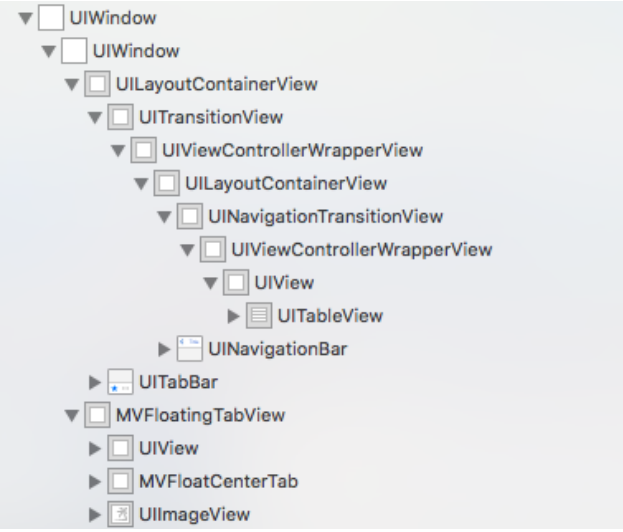
UIWindow
UIWindow
UILayoutContainerView
UITransitionView
UIViewControllerWrapperView
UILayoutContainerView
UINavigationTransitionView
UIViewControllerWrapperView
UIView
UITableView
UINavigationBar
UITabBar
MVFloatingTabView
UIView
MVFloatCenterTab
UIImageView

图 2.5 UI 组件图

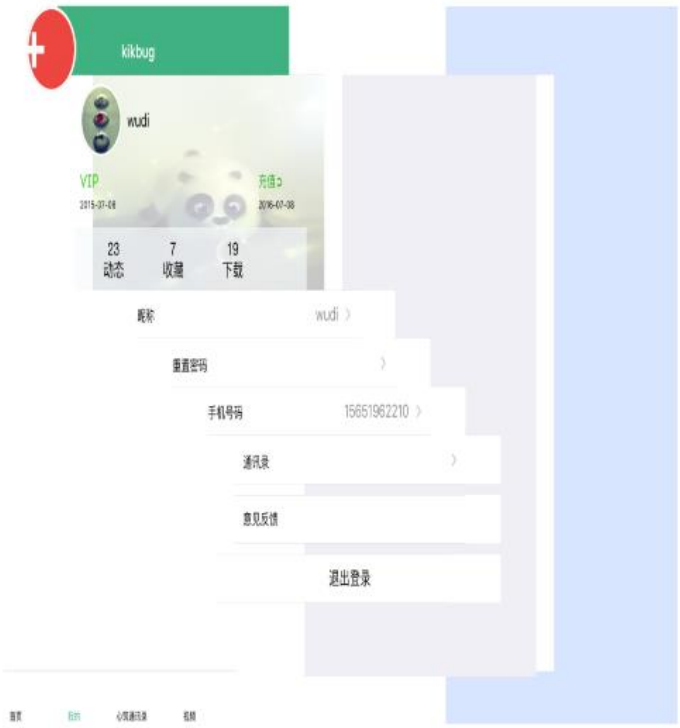
kikbug
wudi
VIP
23
动态
7
收藏
19
下载
昵称
wudi
重置密码
手机号码
15651962210
通讯录
意见反馈
退出登录

图 2.6 层次结构图

最上方的圆形 Button 就是 kikbug 项目生成的，其余部分都是原有项目。只有当悬浮按钮始终保持在最上方，它才能一直接受触屏事件。这就涉及到用户事件响应。

在 iOS 中，UITouch[8]代表触摸事件。所有的 UITouch 都封装在事件中，被特定的对象处理。UITouch 有 5 个属性：1.window 代表触碰产生时所在的窗口。2.view 触碰产生时所在的窗口。3.tapCount：短时间内触碰屏幕的次数，由此判定是单击还是双击。4.timestamp：记录触碰所产生或变化的时间。5.phase 触碰周期的各个状态。响应事件并对并非事件作出处理的对象叫做响应者对象，响应者链表示一系列响应者对象组成事件的传递链。当确定了第一响应者，事件就由第一响应者处理，否则传递给链中的下一个响应者。一般第一响应者是 UIView 或其子类对象，它不处理，就会将事件交给它的 UIViewController 处理，然后是它的 superView，直到顶层视图。顶层视图不处理就交给 UIWindow，再到 UIApplication。此项目中，原型 Button 绘制在 UIWindow 的最顶层，所在的 superView 是一个 UIView，UIView 不存在 UIViewController，直接将事件交有 UIWindow 处理，这样就能保证其他视图不会被 button 所在的 UIView 所覆盖，保证原项目能够正常接收触摸事件。

2.4 本章小结

kikbug 第三方库将收集到的数据存储到本地，然后上传到 UPYun，返回的 URL 通过 Schema 传递给 Kikbug 客户端。这就是整个第三方库所做的事，而其中的难点在于如何收集用户操作信息和合适的架构来尽量少的改变源码。本章节通过讲述 iOS 的文件目录结构，Runntime 的 AOP 机制与使用的 Bugsnag 收集 crash 来描述所用到的理论技术。通过描述委托和协议，UIVIew 触摸事件的传递来描述技术实现。

批注 [CF21]: 不明白啥叫第三方库？你的工作就写你的库

批注 [CF22]: upyun 介绍了？

第三章 项目系统需求分析与概要设计

3.1 Kikbug iOS 第三方类库项目的整体概述

传统的移动应用测试，都是公司内部的测试人员或开发人员兼职测试，耗费了大量的人力物力。而很多测试需求重复，单调，也不需要大量的专业知识。有些bug会在特定场景，特定型号的手机下出现，公司不一定能够充分覆盖所有测试环境，测试条件。因此，Mooctest提出移动应用的众测，将测试需求分配给软件专业的学生，不仅可以提高学生的测试能力，也方便了企业测试其应用程序。

国内关于数据统计的第三方插件，大多数都是为了服务企业统计访问量，日常用户访问量和用户群体分析，而与crash分析有关的第三方插件，往往应用于开发团队或开发者的集成测试。并且大多数涉及到收集应用内部用户行为的第三方插件都是收费的。

本项目主要有三个模块AppDelegate类别模块，数据收集模块和悬浮按钮模块。以下将对三个模块进行概述。

批注 [CF23]:

3.1.1 AppDelegate 类别模块

这一模块是整个项目的主模块，其他模块都与这个模块交互。它负责通知开始收集手机信息，开始监测用户行为以及悬浮按钮的添加和删除。众测人员通过 Kikbug iOS 端跳转到待测 App，然后这一模块通过方法监听到这一事件，整个项目开始启动。众测人员不直接与这一模块进行交互。当 crash 发生时，crash 报告会自动记录到本地，用户行为数据也记录于一个实例变量。当众测人员结束录制，此模块获取本地手机的信息，并接受其他模块传递的数据，将数据存储在本地后，根据悬浮按钮的状态将相应的文件上传至云存储。将云存储地址和其他少量信息随着应用跳转返回给 kikbug，kikbug 上传至 mooctest 服务器。

3.1.2 数据收集模块

这一模块的职责是收集应用程序活动期间产生的用户操作信息和crash信息，同时收集手机信息。这一模块与用户完全没有交互，仅为主模块服务，主模块在程序运行后启动这一模块，在结束测试后终止这一模块。通过改编的bugsnag代码库，此模块可以在监听到crash产生时将crash日志信息存储在本地，并将标记上时间戳的方法调用也存储在文件中。通过方法调配，此模块收集到用户跳转到某个页面，点击某个控件的信息。在结束测试后，应用跳转前，将程序活动时间，手机信息，app版本信息和用户操作信息传递给主模块。这一模块收集的具体信息有：

1）crash日志

2）程序开始运行时间和结束运行时间

3）用户操作信息：具体有UIViewController的跳转，UIViewController的类属性。UIControl的点击，具体类属性（UIButton，UITextfield...）,继承自哪一类控件，控件所在的位置和大小。

4）手机信息：手机设备名称，手机型号，手机放置方向，手机UUID，手机系统版本，电池状态，手机名称，手机分辨率，国际化区域名称。

5）应用信息：应用名称，应用版本。

6）网络信息：运行商，网络类型（流量还是无线）

3.1.3 悬浮按钮模块

这一模块的职责是提供给用户交互的视图。通过这一视图，用户知道自己进入了测试 app，也知道了返回 kikbug 的入口。这一模块将用户的思想传递给主模块，告诉主模块何时结束测试以及返回的数据。悬浮按钮模块与数据收集模块之间没有任何交互。

3.2 系统的需求分析

3.2.1 用例图及用例描述

为了更形象的阐述需求，我们把系统的功能以用例图和用例描述的方式展示。用例图由角色，系统边界和用例组成，它可以帮助我们更好的描述系统内外交互。系统的涉众有众测人员和研究人员两类。众测人员是借助系统明确测试目的和结束测试。研究人员则是研究系统收集到的数据，并通过数据模拟出 bug 出现的特定条件,自动化重现用户操作步骤。如图 3.1 所示：下图为项目的整体用例图。

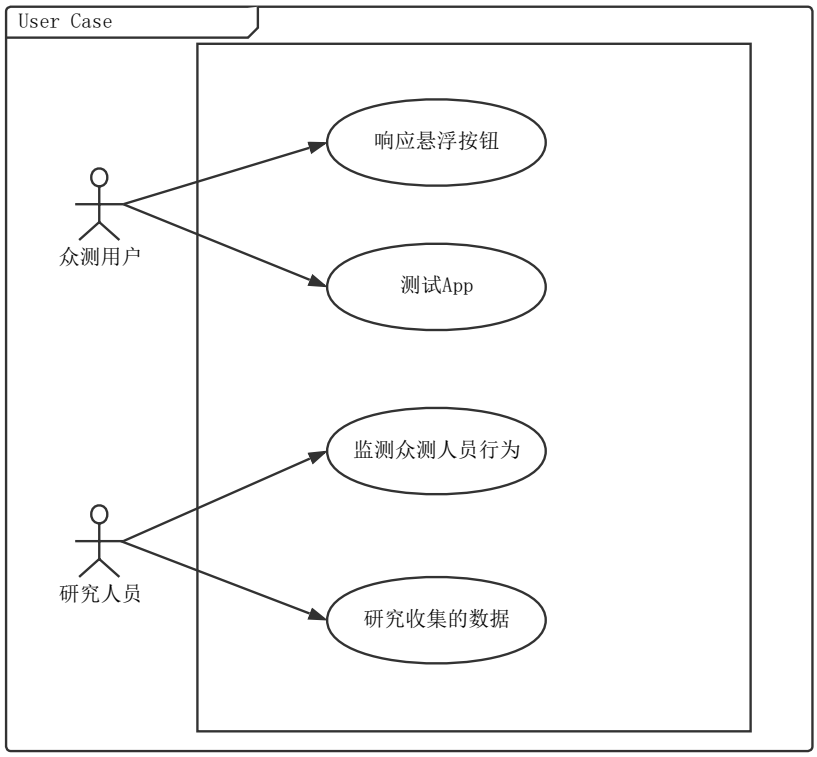

图 3.1 系统用例图

基于以上用例图，继续编写用例描述。

响应悬浮按钮的用例描述如下表 3.1。

用例名称	响应悬浮按钮
主要参与者	众测人员
触发条件	众测人员通过 kikbug 跳转到待测 App
优先级	高
前置条件	众测人员开始测试任务
后置条件	按钮展示相应的动画效果，跳转回 kikbug
流程	1.众测人员点击开始测试 2.悬浮按钮显示

表 3.1 响应悬浮按钮用例

测试 App 的用例描述如下表 3.2。

用例名称	测试 App
主要参与者	众测人员
触发条件	众测人员通过 kikbug 跳转到待测 App
优先级	高
前置条件	众测人员开始测试任务
后置条件	系统记录众测人员的测试行为
流程	1.众测人员点击开始测试 2.测试人员点击视图以完成测试任务

表 3.2 测试 App 用例

监测众测人员行为的用例如下表 3.3。

用例名称	监测众测人员行为
主要参与者	研究人员
触发条件	研究人员打开测试文件
优先级	高
前置条件	测试文件上传成功
后置条件	研究人员查看众测人员行为，判定是否完成测试任务
流程	1.研究人员从网站打开测试文件 2.分析用户测试行为

表 3.3 监测众测人员行为用例

研究测试数据的用例如下表 3.4。

用例名称	研究测试数据
主要参与者	研究人员
触发条件	研究人员打开信息文件
优先级	高
前置条件	测试文件上传成功
后置条件	研究人员分析手机信息与测试行为
流程	1.研究人员从网站打开数据文件 2.分析用户测试行为与手机信息 3.分析 bug 与用户行为及手机信息之间的关系。

表 3.4 研究测试数据用例

顺序图将交互表示成一个二维的图表，其中纵向表示时间轴，横向表示用户参与协作的交互。为了更好地梳理交互顺序，理清逻辑关系，保证系统的可用性，系统的顺序图如下图 3.2 所示：

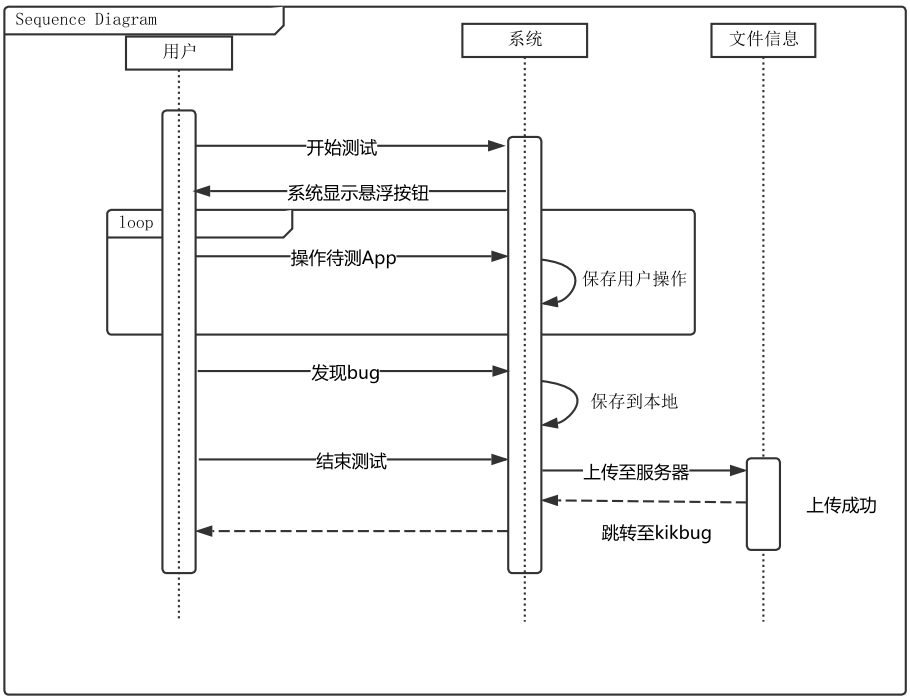

图 3.2 系统顺序图

3.2.3 系统活动图

顺序图可以用来描述常见对象的协作与交互行为，但遇到复杂情况时，顺序图有较大的局限性。活动图能够描述所有状态的流转关系，方便描述工作流程。系统的活动图如下图 3.3 所示：

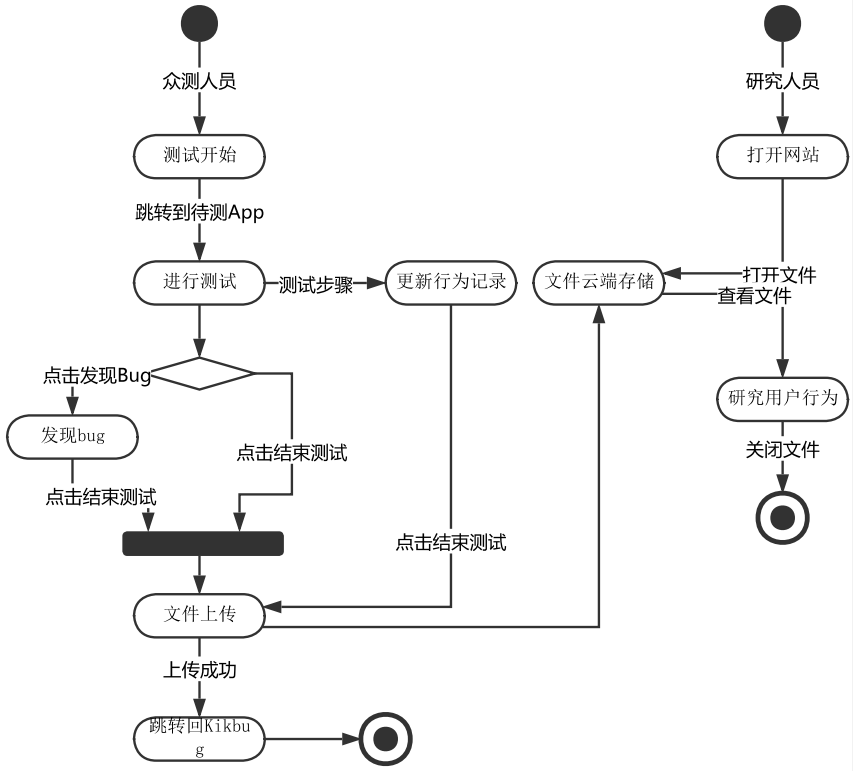

图 3.3 系统活动图

3.2.3 非功能需求描述

本小节描述了系统的非功能需求和约束条件。

系统的非功能需求如下表 3.5 所示：

名称	内容	非功能需求类别

数据完整	当网络连接中断，文件将依然完整，不会被删除或破坏。	可靠性
低出错率	系统具有良好的人机交互，不易出错	可用性
简单上手	用户能够第一时间使用系统的交互	易用性
文件及时删除	在文件上传成功后，本地文件将被删除，不占用手机空间	可用性
便于维护和扩展	系统保留了截图等功能接口，设计合理	可维护性
人性化	当系统处于无网环境，在多次提交失败后，系统会自动跳转至 Kikbug	可用性

表 3.5 系统非功能需求

系统的约束条件如下表 3.6 所示：

约束类别	内容
操作系统约束	在所有苹果手机上均可运行
网络环境约束	上传成功需要联网，不联网也可返回至 Kikbug
版本约束	仅支持 iOS7.0 及以上
法律法规	除实验统计，不泄漏用户任何信息

3.3 系统概要设计

3.3.1 接口描述

系统共有三个模块，其中主模块是 AppDelegate 分类模块。它提供给源程序一个启动接口，当用户跳转至待测 App，源程序通过此接口启动本系统。信息收集模块提供一个启动接口和一个数据获取接口给主模块。主模块启动后，启动接口被调用，信息收集开始。当系统结束前，主模块通过数据获取接口获取数据。悬浮按钮模块负责与用户交互，提供一个记录 bug 和一个结束测试接口。当测

试结束时，悬浮按钮模块通知系统需要获取的数据并上传。上传成功后，通过主模块再跳转会 Kikbug。

3.3.2 概念类图

概念类图是面向对象方法的核心技术，它描述了逻辑类之间的关系说明。下图 3.4 是整个系统的概念类图。左边是主模块。右上方是悬浮按钮模块，右下方是按钮收集模块。

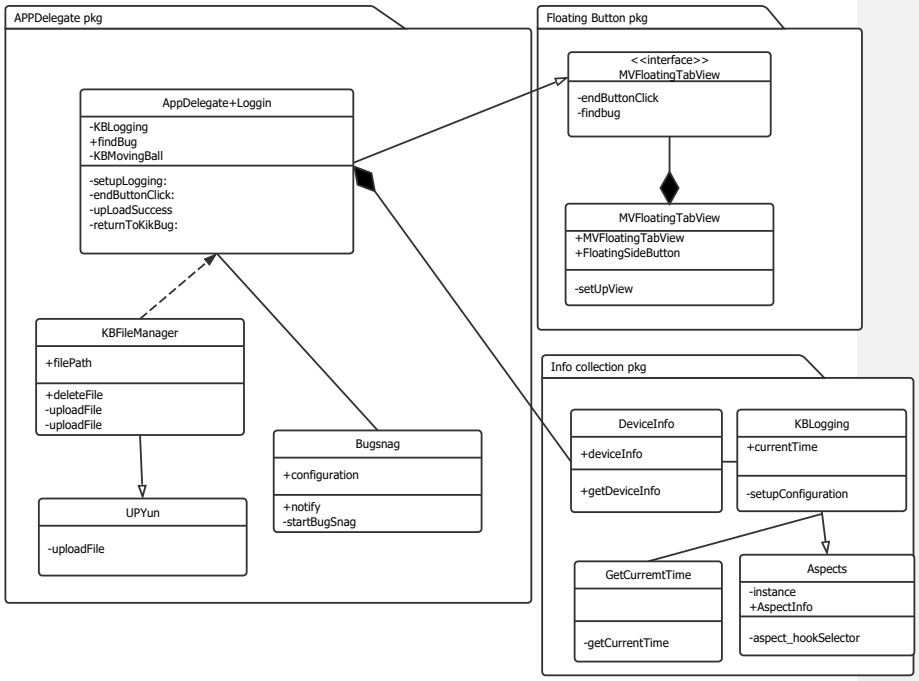

图 3.4 概念类图

3.4 本章小结

本章主要介绍了项目的整体概述，项目的需求分析和项目的概要设计，通过用例图，顺序图，活动图的概念类图详细介绍了项目的整体实现。也描述了项目的非功能需求。本章的内容视为下章更详细的介绍子模块做铺垫。

第四章 项目详细设计与实现

4.1 项目子模块的概述

在第三章概要设计的基础上，为了做更详细的细化和补充，添加了子模块详细设计。子模块详细设计描述了每个模块的布局和逻辑关系。本章主要使用详细设计类图来描述子模块。对于主模块，因其逻辑关系较为复杂，所以再添加顺序图介绍各个模块调用的先后顺序。

4.2 项目详细设计

4.2.1AppDelegate 分类模块详细设计

下图 4.1 是 AppDelegate 分类模块的详细设计类图。AppDelegate 是整个程序的入口，它可以获取到从 kikbug 传来的 taskId 数据，并通过这个 taskId 启动整个第三方项目。APPDelegate+Logging 通过设置 taskId 告知是哪一个任务，Bugsnag 也以这个 taskId 启动，并当发现 bug 时通知记录，BugsnagSink 将收集的数据以 json 数据格式保存到本地。当结束测试按钮被点击，KBFileManager 被调用，通过 UPYun[9]的 uploadFile 方法将本地文件上传，上传成功后删除本地文件。当上传文件成功后，KBFileManager 会通知 AppDelegate+Logging，使其通过 returnToKikBug 的方法返回到 kikbug，并传递参数。

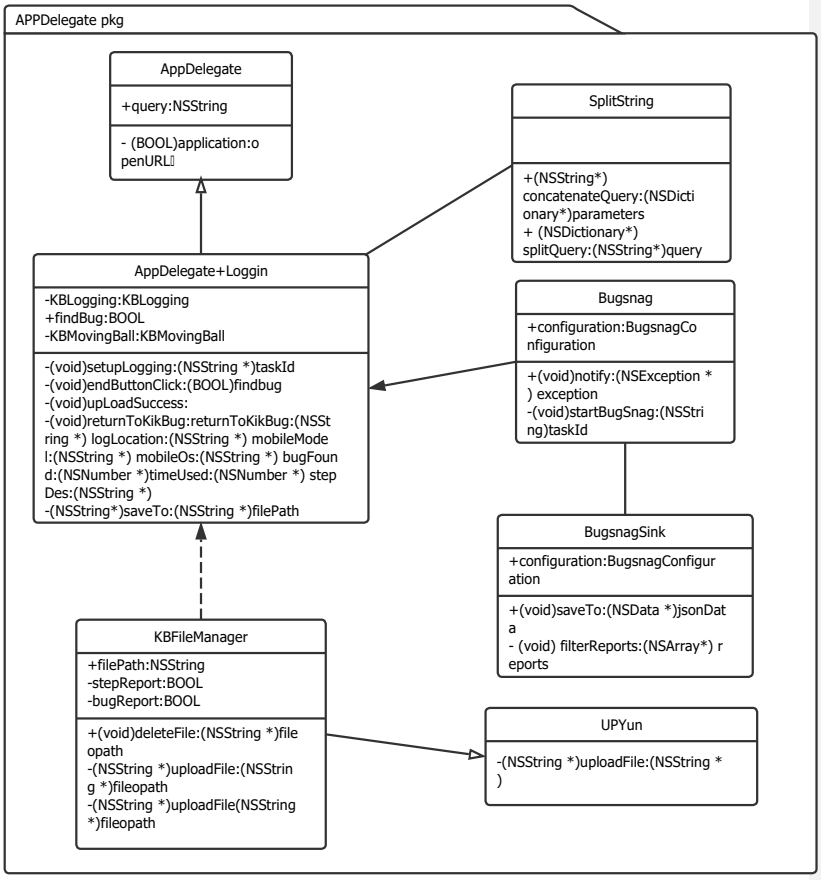

图 4.1 AppDelegate+Logging 设计类图

顺序图：

下图4.2是AppDelegate分类模块的顺序图。图中，用户点击开始测试通过openURL的方法跳转到项目的AppDelegate+Logging中，然后将URL交给SplitString解析，返回Dictionary。当发现bug，AppDelegate+Logging通知Bugsnag，bugsnag将文件保存到本地后返回filePath。结束测试后，分类将保存数据的Dictionary传递给SplitString解析，返回URL。将要上传的文件交给KBFileManager上传。在上传成功后，跳转回Kikbug，结束测试。

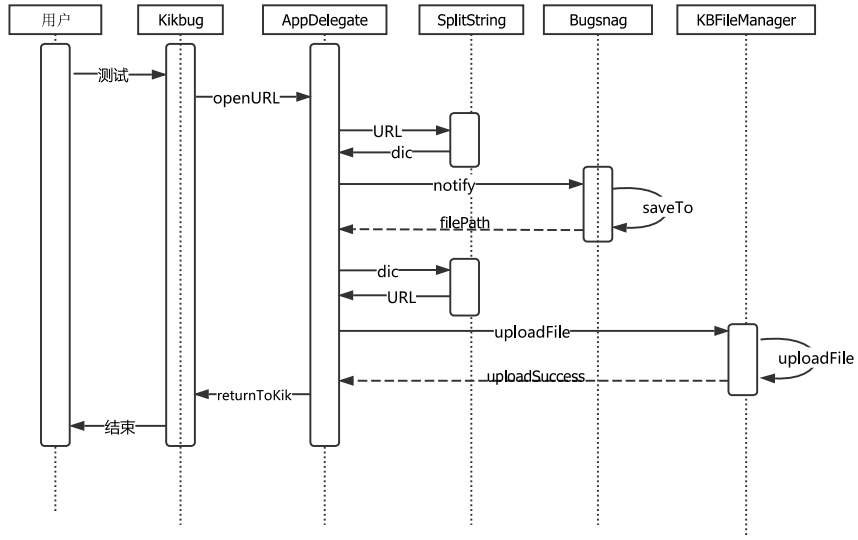

图 4.2 顺序图

4.2.2 信息收集模块详细设计

下图 4.3 是信息收集模块的设计类图。Aspects[10]类封装了方法调配，KBLogging 实现了 Aspects 中的方法。GetCurrentTime 负责提供时间戳，KBLogging 负责监听用户的操作，当监听到页面跳转和用户点击了控件就调用 Aspects 中的方法。通过 userLog 保存数据。DeviceInfo 负责收集手机信息，主模块通过 getDecviceInfo 方法获取已经收集完毕的数据。

批注 [CF24]: 下面的空调整下格式

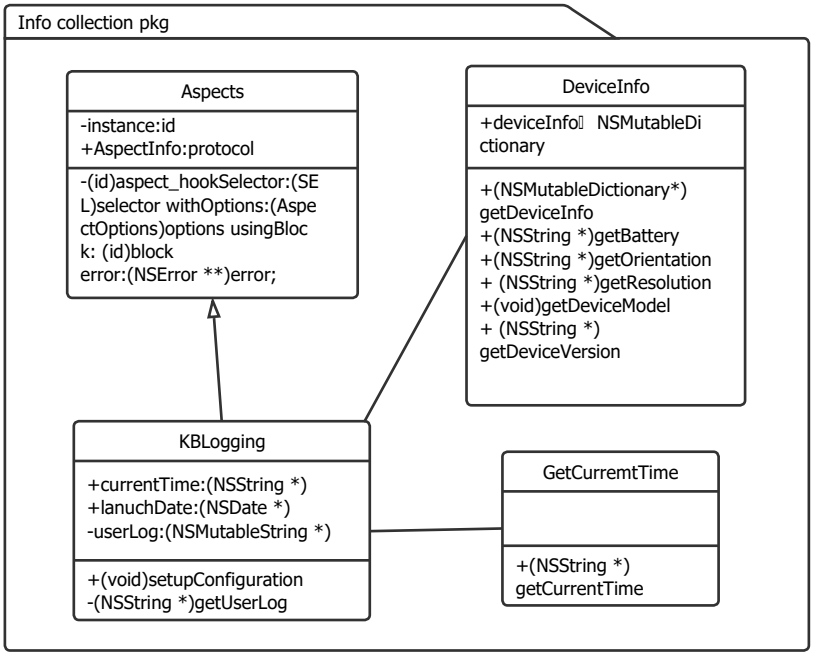

图 4.3 信息模块设计类图

4.2.3 悬浮按钮模块

MVFloatingTabView声明了一个endButtonClick的协议，当结束录制按钮被点击后，AppDelegate+Logging中的方法会被调用，以此监测点击事件并传递数据。MVFloatingTabView有两个button，分别为“发现Bug”和“结束测试”。CenterTab负责按钮的随手指跟踪，SideButton负责切换动画和实现button按钮的其他功能。他们一起组成了悬浮按钮模块1。下图4.4是悬浮按钮模块的设计类图。

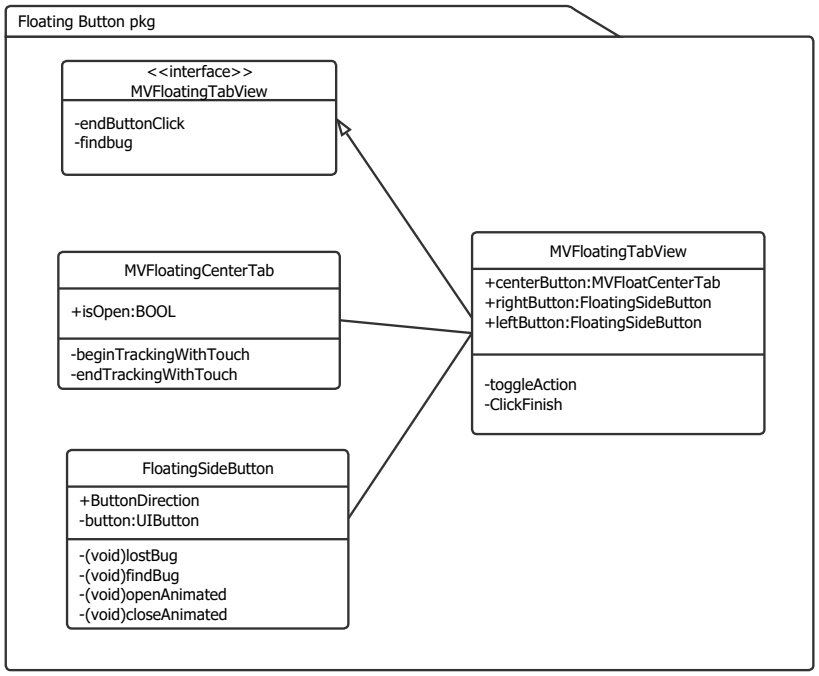

图 4.4 信息收集模块设计类图

4.3 项目的实现

4.3.1AppDelegate 分类模块的实现

主模块主要分为以下几个部分：App跳转，设置启动参数，文件上传还有Bugsnag部分。本小节将会结合源代码讲解其具体实现。

App跳转部分

本部分主要负责从Kikbug跳转到待测应用程序和待测应用程序跳转回Kikbug，并携带参数。下表4.1所展示的代码是当从Kikbug跳转到待测程序，从query中可以获取到taskId。

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    NSString *query = url.query;
    NSLog(@"QUERY FROM KIKBUG : %@",query);
    NSString* taskId = [[SplitString splitQuery:query] objectForKey:@"taskId"];
    [self setupLogging:taskId];
    return YES;
}
```

表 4.1 openURL 代码片段

下表 4.2 中代码负责跳转会 Kikbug，并携带 name，loglocation，mobileInfo，timeUsed 信息。两段代码的跳转都是通过[[UIApplication sharedApplication] openURL:appUrl]这一个方法实现。

```
-(void)returnToKikBug:(NSString *)name logLocation:(NSString *)logLocation
mobileModel:(NSString *)mobileModel mobileOs:(NSString *) mobileOs
bugFound:(NSNumber *)bugFound timeUsed:(NSNumber *)timeUsed stepDes:(NSString
*)stepDes
{
    NSMutableDictionary *allInfo = [[NSMutableDictionary alloc] init];
    [allInfo setObject:[Bugsnag configuration].apiKey forKey:@"taskId"];
    [allInfo setObject:name forKey:@"name"];
    [allInfo setObject:logLocation forKey:@"logLocation"];
    [allInfo setObject:@"none" forKey:@"reportLocation"];
    [allInfo setObject:@"苹果" forKey:@"mobileBrand"];
    [allInfo setObject:mobileModel forKey:@"mobileModel"];
    [allInfo setObject:mobileOs forKey:@"mobileOs"];
    [allInfo setObject:[NSString stringWithFormat:@"%@",timeUsed]
forKey:@"timeUsed"];
    [allInfo setObject:stepDes forKey:@"stepDescription"];
    NSString *url =   [SplitString concatenateQuery:allInfo];
    [self.mvFloat removeFromSuperview];
    NSURL *appUrl = [NSURL URLWithString:[NSString
stringWithFormat:@"KikBug://?%@",url]];
    [[UIApplication sharedApplication] openURL:appUrl];

}
```

表 4.2 returnToKikbug 代码片段

设置启动参数

下表4.3是启动时设置的代码，分别负责KBFileManager的初始化，启动bugsnag的监听。方法中保留了截屏通知的接口。KBLoggin负责用户行为的监听，并记录了App启动时的时间。setUpMVFloat初始化整个悬浮按钮，并将其整个UIView设置在UIWindow的最顶层。Image是可以自定义，其他界面都是固定。

```
-(void)setupLogging:(NSString *)taskId
{
    //设置 KBFileManager
    [KBFileManager setBugNO];
    [KBFileManager setStepNO];
    //启动 Bugsnag 监听
    [Bugsnag startBugsnagWithApiKey:taskId];
    [Bugsnag configuration].autoNotify = YES;
     //注册截屏通知
//          [[NSNotificationCenter defaultCenter] addObserver:self
//
selector:@selector(userDidTakeScreenshot:)
//
name:UIApplicationUserDidTakeScreenshotNotification object:nil];
    //设置启动用户行为监听
    self.kblogging = [[KBLogging alloc]init];
    [self.kblogging setupConfiguration];
    self.kblogging.lanuchDate = [NSDate date];
    //设置悬浮按钮
    [self setUpMVFloat];

}
```

表 4.3 setUpLogging 代码片段

批注 [CF25]: 代码只填关键的。
关键的代码都要加好注释

下表 4.4 展示了设置悬浮按钮的代码

```
    -(void)setUpMVFloat
{
    self.mvFloat = [MVFloatingTabView floatingTabView];
    self.mvFloat.openStateImage = [UIImage imageNamed:@"home.png"];
    self.mvFloat.closeStateImage = [UIImage imageNamed:@"plus.png"];
    self.mvFloat.delegate =self;
    [self.window addSubview:self.mvFloat];
    [self.window bringSubviewToFront:self.mvFloat];
}
```

表 4.4 悬浮按钮设置代码片段

文件上传

刚开始需要检验文件是否上传成功，为了避免重复上传。通过UPYunConfig配置上传的路径与用户名密码。文件上传成功或失败都有对话框提示，当一个上传成功后通过[self removeTempfile:filepath]删除上传文件，并通知AppDelegate执行校验方法，判断Log文件和Bug文件是否都上传成功。当上传失败5次后或两个文件都上传成功，系统通知执行AppDelegate的upLoadSuccess方法。下表4.5是文件上传部分代码片段。

```
+(NSString *)uploadFile:(NSString *)filepath{
        if(stepReport)     return @"loadsuccess";
    [UPYUNConfig sharedInstance].DEFAULT_BUCKET = @"kikbug-test";
    [UPYUNConfig sharedInstance].DEFAULT_PASSCODE = UPYUNKEY;
    __block UpYun* uy = [[UpYun alloc] init];
    uy.successBlocker = ^(NSURLResponse* response, id responseData) {
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"" message:@"上传信息
成功" delegate:nil cancelButtonTitle:@"确定" otherButtonTitles:nil];
        [alert show];
        [(AppDelegate *)[UIApplication sharedApplication].delegate upLoadSuccess] ;
        stepReport = YES;
        [self removeTempfile:filepath];
            NSLog(@"delete sucess");
    };
    uy.failBlocker = ^(NSError* error) {
            UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"上传信息失败"
message:@"请联网后重新上传" delegate:nil cancelButtonTitle:@"确定"
otherButtonTitles:nil];
        [alert show];
        reportNum++;
        if(reportNum>=5)
        {
            [(AppDelegate *)[UIApplication sharedApplication].delegate
upLoadSuccess] ;
        }    };
      NSString *key = [NSString
stringWithFormat:@"/%@/taskID%@/StepInfo.json",[self getDateString],[Bugsnag
configuration].apiKey];
    [uy uploadFile:filepath saveKey:key];
    return [KBFileManager fullImageUrlWithUrl:key];
}
```

表 4.5 文件上传代码片段

Bugsnag

此部分主要修改了配置文件，当crash发生或用户发现bug，就会调用此部分。下图代码主要负责将收集到的数据保存到本地，首先要检测本地是否存在KikBug_log和Crash_log目录，然后以taskId的文件名保存在此目录下。存储手机信息和用户行为的代码与下表4.6类似。下表4.6展示了Bugsnag文件存储代码。

```
-(void)saveTo:(NSData *)jsonData
{
    NSArray *directories=
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,
YES);
    NSString* cachePath = [directories objectAtIndex:0];
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *documentPath = [cachePath
                              stringByAppendingPathComponent:@"KikBug_log"];
    NSString *crashLogPath=[documentPath
stringByAppendingPathComponent:@“Crash_log"]；
    BOOL isDir = YES;
    BOOL isDirExist = [fileManager fileExistsAtPath:crashLogPath isDirectory:&isDir];
    if   (!isDirExist){
        NSLog(@"创建文件");
        BOOL bCreat =[fileManager createDirectoryAtPath:crashLogPath
withIntermediateDirectories:YES attributes:nil error:nil];
        if(!bCreat){
            NSLog(@"Create crashLogPath Directory Failed.");
        }
    }        NSLog(@"doc:%@",crashLogPath);
   NSString* filePath = [crashLogPath stringByAppendingPathComponent:
     [NSString stringWithFormat:@"crash%@.json",
     [Bugsnag configuration].apiKey]];
   NSLog(@"文件存储：%d", [jsonData writeToFile:filePath atomically:YES]) ;
}
```

表 4.6 Bugsnag 文件存储代码片段

4.3.2 信息收集模块的实现

此模块主要分为以下两个部分：KBLogging，DeviceInfo。

KBLogging 主要负责记录，DeviceInfo 负责静态收集。本小节将结合源代码讲解此模块具体实现。userLog 是记录整个用户行为的一个全局变量。aspect_hookSelector：方法封装了整个 method swizzling 需要的代码。@selector 的参数是需要调配的方法名，更改后的方法放在 usingBlock 中。usingBlock 中，首先是获取所捕获的方法的 sender 并获取实例，然后给方法加上时间戳存储到 userLog 中。第一个调配的是 viewDIdAppear，所有的 UIViewController 显示前都会调用 viewDidAppear 方法。第二个调配的 touch 事件中的(touchesEnded:withEvent：),UIController 触发事件也都会调用此方法。下表 4.7 展示了 KBLogging 设置初始化的代码。

```
- (void)setupConfiguration
{
    self.userLog = [[NSMutableString alloc]init];
    // Hook Page Impression
    [UIViewController aspect_hookSelector:@selector(viewDidAppear:)
                                withOptions:AspectPositionAfter
                                 usingBlock:^(id<AspectInfo> aspectInfo) {
                                    NSString *className =
NSStringFromClass([[aspectInfo instance] class]);
                                        [self.userLog appendString:[NSString
stringWithFormat:@"%@ userLog_ViewController:%@,ViewControllerInfo%@ ",
                                                                    [GetCurrentTime
getCurrentTime],className,aspectInfo.instance]];

                                 } error:NULL];
    [UIControl aspect_hookSelector:@selector(touchesEnded:withEvent:)
                                withOptions:AspectPositionAfter
                                 usingBlock:^(id<AspectInfo> aspectInfo) {

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT
, 0), ^{
                                            NSString *className =
NSStringFromClass([[aspectInfo instance] class]);
                                             [self.userLog appendString:[NSString
```

```
stringWithFormat:@"%@ ActionSender:%@ SenderInfo:%@ ",[GetCurrentTime
getCurrentTime],className,aspectInfo.instance]];
                                        });
                                    } error:NULL];
}
```

表 4.7 KBLogging 代码片段

DeviceInfo

通过[UIDevice currentDevice]可以获取手机的基本信息，iPhone显示型号不显示5，5s，6s显示的是iPhone 7,1所以需要建立型号对应表。通过platformString可以获取手机的型号。根据[[NSBundle mainBundle] infoDictionary]可以获取应用程序信息。其他分辨率，网络状态，电池信息也都需要通过UIDevice获取。通过封装，将枚举类型改变成人可读的字符串。下表4.8展示了手机各种手机信息的代码片段。

```
+(NSMutableDictionary *)getDeviceInfo
{
    NSMutableDictionary *deviceInfo = [[NSMutableDictionary alloc] init];
        //手机别名： 用户定义的名称
    NSString* userPhoneName = [[UIDevice currentDevice] name];
   [deviceInfo setObject:userPhoneName forKey:@"手机名称"];
    //设备名称
    NSString* deviceName = [[UIDevice currentDevice] systemName];
    [deviceInfo setObject:deviceName forKey:@"设备名称"];
    //手机系统版本
    NSString* phoneVersion = [[UIDevice currentDevice] systemVersion];
   [deviceInfo setObject:[NSString stringWithFormat:@"iOS %@",phoneVersion]
forKey:@"手机系统版本"];
   //手机型号
    NSString * phoneModel = [self platformString];
    [deviceInfo setObject:phoneModel forKey:@"手机型号"];
    // 当前应用名称
    NSString *appCurName = [infoDictionary objectForKey:@"CFBundleName"];
        [deviceInfo setObject:appCurName forKey:@"当前应用名称"];
    // 当前应用软件版本 比如：1.0.1
    NSString *appCurVersion = [infoDictionary
objectForKey:@"CFBundleShortVersionString"];
    [deviceInfo setObject:appCurVersion forKey:@"当前应用软件版本"];
    // 当前应用版本号码 int类型
    NSString *identifier = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
   [deviceInfo setObject:identifier forKey:@"设备的唯一标示符"];
    [deviceInfo setObject: [DeviceInfo getResolution] forKey:@"设备分辨率"];
```

```
        NSString *mConnectType = [[NSString alloc] initWithFormat:@"%@",info.currentRadioAccessTechnology];
    [deviceInfo setObject:mCarrier forKey:@"运行商"];
    [deviceInfo setObject:mConnectType forKey:@"当前网络类型"];
     [deviceInfo setObject:[DeviceInfo getOrientation ]forKey:@"手机放置方向"];
    [deviceInfo setObject:[DeviceInfo getBattery ]forKey:@"电池状态"];
    return deviceInfo;
}
```

表 4.8 手机信息手机代码片段

4.3.2 悬浮按钮模块的实现

此模块主要分为，MVFloatingCenterTab，FloatingSideButton和MVFloatingTabView。

MVFloatingCentTab

它负责跟踪用户的手指，使悬浮框能够随手指拖动。下表 4.9 展示了跟踪代码片段。

```
    -(BOOL)beginTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event{

    [UIView animateWithDuration:0.2 animations:^{
        self.transform = CGAffineTransformScale(CGAffineTransformIdentity, 0.9, 0.9);
    }];
    return YES;
}

-(void)endTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event{
    [UIView animateWithDuration:0.25 delay:0.0 usingSpringWithDamping:1 initialSpringVelocity:0.5 options:1 animations:^{
        self.transform = CGAffineTransformIdentity;
    } completion:^(BOOL finished) {
    }];
}
```

表 4.9 跟踪手指代码片段

FloatingSideButton

这段代码实现了左右两个button。下表4.10是发现bug时的处理：发现bug时将背景颜色变化为红色，否则回复为原色。

```
-(void)findBug
{
    self.button.backgroundColor = [UIColor redColor];
```

```
}
-(void)lostBug
{
    self.button.backgroundColor = [UIColor colorWithRed:108/255.0 green:197/255.0 blue:219/255.0 alpha:1];
}
```

表 4.10 处理 bug 代码片段

下表 4.11 代码实现了按钮开启时的动画效果，关闭时的代码与下图类似。

```
-(void)openAnimated:(BOOL)shouldAnimate{
    [self setViewToClose];
    self.hidden=NO;
    if(shouldAnimate){
        [UIView animateWithDuration:0.4 delay:0.0 usingSpringWithDamping:1 initialSpringVelocity:0.5 options:1 animations:^{
            _button.transform = CGAffineTransformIdentity;
        } completion:^(BOOL finished) {
        }];
    }else{
        _button.transform = CGAffineTransformIdentity;
    }
}
-(void)setViewToClose{
    _button.transform = CGAffineTransformTranslate(CGAffineTransformIdentity, _direction==kButtonDirectionRight?-_button.frame.size.width:self.frame.size.width, 0);
}
```

表 4.11 按钮开启与关闭代码片段

MVFloatingTabView

下表 4.12 代码执行了悬浮按钮的初始化，首先确定左右按钮的展开方向，然后设置边缘和动画效果。

```
-(void)setUpView{
    _leftButton.direction = kButtonDirectionLeft;
    _rightButton.direction = kButtonDirectionRight;
    _centerButton.layer.cornerRadius = _centerButton.frame.size.width*0.5;
    _centerButton.layer.borderColor = [UIColor whiteColor].CGColor;
    _centerButton.layer.borderWidth = 3.0;

    [_leftButton closeAnimated:NO];
    [_rightButton closeAnimated:NO];
    _isOpen = NO;
```

```
    _findBug = NO;
}
```

表 4.12 按钮初始化代码片段

下表 4.13 代码实现了“点击发现 bug”按钮时的界面变化，将具体的实现方法封装在其它函数中。

```
-(void)toggleLeftButton
{
    if(!_findBug){
        [self.leftButton findBug];
        _findBug =YES;
    }else{
        [self.leftButton lostBug];
        _findBug = NO;
    }
}
```

表 4.13 点击 bug 按钮代码片段

clickFinish方法与“结束测试”按钮绑定，此方法将事件通过委托传递给AppDelegate。当点击按钮后，首先判断AppDelegate是否实现了协议中的方法，如果实现了则调用AppDelegate中的方法。下表4.14展示了结束测试的代码片段。

```
- (IBAction)ClickFinish:(id)sender {
    if ([_delegate conformsToProtocol:@protocol(endButtonDelegate)]&&[_delegate
respondsToSelector:@selector(endButtonClick:)]){
            [_delegate endButtonClick:self.findBug];
        }}
```

表 4.14 结束测试代码片段

4.4 项目运行结果展示

本节展示项目运行时的截图，主要分为两个部分：1.众测用户操作时截图，2.研究人员搜集数据时截图。

4.4.1 众测人员使用截图

众测人员在接收任务后，会跳转到任务详情页，在该页可以点击开始测试。如下图 4.5 所示，跳转到待测 App 后，待测 App 会新增一个悬浮按钮。如下图 4.6 所示。

图 4.5 开始测试截图

图 4.6 待测 App 截图

用户点击开始测试后，系统会自动跳转到待测 App 之中。也只有当通过点击开始测试跳转到待测 App 时，会显示悬浮按钮。用户可以随意拖动悬浮按钮，以避免按钮遮挡住需要点击的组件。当用户发现 Bug 时，用户可以点击悬浮按钮，使其展开，然后点击发现 Bug。如下图 4.7 所示：点击 Bug 按钮被点击后，会显示红色以提示用户。

图 4.8 发现 bug 截图

用户再次点击按钮，可以使其回到折叠状态。当用户完成所有的测试需求，需要点击结束测试按钮，系统此时会上传文件。当文件上传成功后，会弹出上传信息成功提示。如下图 4.9 所示。

图 4.9 上传成功截图

由于没有联网或网络状态不佳等原因导致的上传失败，系统也会给出提示，如下图 4.10 所示。

图 4.10 上传失败截图

当文件上传成功后，或者上传失败 5 次后。系统会重新跳转到 Kikbug 客户端。

4.4.2 研究人员使用截图

用户填写报告之后，会在网页上生成报表。报告列表如下图 4.11 所示。通过报表，可以知道测试的详细数据。

图 4.11 报告列表截图

点击查看按钮，可以查看详细测试报告。详细报告如下图 4.12 所示。

图 4.12 详细报告截图

研究人员可以通过此处统计手机型号所对应的 bug 类型。并查看系统日志和详细报告。通过详细报告，可以得出用户的操作行为，从而复原用户产生 bug 的操作流程。报告内容截图如下图 4.13 所示。


```
{
  "当前应用名称" : "xinZhu",
  "运行商" : "中国联通",
  "设备名称" : "iPhone OS",
  "手机型号" : "iPhone7,2",
  "当前网络类型" : "CTRadioAccessTechnologyLTE",
  "设备分辨率" : "frame {{0, 0}, {375, 667}},宽750.000000,高1334.000000",
  "国际化区域名称" : "iPhone",
  "手机放置方向" : "竖直正立",
  "设备的唯一标示符" : "F977C4F6-8F64-4FBB-99B7-94DB9911311D",
  "用户操作信息" : "\nuserLog_ViewController:XZTabBarController\nuserLog_ViewController:XZNavigationViewController
\nuserLog_ViewController:XZMineViewController\nuserLog_ViewController:UIInputWindowController\nActionSender:MVFloatCenterTab SenderInfo:
<MVFloatCenterTab: 0x15c689d60; baseClass = UIControl; frame = (93 0; 80 80); autoresize = RM+BM; gestureRecognizers = <NSArray:
0x15c5d65b0>; animations = { transform=<CASpringAnimation: 0x15c6a4150>; }; layer = <CALayer: 0x15c68a160>>\nActionSender:UIButton
SenderInfo:<UIButton: 0x15c688c50; frame = (0 0; 111 54); opaque = NO; autoresize = RM+BM; layer = <CALayer:
0x15c688ef0>>\nActionSender:UIButton SenderInfo:<UIButton: 0x15c688c50; frame = (0 0; 111 54); opaque = NO; autoresize = RM+BM; layer =
<CALayer: 0x15c688ef0>>\nActionSender:UIButton SenderInfo:<UIButton: 0x15c688c50; frame = (0 0; 111 54); opaque = NO; autoresize = RM+BM;
layer = <CALayer: 0x15c688ef0>>\nActionSender:UIButton SenderInfo:<UIButton: 0x15c688c50; frame = (0 0; 111 54); opaque = NO; autoresize =
RM+BM; layer = <CALayer: 0x15c688ef0>>",
  "手机系统版本" : "9.3",
  "电池状态" : "正在充电   电量：100%",
  "当前应用软件版本" : "1.0",
  "手机名称" : "刘威廷的 iPhone"
}
```


图 4.13 报告内容截图

4.5 本章小结

本章通过介绍项目各模块的详细设计和项目运行截图，详细的阐述了系统的关键代码和设计细节。本章所展现的代码都是关键的代码，有利于理解重要功能的实现。通过项目截图的展现，能够更加直观，具象的表现出项目的运行。

第五章 总结与展望

5.1 总结

随着移动应用设备越来越普及，将来人均持有量必定会大于等于一，由此移动应用的开发需求也会越来越密集。软件测试作为整个软件工程的一个重要环节，移动应用测试的重要性也逐渐凸显。测试是一个耗时的步骤，往往一个测试需求需要多次测试，且不同环境下测试的结果也会有不同。为了缩减移动应用开发周期，从而将移动应用测试需求以众包得形式发布给广大众测用户，这些用户通常是学生。这不仅大大提高了企业开发应用程序的效率，也方便测试程序在不同手机环境下的可用性，更提高了学生的测试技能。Kikbug客户端引导众测用户接受测试任务，并收集用户测试数据，极大的方便了众测用户实现测试需求。

在完成毕设的几个月间，我们通过每周例会的形式，获取了项目的需求，讨论得出了与Mooctest的接口文档。并实现了项目的概要设计，详细设计，测试和部署。为了完成毕业设计，我阅读了有关书籍与博客，并询问了同学与企业界人员。这些准备都为我成功完成项目夯实了基础。

批注 [CF26]: 不要提幕测

在本次毕业设计中，我学习到了许多知识。例如iPhone私有框架的强大，然后无法通过App Store的审核。以json为格式的数据传输，通过 URL Schema[11]的App跳转，面向切面编程中的method swizzling的实现和iOS中UI的整体框架。

不过，项目中仍有一些需要改进的地方。项目收集的内容都比较有限，无法为研究数据提供更充分的信息。截图功能没有实现自动化，用户仍然需要手动选取截取图片。项目目前依靠KIkbug，无法独立完成整个测试任务。

整个项目仍有很多可以改进的地方，将来仍然可以以此项目为基础，实现更多自动化的功能。为了实现自动化测试，我们仍需努力。

5.2 展望

整个项目有许多地方可以改善。项目截图接口已经预留，在以后的版本中，系统将监听用户的截图事件。kikbug 能够自动呈现用户的截图供用户选择，就如微信的发送图片功能。这样用户能够节省许多选择 bug 截图的时间。

项目作为一个插件，并没有完全隐藏实现，用户能够看到源代码。在之后将把整个项目打成一个 framework 直接插入整个项目中。这样能够极大的提高插件的可移植性和可维护性。

项目的许多功能仍然需要改善，尤其是 crash 与 bug 监测这块。目前项目直接修改的 Bugsnag 和 KSCrash 的开源项目。在未来的版本中，将自己根据 NSException 等方面研究。用户监测这块的数据将会更详细，使得将来的自动化重现用户行为功能能够实现。

参考文献

批注 [CF27]: 知网，scholar 上搜几个相关的论文贴贴

[1] 沙梓社 吴航 刘瑾， iOS 应用逆向工程:分析与实战，北京：机械工业出版社，2014 年 3 月。P1~P10。
[2] 沙梓社 吴航 刘瑾， iOS 应用逆向工程:分析与实战，北京：机械工业出版社，2014 年 3 月。P13~P20。
[3] Bugsna-cocoa 开源社区，https://github.com/bugsnag/bugsnag-cocoa 。
[4] KSCrash 开源社区，https://github.com/kstenerud/KSCrash 。
[5] Matt Galloway,爱飞翔, Effective Objective-C 2.0 52 Specific Ways to Improve Your iOS and OS X Programs，北京：机械工业出版社，2014 年 1 月。P52~P55
[6] Matt Galloway,爱飞翔, Effective Objective-C 2.0 52 Specific Ways to Improve Your iOS and OS X Programs，北京：机械工业出版社，2014 年 1 月。P43~P49
[7] Aaron Hillegass / Mikey Ward， 王蕾 / 吴承耀，Objective-C 编程（第 2 版），武汉：华中科技大学出版社，2015 年 10 月。P161~P163。
[8] UITouch 触摸事件处理，http://justcoding.iteye.com/blog/1473287 。
[9] UPYun iOS SDK ，https://github.com/upyun/ios-sdk 。
[10] Aspects 开源社区，https://github.com/steipete/Aspects 。
[11] 自定义 URL Scheme 完全指南，http://objcio.com/blog/2014/05/21/the-complete-tutorial-on-ios-slash-iphone-custom-url-schemes/ 。

致谢

在论文完成的最后，我要感谢所有指导和帮助过我的同学，学长和老师。

首先，我要感谢我的指导老师陈振宇教授，在我本科的最后阶段给予了极大的帮助和指导。毕业阶段，我选择了陈老师的毕业项目，不仅是对项目有着较大的兴趣，更是因为陈老师大三大四课程中的认真，幽默使我愿意跟随陈老师。陈老师每周三召开例会，我可以听到不仅仅与自己项目有关的知识，更可以听到学长学姐的工作汇报。也正是因为参加了陈老师的毕业项目，我也有幸可以参加南京开发者软件大会。

其次，我要感谢房春荣学长的资料提供与技术支持。由于我对毕业项目的相关知识都了解的很少，所以房春荣学长的支持让我有了开发的初始方向，他的支持对我有启明灯的作用。我经常向学长询问关键问题的解决方案，学长也经常放下手中繁琐的工作向我解答。所以我要诚挚地向房春荣学长的无私奉献表达感谢！

同时，我也很感谢和我同做 Kikbug 项目的刘威廷同学和王天宇同学，他们在我遇到技术问题的时候经常给出修改意见，让我可以顺利完成整个毕业项目。他们丰富的开发经验让解决了许多技术上的细节问题。

最后，我非常感谢大学期间软件学院的培养。开始两年基础的夯实，与最后两年专业知识的培养与技能的训练，让我从一个大一新生成长为一个具有自主学习和开发技能的毕业生。也正是因为软件学院这个大环境，才能有一个积极学习讨论的氛围，我才能够认识优秀的学长和同学。