“**实时实地”软件**

**项目架构设计说明书**

**文档变更记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本 | 变更描述 | 主要负责人 | 完成日期 |
| 1.0 | 初始编写此文档 | 顾芳铭 | 2017.6.18 |
| 1.1 | 变更目录结构，优化部分内容 | 顾芳铭 | 2017.7.12 |
| 1.2 | 图片优化，内容优化 | 顾芳铭 | 2017.7.13 |
|  |  |  |  |
|  |  |  |  |

*小组成员：*

*钟玉生，顾芳铭，蒙科德*

目录

[1. 实时实地App架构简介 3](#_Toc487743983)

[1.1架构设计简述 3](#_Toc487743984)

[1.2架构设计目的 3](#_Toc487743985)

[1.3架构适用范围 4](#_Toc487743986)

[1.4架构设计概述 4](#_Toc487743987)

[2. 架构设计整体说明 4](#_Toc487743988)

[2.1简介 4](#_Toc487743989)

[2.2架构表示方式 4](#_Toc487743990)

[2.3质量属性与架构和设计 4](#_Toc487743991)

[3. 用例视图简介 5](#_Toc487743992)

[3.1核心用例 5](#_Toc487743993)

[3.2用例实现与质量属性 5](#_Toc487743994)

[4. 逻辑视图 6](#_Toc487743995)

[4.1逻辑视图 6](#_Toc487743996)

[4.2分层视图 8](#_Toc487743997)

[4.3架构模式 11](#_Toc487743998)

[4.4设计机制 13](#_Toc487743999)

[5. 进程视图 13](#_Toc487744000)

[6. 数据视图 14](#_Toc487744001)

[7. 部署视图 15](#_Toc487744002)

[8. 质量属性 16](#_Toc487744003)

[8.1 可用性分析及解决方案 16](#_Toc487744004)

[8.2.系统功能分析 17](#_Toc487744005)

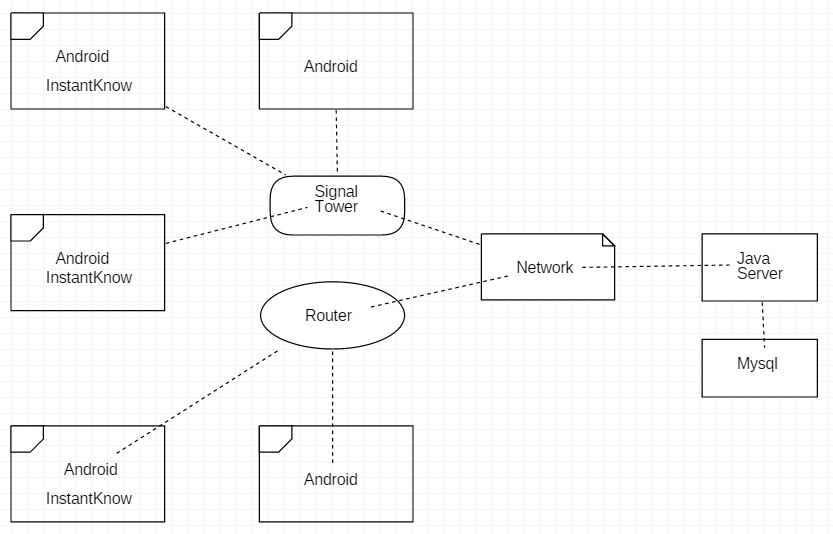
[8.3.可测试性分析 17](#_Toc487744006)

[9. 架构设计文档总结 18](#_Toc487744007)

# 1. 实时实地App架构简介

## 1.1架构设计简述

物理部署意图以简单的图形方式表达清楚实时实地这款APP在实际生产运行环境中的物理结构部署，以下为本项目的物理部署图:



**图1-1 物理部署设计图**

本架构设计文档从不同设计视图角度描述了架构设计的基本理念并围绕质量属性中可用性这一需求描述了架构设计的思路和出发点。

## 1.2架构设计目的

为了保障系统的可用性这一重要的质量属性，基于MVC架构模型给出本系统的架构模式。

架构设计的目的围绕保障系统的基本可用性，高可用性为出发点。系统可用性介绍：可用性是在某个考察时间，系统能够正常运行的概率或时间占有率期望值。考察时间为指定瞬间，则称瞬时可用性；考察时间为指定时段，则称时段可用性；考察时间为连续使用期间的任一时刻，则称固有可用性。它是衡量设备在投入使用后实际使用的效能，是设备或系统的可靠性、可维护性和维护支持性的综合特性。

## 1.3架构适用范围

本架构设计适用于本系统及其子系统的架构设计，其架构设计影响范围包含系统及其相应子系统本身及其架构模式等设计用例。

## 1.4架构设计概述

本章内容主要介绍了项目架构的物理部署图及相应的架构概念引入和架构模式综述。本架构文档主要包括：架构设计整体说明、用例视图简介、逻辑视图，分层视图，架构模式，设计机制，进程视图，数据视图部署视图和质量属性等。

# 2. 架构设计整体说明

## 2.1简介

本部分从整体出发，描述以下几个视图的相关概述内容：

用例视图：面向用例分析及质量属性简述

逻辑视图：面向系统类之间的关系和具体实现

部署视图：面向系统的部署及相关架构

## 2.2架构表示方式

架构主要以MVC架构模型为基础展开。其中必须包含的视图为：

用例视图、逻辑视图、部署视图、数据视图等。

构架目标和约束，主要针对系统质量属性需求中的可用性需求，

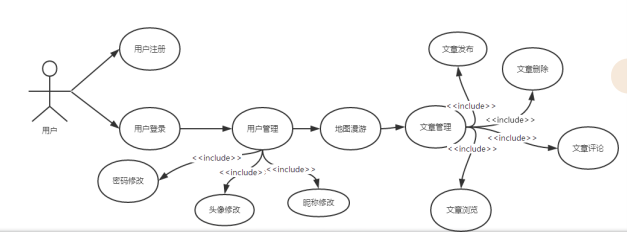
制定相应的计划和实施策略。

## 2.3质量属性与架构和设计

由于本系统设计和架构主要面向可用性质量属性，质量属性是软件质量属性划分为运行期质量属性和开发期质量属性两大类。开发期质量属性其实包含了和软件开发，维护和移植这三类活动相关的所有质量属性，这些是开发人员，开发管理人员和维护人员都非常关心 的，对最终用户而言，这些质量属性只是间接地促进用户需求的满足；而运行期 质量属性是软件系统在运行期间，最终用户可以直接感受到的一类属性，这些质量 属性直接影响着用户对软件产品的满意度。

# 3. 用例视图简介

## 3.1核心用例



主要用例：

普通用户：普通用户首先需要经过登录认证，然后可以使用查看新鲜事，编辑自己的新鲜事，查看自己发布过的历史记录，以及了解自己周围（以地理位置为准）发生的新鲜事等操作，每一个操作形成一个新的用例，这些不同的用例最终组成了系统的各项功能要求。

登录系统：用户登录的用例，执行用户身份审查，权限授予等。

内容管理系统：包括发布内容， 编辑内容，内容历史记录，查看内容，评论等各个用例，综合成为内容管理系统，此系统为实时实地APP的核心。

系统管理员：负责管理普通用户，已发布文章内容的管理，评论管理，系统运行维护等。

以上各个用例之间相互作用，共同构成了系统的功能集合，基于此对系统和APP提出了可用性的基本质量属性要求，并将在后文讨论如何基于当前APP架构实现此质量属性要求。

## 3.2用例实现与质量属性

考虑到系统主要面向可用性这一质量属性需求，故用例实现上主要向可用性作考虑。

1.方便操作，操作流程简单。

2.尽量从用户角度出发，以方便使用本产品：例如

a.用户在进行文章编写的时候提供分类标签让用户方便的将自己的文章归类。

b.在编辑栏提供编辑辅助工具方便用户自定义自己的文章。

3.支持第一次使用的用户能很快的熟练流畅的使用这款软件。

4.容错能力：软件具有一定的容错能力，在非硬件故障或非通讯故障时，软件能保证正常运行，并有足够的提示信息帮助用户有效正确的完成任务。

5.操作完成时有统一规范的提示信息。例如：删除文章时会弹出提示是否确认删除。

6.软件使用时就有足够多的可浏览的信息。

# 4. 逻辑视图

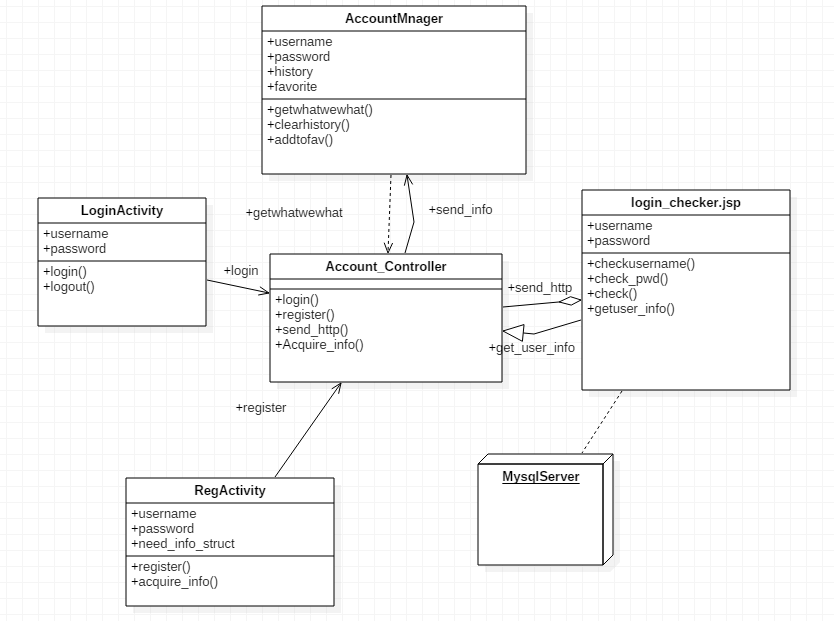
本章详细描述各架构关键性视图的设计及细节。

## 4.1逻辑视图

本节主要介绍系统的逻辑视图。通过面向对象的类图方式展现系统的逻辑架构部署和主要功能类之间的关联分析，来明确系统的功能性需求设计，以及通过何种方式满足系统的可用性，可测试性等重要质量属性。

逻辑视图（功能类图）一：

登录/注册类图



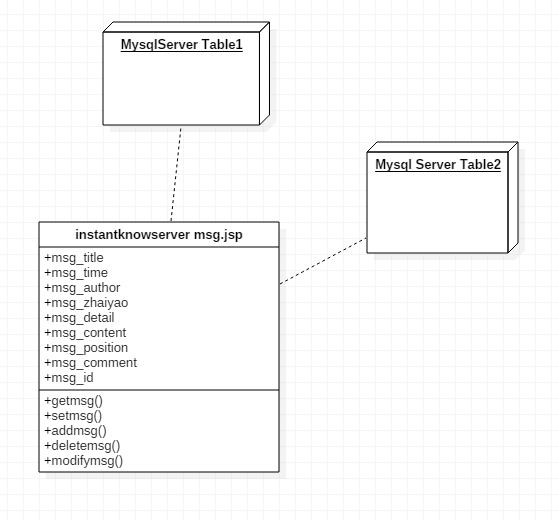
用户通过LoginActivity，RegActivity等类与Account\_Controller交互，Account\_Controller将信息传递到服务器的jsp页面，Javaweb与mysql后端交互后确认信息的正确性后返回前端。

AccountManager负责处理用户信息的保存等。

逻辑视图（功能类图）二:

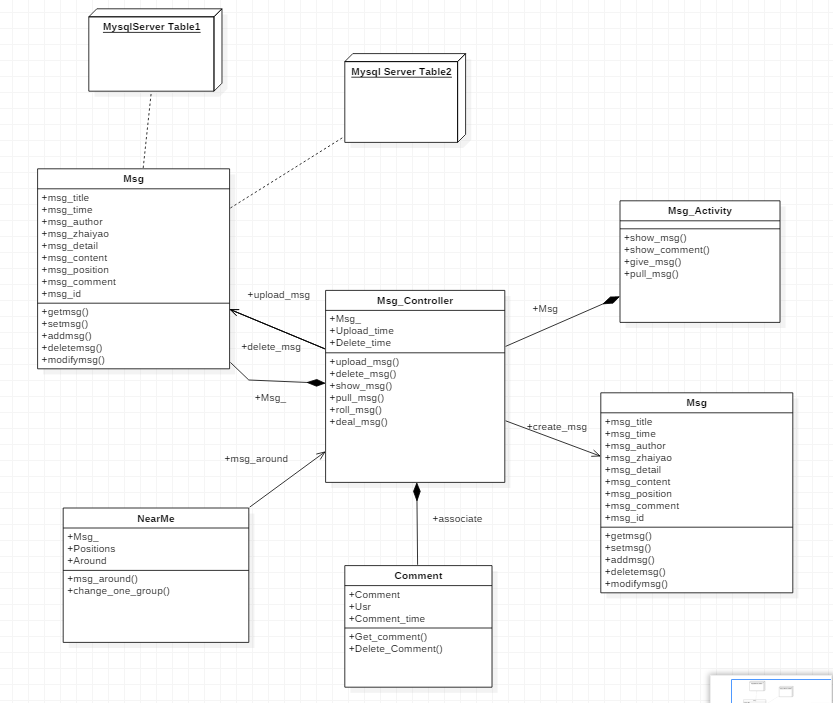
信息发布，收集，编辑，推送功能模块类图

1)服务器端类图示意：



部署在javaweb上的服务器的核心处理逻辑即上图所示的msg类（具体开发时名称可能不同），此类主要功能为处理APP的信息发布，收集，编辑，推送功能，并利用这些功能实现不同用户的需求，最终将处理好的数据放入不同的mysql server数据表中，或者在用户请求这些数据时将他们从表中取出。

APP与服务端交互类图：



服务器端与APP端的交互主要集中于Comment(评论)功能和Msg(新鲜事/消息)的编辑，发布，上传，删除，下载，展示等功能，因为目前阶段依然是主要从功能点出发，因而需要更多的辅助类来达成上述功能实现。

## 4.2分层视图

由于项目采用的是MVC-MVP的架构设计风格，故包设计和分类遵从MVC-MVP的架构设计模式，具体项目包结构如图4-2所示，

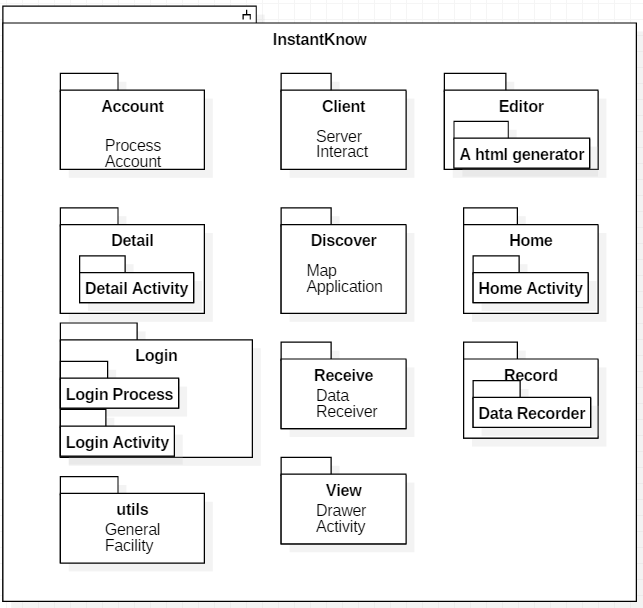
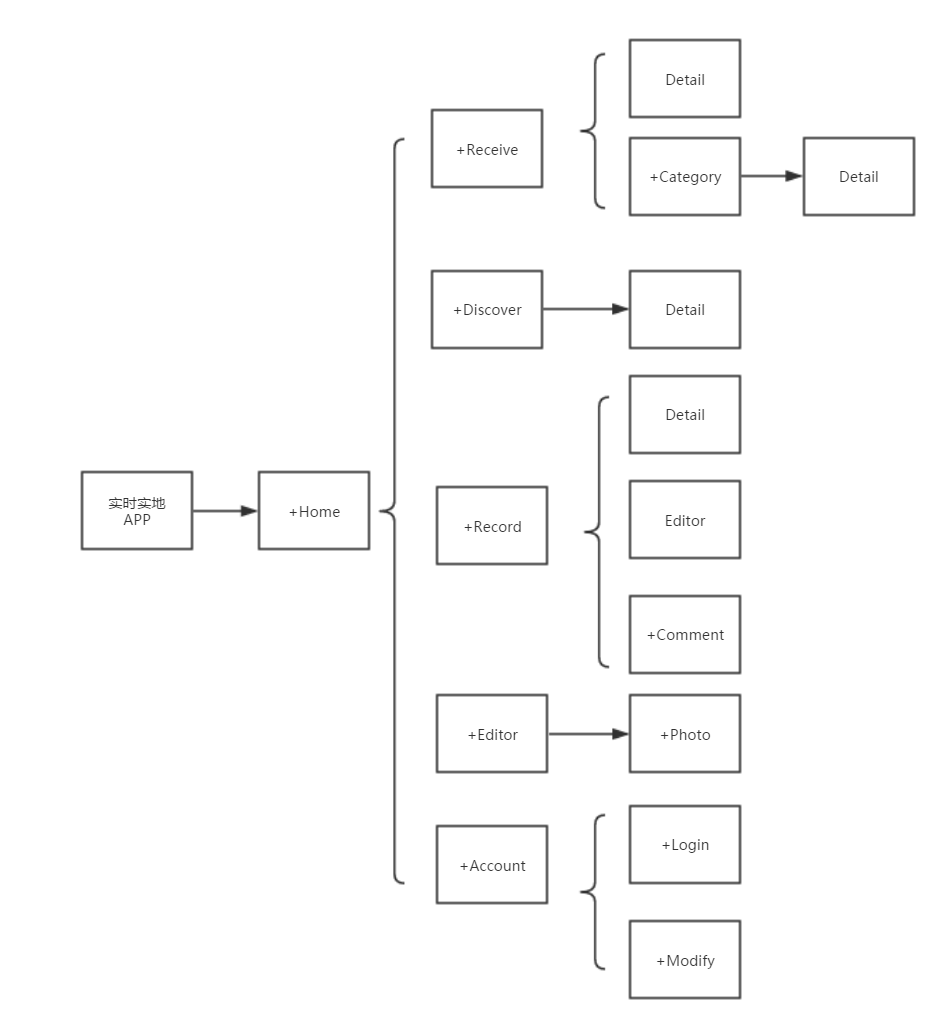
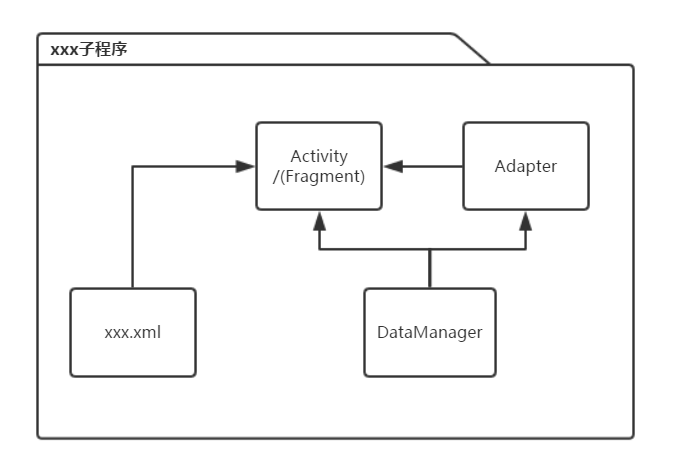


图4-2 实时实地App项目项目包结构

系统结构与子系统分层视图：



对于每一个子程序的内部结构，结构部署如下：



根据MVC-MVP架构设计模式，实时实地APP中大量应用了前端UI-数据适配Adapter-数据存储Model的设计模式，并且成功在一些包内设置了单例类，根据这样的设计模式，可以很轻松的实现不同Activity之间复杂的Context切换，从而节约了开发成本。

APP前端显示主要使用Activity结合Fragment来实现，合理的包设计使得各包和各模块之间的耦合度非常低，而同时拥有非常高的内聚度，成功秉持了高内聚，低耦合的设计原则。

以Account包为例，其中的AccountFragment功能设计简洁，目的即保存用户的个人信息（登录后的）并提供存取和修改查找等的接口，而作为Login模块的数据来源，Account包可以与LoginHandler相配合，从而实现数据与逻辑处理的分离，实现高效而安全的设计理念。

同时，utils包中包含了各种通用工具，比如统一管理本地缓存的DiskLruCache类，实现了一个简单的LRU算法，非常高效的管理本地缓存机制；以及FileUtils类包含了各种处理文件读写的方法实现；NetworkUtil则以相对通用的方式预先定义了几个以HTTP协议访问互联网上服务器的方法，这对于软件必须的网络传输而言是非常有必要的技术。

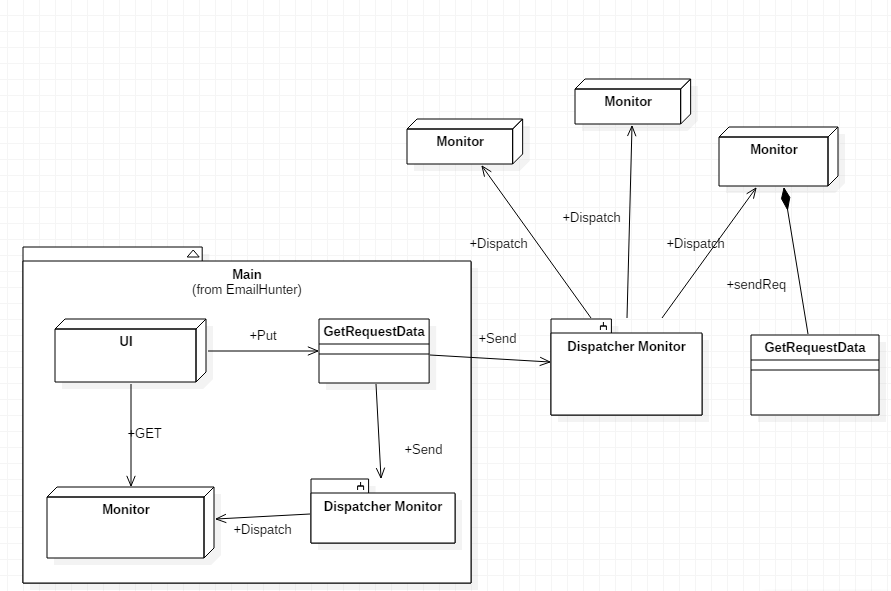
Editor包中包含了非常重要的部分：一个适应性非常好的用户用编辑器，用户可以利用此编辑器对自己即将发布的文章进行编辑，编辑过的文章将直接被生成为HTML文档，经Android端CSS优化后，可直接被实时实地APP内嵌的webview组件直接读取并显示给用户，方便和快捷度上都有一定的提示。

Login包则包含两个子模块于其中， 一个是用来处理登录，与后台进行交互的LoginProcess类，另一个则是负责和用户交互的LoginActivity，之所以这样设计，是因为这不仅对用户是友好的，也实现了数据和功能相分离，UI和后台逻辑相分离的设计原则，是值得采用的一种设计原则。

## 4.3架构模式

主要利用了MVC架构设计模式，此模式已在前述逻辑视图、分层视图中得到了体现，并通过APP的数据管理和显示得到进一步显示。

部署爬虫（APP初始化数据和维持数据获取部分）



程序目标是使用一只分布式的，可循环不停爬取信息的搜索爬虫，另外借助附带的信息过滤功能，快速从整个互联网上获取按照时间新鲜度和火热程度排序的新鲜事物，并通过JavaWeb部署的相应接口，直接将获取到的数据脱敏和规范化后推送到用户的APP端，作为初始化数据来吸引用户并且在系统缺乏热度时采用这种办法来提高用户阅读的兴趣。这一手段将成为程序架构设计中如何保证系统持续可用性的重要一环。

爬虫基本的工作模式如上图所示，虽然爬虫并不是系统功能的主要实现者，但是作为新鲜数据源的获取者，的确能够对系统的上线和吸引用户起到非常关键的作用，因而在此处赘述WebSpam系统的工作模式以及我们是如何打算将WebSpam使用在我们的系统中以确保系统中总是有足够的新鲜信息。

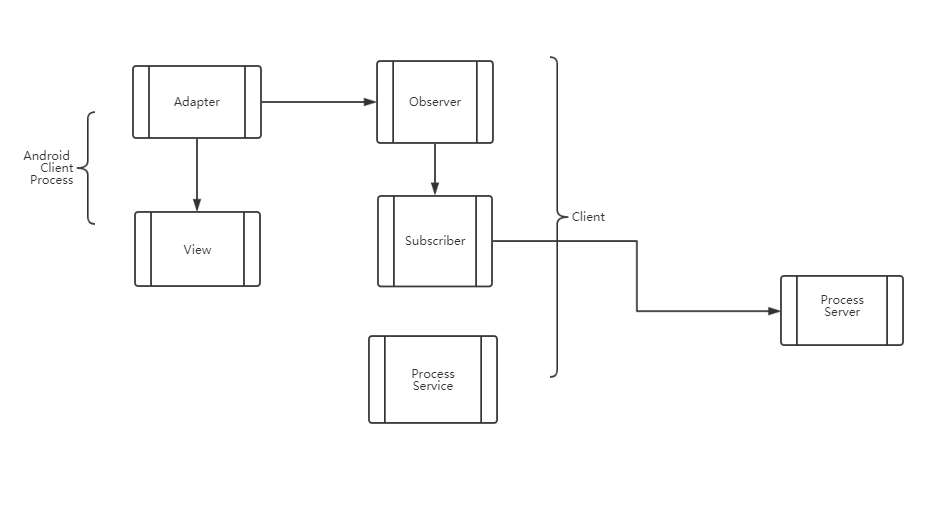
## 4.4设计机制

子系统与架构对应关系

|  |  |  |
| --- | --- | --- |
| 层次 | 类、模块 | 功能职责 |
| View | Xxx.xml | 定义用户交互界面 |
| Model | Datamanager | 处理用户输入数据，维护应用自身数据 |
| Controller | Activity、Fragment、adapter | 负责界面更新处理：包括控件显示，数据展示，用户交互监听 |

# 5. 进程视图

如下的进程视图进一步阐明了AndroidClient端和Server端的通信结构以及如何做到异步加载请求内容的，通过Subscribe-Observe的方式拉取消息内容，推送给APP.



# 6. 数据视图

数据模型及实体-关系图：



# 7. 部署视图

带WebSpam的系统结构部署图如下图所示：

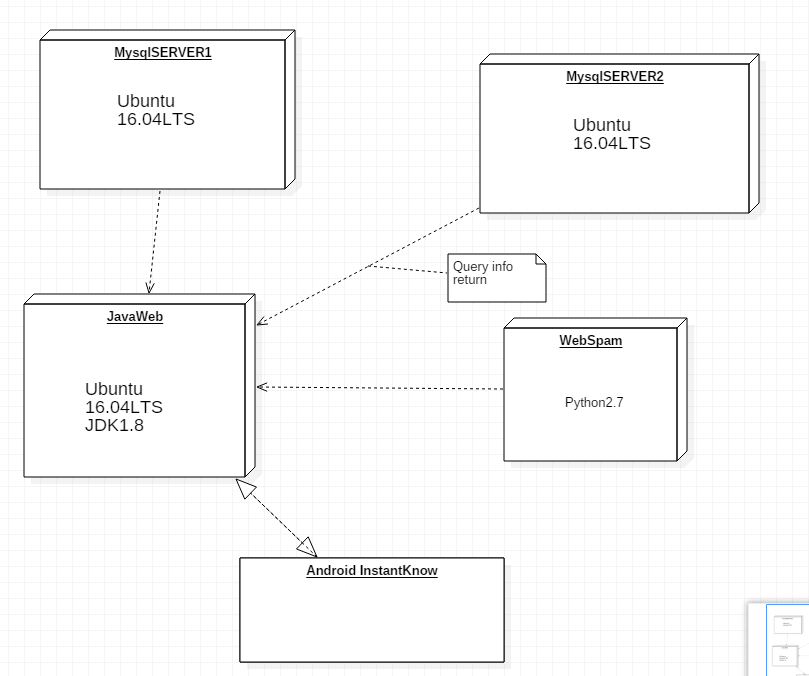


图3-3.实时实地APP系统结构部署图

各部分相互作用，相互协调，共同维持系统的可用性，其中的两个核心为部署在服务器端的JavaWeb和部署在用户手机端的Android InstantKnow App.

同时，利用Mysql和WebSpam实现了数据的自动存取，提升了系统处理和完善数据的效率。

概括了项目APP部署上线后的物理环境，本节将介绍项目APP的包结构设计。

# 8. 质量属性

## 8.1 可用性分析及解决方案

系统运行流程设计（此视图是针对系统可用性需求给出的特殊视图）

可用性是一个系统能够在一个时间或一个瞬间考察其能够正常提供服务的概率或该时间占总时间的期望值。为了提升本项目APP的可用性，必须从服务器端和用户端两个方面深入考虑，由此首先提出考虑系统可用性需求的特殊流程设计图（图1-3所示）：

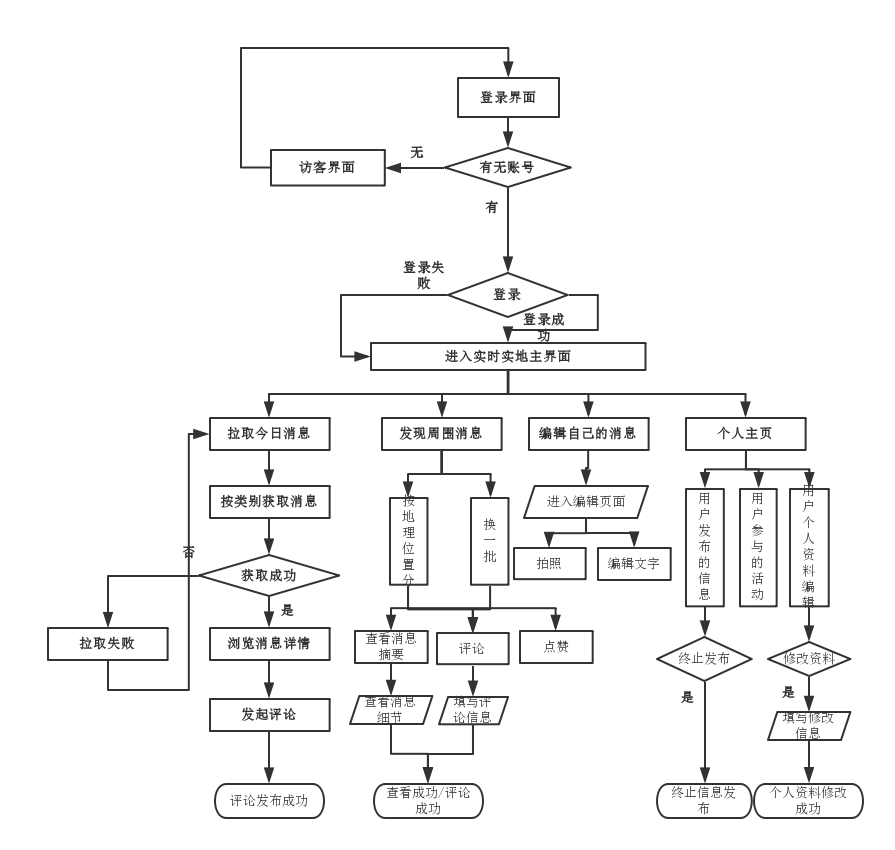


图1-3.系统运行流程设计

由上图我们可以分析得出本系统运行的主成功逻辑可以分为4条，每条逻辑最终的执行结果的成功与否都直接对系统的可用性产生影响，故要使本项目APP实现高可用性这一质量属性要求，则必须要求上述4条执行逻辑都不能出现失败的情形，具体而言，意味着用户的操作不能够出现单点失败，或者系统出现未预期的影响用户体验的情形。

## 8.2.系统功能分析

上述内容已经使得系统功能处于一个较为完善的阶段，此处仍然重点阐述一下WebSpam部分。

通过WebSpam，我们能够在海量的信息中搜集到自己需要的信息，我们需要的信息主要分为两类：

具有时效性的消息，类似新闻，但是需要比新闻更多一点趣味性，少一点时效性。

时间延长不太大的消息，这要求消息的发布时间不能过晚，应该在限定之内尽可能是刚刚发布的消息最为合适。

上述两类消息我们都可以通过一只分布式的WebSpam来实现，组员有实现网络爬虫和基本数据挖掘、分析的经验，因而可以通过一只分布式爬虫获取到相关信息，并且将信息进行一定的预处理，删除无关和无价值的信息后保存到服务器上，然后寻找合适的时机将之推送给用户即可。

## 8.3.可测试性分析

软件的可测试性是指软件发现故障并隔离、定位其故障的能力特性，以及在一定的时间和成本前提下，进行测试设计、测试执行的能力。James Bach 这样描述可测试性：软件可测试性就是一个计算机程序能够被测试的容易程度。

本项目系统使用了模块化非常高的设计，做到了高内聚，低耦合，因而降低了测试分析的难度，基于源码，可以做到:  
白盒测试

符号执行测试

动态逻辑测试

另外我们还打算对系统执行：

黑盒测试

模糊测试

服务器压力测试

APP数据安全性测试

# 9. 架构设计文档总结

本架构设计文档从实际出发，结合项目工程中遇到的实际情况，总结并研究了以上几个方面的内容，并通过概要设计，详细设计，和系统可用性这一质量属性等重点研究了如何较好地实现本系统的架构模式。