软件设计文档

易闲圈---挣闲钱的 APP

目录

一、 绪论

1.1研究的背景和意义

互联网技术自从诞生（1969 年）以来已经走过了快 50 个年头，他是人类发展史上的重大变革，是每个人的免费而又无尽的巨大图书馆，是每个人通向世界的一扇窗，为人类社会带来的便利更是无法用数字来衡量。

2008 年以来，智能手机热潮席卷全球。随着移动操作系统的更新迭代，移动应用也像雨后春笋一样：娱乐、音乐视频、生活、社交等等应用应接不暇，渗透到我们生活中的方方面面，给我们的生活，工作和学习带来了无尽的便利。

社交应用，作为移动应用的中坚力量，几乎每个人的手机上都有至少一两款，每个人每天都会不厌其烦的打开一次又一次。各大平台都有很多社交应用，微博，微信，淘宝，支付宝等 APP 更是其中的杰出代表。他的迅速，便捷，廉价等等特点使得这些应用深受人们的喜爱，甚至大有取代主流媒体的驱势。

而且，对于每个大学生来说，如果有一款 APP 能够了解周围同学的一些需要帮助的动态，那我们就可以通过这款 APP 来和他们取得联系并帮助他们解决。这样就一举两得了，作为希望得到帮助的人来说，自己花了小钱完成了自己的任务，作为提供帮助的人来说，用自己闲暇的时间去挣了一些零花钱，何乐而不为呢。

1.2项目简介

挣闲钱是大学生通过做任务挣钱的一款 APP，它属于以运营为中心的服务软件，也可以理解为面向大学生的专业“众包系统”。系统非常简单：有一个云服务中心，其业务在不断完善中；每个大学生都装有“挣闲钱”客户端；一些机构，当然也可以是大学生，称为“奶牛”，他们提供任务给平台。基本业务是。奶牛发布任务要求与薪酬，系统推送到客户端，学生完成任务可获得系统内部的“闲钱币”，闲钱币可用于发布任务或提现（当然，提现这个没有做）。系统不支持零元交易。

二、 系统需求分析

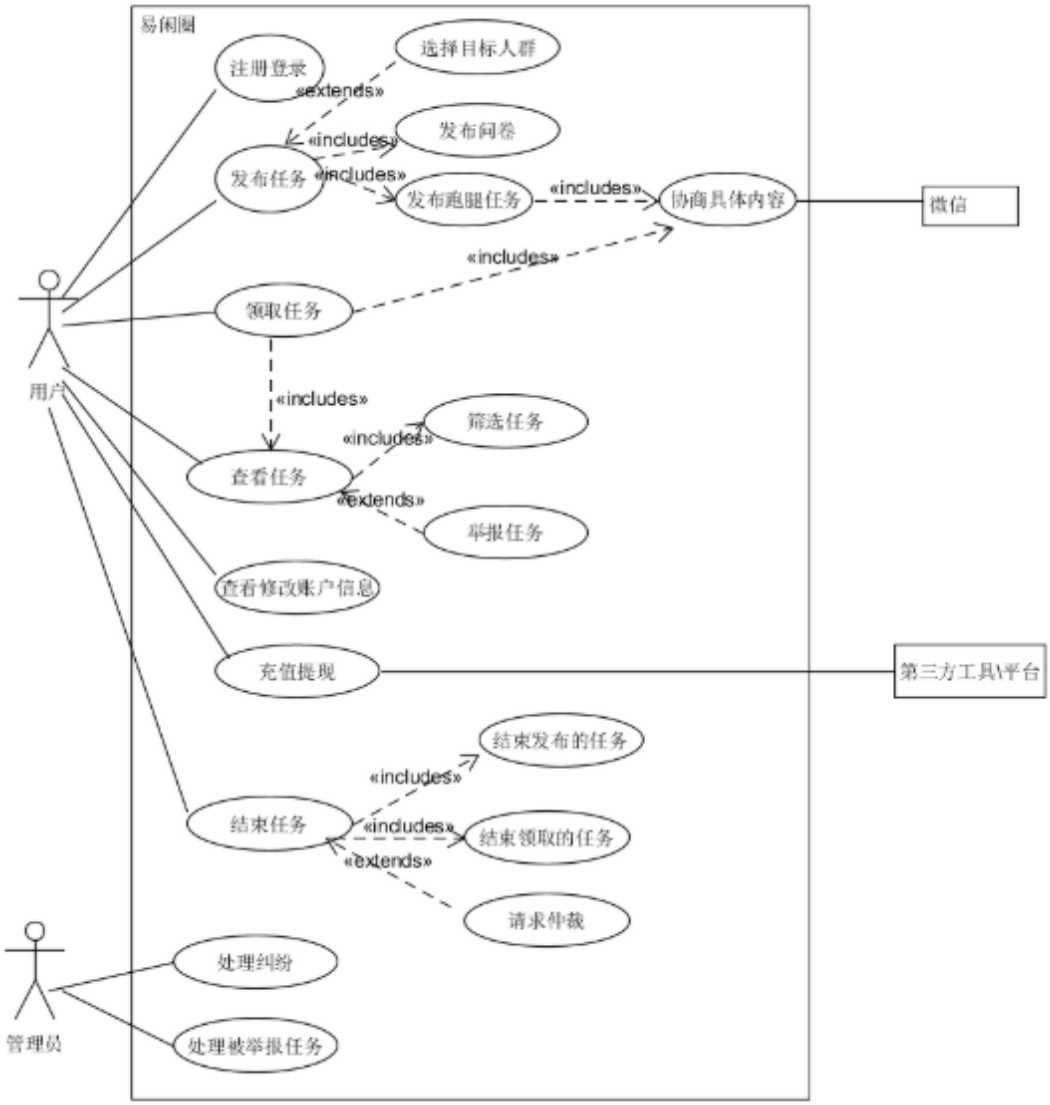
2.1 需求分析的背景

在完成了针对图片大学生闲时时间活动的前期调查，同时与多位大学生进行了全面深入地探讨和分析的基础上，提出了这份软件需求规格说明书。

此需求规格说明书对易闲圈 APP 系统软件做了全面细致的用户需求分析，明确所要开发的 APP 软件应具有的功能、性能与界面，使系统分析人员及软件开发人员能清楚地了解用户的需求，并在此基础上进一步提出概要设计说明书和完成后续设计与开发工作。本说明书的预期读者为用户、业务或需求分析人员、测试人员、项目管理人员。

2.2 需求分析的任务概述

系统用例图如下：



2.2.1 目标

- 开发目标：人们交流的方式多种多样，但是，我们在交流的同时希望更多的了解对方的信息，本软件即是根据此需求进行开发的。
- 应用目标：让用户能够通过注册信息，登录并更新及查询自己的信息，在此基础上，用户还能够有效的掌握和共享其它资源，但是不能更新。从而促进了信息管理的规范化和集成化，使得用户之间的交流更加的便捷。

2.2.2 用户的特点

本 APP 软件产品的最终用户来源主要是在校大学生，相同之处则是为了在闲时挣闲钱来充实自己，管理人员则可以随时更新软件的项目，以及查询和维护信息。本软件面向大学生，用户使用过程中的操作也不复杂。

2.2.3 假定和约束

本 APP 软件产品为大学生用户使用，然而，由于技术原因，本软件在功能上还不够完善，例如：无法提现，举报任务的功能还存在瑕疵，因此对于用户的需求还无法完全实现。

2.3 系统对主要功能的规定

2.3.1 系统功能模块

- 用户注册、登录模块
- 用户修改账户信息模块
- 发布任务者发布任务模块
- 发布任务者取消任务模块
- 发布任务者确认完成任务模块
- 用户查看任务模块
- 领取任务者领取任务模块
- 领取任务者确认完成任务模块
- 用户举报任务模块

2.3.2 功能描述

下面详细描述一下各个功能模块：

1）用户注册、登录模块

- 用户注册：仅限于用户第一次使用该 APP 软件的用户。在注册过程中，即将注册的用户必须需要根据要求填写用户名、密码、确认密码、注册邮箱等，在填写过程中系统会有对应的提示，有的必填项会明确显示，对于其他比如说性别、出生年月等信息为可填，注册成功后将进入个人信息模块。

- 用户登录：仅限于已注册用户和管理人员进行操作。该模块主要是用于用户登录，用户和管理人员输入用户名和正确的密码即可进入对应的界面。该界面的修改账户信息控件可转到修改账户信息功能。

2）用户修改账户信息模块

该模块仅限于已在该 APP 上注册过且登录成功的用户，用户通过成功登录自己的账户，然后对自己的账户信息进行修改，修改完之后就进行提交，后台然后对其进行判断，返回修改结果。

3）发布任务者发布任务模块

因为我们的 APP 没有实现充值和体现功能，所以，用户在注册时机会有一个初始账户余额值，然后用户就可以根据自己的需要进行发布任务和设置完成任务的佣金。

4）发布任务者取消任务模块

发布任务者可在任务截止时间之前取消任务，若取消任务时该任务还没有被人领取，则发布者成功取消任务；若任务已被领取，则可申请系统仲裁。

5）发布任务者确认完成任务模块

发布任务的人可以在一段时间后确认任务完成，此时领取该任务并顺利完成任务的人会自动收到任务酬劳；若发布任务者没有确认任务完成，一段时间后系统将自动判定该任务已完成。

6）用户查看任务模块

用户登录自己的账号，进入查看任务页面，用户可查看平台上已发布的任务，可对任务进行筛选;点击任务可查看任务详情。

7）领取任务者领取任务模块

用户通过先前的查看任务，选择自己心仪的任务进行领取。

8）领取任务者确认完成任务模块

领取任务的人在完成任务之后可以登录系统确认任务完成；若领取任务者在任务 Deadline 之前顺利完成任务，则系统将任务酬劳发送到该完成者账户，若任务完成时间超过截止时间，则系统判定任务完成失败，领取任务者将不会获得酬劳。

9）用户举报任务模块

用户通过查看任务，发现有明显的的欺骗行为或者用户完成了任务却没有收到佣金，然后向系统进行举报该任务，系统对其进行判断，做出相应的处罚。

2.3.3 系统主要功能的数据流程图

1.用户注册、登录模块

注册、登陆流程图如图所示：

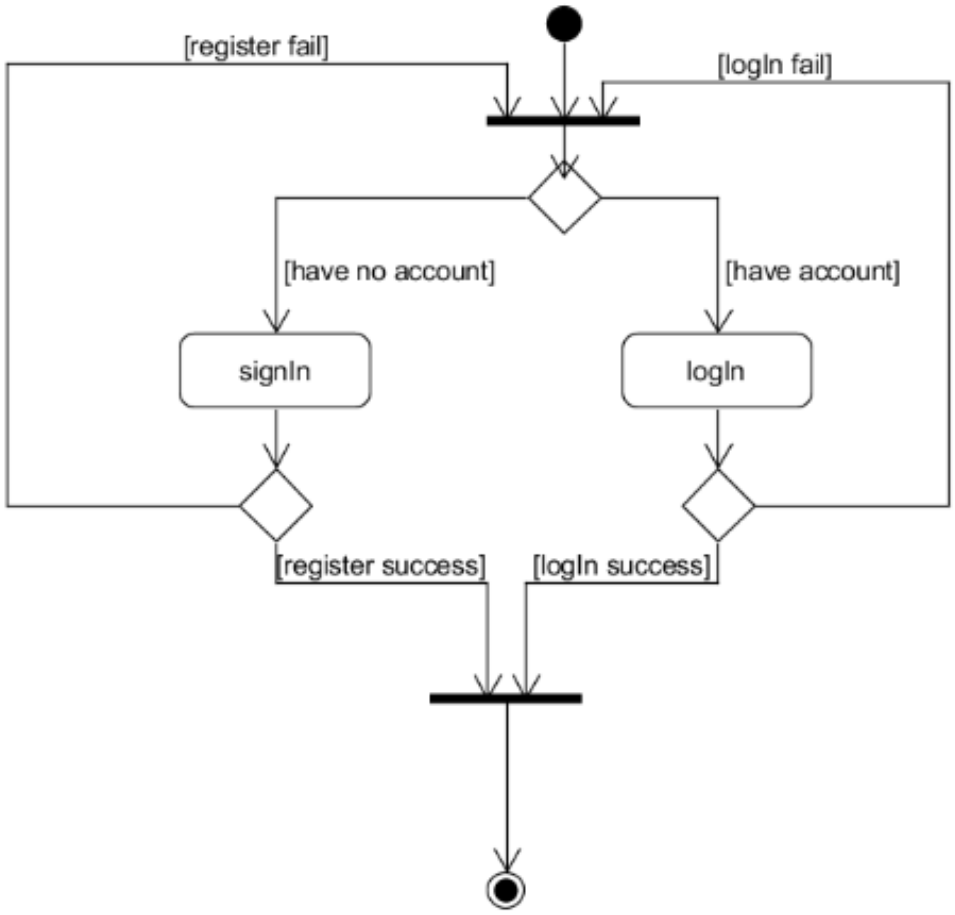


Figure 1 **注册登录**

2. 用户修改账户信息模块

用户修改账户信息流程图如图所示：

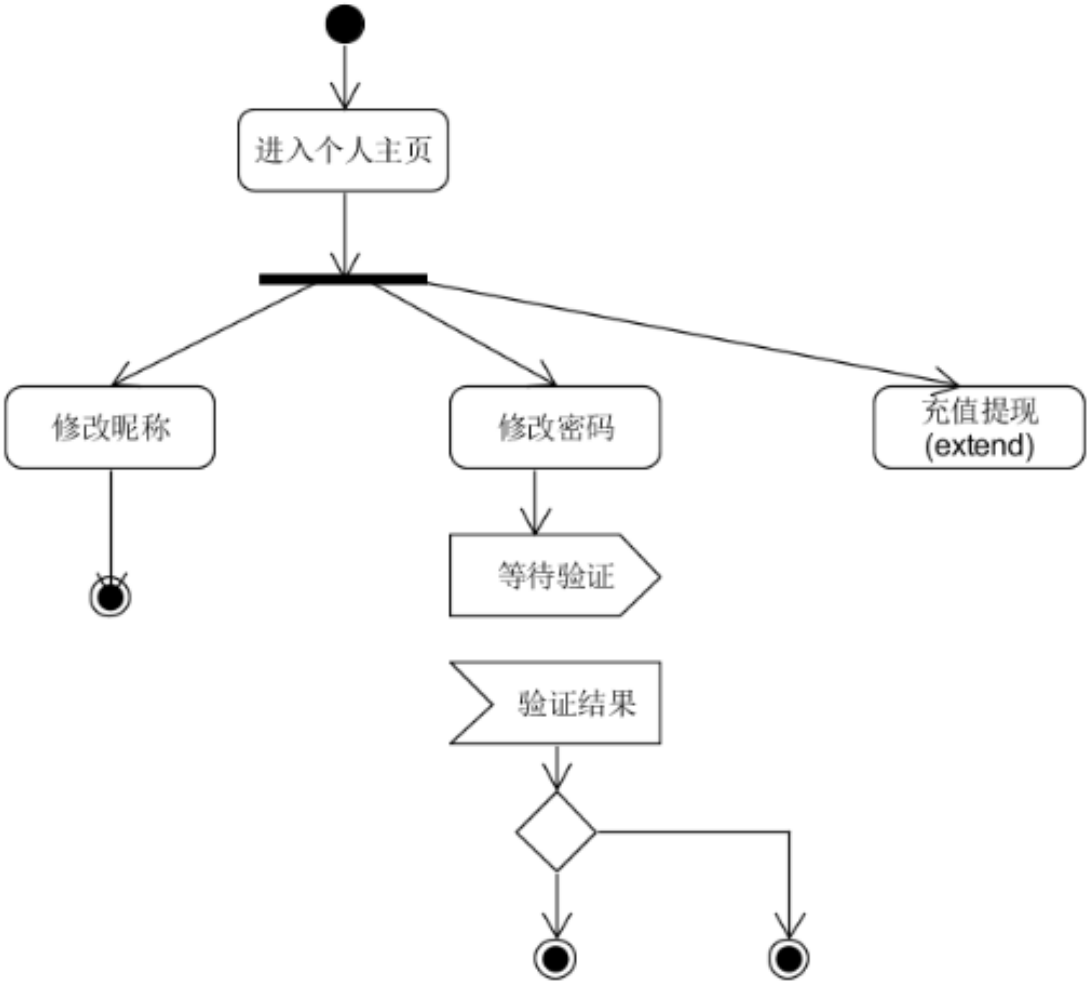


Figure 2 **用户修改账户信息**

3. 发布任务者发布任务模块

发布任务者发布任务流程图如图所示：

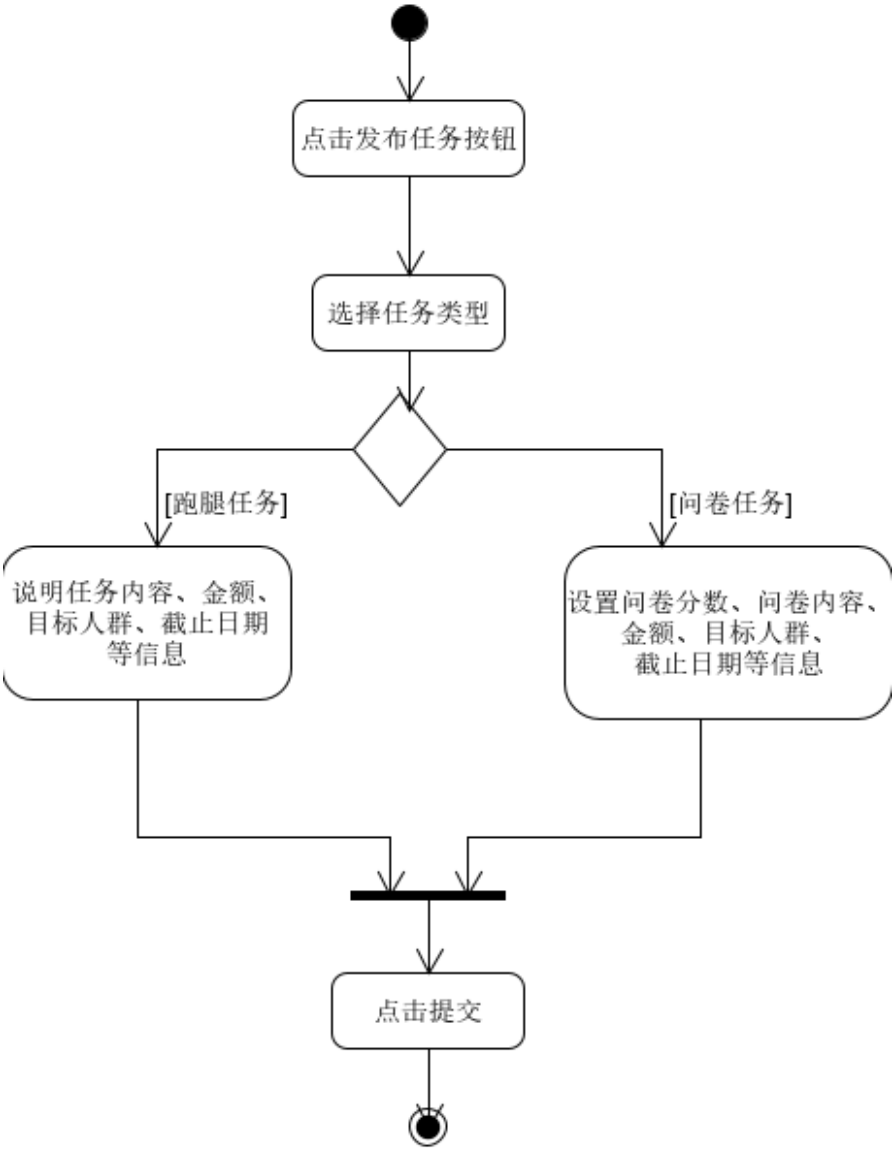


Figure 3 发布任务者发布任务

4. 发布任务者取消任务模块

发布任务者取消任务流程图如图所示：

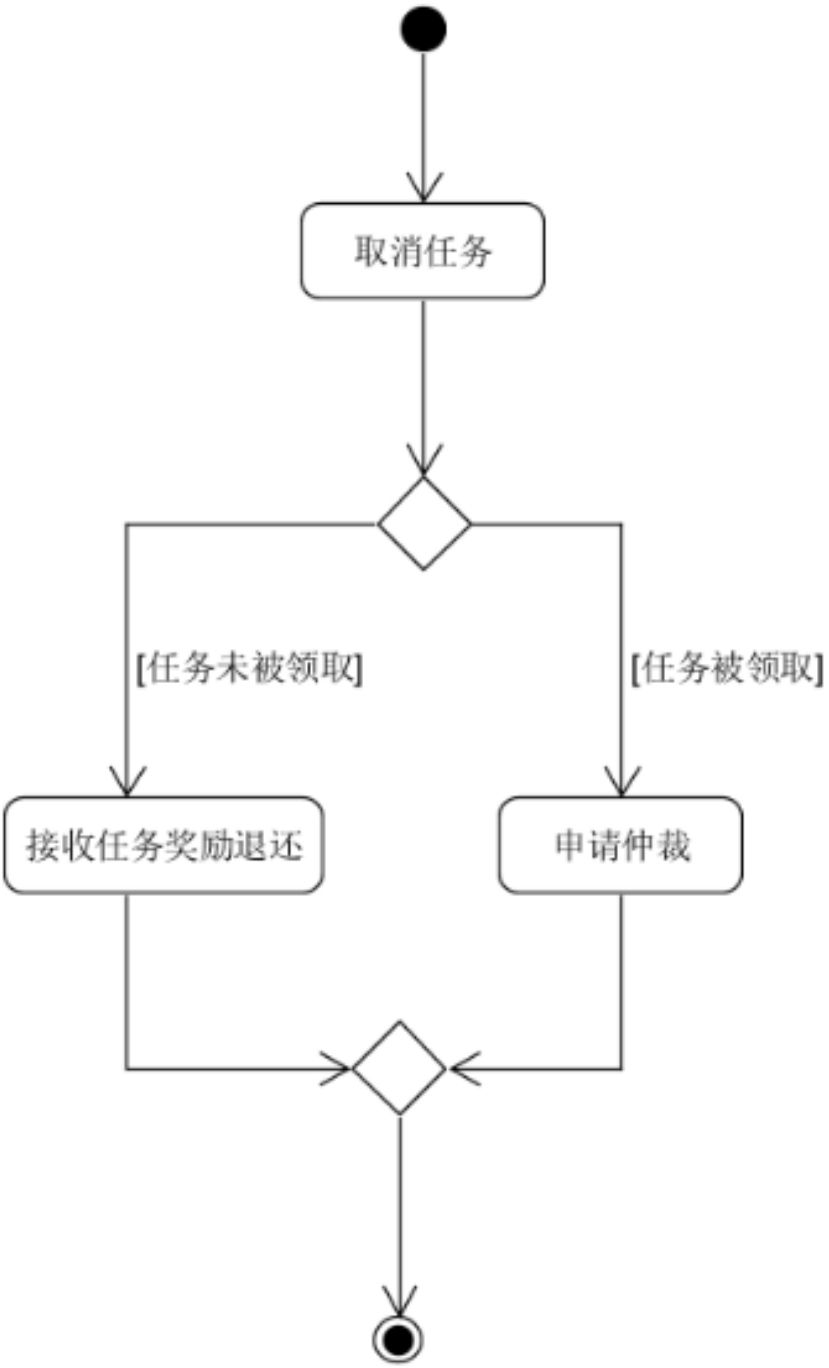


Figure 4 发布任务者取消任务

5. 发布任务者确认完成任务模块

发布任务者确认完成任务流程图如图所示：

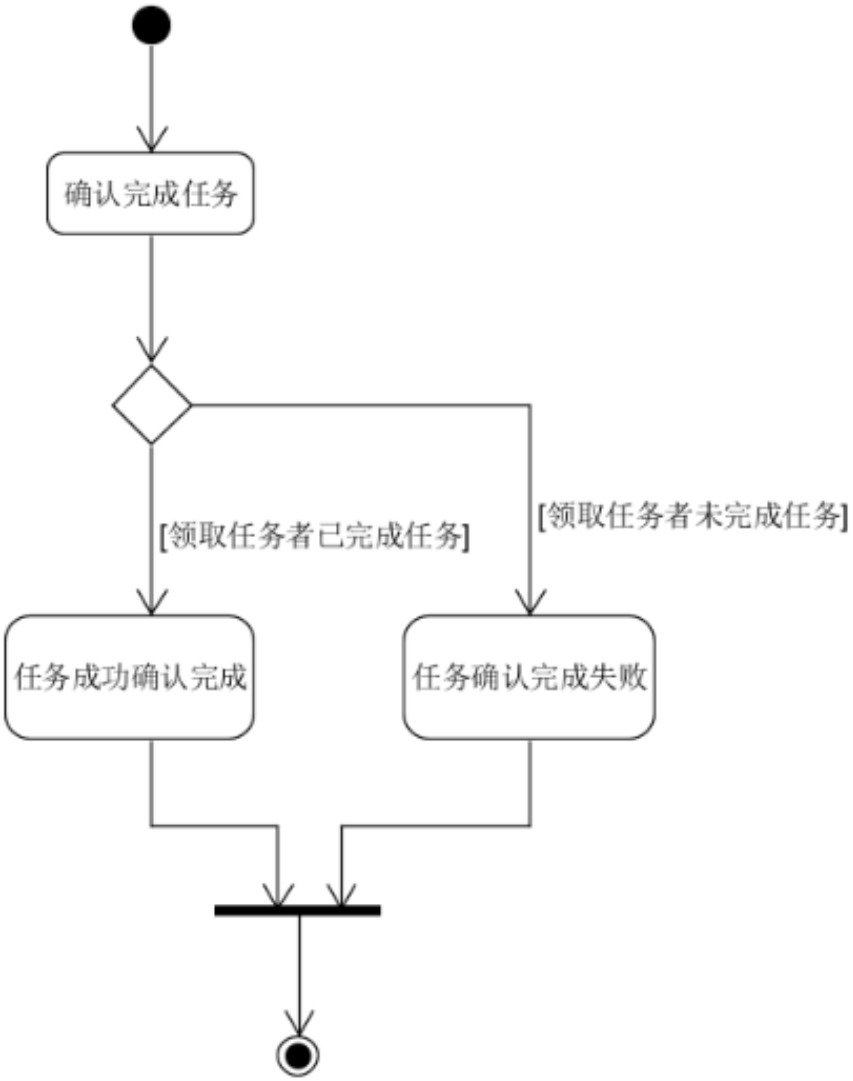


Figure 5 发布任务者确认完成任务

6. 用户查看任务模块

用户查看任务流程图如图所示：

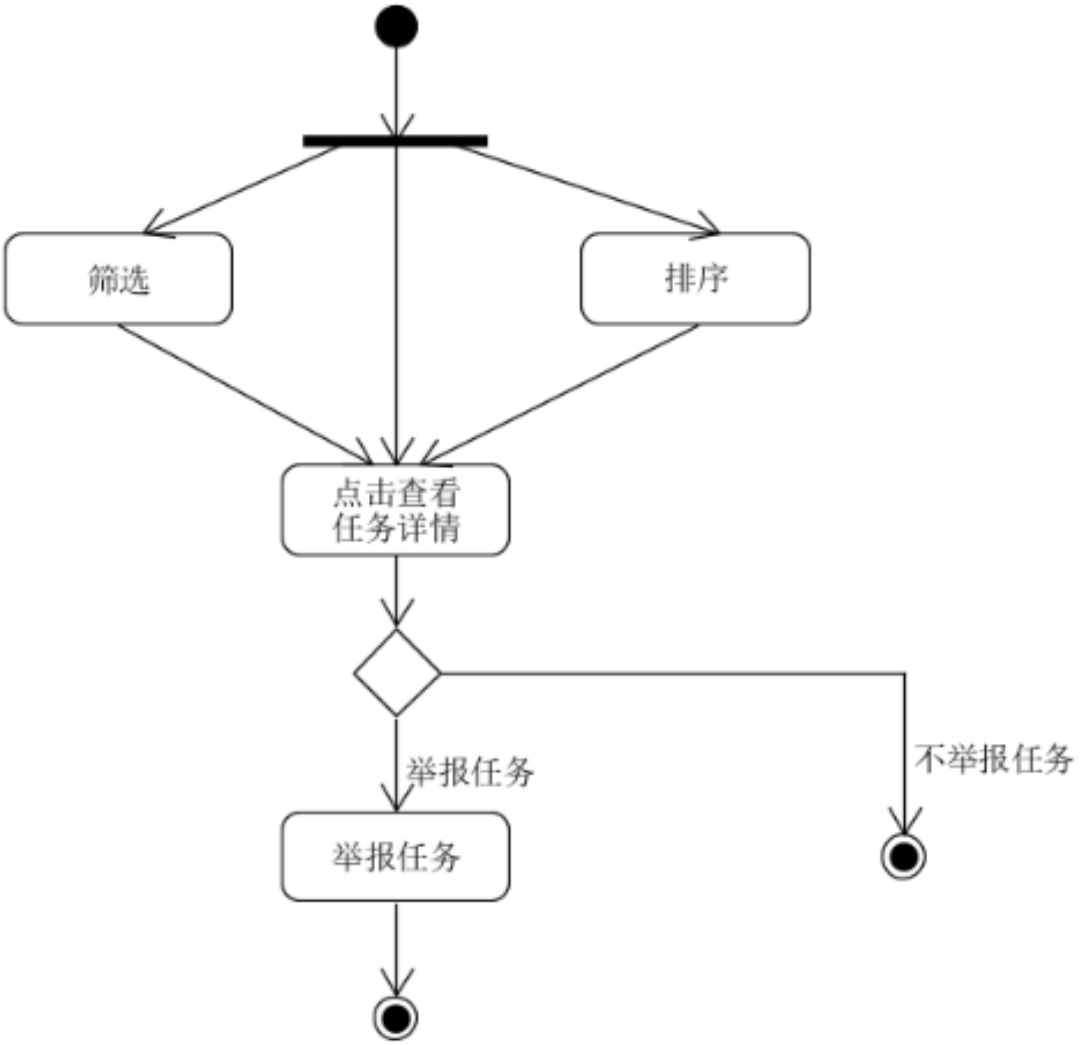


Figure 6 **用户查看任务**

7. 领取任务者领取任务模块

领取任务者领取任务流程图如图所示：

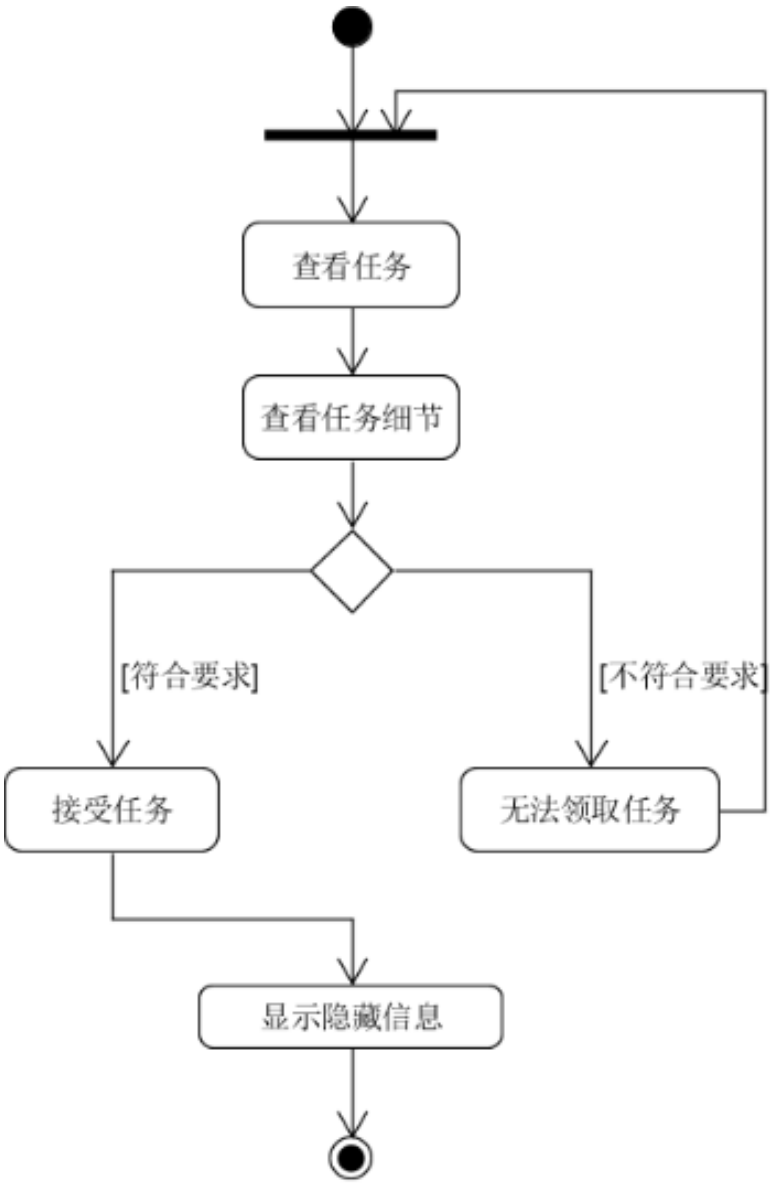


Figure 7 **领取任务者领取任务**

8. 领取任务者确认完成任务模块

领取任务者确认完成任务流程图如图所示：

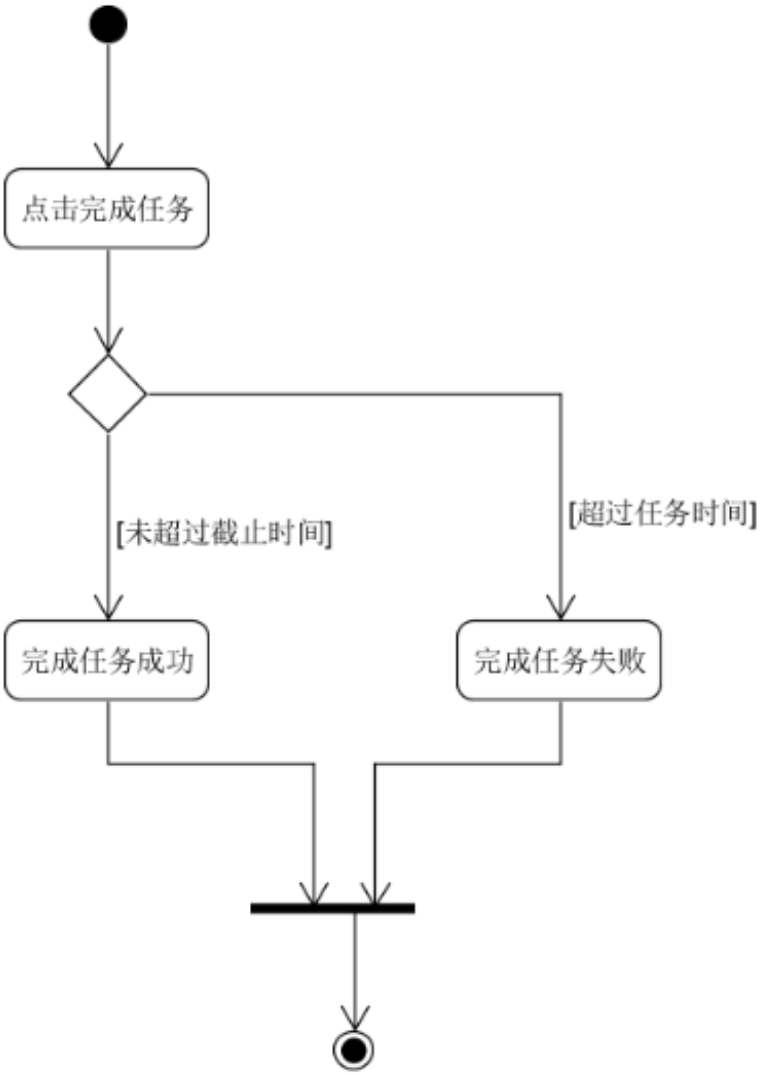


Figure 8 **领取任务者确认任务完成**

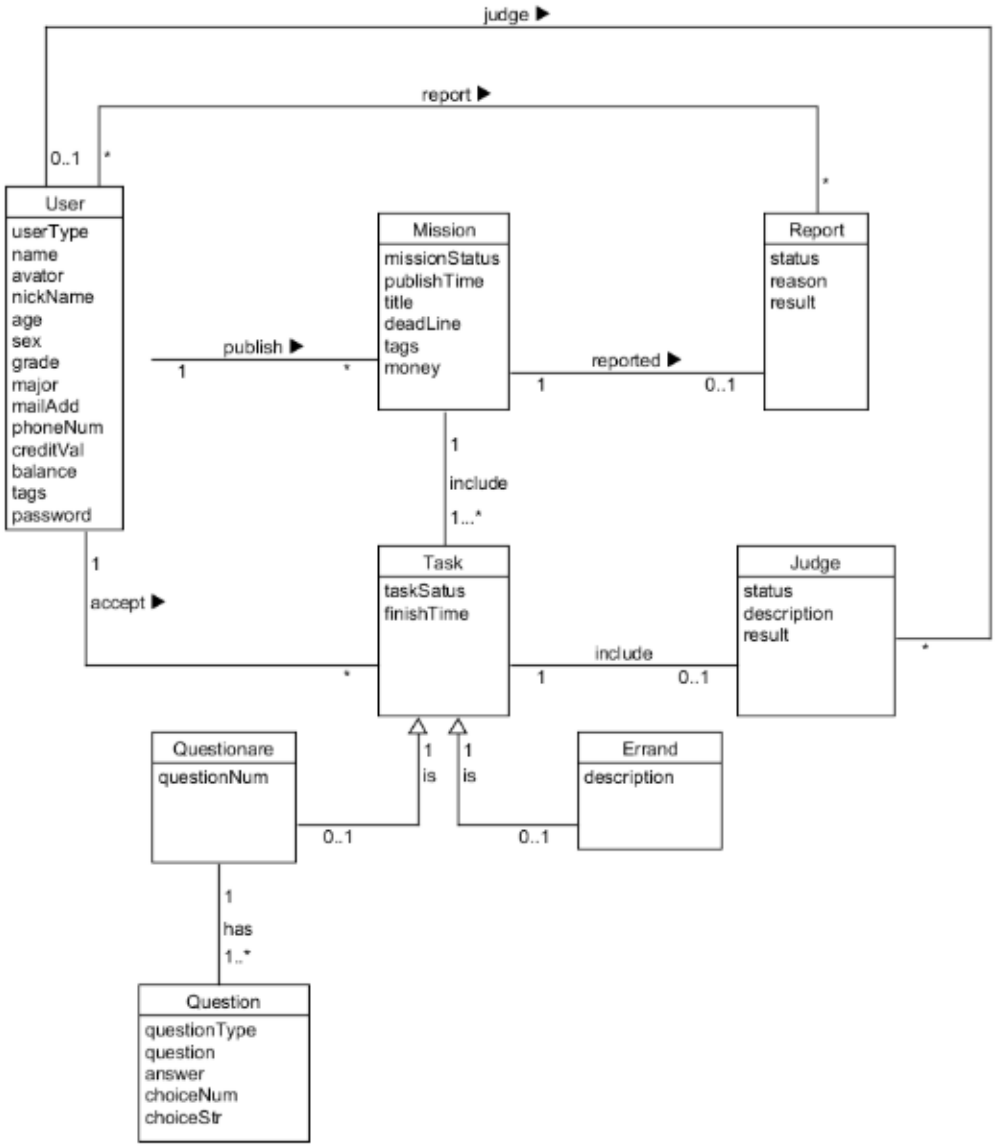
9. 用户举报任务模块

用户举报任务流程图如图所示：

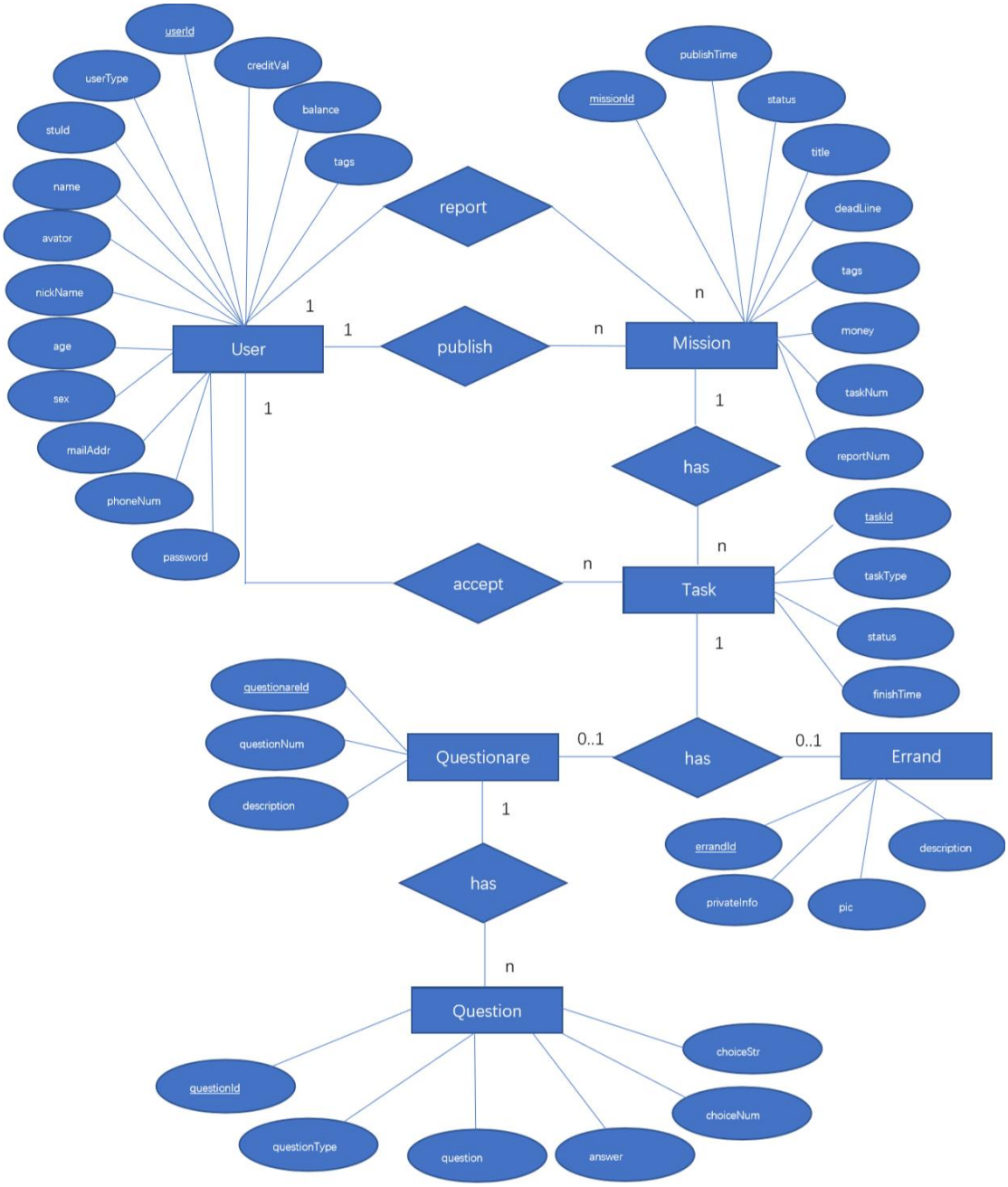
Figure 9 **举报任务**

三、 系统设计与实现

系统领域模型如下：



数据库 E-R 模型：



在需求分析通过之后，则进入项目的具体设计阶段，在此阶段需要完成的有详细功能设计，数据库设计，界面设计，软件结构设计，然后编码完成功能实现，最终通过测试之后，交付使用。

3.1 系统数据分析

通过对需求分析的用例分析，可以得出易闲圈的 字典分析和概念设计的实体。

3.1.1 系统的数据字典分析

数据字典是用来定义数据流图中各个成分的具体含义，它以一种准确的、无二义性的说明方式为系统的分析、设计及其维护提供了有关元素的一致定义和详细的描述。它与数据流图共同构成了系统的逻辑模型，是需求规格说明书的重要组成部分。数据字典有以下四类：数据流、数据项、数据存储、处理逻辑。数据项是组成数据流和数据存储的最小元素。源点、终点不在系统之内，故一般不在字典中说明。系统数据字典，如下表所示：

数据流名	来源	去向	说明
登录	用户输入的用户名和密码	验证后进入系统界面	用户进入界面
用户注册	用户填写的注册信息	注册成功后可直接登录系统	注册信息自动写入到后台数据库
个人资料	用户点击个人档案	修改资料或头像	修改后的资料自动更新后台数据库
发布任务者发布任务	用户登录 APP，选择发布任务	通过选择要发布的任务，然后填写相应信息	点击提交后，就写入了后台数据库
发布任务者确认完成	发布任务点检确认完成任务	后台将对该任务的状态更改为已完成	发布者点击后，后台就进行了相应的操作
用户查看任务	用户登录后，可选择进入查看页面	后台将根据用户的标签进行推荐任务	
领取任务	用户查看任务，可点击领取任务进行领取	将该任务 accUserId 设为该用户的 ID	点击领取后，就写入了后台数据库
领取任务者确认完成	用户登录后选中该任务后，点击完成	等待发布者的确认完成	根据发布者的回馈进行操作

举报任务（扩展）	查看任务后，点击举报	后台对其进行判断	后台判断后进行反馈

3.1.2 数据项的条目

数据项条目入表所示：

- User:主键：UserId

名称	类型	描述
userId	int(11)	用户 id
userType	int(11)	用户类型
stuId	varchar(32)	用户学号
name	varchar(32)	用户真实姓名
avator	varchar(128)	用户头像 URL
nickName	varchar(32)	用户昵称
age	int(11)	用户年龄
sex	int(11)	用户性别
grade	int(11)	用户年级
mailAddr	varchar(64)	用户邮箱
major	varchar(32)	用户专业

名称	类型	描述
phoneNum	varchar(64)	用户手机号码
creditVal	int(11)	用户信誉值
balance	double	用户余额
tags	varchar(255)	用户标签
password	varchar(255)	用户密码

物理模型：

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| userId    | int(11)      | NO   | PRI | NULL    | auto_increment |
| userType  | int(11)      | YES  |     | NULL    |                |
| stuId     | varchar(32)  | YES  |     | NULL    |                |
| name      | varchar(32)  | YES  |     | NULL    |                |
| avator    | varchar(128) | YES  |     | NULL    |                |
| nickName  | varchar(32)  | YES  |     | NULL    |                |
| age       | int(11)      | YES  |     | NULL    |                |
| sex       | int(11)      | YES  |     | NULL    |                |
| grade     | int(11)      | YES  |     | NULL    |                |
| major     | varchar(32)  | YES  |     | NULL    |                |
| mailAddr  | varchar(64)  | YES  |     | NULL    |                |
| phoneNum  | varchar(32)  | YES  |     | NULL    |                |
| creditVal | int(11)      | YES  |     | NULL    |                |
| balance   | double       | YES  |     | NULL    |                |
| tags      | varchar(255) | YES  |     | NULL    |                |
| password  | varchar(255) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
```

◆ Mission: 主键：missionId；外键：userId

名称	类型	描述
missionId	int(11)	任务 id

名称	类型	描述
publishTime	varchar(32)	任务发布时间
missionStatus	int(11)	任务状态
title	varchar(64)	任务标题
deadLine	varchar(32)	任务截止日期
tags	varchar(255)	任务标签
money	double	任务奖金
userId	int(11)	任务发布者的 id
taskNum	int(11)	任务需要的人数
reportNum	int(11)	任务被举报的次数

物理模型：

```
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| missionId     | int(11)      | NO   | PRI | NULL    | auto_increment |
| publishTime   | varchar(32)  | YES  |     | NULL    |                |
| missionStatus | int(11)      | YES  |     | NULL    |                |
| title         | varchar(64)  | YES  |     | NULL    |                |
| deadLine      | varchar(32)  | YES  |     | NULL    |                |
| tags          | varchar(255) | YES  |     | NULL    |                |
| money         | double       | YES  |     | NULL    |                |
| userId        | int(11)      | YES  | MUL | NULL    |                |
| taskNum       | int(11)      | YES  |     | NULL    |                |
| reportNum     | int(11)      | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
```

- Task：主键：taskId；外键：pubUserId，accUserId，missionId

名称	类型	描述
taskId	int(11)	任务 id
taskType	int(11)	任务类型
taskStatus	int(11)	任务状态
finishTime	varchar(32)	任务完成日期
pubUserId	int(11)	发布者的 id
accUserId	int(11)	接受者的 id
missionId	int(11)	对应的 mission 的 id

物理模型：

```
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| taskId     | int(11)     | NO   | PRI | NULL    | auto_increment |
| taskType   | int(11)     | YES  |     | NULL    |                |
| taskStatus | int(11)     | YES  |     | NULL    |                |
| finishTime | varchar(32) | YES  |     | NULL    |                |
| pubUserId  | int(11)     | YES  | MUL | NULL    |                |
| missionId  | int(11)     | YES  | MUL | NULL    |                |
| accUserId  | int(11)     | YES  | MUL | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

◆ Questionare:主键：questionareId，外键：taskId

名称	类型	描述
questionareId	int(11)	问卷 id

名称	类型	描述
taskId	int(11)	问卷对应任务的 id
questionNum	int(11)	问卷中问题的数目
description	varchar(255)	问卷描述

物理模型：

```
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| questionareId | int(11)      | NO   | PRI | NULL    | auto_increment |
| taskId        | int(11)      | YES  | MUL | NULL    |                |
| questionNum   | int(11)      | YES  |     | NULL    |                |
| description   | varchar(255) | YES  |     | NULL    |                |
| title         | varchar(255) | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
```

- Errand：主键：errandId，外键：taskId

名称	类型	描述
errandId	int(11)	跑腿 id
pic	varchar(255)	跑腿对应的图片 URL
taskId	int(11)	跑腿对应任务的 id
description	varchar(255)	跑腿描述
privateInfo	varchar(255)	接受跑腿后显示的联系方式等隐私信息

物理模型：

```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| errandId    | int(11)      | NO   | PRI | NULL    | auto_increment |
| description | varchar(255) | YES  |     | NULL    |                |
| pic         | varchar(255) | YES  |     | NULL    |                |
| taskId      | int(11)      | YES  | MUL | NULL    |                |
| privateInfo | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

◆ Question：主键：questionId，外键：questionareId

名称	类型	描述
questionId	int(11)	问题 id
questionType	int(11)	问题类型
question	varchar(255)	问题题干
answer	varchar(255)	问题的回答
choiceNum	int(11)	选择题可以选择的项目数量
choiceStr	varchar(255)	选择题所有选项的内容
questionareId	int(11)	问题对应问卷的 id

物理模型：

```
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| questionId    | int(11)      | NO   | PRI | NULL    | auto_increment |
| questionType  | int(11)      | YES  |     | NULL    |                |
| question      | varchar(255) | YES  |     | NULL    |                |
| answer        | varchar(255) | YES  |     | NULL    |                |
| choiceNum     | int(11)      | YES  |     | NULL    |                |
| choiceStr     | varchar(255) | YES  |     | NULL    |                |
| questionareId | int(11)      | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
```

3.1.3 数据存储条目

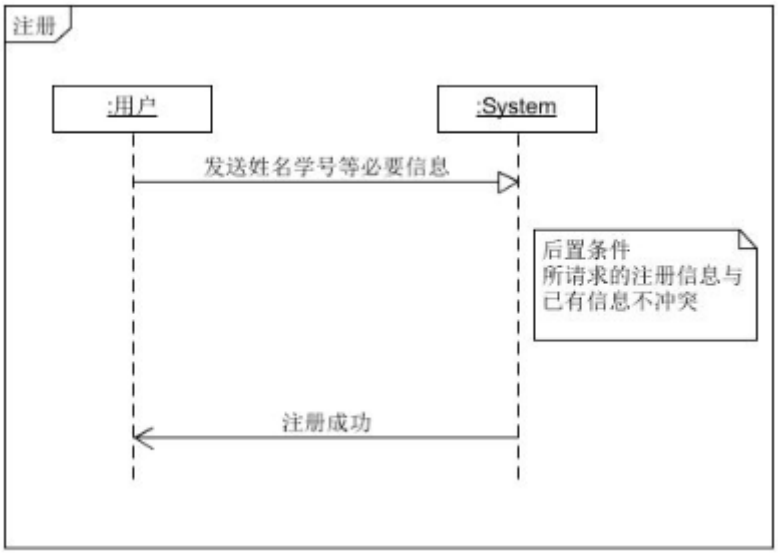
数据存储条目表如下所示：

数据存储名称	简述	组成
用户注册信息	存放用户的注册信息	userId+userType+stuId+name+avator+nickName+age+sex+grade+major+mailAddr+phoneNum
用户发布问卷任务信息	存放用户发布的问卷信息	Mission{taskNum+title+deadline+tags+money}+task+questionare{questionNum+description+title+question{questionType+question+choiceNum+choiceStr}}
用户发布跑腿任务信息	存放用户发布的跑腿信息	Mission{taskNum+title+deadline+tags+money}+task+errand{description+privateInfo+pic}

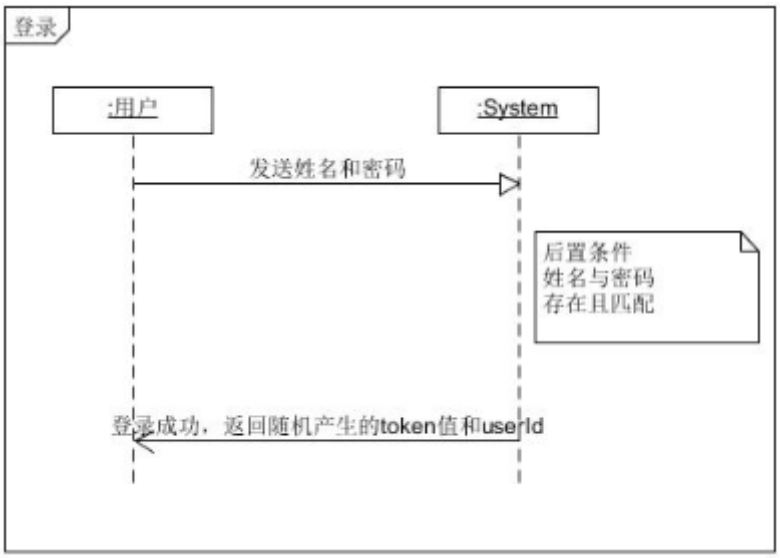
3.2 系统功能模型

我们根据我们系统的需求，主要讲系统分为三个功能模型，如下：一位注册功能模型、二为登录功能模型、三位任务功能模型：

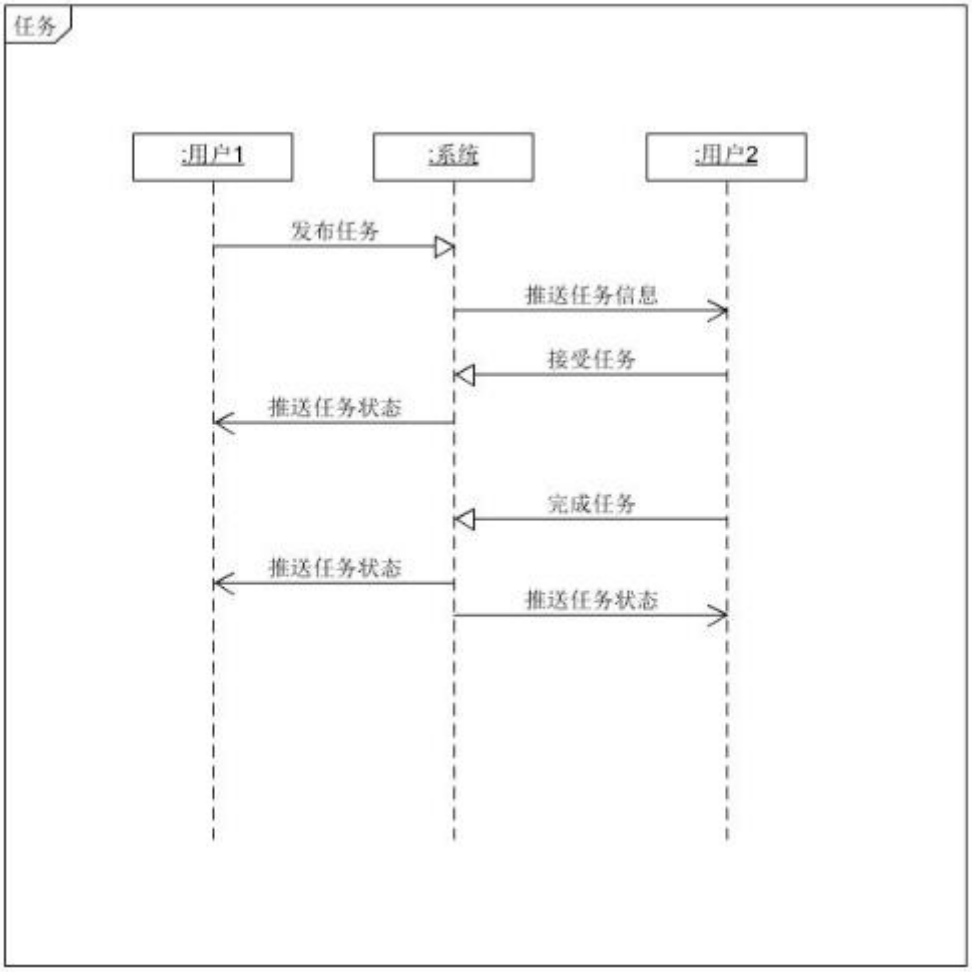
- **注册**



◆ 登录


登录
:用户
:System
发送姓名和密码
后置条件
姓名与密码
存在且匹配
登录成功，返回随机产生的token值和userId

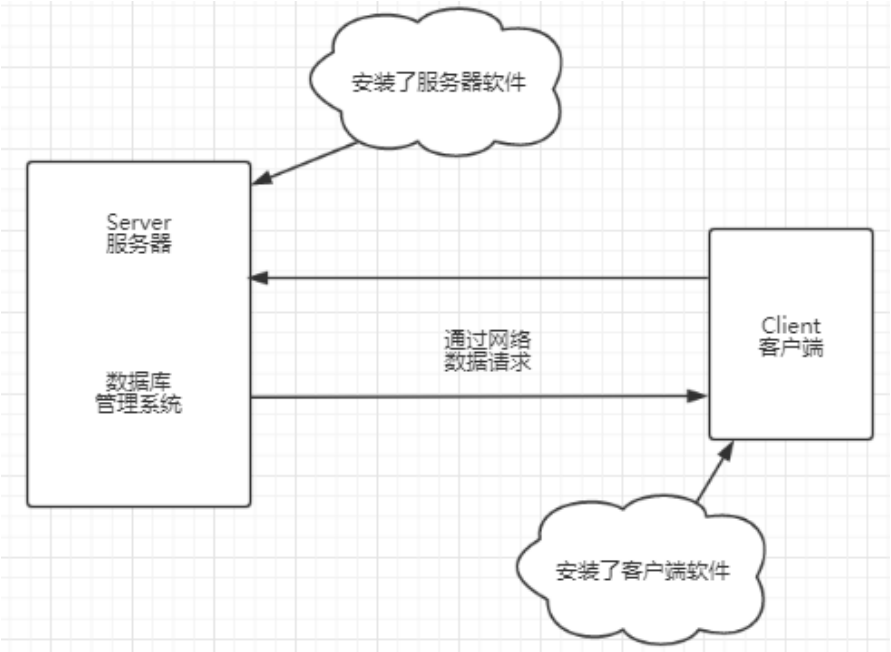
◆ 任务


任务
:用户1
:系统
:用户2
发布任务
推送任务信息
接受任务
推送任务状态
完成任务
推送任务状态
推送任务状态

四、 技术选型理由

4.1 项目总体框架----C/S 架构

C/S 架构软件（即客户机/服务器模式）分为客户机和服务器两层：第一层是在客户机系统上结合了表示与业务逻辑，第二层是通过网络结合了数据库服务器。简单的说就是第一层是用户表示层，第二层是数据库层。客户端和服务器直接相连，这两个组成部分都承担着重要的角色。服务器通常采用高性能的 PC、工作站或小型机，并采用大型数据库系统，如 Oracle、MySQL Server 等。客户端需要安装专用的客户端软件。如下图：



它具有的优点：

- 安全性：需要其特定的客户端，所以面向对象比较确定，将所进行的信息安全处于一个可控的范围。
- 效率：客户端的服务器直接相连，省却了中间环节，数据的传输比较快。
- 个性化：有特定的客户端，所以可以在较大程度上满足客户的个性化要求。
- 稳定性：结构比较稳定，有较强的事务处理能力，可以实现较复杂的业务逻辑。

在本次系统开发中，我们使用基于 C/S 架构的应用模式和开发技术，很好的满足基于网络环境下的挣闲钱的需求，可以很容易的在互联网上提供服务与获得服务。

4.2 后端技术

4.2.1 Nginx 反向代理

- **源代码：配置文件**

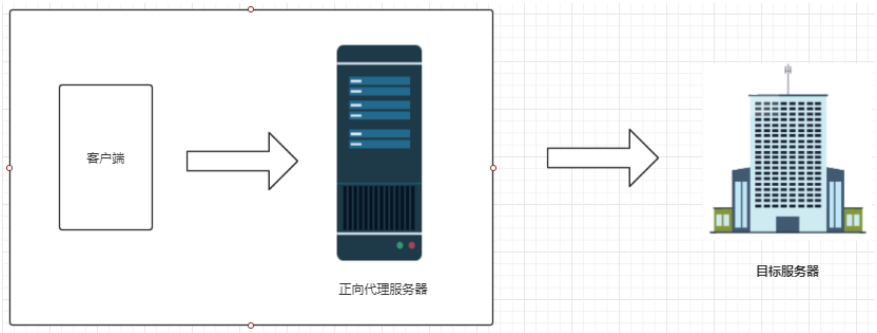
  https://github.com/sysuz4/AppServer/blob/master/default.conf

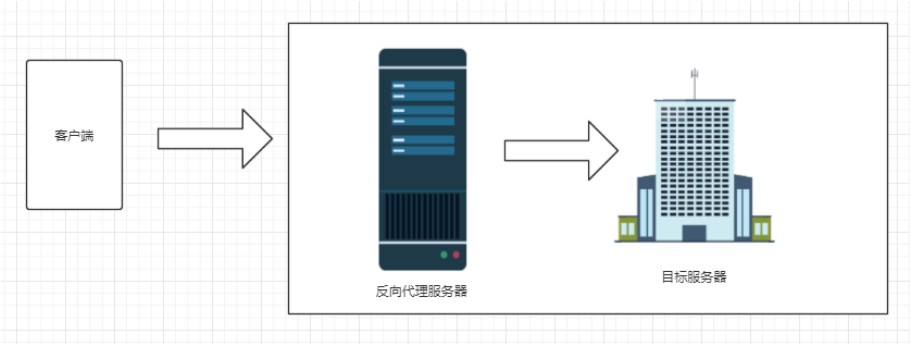
- **选择理由：**提高访问速度，防火墙作用以及通过代理服务器访问不能访问的目标站点。

- **Nginx 简介：**Nginx 服务器的反向代理服务是其最常用的重要功能，由反向代理服务也可以衍生出很多与此相关的 Nginx 服务器重要功能。在 Java 设计模式中，代理模式是这样定义的：给某个对象提供一个代理对象，并由代理对象控制原对象的引用。代理简单来说，就是如果我们想做什么，但又不想直接去做，那么这时候就找另外一个人帮我们去做。Nginx 主要能够代理的协议有：http/https，ICMP/POP/IMAP，RTMP，其中用到的最多的就是做 Http 代理服务器。

- **反向代理和正向代理的区别**就是：正向代理是代理客户端，反向代理是代理服务器。反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器 IP 地址。

  下面我们通过两张图来对比正向代理和方向代理：





总结起来还是一句话：正向代理是代理客户端，反向代理是代理服务器。

4.2.2 Nginx 静态服务器

- **源代码：配置文件**
  https://github.com/sysuz4/AppServer/blob/master/default.conf
- **选择理由：**轻量级同样起 web 服务比 apache 占用更少内存及资源，抗并发 Nginx 处理请求异步非阻塞而 Apache 则阻塞型高并发下 Nginx 能保持低资源低消耗高性能，高度模块化设计编写模块相对简单。
- **优点：**Nginx 可以作为静态 web 服务器，Nginx 在实际运维中，用到最多的地方是反向代理服务器，或负载均衡服务器。
  - 各功能基于模块化设计，扩展性好。
  - 支持平滑重启，实现应用不下线部署
  - 在多并发请求模型下，内存消耗低
  - 支持事件驱动模型和异步 I/O
  - 支持 FastCGI 协议将动态请求发往后端的动态 web 服务器，支持 WSGI（Python）等协议
  - master/worker 架构模型：一个 master 主进程，可以生产多个 worker 子进程

4.2.3 MySQL 数据库

- **源代码：**
  https://github.com/sysuz4/AppServer/tree/master/swsad/src/main/java/com/zgl/swsad/mapper
- **选择理由：MySQL 免费开源、性能出众**
- **MySQL 简介：**MySQL 是一个小型关系型数据库管理系统，开发者是瑞典 MySQL AB 公司。在 2008 年 1 月 16 号被 Sun 公司收购。然而 2009 年，SUN 公司又被 Oracal 收购.对于 Mysql 的前途，没有任何人抱乐观的态度。目前 MySQL 被广泛地应用在 Internet 上的中小型网站中。由于其体积小、

速度快、总体拥有成本低，特别是开放源码这一特点，许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。My SQL 有以下特性：（1）使用 C 和 C++编写，并使用了多种编译器进行测试，保证源代码的可移植性。

（2）支持 AIX、FreeBSD、HP-UX、Linux、Mac OS、Novell Netware、OpenBSD、OS/2 Wrap、Solaris、Windows 等多种操作系统 。

（3）为多种编程语言提供了 API。这些编程语言包括 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby 和 Tcl 等。

（4）支持多线程，充分利用 CPU 资源 。

（5）优化的 SQL 查询算法，有效地提高查询速度。

（6）既能够作为一个单独的应用程序应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他的软件中提供多语言支持，常见的编码如中文的 GB 2312、BIG5，日文的 Shift_JIS 等都可以用作数据表名和数据列名。

（7）提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径 。

（8）提供用于管理、检查、优化数据库操作的管理工具。

（9） 可以处理拥有上千万条记录的大型数据库。

4.2.4 SpringBoot 框架

- **源代码：整个后台都是用 SpringBoot 搭建的**
- **选择理由：**使编码变得简单，SpringBoot 采用 JavaConfig 的方式，对 Spring 进行配置，并且提供了大量的注解，极大的提高了工作效率。
- **SpringBoot 简介：**我们大家都知道 Spring，Boot 是启动的意思，所以 SpringBoot 其实是一个启动 Spring 项目的一个工具，从根本上讲，SpringBoot 就是一些库的集合，它能够被任意项目的构建系统所使用。Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化新 Spring 应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot 致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。以前在写 spring 项目的时候，要配置各种 xml 文件，还记得曾经被 ssh 框架支配的恐惧。随着 spring3，spring4 的相继推出，约定大于配置逐渐成为了开发者的共识，大家也渐

渐的从写 xml 转为写各种注解，在 spring4 的项目里，你甚至可以一行 xml 都不写。虽然 spring4 已经可以做到无 xml，但写一个大项目需要茫茫多的包，maven 配置要写几百行，也是一件很可怕的事。现在，快速开发一个网站的平台层出不穷，nodejs，php 等虎视眈眈，并且脚本语言渐渐流行了起来（Node JS，Ruby，Groovy，Scala 等），spring 的开发模式越来越显得笨重。在这种环境下，spring boot 伴随着 spring4 一起出现了。spring boot 并不是一个全新的框架，它不是 spring 解决方案的一个替代品，而是 spring 的一个封装。所以，你以前可以用 spring 做的事情，现在用 spring boot 都可以做。现在流行微服务与分布式系统，springboot 就是一个非常好的微服务开发框架，你可以使用它快速的搭建起一个系统。同时，你也可以使用 spring cloud（Spring Cloud 是一个基于 Spring Boot 实现的云应用开发工具)来搭建一个分布式的网站。

4.2.5 RESTful API

源代码：用 swagger editor 打开进行编辑

https://github.com/sysuz4/AppServer/blob/master/swagger.json

选择理由：我们基于 RESTful API 与传统 API 相比，具有的优点（下边有介绍），我们选择了 RESTful API

简介：REST 全称是 Representational State Transfer，中文意思是表述性状态转移。它首次出现在 2000 年 Roy Fielding 的博士论文中，Roy Fielding 是 HTTP 规范的主要编写者之一。他在论文中提到："我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。REST 指的是一组架构约束条件和原则。"如果一个架构符合 REST 的约束条件和原则，我们就称它为 RESTful 架构。

REST 本身并没有创造新的技术、组件或服务，而隐藏在 RESTful 背后的理念就是使用 Web 的现有特征和能力，更好地使用现有 Web 标准中的一些准则和约束。虽然 REST 本身受 Web 技术的影响很深，但是理论上 REST 架构风格并不是绑定在 HTTP 上，只不过目前 HTTP 是唯一与 REST 相关的实例。所以我们这里描述的 REST 也是通过 HTTP 实现的 REST。

RESTful API 的优点：

- 前后端分离，减少流量
- 安全问题集中在接口上，由于接受 json 格式，防止了注入型等安全问题
- 前端无关化，后端只负责数据处理，前端表现方式可以是任何前端语言
- 前端和后端人员更加专注于各自开发，只需接口文档便可完成前后端交互，无需过多相互了解
- 服务器性能优化：由于前端是静态页面，通过 nginx 便可获取，服务器主要压力放在了接口上

4.3 前端技术

4.3.1 Android Studio

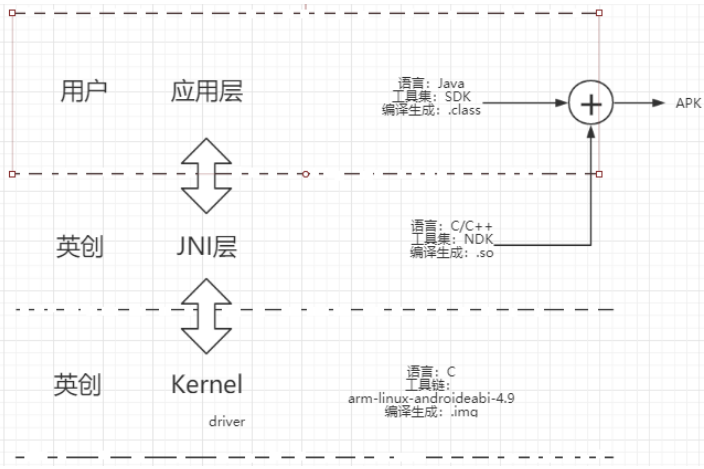
源代码：整个前端的代码都是 Android Studio 的代码

选择理由：我们基于做一个安卓 APP 的考虑，又学过这门课

简介：Android Studio 是谷歌推出的一个 Android 集成开发工具，基于 IntelliJ IDEA. 类似 Eclipse ADT，Android Studio 提供了集成的 Android 开发工具用于开发和调试。在 IDEA 的基础上，Android Studio 提供:

- 基于 Gradle 的构建支持
- Android 专属的重构和快速修复
- 提示工具以捕获性能、可用性、版本兼容性等问题
- 支持 ProGuard 和应用签名
- 基于模板的向导来生成常用的 Android 应用设计和组件
- 功能强大的布局编辑器，可以让你拖拉 UI 控件并进行效果预览

Android 应用程序主要使用 Java 语言编写,要用到开发工具集 SDK(Software Development Kit，提供 java 编译工具、Android 系统 API 等，可以直接在 Android Studio 中下载)。当应用程序中要直接访问硬件，或者需要提高运行效率时，需要将访问硬件、复杂逻辑部分使用 C/C++实现。要在 Android Studio 中开发编译 C/C++代码，需要用到工具集 NDK（Native Development Kit，提供 C/C++编译工具、API、打包工具等，可直接在 Android Studio 中下载)。使用 NDK 可以将 C/C++源码编译成动态链接库，供 Java 调用。由于 Java 语言要调用 C/C++函数需要用到 JNI（Java Native Interface）技术，这就要求使用 NDK 开发 C/C++时，C/C++源码要符合 JNI 规范要求。



为方便 Android 用户专注于 Android 应用层（Java 语言）的开发，英创公司对所支持的硬件接口均提供了符合 JNI 规范的 C/C++动态链接库，用户只需要加载英创的动态链接库，就可以在纯 Java 语言环境中调用动态链接库中的函数，达到访问硬件资源的目的。如图 4 所示，用户的工作只是应用层的 java 程序，英创已完成了其他部分工作。这篇文章会介绍 Android Studio 的环境搭建，并以 Step2_SerialPort 为例，来介绍使用 Android Studio 开发、调试、安装应用到 ESM6802 的过程。英创对已支持的硬件接口提供对应的应用程序示例，包括 C/C++部分的 so 文件，供用户参考。

五、 软件设计技术

5.1 MVC

源代码：整个后端代码框架

选择理由：耦合性低；重用性高；生命周期成本低；部署块；可维护性高；有利软件工程化管理。

简介：模型-视图-控制器模式，也称为 MVC 模式（Model View Controller）。用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC 被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。它把软件系统分为三个基本部分：

- 模型（Model）：负责存储系统的中心数据。
- 视图（View）：将信息显示给用户（可以定义多个视图）。

- 控制器（Controller）：处理用户输入的信息。负责从视图读取数据，控制用户输入，并向模型发送数据，是应用程序中处理用户交互的部分。负责管理与用户交互控制。

视图和控制器共同构成了用户接口。且每个视图都有一个相关的控制器组件。控制器接受输入，通常作为将鼠标移动、鼠标按钮的活动或键盘输入编码的时间。时间被翻译成模型或试图的服务器请求。用户仅仅通过控制器与系统交互。

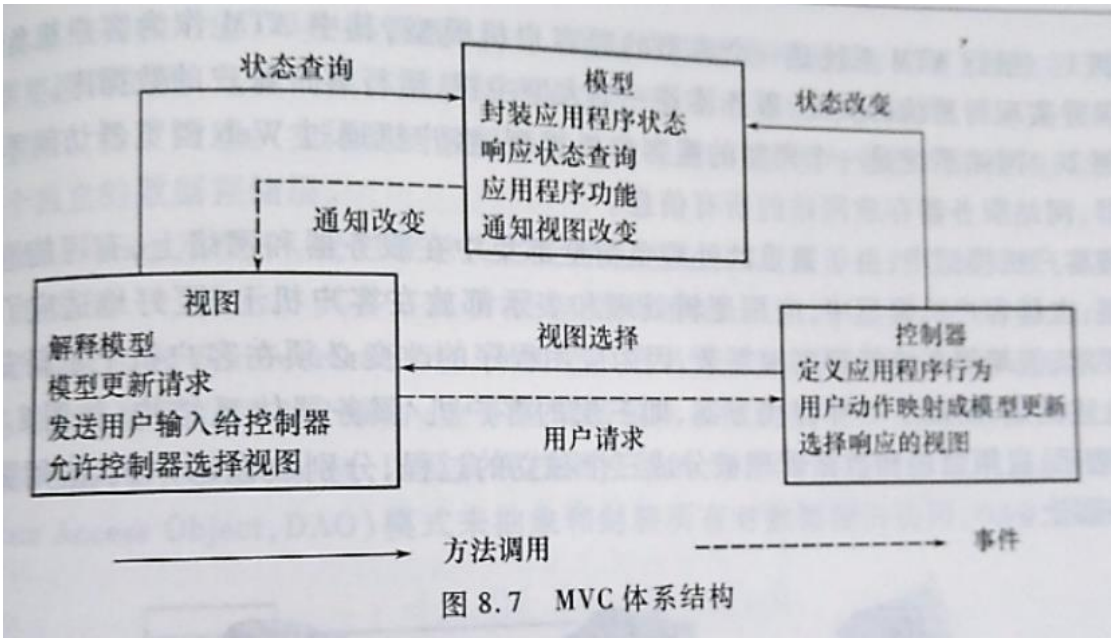


图 8.7　MVC 体系结构

5.2 Object-Oriented Programming

- **源代码**：前端几乎代码都是面向对象编程
- **选择理由**：这样使得我们的项目已维护、质量高、效率高、易扩展
- **简介**：面向对象的基本思想是使用类,对象,继承,封装,消息等基本概念进行程序设计。面向对象方法的三个基本特征:

  - 封装性：将对象的实现细节隐藏起来,通过一些公共的接口方法来供外部调用对象的功能
  - 继承性：是面向对象实现的的重要手段,子类继承父类,子类直接获得父类的非 private 属性和方法

- 多态性:子类对象可以赋值给父类对象引用,但运行的时候仍然表现出子类的行为特征,同一个类型的对象在执行同一个方法时,可能表现出不同的特征

在项目中，对各对象、组件进行了封装，只留给外部接口，调用它的其他组件不用知道其内部的具体实现方式，每个组件拥有自己的方法与行为，在解决整个事务的某问题中行使具体职责。

5.3 Structure Programming

- **源代码**：后端以及前端中函数的实现都是以顺序结构、选择结构、循环结构之一写的。
- **选择理由**：基于结构化程序设计子模块之间的不会相互影响以及该结构的优点，我们选择了结构化程序设计。
- **简介**：结构化程序设计是进行以模块功能和处理过程设计为主的详细设计的基本原则。结构化程序设计是过程式程序设计的一个子集，它对写入的程序使用逻辑结构，使得理解和修改更有效更容易。
- **结构化程序设计的优点**：结构化程序中的任意基本结构都具有唯一入口和唯一出口，并且程序不会出现死循环。在程序的静态形式与动态执行流程之间具有良好的对应关系。由于模块相互独立，因此在设计其中一个模块时，不会受到其它模块的牵连，因而可将原来较为复杂的问题化简为一系列简单模块的设计。模块的独立性还为扩充已有的系统、建立新系统带来了不少的方便，因为我们可以充分利用现有的模块作积木式的扩展。

六、 架构设计

6.1 架构问题

- 服务端
  - **鉴权**：由于用户的不同，不同用户能够访问的资源类型也不相同，因此要对 restful API 中资源的访问进行鉴权，用户只能访问个人的信息，以及其它用户的一些非敏感信息,并且部分 API 是允许未登录用户调用，而部分 API 是必须登录用户才能调用的，考虑到这样的鉴权需要，需要一个简单有效的鉴权方案。

- **不同资源的获取：**一些静态资源，例如图片这些，通过数据库来读取的话速度比较慢，而一些文本类型的资源就可以通过关系型数据来进行读取。因此需要有不同类型的服务器来针对不同类型资源的请求进行服务。
- **分布式负载：**当所有请求都在一个服务器上进行处理的时候，可能会导致等待时间较长或者导致无响应的情况，因此需要进行分布式的负载。
- **可扩展的结构：**服务端需要一个可扩展的结构对服务进行分层，从而兼顾开发效率和运行效率。
- **自动测试与部署：**为了简化运维难度，需要在软件架构中考虑并设计自动测试与部署方案。

6.2 解决方案

◆ 服务端

- **鉴权：**服务端在路由层之后加入一个可选的鉴权层，通过鉴权层将 API 权限分为以下两类：

  - 只有登录用户才能调用的 API
  - 任何人都能调用的 API

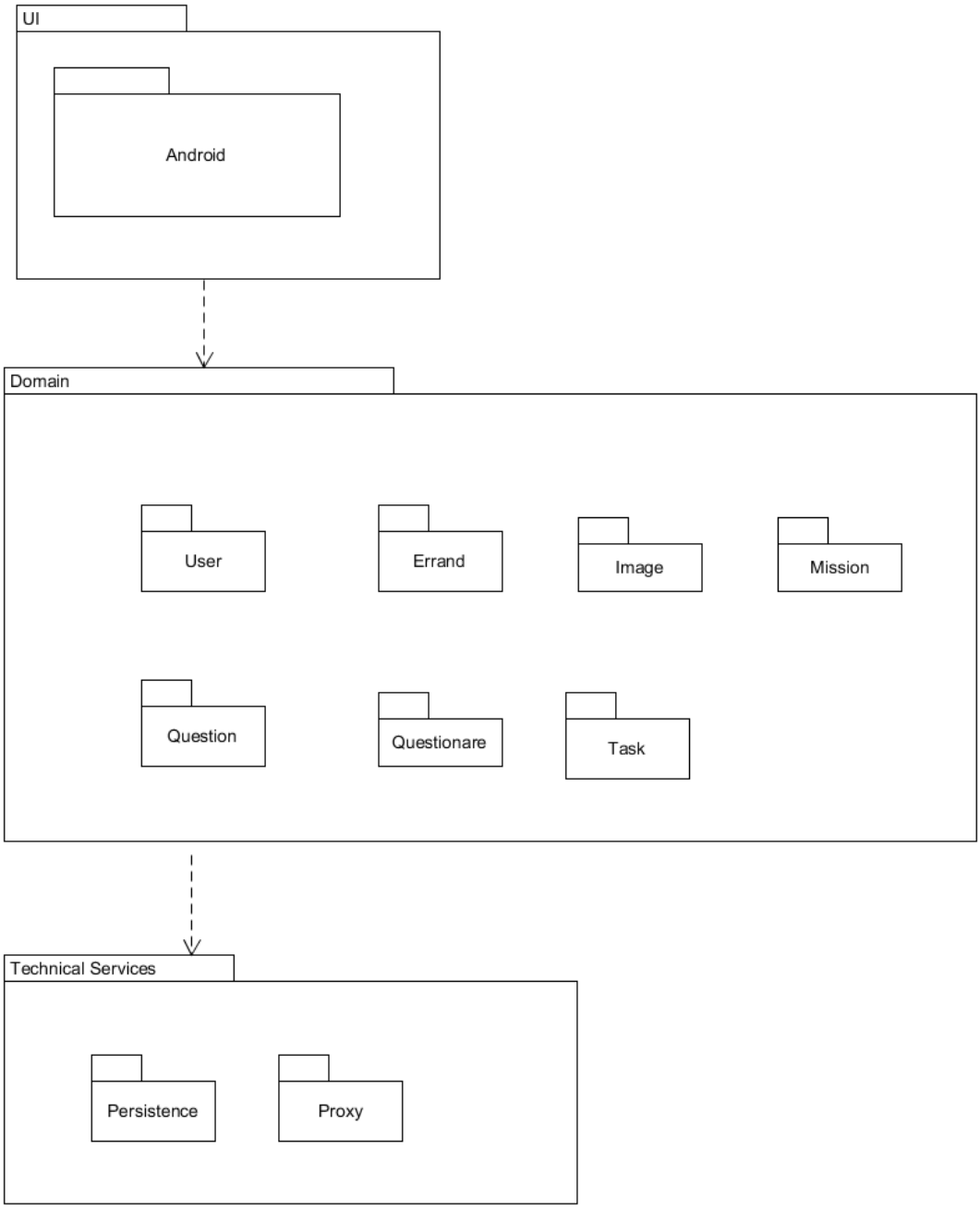
  并且在路由层中对登录用户与请求的资源进行对应的鉴权，部分个人敏感资源只有该用户自己才能进行访问。

  考虑到我们的客户端主要为安卓客户端，因此我们主要采用 token 机制进行鉴权处理，我们主要采用的是 JWT 的授权规范。

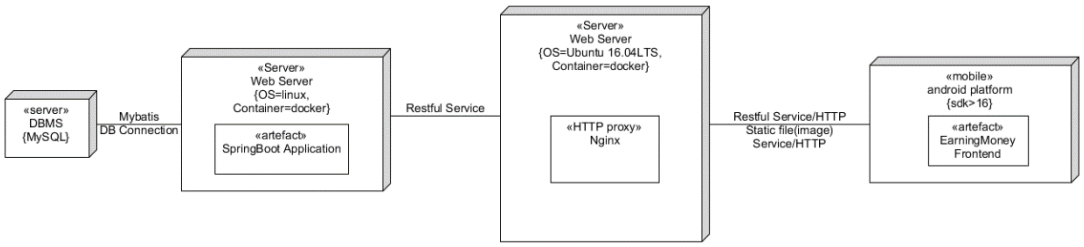
- **不同资源的获取：**对于图片这种静态资源，我们通过 nginx 作为静态资源服务器，处理这些静态资源，与此同时，nginx 也作为代理服务器，将非静态资源的请求转发给 springboot 服务端进行处理。
- **分布式负载：**分布式负载同样也是通过 nginx 进行实现，可以通过 nginx 将请求分发给不同的服务端进行处理。
- **可扩展的结构：**服务端结构分为六层：
  - Router：路由层，根据 API 路由到不同 Controller

- Authorize：鉴权层，提供 API 鉴权服务
- Controller：控制层，处理 HTTP 请求，收集参数，检查参数类型和合法性，并调用相应的 Service
- Service：服务层，处理业务逻辑
- mapper：处理数据库的交互
- Model：数据层，数据模型

6.3 逻辑视图



6.4 物理视图



七、 模块划分

7.1 后端模块划分

后端我们主要分成了六层

- Router：路由层，根据 API 路由到不同 Controller（由框架内部实现）
- Authorize：鉴权层，提供 API 鉴权服务，用于拦截和过滤请求，验证请求头中的登录用户 token 是否合法，拦截非法的请求
- Controller：控制层，处理 HTTP 请求，收集参数，检查参数类型和合法性，并调用相应的 Service
- Service：服务层，处理业务逻辑
- mapper：处理数据库的交互
- Model：数据层，数据模型

其中：

◆ Controller：控制层中又根据 restful API 的设计，划分成了以下几个模块：

- Token 模块，用于处理用户的登录请求，返回登录成功生成的 token 与 token 有效性的检验
- Image 模块，用于处理图片的上传请求与保存
- User 模块，用于处理用户的创建，修改，查询请求
- Mission 模块，用于处理发布新建任务，以及对发布任务的相关信息查询请求

- Task 模块，用于处理接受个人任务，提交个人任务完成情况，查询个人任务相关信息的请求

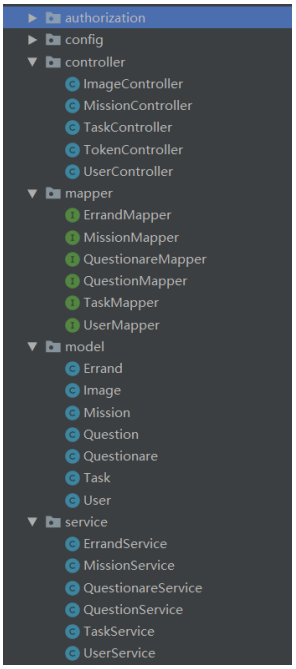
（Mission 和 Task 两个模块的划分是因为发布一个 mission，可以被多个用户接受，因此需将这两个模块进行划分，具体可见领域模型）

- Service：服务层中又因为具体任务中分成了问卷和跑腿两种任务，这两种任务即有部分相同的业务逻辑要求，又不部分不同的业务逻辑要求，所以还要进行进一步的模块划分，同样的问卷中又包含了不同种类的问题，为了将业务逻辑进行细分，在服务层中我们主要划分的模块为：
  - User 模块，用于处理用户创建修改等的一些业务逻辑要求
  - Mission 模块，用于处理创建一个抽象任务的业务逻辑要求
  - Task 模块，用于处理接受个人抽象任务的逻辑要求
  - Questionare 模块，用于处理具体到问卷的任务中的业务逻辑要求
  - Question 模块，用于处理具体到问卷中每个问题的业务逻辑要求
  - Errand 模块，用于处理具体到跑腿任务中的业务逻辑要求
- mapper：处理数据库的交互层，对数据库进行数据持久化操作，他的方法语句是直接针对数据库操作的，所以我们在这层划分的模块主要为对存储与数据库的信息进行操作，这这层中我们划分的模块为：对应各自的持久化操作
  - User 模块
  - Mission 模块
  - Task 模块
  - Questionare 模块
  - Question 模块
  - Errand 模块

  （我们的图片是作为静态资源存储，并不是存储在数据库中的，因此在数据库交互层中没有 image 模块）

- model 层：存放我们的实体类，与数据库或静态存储的属性值基本保持一致。

- User 模块
- Mission 模块
- Task 模块
- Questionare 模块
- Question 模块
- Errand 模块
- Image 模块，与保存图片的名字信息保持一致



7.2 前端模块划分

前端我们使用的是安卓移动端，因此我们的模块划分主要是根据页面进行划分，此外还有处理网络请求的模块。

7.2.1 页面模块的划分

每个页面包括　了布局文件和逻辑代码两部分：

- 登录注册页面
- 查看所有任务页面
- 查看个人发布与接收任务页面
- 查看任务详情页面
- 个人信息页面
- 创建问卷页面

- 创建跑腿任务页面
- 填写问卷页面
- 完成跑腿任务页面

7.2.2 service 模块

主要用于管理网络请求中的 URL

7.3 前后端共有的模块

- Constant 模块：用于定义用户，任务等的状态，便于协调。
- Util 模块：用于定义一些在其它模块中普遍要用到的函数，比如说将 java 中 Date 类型转换成我们定义是字符串格式函数等。