

江 苏 科 技 大 学

本 科 毕 业 设 计 (论 文)

学 院 _____计算机学院

专 业 _____计算机科学与技术

学生姓名 _____刘东旭

班级学号 _____152210708223

指导教师 _____景国良

二零一九年五月

江苏科技大学本科毕业论文

基于增强现实 AR 的智能社交网络研究

Intelligent social network research
based on augmented reality

摘要

现今市场上的社交网络平台以即时通讯、泛媒体、短视频三大类为主，信息体量大，内容偏重娱乐，甚至精心设计成瘾机制吸引用户，对大学学子产生了越来越多的负面影响。

本文论述了一个基于增强现实的智能社交系统，该系统以社交内容为交互中心，在增强现实技术帮助下，将仿真的社交信息和现实中的社交场景结合起来，借助不同社交场合同有的特殊性对社交信息进行筛选和分类，引导用户专注于发布在 AR 场景中的学习计划（或其它的生活目标），有效地避免了信息超载，减少了无效、低效社交。该系统采用客户端-服务器架构，客户端基于 Swift 编程语言和 Cocoa Touch 框架实现，应用 ARKit 2.0 开发增强现实界面，并使用 Core Data 框架实现本地数据持久化。系统服务器接口的开发应用 go-gin 微型框架，使用 gorm 框架驱动 Mysql 数据库，实现服务器端数据存取。

关键词： 增强现实；社交网络；智能社交；ARKit2.0；Go

Abstract

The social networking platform on the market today is dominated by instant messaging, pan-media and short video which feature large volume of information, inclination to entertainment content and well-designed addiction mechanism to attract users, causing more and more negative effects on university students.

This paper proposes an intelligent social system based on augmented reality and makes content on social media to be the interaction center. With the help of augmented reality technology, it combines simulated social information with real-life social scenes, screening and categorizing social information by the specialty of different social scenes, and guiding users to focus on learning plans (or other life goals) published in AR scenarios, which, as a result, effectively avoids information overload and reduces invalid and inefficient social interaction. The system adopts a client-server architecture, and the client is based on the Swift programming language and the Cocoa Touch framework. ARKit 2.0 is used to develop an augmented reality interface, and Core Data framework for local data persistence. The development of the system server interface uses the go-gin micro framework, as well as the gorm framework to drive the Mysql database, and therefore realizes server-side data access.

Keywords: Augmented reality; Social network; Smart social; ARKit2.0; Go

目 录

第一章 绪论	1
1.1 选题背景与意义	1
1.2 发展现状	2
1.3 主要研究内容	2
1.4 组织结构	3
第二章 开发技术与环境	4
2.1 开发技术	4
2.1.1 MVC 模式	4
2.1.2 Swift 语言和 Cocoa Touch 框架	4
2.1.3 ARKit2.0	5
2.1.4 Core Data	5
2.1.5 Go 语言及相关框架	5
2.1.6 Restful 接口设计规范	6
2.2 开发环境	7
第三章 可行性研究与需求分析	8
3.1 可行性研究	8
3.2 需求分析	8
3.2.1 功能需求	8
3.2.2 性能需求	10
3.2.3 出错处理需求	10
3.2.4 接口需求	10
3.2.5 数据需求	11
3.2.6 安全与保密需求	16
3.3 数据流图	16
3.3.1 顶层数据流图	16
3.3.2 数据流图的分解	17
3.4 状态转换图	21

第四章 应用总体设计	23
4.1 系统功能概述	23
4.2 系统结构	24
4.2.1 客户端	24
4.2.2 服务器	25
4.3 接口设计	27
4.3.1 用户管理	27
4.3.2 内容管理	32
4.4 测试环境	39
第五章 客户端设计与实现	40
5.1 项目目录简介	40
5.1.1 Assets.xcassets	40
5.1.2 Resources	40
5.1.3 Extensions	40
5.1.4 Scenes&Models	41
5.1.5 其它文件	41
5.2 Model	42
5.2.1 用户	42
5.2.2 内容	43
5.2.3 用户与内容的关系	45
5.3 View, Controller	45
5.3.1 页面导航	46
5.3.2 用户管理	46
5.3.3 AR 场景	48
5.3.4 内容管理	48
第六章 服务器设计与实现	51
6.1 开发目录介绍	51
6.2 数据库	51
6.2.1 数据库连接	52
6.2.2 数据表定义	52

6.3 网络接口 -----	55
6.3.1 路由分发-----	55
6.3.2 接口实现-----	57
第七章 系统测试-----	63
7.1 功能测试-----	63
7.1.1 客户端-----	63
7.1.2 服务器-----	65
7.2 性能测试-----	69
7.3 兼容性测试-----	70
7.4 测试结果-----	70
结论-----	71
致谢-----	72
参考文献-----	73

第一章 绪论

近年来，随着智能手机的普及和移动互联网的兴起，以手机 QQ、微信为代表的移动社交应用早已取代了论坛、贴吧等早期社交网站，成为主流社交渠道。同时，以微博为代表的泛媒体社交平台、以知乎为代表的网络问答社区也逐渐将发展重点转向移动应用；而诸如哔哩哔哩这样的老牌视频弹幕网站，抖音、快手这样方兴未艾的短视频分享平台，更是借助高速发展的流媒体技术成为移动互联网的中流砥柱。

即时通讯、泛媒体平台、短视频，小小的智能手机几乎可以满足年轻用户的所有社交、娱乐需求，而内容推荐算法的日渐成熟，瀑布流式布局的应用又使大部分社交、娱乐资讯失去了明确的停止信号，用户的私人时间在不知不觉中就被消耗在了一条接一条的聊天信息、30秒又30秒的短视频中，真正能思考、学习、创造的时间少之又少。据统计，在2007年至2017年之间，智能手机用户在私人时间平均屏幕使用时长增加了一倍之多，而且情况在这之后只会愈演愈烈。社交媒体的初衷是方便人们的生活，是以人为本的，却在发展过程中渐渐吞噬了用户的生活。

过度依赖社交媒体诚然是不健康的，但一味地要求用户自律，自行控制社交应用的使用频率，也是不现实的。如何利用新的技术改进移动社交媒体，提供一个崭新的、以帮助用户提高执行力、自制力为主旨的社交平台，才是本文的研究主题。

1.1 选题背景与意义

自互联网诞生以来，高速发展的社交网络服务持续不断地降低交流成本，丰富沟通手段，从起初的电子邮件、BBS，到如今的即时通讯软件和博客、短视频平台，交流速度越来越迅速，交流空间越来越广阔，交流对象也从最初的一对一变成了整个社交圈子。没有了传统沟通方式所固有的空间与时间上的限制，诚然信息交流的效率有了前所未有的提升，但信息的量级却逐渐超过了用户的处理能力。在现有的信息组织方式中，即时通讯软件粗暴地将整个人际关系网络搬上了互联网，不舍昼夜地运转，而博客类、短视频类的社交平台则利用成瘾机制，千方百计提高用户粘度，以获取更多流量。

本文尝试将增强现实技术与当前主流的社交网络架构相结合，将虚拟的社交信息与现实的社交场所相结合，提供一个全新的社交信息组织方案。这里的“新”，不只是技术上的“新”，更是基本理念上的“新”，包括以下三点：

- (1) 为用户提供明确的停止信号，摒弃瀑布流式布局；

- (2) 以满足用户的深层次需求为主导，而不是单纯的消耗用户时间，无下限地吸引用户的关注以赚取流量；
- (3) 帮助用户提高自制力，而不是处心积虑地利用成瘾性机制破坏用户的自律。

以这三点理念为指导，新平台

- 学习问答社区的做法，以用户提交的内容为互动核心，在链接其他用户的同时，将社区的注意力导向内容本身，最大程度上避免信息超载；
- 应用逐渐成熟的增强现实技术，开发简洁生动的 AR 交互界面，实现虚拟内容的具象化，提高人机交互的感染力；
- 引导用户在合适的场合公布自己的短期计划，将社交关注度转化为具有鼓励意义的正反馈。

在提高用户执行力的同时，为用户构筑一个反应即时而又富有安全感的社交环境，使每个用户在不多的私人时间里，过滤低效无用社交信息、摆脱碎片化的娱乐资讯的诱惑，关注自己，提升自己，成为更好的自己。

1.2 发展现状

随着移动互联网的蓬勃发展，社交网络平台的发展模式呈现多元化，除上文列及的即时通讯、泛媒体社交、短视频三类平台之外，还有许多定位更为精准，目标用户群体特征更为明显的平台，也有了长足的发展，如：

- 陌陌，主打陌生人社交，用户量超 1 亿；
- 探探，以异性交友为核心，用户注册量达 9000 万；
- 超级课程表，针对大学生市场，课表交友，总用户超 3632 万。

其中，探探已被陌陌收购，而超级课程表由于改版，弱化了课程表功能而加强社交属性，正渐渐流失用户。

1.3 主要研究内容

掌握在人机交互可视化界面的开发基本流程，学会分布响应用户操作、创造增强现实的社交网络环境，实时互动，信息共享。

使用 E-R 模型设计数据库，并使用 MySQL 关系数据库管理系统创建相关数据表和索引，以高效地存储、检索应用数据。

学习 Restful API 设计规范，设计 Web 数据接口；学习 Swift 语言及 IOS 开发框架，深入学习 ARKit 框架，构建 AR 交互界面。

1.4 组织结构

全文共有七章，每章的主要内容如下：

第一章是绪论，阐述了本文的选题背景和意义以及国内外的发展现状，总结了研究内容。

第二章是相关技术简介，对开发本系统需要用到的一些技术进行了介绍。

第三章是可行性研究及需求分析，从技术和应用两个方面进行了可行性分析，又通过需求分析导出了系统的逻辑模型。

第四章是应用总体设计，给出了系统架构，分析了系统流程。

第五章是客户端设计与实现，分模块详细论述了客户端的实现。

第六章是服务器设计实现，对服务器各个模块的接口实现和整个服务器的路由分发进行了详细阐述。

第七章是系统测试，运用黑盒测试的方法对系统进行功能测试，并使用专业的接口测试软件和 iOS 测试环境对系统进行综合测试，包括性能测试与兼容性测试。

第二章 开发技术与环境

本系统采用客户端-服务器设计架构，客户端的实现基于 swift 语言和 Cocoa Touch 开发框架，页面组织应用 MVC 模式，其中增强现实场景的实现应用 ARKit2.0 框架，数据持久化则应用 Core Data 框架，而服务器的实现基于 golang 语言，其中路由分发部分应用 go-gin 框架，数据模型的开发应用了 gorm 框架，并通过 gorm 提供的接口驱动 mysql 数据库存取数据。本章介绍系统开发过程中涉及的各项技术，并对整个开发环境作出说明。

2.1 开发技术

本系统实现的技术难点是增强现实场景的构建和虚拟社交信息的仿真，以及服务器接口对客户端请求的响应。增强现实场景的实现应用了 ARKit2.0 框架，客户端开发应用 MVC 模式，服务器接口的开发基于 golang 技术栈。

2.1.1 MVC 模式

MVC 模式（Model - view - controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）^[1]。

- 控制器 Controller，负责请求的处理和转发；
- 视图 View，显示在屏幕上的用户界面，响应用户事件，接受用户输入；
- 模型 Model，程序员实现的算法、编写的功能模块、数据管理和数据库设计。

iOS 程序开发深度应用了 MVC 模式，充分理解 iOS 的 MVC 模式，有助于客户端程序的组织合理性。

2.1.2 Swift 语言和 Cocoa Touch 框架

Swift 是 Apple 在 WWDC2014 发布的一门编程语言，用来编写 macOS 10 和 iOS 应用程序^[2]。Swift 有着现代开发语言的很多优秀的特性，如面对对象，多继承，协议与扩展对类进行的补充，key path 语法，optional type 以及泛型下标，继承 object c 的优点的同时，又为开发者提供了更多的便利。

Cocoa Touch 由苹果公司提供的软件开发 API，用于开发 iPhone、iPod、iPad 软件。它为开发者快速构建图形应用程序提供了可能。Cocoa Touch 提供了很多 macOS 平台上久经考验的模式，同时又特别专注于基于触摸的开发接口和性能优化^[3]。其中，UIKit 框架为开发者提供了基本的用户界面控件，包括视图、视图控制器、按钮等图元

控件，为用户界面的设计和实现提供了极大的便利。

UIKit 基于 Foundation 框架。Foundation 同样是 Cocoa Touch 的一部分，为开发者提供了文件处理、网络连接、字符串处理等其他基础类库。

Cocoa Touch 还包含其他框架，如 ARKit 2.0，将在下文进行介绍。

2.1.3 ARKit2.0

增强现实 (augmented reality, AR) 技术是将计算机生成的虚拟信息叠加到用户所在的真实世界的一种新兴技术，是虚拟现实技术的一个重要分支。它以虚实结合、实时交互、三维注册为特点，利用附加的图片、文字信息对真实世界进行增强，提高了用户对现实世界的感知能力，提供了人类与世界沟通的新的方式^[4]。

在本系统的开发过程中，主要用到了苹果公司提供的 ARkit 2.0 开发框架。ARKit 是苹果在 2017 年 WWDC 推出的增强现实开发平台，并于 2018 年发布第二版^[5]。开发人员可以使用这套工具为 iPhone 和 iPad 设备开发增强现实应用程序。

ARKit 集成了设备运动跟踪，摄像机场景捕捉，高级场景处理等高度封装的开发接口，有着很好的显示便利性，为开发者简化了构建 AR 应用的任务。开发者可以使用这些技术，以 iOS 设备的后置、前置摄像头为媒介，创建多种 AR 体验。

相比 ARKit，ARKit2.0 联合各大厂商推出了通用 AR 数据模型 USDZ，增强了对移动的 2D 图像和 3D 物体的追踪能力，并推出了多人联机模式。

2.1.4 Core Data

Core Data 也是 Cocoa Touch 的一部分，是 iOS 3.0 之后出现的一种数据持久化方案。Core Data 提供了一种对象关系映射的存储方案，直接将对象存储到数据库中去，同时将数据库中的数据转化为 Swift 对象，过程中不用编写任何 SQL 语句^[6]。

Core Data 将关系模型分解为属性和关系，进行数据库设计的同时，就完成了本地数据库的开发。

2.1.5 Go 语言及相关框架

Go (又称 golang) 是一门用于并发编程的命令式编程语言。它有 C 风格的语法，内建垃圾回收机制，使用 Go 协程(goroutines)和信道(channels) (一种基于 Hoare 的“通信顺序进程”理论的协程^[7]) 提供内建的并发支持。

Go 程序以包的形式组织。所谓的包本质上就是一个包含 Go 文件的文件夹。包内的所有文件共享相同的命名空间，而包内的标志符有两种可见性：以大写字母开头的符号

对于其他包是可见的，而其他符号则是该包私有的。Go 语言通过严格的包模型依赖关系检查机制来加快程序构建的速度。

Go 语言对于网络通信、并发和并行编程有着极好的支持，因为它本来就是以更好地利用大量的分布式和多核的计算机为设计目标的。Go 语言的设计者通过 Go 协程(goroutine)这种轻量级线程的概念来实现这个目标，然后通过信道(channel)来实现各个协程(goroutine)之间的通信。

本系统应用 gorm 实现数据存取，并应用 go-gin 框架实现服务器接口。

gorm 是一个优秀的对象关系映射框架，可以驱动多个不同引擎的数据库，本系统运用 Mysql 数据库管理系统存储数据。

go-gin 是一个基于 net/http 包的微型框架，对原生的 http 接口进行了很好的封装，为服务器接口的实现提供了很大的便利。

2.1.6 Restful 接口设计规范

REST，即 Representational State Transfer 的缩写。

Resource（资源）：对象的单个实例，包括文本、歌曲、视频、某种服务，总之就是一个具体的实在。实例可以用一个 URI（统一资源定位符）标记定位，每个资源对应一个特定的 URI，类似指针。要获取某一资源，访问它的 URI 就可以。URI 是每一个资源独一无二的识别符，或者说，是它的地址。

集合：对象的集合。

第三方：使用接口的开发者。

表现层（Representation）：“资源”是一种信息实体，它可以有多种外在表现形式。“资源”具体呈现出来的形式，就是它的“表现层”（Representation）。

状态转化（State Transfer）：访问一个网站，即意味着客户端和服务器发生交互。交互过程中，一定会涉及到数据和状态的变化。而互联网应用层通信协议 HTTP 协议是一个无状态协议。这就是说，所有的网络连接状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过 HTTP 协议提供的几个规定动作让服务器端发生“状态转化”（State Transfer）。这种转化是建立在表现层之上的，即“表现层状态转化”。

具体来说，就是运用 HTTP 协议里面四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源。

总结来说，RESTful 架构即是：

- 每一个 URI 代表一种资源；
- 客户端和服务器之间，传递这种资源的某种表现层；
- 客户端通过四个 HTTP 动词，对服务器端资源进行操作，实现“表现层状态转化。

2.2 开发环境

本系统的服务器开发环境及运行环境为 ubuntu 18.04，客户端开发环境为 macOS 10.14，开发工具有 Xcode 集成开发环境及 VS Code 编辑器。

Xcode 是由 Apple Inc 开发的，运行在操作系统 macOS 上的集成开发工具(IDE)。Xcode 是开发 macOS 桌面程序和 iOS 移动应用程序的最主流的开发工具，有着丰富的文档支持。Xcode 高度可视化的开发界面，页面的设计、数据模型的实现都可以通过简洁的图形用户界面完成，同时视图和视图控制器之间的数据绑定也可以很智能地通过拖曳连线等方式完成。

Visual Studio Code 是一个运行于 macOS、Windows 和 Linux 之上的，编写现代 Web 和云应用的跨平台源代码编辑器。VS Code 简便易用，有着丰富的第三方扩展，是当前最受欢迎的 golang 开发编辑器。

第三章 可行性研究与需求分析

合理的可行性分析和详尽的需求分析能够帮助我们规避风险，搭建良好的系统架构。本章对基于增强现实的智能社交应用进行可行性研究，并给出需求分析。

3.1 可行性研究

经过详细的调查研究，接下来将从技术实现和应用推广两个方面，简单的对项目进行可行性分析。

(1) 技术条件

在进行平台开发时，主要解决两个问题：

- ① 开发客户端应用，并设计实现一个符合用户需求的 AR 交互界面。
- ② 搭建服务器，存储用户信息。

经过前期调研，平台选用 iOS 系统为客户端操作系统，应用 ARkit2.0 框架实现 AR 交互，使用 Golang 语言实现符合 Restful 规范的网络接口，并建立 Mysql 数据库存储数据，相关技术已在前文有所介绍。

(2) 市场推广

本平台以现实生活中的学习、工作场景为应用场景，目标用户群体定位群体，有着广阔的应用市场。

3.2 需求分析

需求分析是软件定义时期的最后一个阶段，它的基本任务时准确地回答“系统必须做什么”这个问题^[8]，具体分析，本系统的业务需求是：

- (1) 改变现有高校社交网络逐渐娱乐化、碎片化的趋势；
- (2) 联系现实和网络，将特定场合的凝聚力和良好氛围通过手机拓延到社交网络；
- (3) 辅助用户进行自律，提高平台用户的执行力。

明确业务需求之后，进行需求分析和业务建模，得出系统功能需求和数据需求，进而得到系统的功能模型、数据模型与行为模型^[9]。

3.2.1 功能需求

完善、安全的用户管理系统是一个社交应用得以存在的基础，而为了满足目标用户群体对新平台的根本需求，一个逻辑清晰的内容共享系统和生动便捷的 AR 交互界面是必不可少的。

本系统可以分为三个主要功能模块，用户管理、内容共享和 AR 交互。

(1) 用户管理

出于保密性的考虑，平台可以通过邮箱直接注册，无需绑定手机号。

- ① 注册：新用户通过填入邮箱、密码注册，保证用户唯一性；
- ② 密码修改：用户在需要时可以修改密码，但必须经过邮箱验证；
- ③ 用户信息管理：用户在注册后，可以添加、修改账户信息，包括年龄等基本信息，以及保密选项。用户可以选择将账户相关信息对整个社区开放，也可以设置为好友可见；
- ④ 用户交流：用户之间不需要直接交流。关注其他用户之后，可以在第一时间接受对方的动态。动态的详情只有在特定的现实场景即该动态的发布场景才是可见的。

(2) 内容共享

用户可以在平台上发布文字内容，包括短期计划，心得交流，问答。所有在场的用户都可以通过 AR 界面评论、点赞或关注当前场景中已发布的内容。

- ① 内容的发布：用户可以直接进入编辑页面，发布内容。除了文字，编辑页还提供“进度条”、“子任务”等 UI 控件，方便用户编辑和发布短期计划。编辑完成并确认发布后，内容将以 3D 文字模型的形式展示在 AR 界面中，并与发布场景的位置信息绑定；
- ② 内容更新及删除：用户可以通过内容列表对已发布内容进行管理，包括
 - 更新进度；
 - 查看其他用户的评论，并进行回复；
 - 删除内容。
- ③ 相关动态通知：在 APP 启用的情况下，若用户所在位置存在内容，平台将会发出信号通知用户。（通知功能是可选的。）用户所发布的内容收到其它用户的反馈之后，平台也会发出相应通知。除此之外，不会以任何形式的提示打扰用户，毕竟应用存在的目的之一就是营造专注的氛围。

(3) AR 交互

AR 交互界面高度依赖于现实场景存在的，所以首先要识别用户所在场景，接着才能通过后置摄像头展示 AR 交互界面。

- ① 场景识别：通过获取手机所处位置信息在一定精度范围内识别并记录场景；

- ② AR 信息发布：识别场景之后，若该场景中存在已发布内容，那么便通过内容热度将之排序，而后生成相应的 3D 形象，显示在用户的 AR 界面中。一般内容直接通过 3D 文字展示，而短期计划比较特殊，不仅要显示各个子任务，还需要通过颜色的不同标志子任务是否完成，并通过百分比简明扼要地展示整个计划的进度；
- ③ 交互：用户可以在 AR 界面中通过触控手势、悬浮按钮与已发布的内容进行交互，包括评论、点赞与关注。在 AR 界面中也可通过悬浮按钮直接进入内容编辑页面，编辑发布新内容。

3.2.2 性能需求

响应时间：客户端各种操作事务的平均响应时间不超过 1s；服务端响应时间不超过 200ms；

吞吐量：2Mbps；

并发用户数量：满足同时执行某一操作的用户数量 200 人；

内存：运行内存占用不超过 200M。

3.2.3 出错处理需求

数据获取失败时，会给出失败原因以及响应的解决方案；

程序崩溃时，自动生成错误日志，并给用户提供反馈的接口。

3.2.4 接口需求

包括用户管理系统所需接口，内容共享系统所需接口。

(1) 用户接口需求

- 注册与登录：包含两个字段，用户名与密码；用户名要求为真实存在的邮箱号，密码由 6~20 位数字、字母和特殊符号组成；
- 修改账户信息：包含 Session id、待修改信息等多个字段；待修改信息一般包括年龄、性别、头像、简介等用户个人信息；
- 修改个人设置：包含 Session id、相关设置等多个字段；个人设置包括打开/关闭动态通知；
- 浏览个人信息：只包含 Session id，返回账户名及其它个人信息；
- 浏览自身已发布内容：只包含 Session id，返回用户已发布的每一条内容的 ID；
- 重设密码请求：只包含用户名字段，向用户名所在邮箱发送验证码；

- 重设密码验证：包含用户名与验证码；
- 重设密码：包含用户名与密码。

(2) 内容接口需求

- 发布内容：需要提供发布者 Session id，内容名，内容简介，详细描述，以及可能存在的子任务信息；
- 更新内容：需要提供发布者 Session id，内容 ID，以及对应的修改后的内容；若修改了子任务状态，则需提供子任务 ID；
- 删除内容：需要提供发布者 Session id，内容 ID；
- 与内容交互：需要提供交互者 Session id，内容 ID，对应的交互内容；这里主要有评论、点赞两种；凡是评论、点赞过的内容将自动关注。

3.2.5 数据需求

为了将本系统的数据需求清晰、准确地描述出来，建立概念性数据模型并绘制数据项表与 E-R 图。数据模型中包含三种互相关联的信息：数据对象、数据对象的属性及数据对象彼此间相互连接的关系，数据对象由数据模型和数据项表定义和描述，而数据对象直接的关系主要由 E-R 图进行体现^[10]。

(1) 数据模型

用户：用户账号，邮箱号，密码；

用户信息：用户账号，用户名，头像，性别，年龄，座右铭；

用户设置：用户账号，推送设置；

场景：场景编号，经度，纬度；

内容：内容编号，发布者编号，场景编号，类型，标题，综述，进度，缩略图，发布时间，更新时间，完成时间(只针对短期计划类型的内容)；

子任务：子任务编号，内容编号，内容，是否完成；

评论：发布者编号，被评论内容编号，评论内容，发布时间；

点赞：发布者编号，被赞内容编号，踩/赞，发布时间。

(2) 数据项表

一个数据模型的数据元素可由若干个数据项(data item)组成。数据项是数据记录中最基本的、不可分的有名数据单位，是具有独立含义的最小标识单位^[11]。数据项表中给出了各个数据项的编号、别名、简述、数据项的长度、类型、数据项的取值范围，如表

3-1 所示。

备注栏对各数据项的含义及约束作出了说明。

表 3-1 数据项表

表名	字段名	数据类型	备注
user	u_id	integer	用户帐号, 主键
	email	varchar(20)	邮箱号, 非空, unique
	password	varchar(20)	密码, 非空
profile	u_id	integer	用户账号, 主键, 外键
	name	varchar(20)	用户名, 非空
	photo	varbinary(1024)	用户头像
	sex	varchar(1)	用户性别, '0'、'1'或空
	age	integer	用户年龄
setting	saying	varchar(100)	座右铭
	u_id	integer	用户账号, 外键, 主键
	push	varchar(1)	推送设置
location	l_id	integer	场景编号, 主键
	lng	double	经度, 非空
	lat	double	纬度, 非空
event	e_id	integer	内容编号, 主键
	u_id	integer	创建者账号, 外键, 非空
	l_id	integer	发布场景编号, 外键, 非空
	type	tinyint	类型, 非空, 默认为感言 0
	title	varchar(20)	标题, 非空
	summary	varchar(100)	综述
	progress	float	进度
	pic	varbinary(1024)	缩略图
	s_time	timestamp	发布时间, 非空
	u_time	timestamp	更新时间, 非空
sub_event	e_time	timestamp	计划完成时间, 非空
	s_id	integer	子任务编号, 主键
	e_id	integer	所在内容编号, 外键, 非空
	s_content	varchar(42)	内容描述, 非空
comment	completed	varchar(1)	是否完成, 非空
	c_id	integer	评论编号, 主键
	u_id	integer	发布者账号, 外键
	e_id	integer	被评论内容编号, 外键
	c_content	varchar(100)	评论内容, 非空
like	time	timestamp	发布时间, 非空
	u_id	integer	发布者账号, 外键, 主键
	e_id	integer	被赞内容编号, 外键, 主键
	islike	varchar(1)	赞或踩, 非空
	time	timestamp	发布时间, 非空

该表主要列举了用户基本信息表 user、用户个人信息表 profile、用户设置表 setting、场景表 location、内容基本信息表 event 子任务表 sub_event、评论表 comment、点赞表 like。

like 表的各项属性和相关约束。

Comment 表与 like 表的不同之处在于，单一用户可以针对单个内容进行多次评论，而单一用户只能对单个内容点赞一次，撤销点赞行为是对已创建的点赞实体进行修改。

(3) ER 图

ER 图可以将数据对象、数据对象属性及数据对象之间的联系直接明了地体现出来。由于各个实体的属性已经在数据项表中有了详细的描述，为了简洁起见，这里首先给出各个实体及其属性的关系图，而后给出各个实体之间的关系图。

① 用户与用户设置

用户实体存储系统用户的基本信息，有三个属性，其中账号由系统自动生成，邮箱和密码由新用户在注册时提供，如图 3-1 所示。

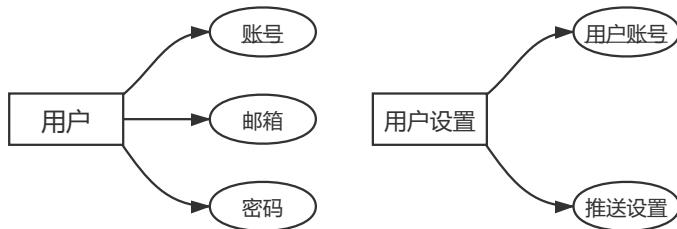


图 3-1 用户与用户设置实体图

② 用户设置

用户设置实体存储用户的相关设置，以用户账户为主键，推送设置为属性，如图 3-1 所示。

③ 用户信息

用户信息实体存储用户的个人信息，以用户账户为主键，以用户名、头像、性别、年龄、座右铭，如图 3-2 所示。

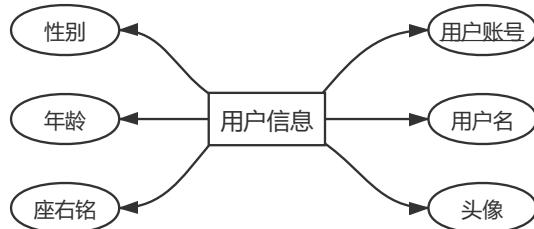


图 3-2 用户信息实体图

④ 内容

内容实体比较复杂，以创建者账号、发布场景编号为外键，自动生成的内容编号为主键，存储一个内容的类型、标题、综述、进度、缩略图、增删改时间等信息，如图 3-

3 所示。

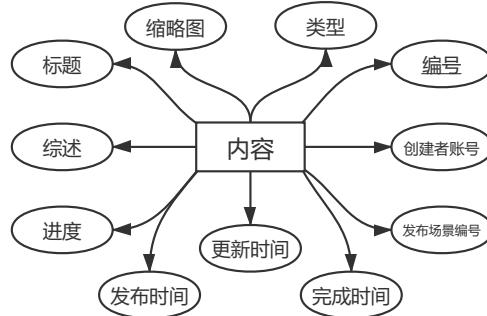


图 3-3 内容实体图

内容的类型有三种，短期计划、陈述性感言、提问。进度是短期计划类内容的派生属性，由各子任务的完成情况计算得出。

⑤ 子任务

短期计划类的内容需要额外的子任务实体来描述其详细步骤。子任务实体以其父内容的编号为外键，自动生成的子任务编号为主键，只有具体内容和是否完成两个属性，如图 3-4 所示。

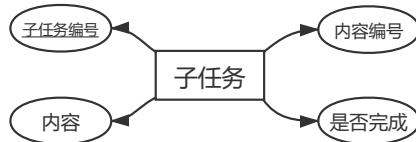


图 3-4 子任务实体图

当子任务的状态改变时，父任务的相关信息如进度、更新时间应该随之改变。

⑥ 场景

场景实体存储各个社交场所的地理位置信息，主键是场景编号，由系统自动生成，有经度、纬度两个属性，使用 WSG84 编码存储信息，如图 3-5 所示。

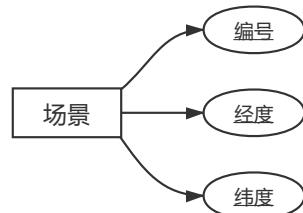


图 3-5 场景实体图

⑦ 评论

评论实体存储用户对某个内容的评论，以自动生成的编号为主键，发布者账号和被

评论的内容编号为外键，具体的评论内容和评论时间为一般属性，如图 3-6 所示。

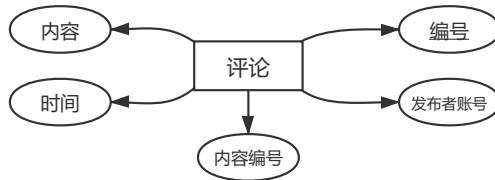


图 3-6 评论实体图

⑧ 点赞

与评论实体相似，但不同的是一个用户针对同一个内容只能创建一个点赞实体，因此该实体以内容编号和发布者账号这两个外键为联合主键，如图 3-7 所示。

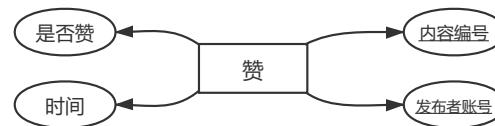


图 3-7 点赞实体图

给出各个实体之后，可以根据他们之间的关系将之分为三组，用户、用户信息为一组，存储用户的基本信息和个人信息，场景、内容、子任务为一组，描述内容信息，评论、点赞实体为一组，描述用户和内容之间的关系。

各个实体的关系图如图 3-8 所示。

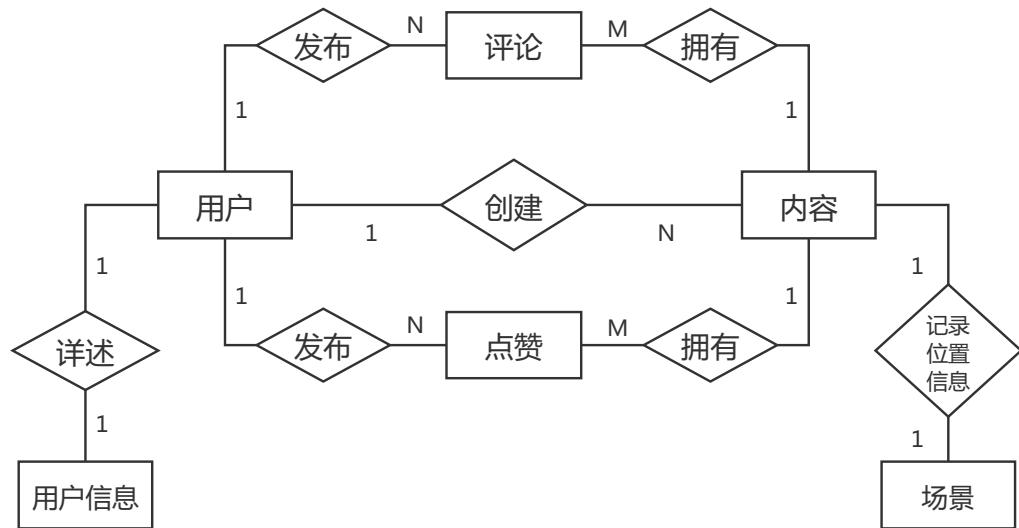


图 3-8 系统实体关系图

据图分析可知：

- (1) 用户信息实体对用户实体的进一步描述，两者之间是一对一联系；

- (2) 用户可以创建多个内容，即用户实体与内容实体有着一对多的联系；
- (3) 用户可以评论、点赞多个内容，而一个内容也可被多个用户评论、点赞，因此用户实体与内容实体之间可以借由评论、点赞实体产生多对多联系。用户和评论、点赞之间是一对多联系，评论、点赞和内容之间也是一对多联系；
- (4) 场景实体描述了内容实体的地理信息，两者之间是一对一联系；
- (5) 一个内容可以有多个子任务，两者之间是一对多联系。

3.2.6 安全与保密需求

用户信息加密存储，保证身份信息安全；用户密码使用 hash 算法进行加密，只存储 hash 值。

客户端和服务器之间通过 https 协议进行通信，保障链接安全。

公告信息因其公开性，不做加密处理。

3.3 数据流图

抽象的模块划分只是系统设计的第一步，详细的设计还要求我们对各个模块进行逐步求精^[12]。数据流图描绘了数据在软件中流动和被处理的过程，是重要的软件设计工具。本章列举顶层数据要素，并绘制顶层数据流图，而后通过面向数据流的自顶向下逐步求精方法，从顶层数据流着手，对平台功能进行结构化分析，得出各个子系统的数据流图，为之后求解软件结构打好了基础。

3.3.1 顶层数据流图

顶层数据流图将系统整个抽象为一个事务，与数据源点交换用户和内容信息，如图 3-9 所示。

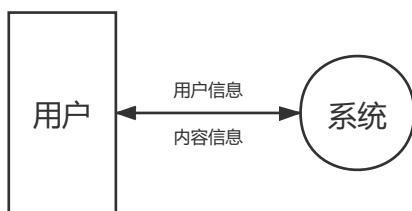


图 3-9 顶层数据流图

进一步分析数据流图之前，首先要对系统功能进行抽象。这里采用事物流模型，将整个系统的功能抽象为用户管理和内容管理两个事务，作为层次化分析的基础。

相关处理包括用户信息处理，内容处理，输出，数据存储包括用户表、内容表，最终将数据流通过客户端界面展示给用户。

二级数据流图如表 3-2 所示。

表 3-2 二级数据流元素列表

源点/终点	处理	数据流	数据存储
用户	用户信息处理	用户信息：邮箱+密码+个人设置	用户表
	内容处理	内容信息：标题+总结+详情+时间戳	内容表
	输出	内容：用户信息+内容信息	场景信息表

整合以上要素，绘制二级数据流图，如图 3-10 所示。

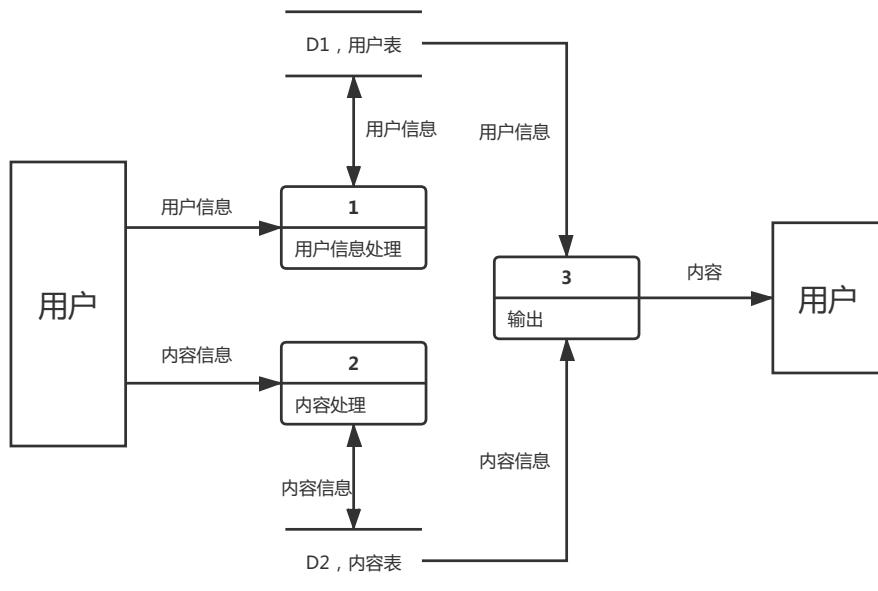


图 3-10 二级数据流图

在二级数据流图中，数据源地与数据终点都是用户。用户向系统输入个人信息和内容信息作为数据流，经过处理整合后输出结构化的内容。相应地得到经过结构化整合的用户信息和经由内容管理模块处理过的内容信息作为输出。

需要说明的是。图中 D1、D2 并不代表实际数据表设计，只是出于简洁对数据流进行的抽象，实际的用户表和内容表是由多个数据表构成。

3.3.2 数据流图的分解

根据二级数据流图，可以将 1、2、3 三个处理进一步分解，分别绘制详细数据流图。

(1) 用户信息处理

这个处理可以分解为用户注册、用户登录、个人信息编辑、用户设置修改、密码修改五个相对独立的处理，同时数据存储也出于方便分解为三个数据表，即用户基本信息表（账号+邮箱+密码）、个人信息表、用户设置表，以提高数据查询效率。

这五个子处理中，每个处理又可以分为两个相对独立的层次，即客户端处理与服务器处理，因此图中设置泳道，体现从用户到客户端、从客户端到服务器又经由客户端回到用户的整个数据流动过程，如图 3-11 所示。

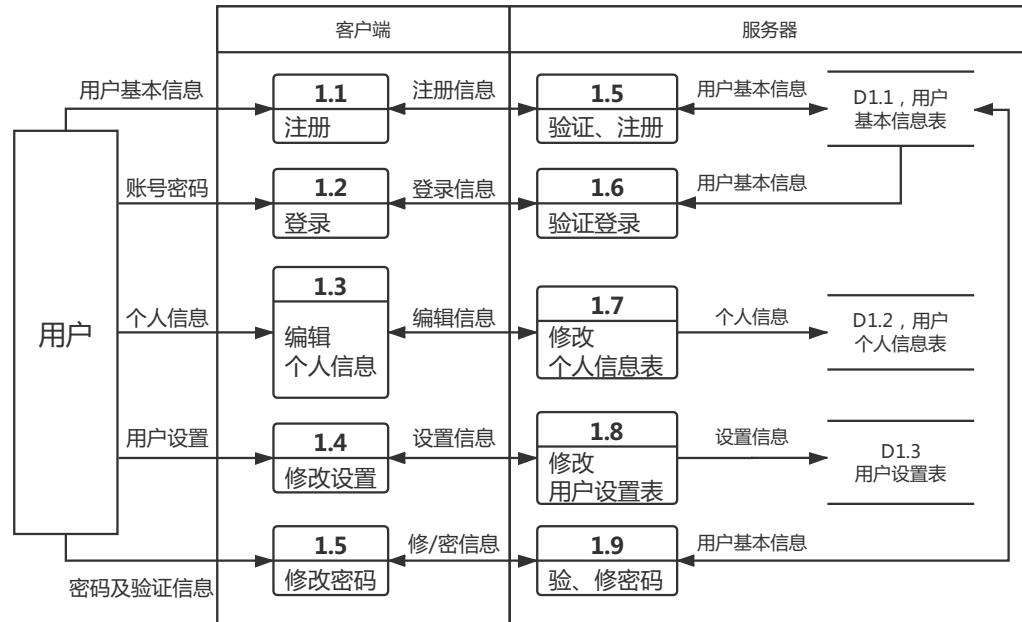


图 3-11 用户输入数据流图

客户端和服务器之间所传输的数据流不仅包括用户输入的信息，还包括系统控制信息，如错误代码。

验证登录过程是较为重要的过程，涉及到数据安全，这里进行进一步分解，如图 3-12 所示。

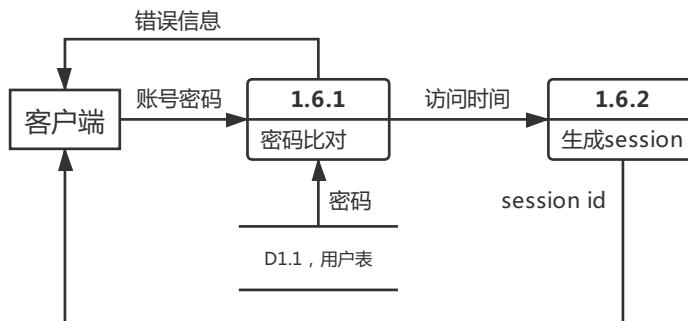


图 3-12 验证登录详细数据流图

系统应用 Session 机制保存 https 链接状态。

(2) 内容处理

这个处理可以分解为新建内容、更新内容、删除内容、评论内容（评论包括发表文

字评论以及点赞）四个子处理。以用户为数据源点，主要信息流包括内容信息、更新信息、删除、评论信息以及内容编号。其中评论信息可细分为文字评论喝点赞评论，而内容信息也可以进一步细分为基本内容信息和子任务信息，如图 3-13 所示。

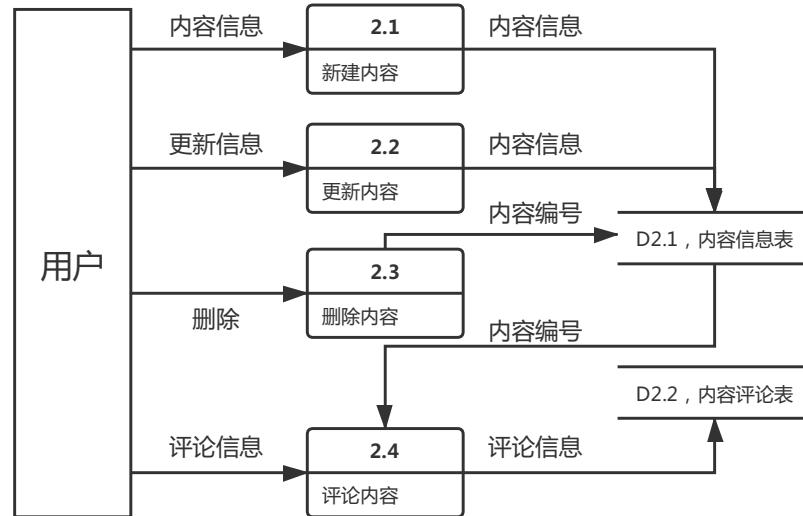


图 3-13 内容处理数据流图

2.1-2.4 这四个处理可以进一步进行求精，这里重点给出更新内容、评论内容两个处理的详细数据流图。

① 新建内容

创建内容不仅需要用户输入的内容信息，还需要用户 id 以标记用户的创建者，以及位置信息以记录创建内容时用户所在的场景，数据流图如图 3-14 所示。

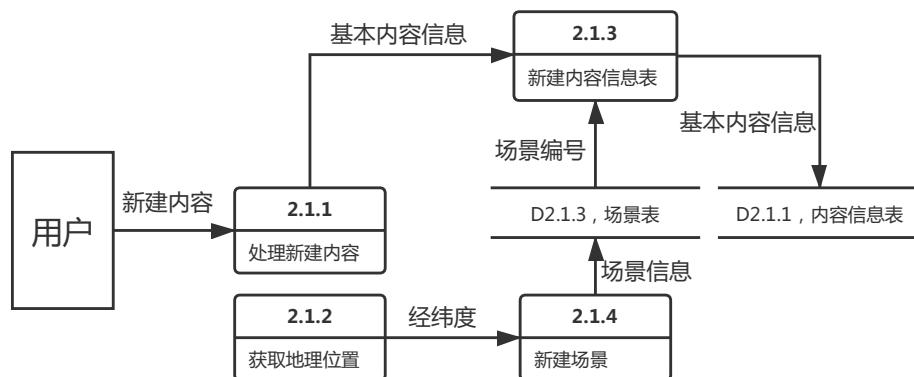


图 3-14 新建内容数据流图

图中将 D2.1 内容信息表进一步细化，分解为 D2.1.1 内容信息表与 D2.1.3 场景表，场景表只存储场景的经纬度和场景编号，通过记录场景编号的方式可以使多个内容共用一个场景，以实现多条内容在 AR 场景中同时显示的功能。

获取地理位置这一处理无需用户提供额外信息，直接通过调用 iOS 系统接口完成，具体算法和实现将在模块算法和系统实现中进一步阐述。

D2.1.2 为子任务表，存储短期计划类内容中的子任务。

② 更新内容

这一处理可以分解为客户端处理、服务端处理层次，在服务器端，又细分为更新基本内容和新建子任务两个子处理，如图 3-15 所示。

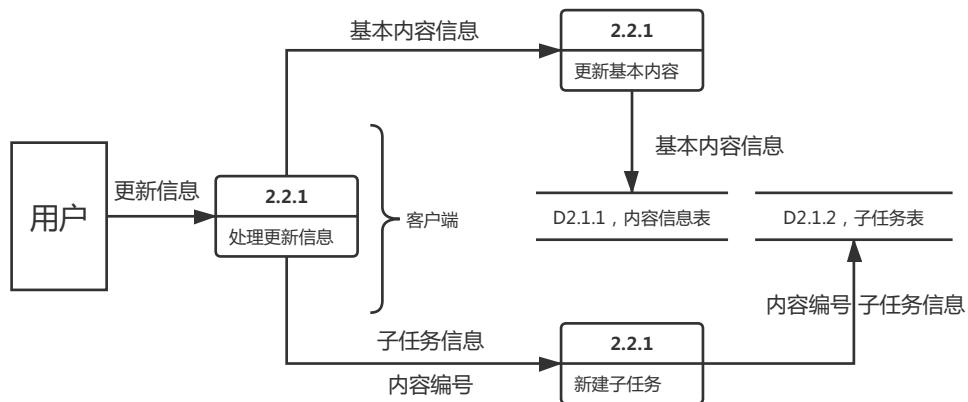


图 3-15 更新内容数据流图

③ 评论内容

这一处理涉及到用户和内容两个实体，又与 Session 机制有关，较为复杂，如图 3-16 所示。

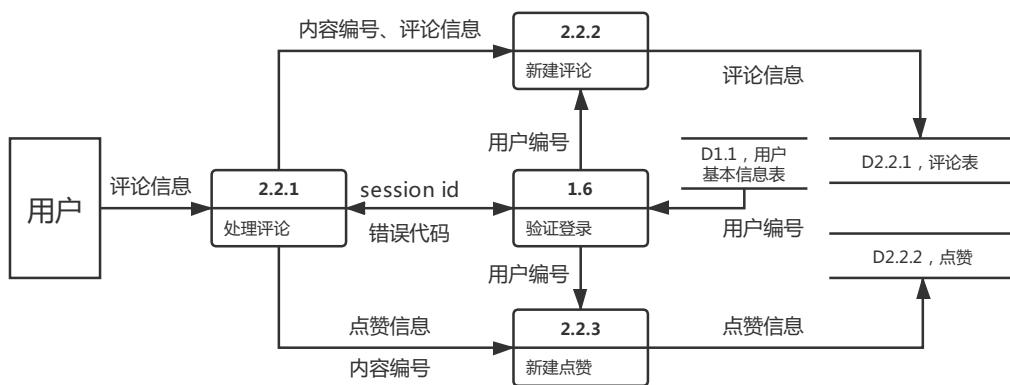


图 3-16 评论数据流图

图中验证登录处理从客户端获得 Session id，以访问 D1.1，若 Session id 无效，则返回错误代码，重新登录。

(3) 输出

处理输出也需要从客户端和服务端两个层面来考虑，客户端从服务器获取结构化数据，而后在相关页面上予以显示，包括个人信息、用户设置状态、内容列表、内容详情、

内容评论、AR 场景，而服务器则接收客户端请求，给出相应数据，包括返回用户信息、返回内容信息、返回评论信息。

数据流图如图 3-17 所示。

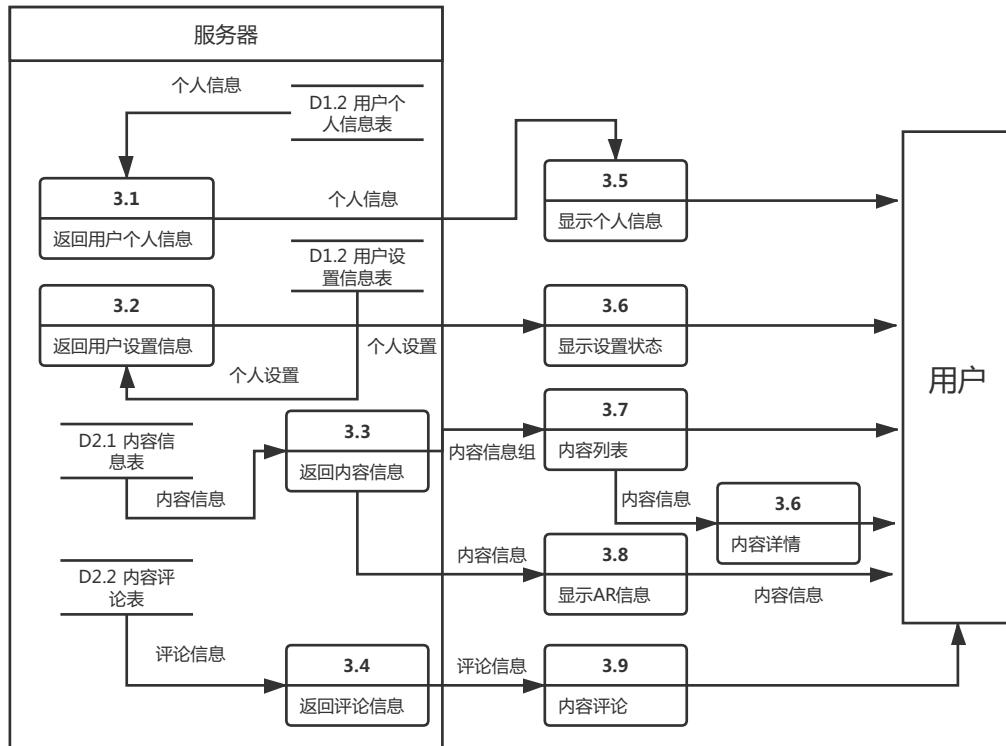


图 3-17 输出数据流图

其中显示 AR 信息处理是客户端功能的重要部分，也是整个系统的重要部分，需要进一步分解，数据流图如图 3-18 所示。

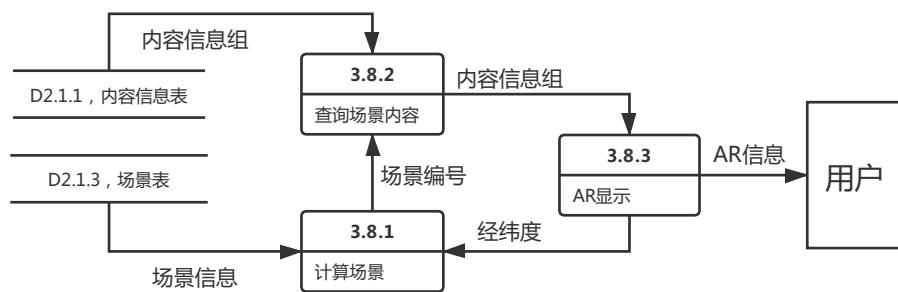


图 3-18 AR 显示数据流图

AR 显示与计算场景、查询场景内容这三个似乎形成循环结构，但需要注意的是这里只是给出所有可能的数据流向，实际的流程算法将在系统流程分析中给出。

3.4 状态转换图

状态转换图可以指明发生特定事件后系统将作出哪些动作，运用状态转换图描绘系

统的状态及引起系统状态转换的事件，可以对系统的行为进行建模^[13]，为详细设计和实现夯实基础，如图 3-19 所示。

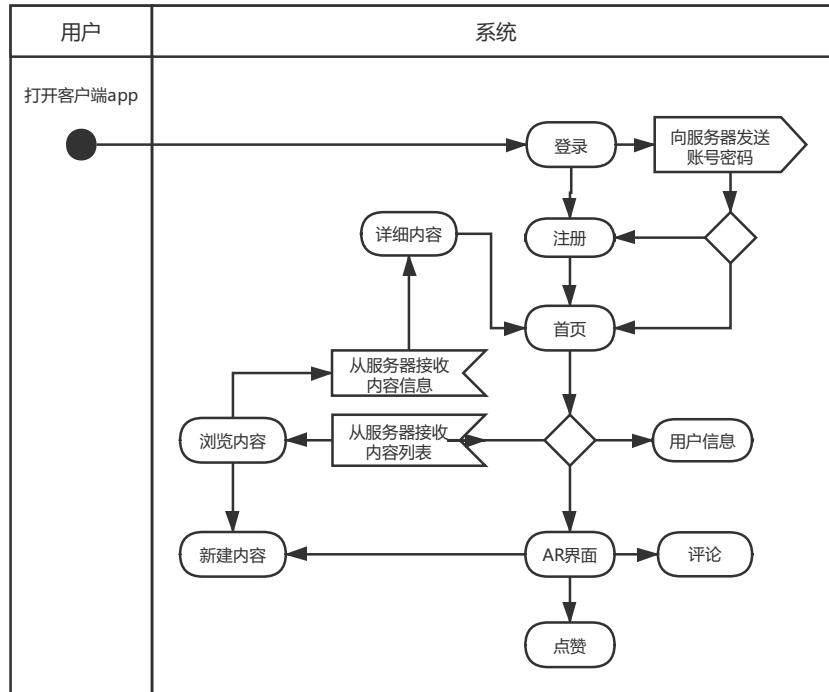


图 3-19 系统状态转换图

用户进入系统之后，首先需要登录以获取数据访问权限，若无账号，则需要进行注册，注册成功之后，进入首页。通过首页可以进入三个新状态：查看用户信息、浏览内容列表或进入 AR 界面。

转换图中，服务器被抽象为一个数据容器，在需要的时候接收、发送数据，这些活动对用户来说是完全透明的。

第四章 应用总体设计

本系统主要由服务器和客户端两大子系统构成，服务器负责数据存取与计算，客户端负责提供人机交互接口，包括 AR 交互界面。本章以前文给出的需求分析为依据，首先分析系统构成，绘制状态转换图，给出物理设计方案，接着面向数据流进行总体设计，给出系统结构，将系统划分为多个模块，针对各个模块进行精细化求解，最后给出模块接口。

4.1 系统功能概述

根据需求分析对系统的各个功能模块进行描述，并给出用例图。

用例图是指由参与者（Actor）、用例（Use Case），边界以及它们之间的关系构成的用于描述系统功能的视图^[14]。用例图（User Case）是外部用户（被称为参与者）所能观察到的系统功能的模型图。

客户端主要功能包括：

- (1) 为用户注册、登录、个人信息查看与修改提供人机交互界面；
- (2) 为密码修改提供成体系的人机交互页面，包括申请按钮、验证页面与重设页面；
- (3) 为用户新建、修改、查看、删除内容提供人机交互页面；
- (4) 构建 AR 场景，并为用户提供内容创建、交互接口；交互包括评论、点赞。

创建内容页面应该是富文本的，因为短期计划作为比较重要的内容类型有子任务等非文本内容。创建内容界面的入口应该有两个，既可以在内容列表界面创建新内容，也可以在 AR 场景界面创建新内容。查看内容时给出已发布内容的列表，并根据内容更新最新时间进行排序。

客户端程序用例图如图 4-1 所示。

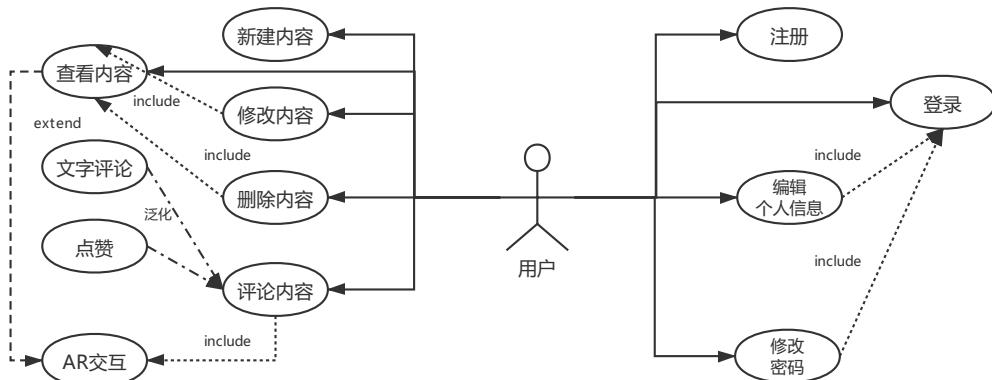


图 4-1 客户端程序用例图

服务器主要功能包括：

- (1) 存储并管理用户信息，应答客户端发出的注册、登录、修改、查询请求；在保证信息安全的情况下，应答客户端发出的密码修改请求；密码修改需要通过邮箱验证，依次响应密码修改申请、用户身份验证、修改密码三个请求；
- (2) 存储并管理用户发布的内容，应答客户端发出的发布、修改、查询、删除请求；针对某一具体内容，存储并管理其它用户对此内容的评论、点赞信息，应答客户端发出的评论内容查询请求，返回评论列表；
- (3) 应答 AR 场景中的交互操作，包括：
 - 用户与 AR 场景的交互：打开 AR 界面时响应客户端发出的内容查询请求，返回根据热度排序之后的内容列表；
 - 用户与 AR 场景中的具体内容的交互：用户评论或点赞某个内容时，响应从客户端发来的评论、点赞创建请求，创建新的评论、点赞信息，存储并更新热度。热度更新之后，根据新的顺序重新显示 AR 内容。
- (4) 根据用户当前所处场景与通知偏好即使向用户推送相关内容；通知偏好由用户通过客户端接口自行设定，推送结果应当是一个或一组结构化的内容。

服务器用例图如图 4-2 所示。

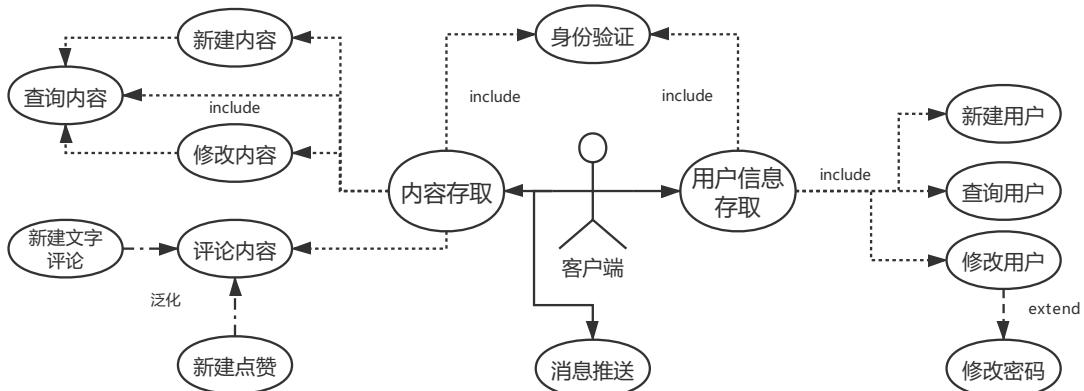


图 4-2 服务器程序用例图

4.2 系统结构

通过对数据流图的逐步求精，依据软件设计常用的启发式规则设计系统模块，给出系统层次图。本系统可以简单地分为客户端与服务器两个子系统，两者之间通过网络接口进行数据传输。

4.2.1 客户端

客户端由以下四个主要模块构成。

- (1) 用户管理模块：与服务器端各子模块相对应，分为注册、登录、用户信息查询与设置、密码修改等 5 个子模块，为相应操作提供人机交互界面；
- (2) 内容管理模块：与服务器端各子模块相对应，为内容的创建、查询、更新与删除提供人机交互界面；
- (3) AR 场景模块：绘制 AR 虚拟实体，显示场景中的内容，并提供评论、点赞、创建新内容等人机交互接口；
- (4) 出错处理模块：处理从服务器接收到的错误代码，在程序崩溃时记录上下文及错误信息。

客户端的每个模块都有相应人机交互界面，各页面所见即所得，管理模块即包含了相关信息的输出，结构层次如图 4-3 所示。



图 4-3 客户端层次结构图

出错处理模块是基础功能模块，其它三个模块通过错误信息与之耦合。接收到相关错误代码（网络错误）或错误信息（其它错误）之后，该模块将通过浮动窗口与用户交互，说明错误原因。若程序直接崩溃，则保存现场信息并在用户的同意下上传至服务器。

4.2.2 服务器

服务器端由用户管理、内容管理、场景计算、Session 对象管理、推送五个模块构成，各模块相互独立，通过数据耦合。Session 对象存储特定用户会话所需的属性及配置信息，以保存服务器与客户端之间的链接状态^[15]。

- (1) 用户管理，包含 5 个子模块，分别是：
 - ① 注册：根据客户端发送的邮箱和密码注册账号，创建用户表和用户信息表；
 - ② 登录：验证客户端发送的邮箱和密码对，返回登录结果，登录成功发挥 Session id，登录失败则返回错误信号；
 - ③ 修改个人信息：通过登录验证后，修改个人信息；

- ④ 用户设置：通过登录验证后，修改用户设置，如推送设置；
- ⑤ 修改密码：根据客户端发送的邮箱号经验证后修改用户密码；
- ⑥ 用户信息查询：根据客户端发送的 Session id 查询并返回用户信息。
- (2) 内容管理，包含 4 个子模块，分别是：
 - ① 新建内容：接收创建者的 Session id 和内容详细信息，创建内容数据实体；
 - ② 更新内容：根据客户端提交的内容 id 及其它信息更新内容；
 - ③ 删除内容：查询内容 id 所在表，删除整行；
 - ④ 评论内容：包含文字评论和点赞两个模块。根据评论者账号和内容 id 新建 comment 表或修改 like 表属性；
- ⑤ 查询内容：接收客户端所在场景 id，查询当前场景中的所有内容，返回内容列表，或只接收明确的内容 id，返回内容详细信息；
- (3) 场景计算：根据场景 id 从数据库查询内容列表，并根据需要对内容列表进行排序，如根据热度排序，根据发布时间排序。这一模块不对外开放接口，由推送模块或查询模块调用；
- (4) Session 管理：生成 Session，配合客户端 cookie 保存链接状态；
- (5) 推送：根据用户设置和内容更新信息向用户发送推送消息。

服务器层次图如图 4-4 所示。

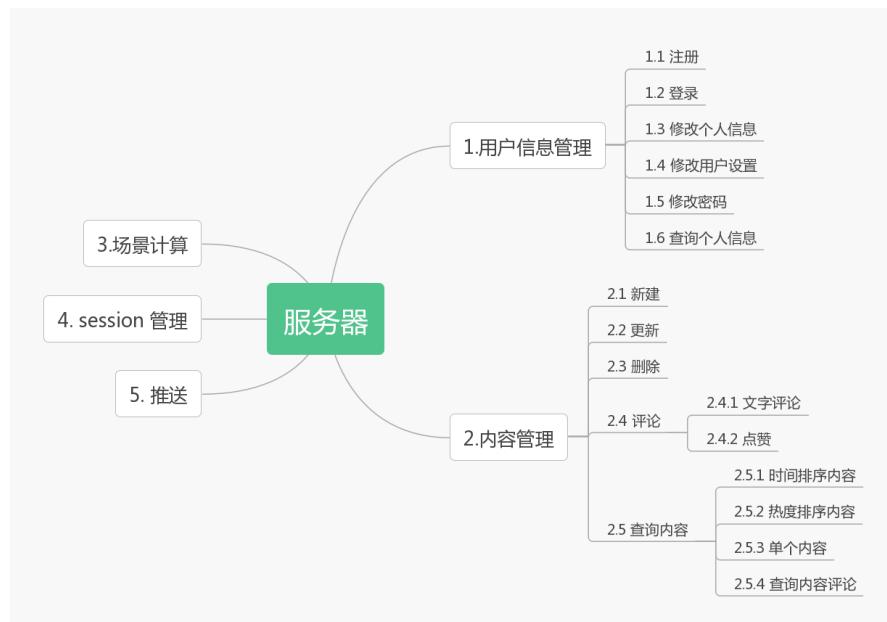


图 4-4 服务器测层次结构图

与客户端产生数据传输的主要是用户信息管理模块和内容管理模块，这两个模块每一个子模块都对应着一个网络接口函数。

4.3 接口设计

服务器接口是沟通客户端和服务器的桥梁，以 restful 设计规范为设计原则，分为用户管理和内容管理两个模块。

4.3.1 用户管理

用户管理模块的各个接口处理用户信息，包括注册、登录、个人信息管理、用户设置、密码修改五个子模块。

为了在保证用户数据安全的同时，保存链接状态，方便客户端访问服务器数据，服务器应用了 Session 机制。Session 管理模块与用户管理模块通过 Session id 耦合。

(1) 注册

接口说明如表 4-1 所示。

表 4-1 注册接口

接口名称	com.mbt.eyes.people
接口地址	https://www.mbt.com/eyes/v1/people
接口协议	Restful, HTTPS

这一接口包含两个方法：

① isNew

方法描述：该方法验证客户端上传的邮箱号是否已注册。

接口地址：/new/:email

动作类型：GET

请求参数：如表 4-2 所示。

表 4-2 isNew 请求参数

参数名	参数类型	含义	备注
email	string	邮箱号	URL 参数，必须

响应参数：如表 4-3 所示。

表 4-3 isNew 响应参数

响应类型	响应描述
string	说明验证结果

若传入的请求参数 email 是从未注册过的新邮箱号，就返回标准响应码 200，表示允许用户继续注册，否则，返回响应码 403，禁止用户使用这一邮箱注册账号。

② signUp

方法描述：注册新用户。以 email、password 以及自动生成的 user_id 构成新的用户基本信息对象(User)，插入用户基本信息表(user)中，并根据 user_id 生成新的用户个人信

息对象(Profile)，插入用户个人信息表(profile)中。新的 Profile 对象如图 3-2 所示，除了用户账户由 user_id 赋值，其它各个属性在新建时皆为空。

接口地址：/signup

动作类型：POST

请求参数：如表 4-4 所示。

表 4-4 signUp 请求参数

参数名	参数类型	含义	备注
email	string	邮箱号	表单参数，必须
password	string	密码	表单参数，必须

响应参数：如表 4-5 所示。

表 4-5 signUp 响应参数

响应类型	响应描述
User	注册成功后返回用户基本信息，JSON 格式
String	注册失败后提醒用户稍后再试

User 类型是用户自建的结构体，详细信息见图 3-1。

(2) 登录

登录接口从客户端接收邮箱号与密码，查询数据库，批判成功后记录链接状态，生成 Session 并返回 Session id，接口说明如表 4-6 所示。

表 4-6 登录接口

接口名称	com.mbt.eyes.signin
接口地址	https://www.mbt.com/eyes/v1/signin
接口协议	Restful, HTTPs

这个接口只有一个 signIn 方法：

方法描述：从 form 表单中获得邮箱号和密码，创建用户基本信息对象并插入到数据库中，返回该对象的 JSON 字符串。登录之后，应用 Session 机制生成 Session 对象，并保存用户基本信息，生成唯一的 Session id，返回客户端，保存登录状态。

动作类型：POST

请求参数：如表 4-7 所示。

表 4-7 signIn 请求参数

参数名	参数类型	含义	备注
email	string	邮箱号	URL 参数，必须
password	string	密码	form 表单参数，必须

响应参数：如表 4-8 所示。

表 4-8 signIn 响应参数

响应类型	响应描述
User	登录成功后返回用户基本信息, JSON 格式
String	登录失败后返回错误信息

(3) 个人信息管理

这个接口的主要功能是在客户端和服务器数据库之间传送用户个人信息，包含两个方法，查询及设置。

接口说明如表 4-9 所示。

表 4-9 个人信息管理接口

接口名称	com.mbt.eyes.user
接口地址	https://www.mbt.com/eyes/v1/user
接口协议	Restful, HTTPs

① 查询 userProfile 方法

方法描述：从服务器获取用户个人信息，JSON 格式

接口地址：/:user_id

动作类型：GET

请求参数：如表 4-10 所示。

表 4-10 userProfile 请求参数

参数名	参数类型	含义	备注
user_id	string	账号	URL 参数, 必须
Session_id	string	Session id	form 表单参数, 必须

通过 Session_id 查询连接状态，若未登录，则返回错误提示。确认已经登录之后，通过 user_id 查询个人信息并返回 Profile 对象。

响应参数：如表 4-11 所示。

表 4-11 userProfile 响应参数

响应类型	响应描述
profile	查询成功后返回个人信息, JSON 格式
String	返回错误信息

② 编辑 editUser 方法

方法描述：根据客户端上传的用户账号 user_id 查询用户个人信息，进行编辑，即更新 profile 表。

接口地址：/profile

动作类型：POST

请求参数：如表 4-12 所示。

表 4-12 editUser 请求参数

参数名	参数类型	含义	备注
user_id	string	账号	form 表单参数, 必须
Session_id	string	Session id	form 表单参数, 必须
name	string	名字	form 表单参数
age	int	年龄	form 表单参数
sex	bool	性别	form 表单参数
saying	string	座右铭	form 表单参数
photo	byte	头像	form 表单参数

响应参数：如表 4-13 所示。

表 4-13 editUser 响应参数

响应类型	响应描述
profile	编辑成功后返回个人信息, JSON 格式
String	返回错误信息

editUser 方法所提交的 Post Form 参数中，除了 user_id 和 Session_id 为必须参数，其它参数均为可选参数，但至少提交一个可选参数，否则返回输入错误。

(4) 用户设置

用户设置接口的功能是在客户端和服务器之间传送设置信息，接口说明如表 4-14 所示。

表 4-14 用户设置接口

接口名称	com.mbt.eyes.user.setting
接口地址	https://www.mbt.com/eyes/v1/user/setting
接口协议	Restful, HTTPs

该接口只有一个方法，setUser。

方法说明：编辑用户设置，即更新 setting 表。

动作类型：POST

请求参数：如表 4-15 所示。

表 4-15 setUser 请求参数

参数名	参数类型	含义	备注
user_id	string	账号	form 表单参数, 必须
Session_id	string	Session id	form 表单参数, 必须
push	bool	推送设置	form 表单参数, 必须

响应参数：4-16 所示。

表 4-16 setUser 响应参数

响应类型	响应描述
setting	编辑成功后返回用户设置信息, JSON 格式
String	返回错误信息

(5) 密码修改

当用户需要修改密码时，客户端首先要对用户真实身份进行验证，因此密码修改接口包含询问、验证、修改三个方法，接口说明如表 4-17 所示。

表 4-17 密码修改接口

接口名称	com.mbt.eyes.user.password
接口地址	https://www.mbt.com/eyes/v1/ /user/password
接口协议	Restful, HTTPs

① 询问 resetAsk 方法

方法描述：向服务器申请重设密码。

接口地址：/:user_id

动作类型：GET

请求参数：如表 4-18 所示。

表 4-18 resetAsk 请求参数

参数名	参数类型	含义	备注
user_id	string	账号	URL 参数，必须
Session_id	string	Session id	form 表单参数，必须

通过 Session_id 查询连接状态，若未登录，则返回错误提示。确认已经登录之后，

通过 user_id 查询用户邮箱账号，向该邮箱发送验证码，进入验证阶段。

响应参数：如表 4-19 所示。

表 4-19 resetAsk 响应参数

响应类型	响应描述
String	验证码发送之后，提醒用户及时登录邮箱查看，完成验证。
String	返回错误信息

② 邮箱验证 resetVerify 方法

方法描述：提交用户从邮箱获得的验证码，与 resetAsk 方法生成的验证码相比对，若匹配成功则进入密码修改阶段，返回 200，否则返回 404。

接口地址：/verification

动作类型：POST

请求参数：如表 4-20 所示。

表 4-20 resetVerify 请求参数

参数名	参数类型	含义	备注
user_id	string	账号	form 表单参数，必须
Session_id	string	Session id	form 表单参数，必须
verify_code	string	名字	form 表单参数，必须

响应参数：如表 4-21 所示。

表 4-21 resetVerify 响应参数

响应类型	响应描述
bool	匹配成功之后返回 true，否则返回 false
String	返回错误代码

用户身份验证成功之后，返回“true”，进入密码修改阶段。

③ 修改 passwordReset 方法

方法描述：从客户端接收用户账户、Session id 与新密码，验证链接状态之后修改用户基本信息表，更新密码。

接口地址：

动作类型：PUT

请求参数：如表 4-22 所示。

表 4-22 passwordReset 请求参数

参数名	参数类型	含义	备注
user_id	string	账号	form 表单参数，必须
Session_id	string	Session id	form 表单参数，必须
password	string	密码	form 表单参数，必须

响应参数：如表 4-23 所示。

表 4-23 passwordReset 响应参数

响应类型	响应描述
User	密码修改成功后返回用户基本信息表，JSON 格式
String	返回错误代码

密码修改成功后，返回用户基本信息，其中密码只存储哈希值，以保障数据安全。

4.3.2 内容管理

内容管理模块的各个接口处理内容信息，包括创建内容、更新内容、删除内容、查询内容、评论内容五个子模块。

(1) 创建内容

创建内容模块的功能是从客户端接收内容信息并创建新的内容实例，插入内容表(event)，接口说明如表 4-24 所示。

表 4-24 创建内容接口

接口名称	com.mbt.eyes.event.creation
接口地址	https://www.mbt.com/eyes/v1/event/creation
接口协议	Restful, HTTPs

由于内容数据中包含场景信息，因此在创建新内容时需要查询并根据需要创建或关

联场景实体。这个接口只有一个 `createEvent` 方法。

方法描述：从 `form` 表单中获得内容信息表各个数据项，创建内容对象并插入到数据库中，返回该对象的 JSON 字符串。

动作类型：POST

请求参数：如表 4-25 所示。

表 4-25 signIn 请求参数

参数名	参数类型	含义	备注
<code>creator_id</code>	string	创建者账号	form 表单参数，必须
<code>lat</code>	string	场景经度	form 表单参数，必须
<code>lng</code>	string	场景纬度	form 表单参数，必须
<code>type</code>	uint	类型	form 表单参数，必须
<code>title</code>	string	标题	form 表单参数，必须
<code>summary</code>	string	综述	form 表单参数，必须
<code>progress</code>	float	进度	form 表单参数，必须
<code>pic</code>	binday data	缩略图	form 表单参数，必须

响应参数：如表 4-26 所示。

表 4-26 signIn 响应参数

响应类型	响应描述
<code>Event</code>	创建成功后返回内容对象，JSON 格式
<code>String</code>	创建失败后返回错误信息

创建内容的过程中，`form` 表单中提交的经度和纬度信息将与 `location` 表中所有场景的经纬度进行比对，若该经纬度落入已存在场景的范围，则内容对象的 `location_id` 属性值即是查询到的已有场景的 `id`，否则，新建场景并将 `id` 赋值给内容对象的 `location_id`。

(2) 更新内容

更新内容模块的功能是对已有模块的数据进行修改，接口说明如表 4-27 所示。

表 4-27 更新内容接口

接口名称	com.mbt.eyes.event
接口地址	https://www.mbt.com/eyes/v1/event
接口协议	Restful, HTTPs

这一接口包含三个方法：

① updateEvent

方法描述：该方法验证客户端上传的邮箱号是否已注册。

接口地址：/updation

动作类型：PUT

请求参数：如表 4-28 所示。

表 4-28 updateEvent 请求参数

参数名	参数类型	含义	备注
event_id	string	内容编号	form 表单参数，必须
Session_id	string	Session id	form 表单参数，必须
title	string	标题	form 表单参数
summary	string	综述	form 表单参数
pic	binday data	缩略图	form 表单参数

响应参数：如表 4-29 所示。

表 4-29 updateEvent 响应参数

响应类型	响应描述
Event	更新成功后，返回内容对象，JSON 格式
String	返回错误代码

注意直接更新方法的请求参数后三项是可选的，但是必须有一个。直接更新不能修改类型 type、进度 process 和场景编号 location_id，类型和场景一旦创建就不能更改，而进度是短期计划类型内容的一个派生属性，需要根据子任务的完成程度计算。

② addStep

方法描述：为短期计划类型的内容添加子任务。

接口地址：/step

动作类型：POST

请求参数：如表 4-30 所示。

表 4-30 addStep 请求参数

参数名	参数类型	含义	备注
event_id	string	内容编号	form 表单参数，必须
content	string	子任务描述	form 表单参数，必须
completed	bool	是否完成	form 表单参数，必须

响应参数：如表 4-31 所示。

表 4-31 addStep 响应参数

响应类型	响应描述
event	子任务创建成功后返回内容对象，JSON 格式
String	返回错误代码

Event 对象包含了子任务信息。

③ updateStep

方法描述：更新某个短期计划的子任务信息。

接口地址：/step/updation

动作类型：PUT

请求参数：如表 4-32 所示。

表 4-32 updateStep 请求参数

参数名	参数类型	含义	备注
event_id	uint	内容编号	form 表单参数，必须
id	uint	子任务编号	form 表单参数，必须
content	string	子任务描述	form 表单参数，必须
completed	bool	是否完成	form 表单参数，必须

响应参数：如表 4-33 所示。

表 4-33 updateStep 响应参数

响应类型	响应描述
event	子任务更新成功后返回内容对象，JSON 格式
String	返回错误代码

Event 对象包含了各子任务信息。

(3) 删除内容

从数据库删除内容对象，并删除相应的子任务、评论、点赞对象。

event_id 在 eventStep、comment 和 like 表中都被设置为外键，并在数据库创建时应用级联删除，因此内容对象被删除时相关信息可以一起被删除。

接口说明如表 4-34 所示。

表 4-34 删除内容接口

接口名称	com.mbt.eyes.event.deletion
接口地址	https://www.mbt.com/eyes/v1/event/deletion
接口协议	Restful, HTTPs

这个接口只有一个 deleteEvent 方法。

方法描述：从 form 表单中获取内容编号和用户账号，删除内容对象。删除操作需要验证用户权限，若验证失败权限不足，返回 503 错误。

动作类型：DELETE

请求参数：如表 4-35 所示。

表 4-35 deleteEvent 请求参数

参数名	参数类型	含义	备注
event_id	uint	内容编号	form 表单参数，必须
user_id	uint	用户账号	form 表单参数，必须
Session_id	string	Session id	form 表单参数，必须

根据 Session_id 可获得 Session 对象，其中保存的用户信息是身份验证的凭证。

响应参数：如表 4-36 所示。

表 4-36 deleteEvent 响应参数

响应类型	响应描述
String	返回响应码

成功删除则返回响应码 200，否则返回响应错误码。

(4) 查询内容

查询内容模块的功能是响应各种查询请求，返回内容信息，包括查询某一用户所有内容、查询某一场景所有内容、根据内容编号查询单个内容。若查询得到的内容对象不止一个，则返回 JSON 数组，若查询到的结果为空，则返回 null 字符串。接口说明如表 4-37 所示。

表 4-37 查询内容接口

接口名称	com.mbt.eyes.event
接口地址	https://www.mbt.com/eyes/v1/event
接口协议	Restful, HTTPs

这一接口包含三个方法：

① fetchAllEvents

方法描述：通过场景 id 查询数据库获取所有内容对象，返回对象数组。

接口地址：/all/:loc_id

动作类型：GET

请求参数：如表 4-38 所示。

表 4-38 fetchAllEvents 请求参数

参数名	参数类型	含义	备注
loc_id	string	内容编号	URL 参数，必须
Session_id	string	Session id	form 表单参数，必须

响应参数：如表 4-39 所示。

表 4-39 fetchAllEvents 响应参数

响应类型	响应描述
Event 数组	查询成功后，返回内容对象数组，JSON 格式
String	返回错误代码

② fetchSingleEvent

方法描述：为短期计划类型的内容添加子任务。

接口地址：/one/:event_id

动作类型：GET

请求参数：如表 4-40 所示。

表 4-40 fetchSingleEvent 请求参数

参数名	参数类型	含义	备注
event_id	uint	内容编号	URL 参数，必须
Session_id	string	Session id	form 表单参数，必须

响应参数：如表 4-41 所示。

表 4-41 fetchSingleEvent 响应参数

响应类型	响应描述
Event	返回内容对象，JSON 格式
String	返回错误代码

(3) myEvent

方法描述：返回当前登录用户的所有内容对象。

接口地址：/mine

动作类型：GET

请求参数：如表 4-42 所示。

表 4-42 myEvent 请求参数

参数名	参数类型	含义	备注
user_id	uint	用户账号	form 表单参数，必须
Session id	string	Session id	form 表单参数，必须

响应参数：如表 4-43 所示。

表 4-43 myEvent 响应参数

响应类型	响应描述
Event 数组	返回内容对象数组，JSON 格式
String	返回错误代码

若该用户未创建内容，则返回空数组。

(5) 评论内容

内容评论模块响应两种评论请求，接口说明如表 4-44 所示。

表 4-44 评论内容接口

接口名称	com.mbt.eyes.event
接口地址	https://www.mbt.com/eyes/v1/event
接口协议	Restful, HTTPs

这一接口包含两个方法：

(1) likeEvent

方法描述：根据提交的用户账号和内容编号，创建或修改 Like 对象。验证用户账户与内容编号存在性的同时，还需要用 Session id 验证链接状态，取得存取权限。针对同一组用户账户和内容编号，只能创建一个 Like 对象。

接口地址：/like/:user_id/:event_id

动作类型：PUT

请求参数：如表 4-45 所示。

表 4-45 likeEvent 请求参数

参数名	参数类型	含义	备注
user_id	uint	用户账号	URL 参数，必须
event_id	uint	内容编号	URL 参数，必须
Session_id	string	Session id	form 表单参数，必须

响应参数：如表 4-46 所示。

表 4-46 likeEvent 响应参数

响应类型	响应描述
Like	操作成功后，返回 Like 对象
String	返回错误代码

若该用户已经对该内容进行过点赞操作，则修改已存在的 Like 对象，否则新建 Like 对象。

② commentEvent

方法描述：根据提交的用户账号和内容编号创建文字评论对象 Comment 并插入数据库中。

接口地址：/comment/:user_id/:event_id

动作类型：POST

请求参数：如表 4-47 所示。

表 4-47 commentEvent 请求参数

参数名	参数类型	含义	备注
user_id	uint	用户账号	URL 参数，必须
event_id	uint	内容编号	URL 参数，必须
Session_id	string	Session id	form 表单参数，必须
content	string	评论文字	form 表单参数，必须

响应参数：如表 4-48 所示。

表 4-48 commentEvent 响应参数

响应类型	响应描述
Comment	返回评论对象，JSON 格式
String	返回错误代码

评论接口是用户管理模块与内容管理模块的耦合接口，在 Session 对象的辅助下，可以通过传递用户账号(user_id)和内容编号(event_id)达到数据耦合。

数据耦合指两个模块在相互调用时只传递简单的数据值。数据耦合联系简单，耦合度低，模块独立性好，模块间的影响最小，是最理想的一种耦合形式^[16]。

4.4 测试环境

- 服务器: Ubuntu Server 18.04 LTS
- 客户端: iOS12.2
- 编程软件: Visual Studio Code/XCode
- 数据库管理系统: Mysql
- 客户端测试机: iPhone 7

第五章 客户端设计与实现

在第四章中，已经将客户端各功能模块进行了划分，接下来将结合项目文件对服务器系统中各个模块的实现进行详细描述。

5.1 项目目录简介

客户端开发环境为 Xcode 10.2，项目目录如图 5-1 所示。

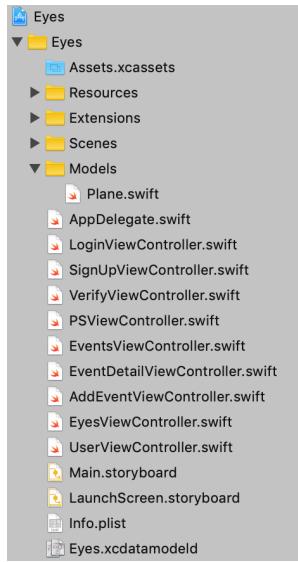


图 5-1 客户端开发目录

其中 Eyes 为工程总目录，下文将对各子目录详细内容进行详细阐述。

5.1.1 Assets.xcassets

图像资源文件管理文件夹，存储项目中用到的所有静态图像文件，并且能够为屏幕分辨率的不同多个设备设置多套图片资源；项目开发过程中用到的图像资源有应用 icon，按钮图标，AR 材质，均存放在这一资源文件夹中。

5.1.2 Resources

存放项目颜色主题；为了方便管理颜色，在 Resources 新建一个 Colors.xcassers 文件，存放主题颜色，并通过 Themes.swift 文件定义为静态颜色常量，方便开发过程中调用。事先设定好颜色主题可以使客户端系统颜色风格趋于统一，避免驳杂。

5.1.3 Extensions

扩展文件夹，存储类扩展文件。开发过程中，需要对一些框架定义的类进行功能扩展，如对将按钮扩展为圆角按钮或悬浮按钮，这就需要定义扩展文件。已定义的扩展文件包括：

- (1) SCNVector3Extensions.swift, 扩展 SCNVector3 类; SCNVector3 类是 SceneKit 中用于描述三维向量的类, 而三维向量可以用来描述空间中的位置。这里的扩展文件中定义了两个函数, distance, 用于求两向量之间的几何距离, 以及和 positionFromTransform, 进行矩阵到向量的变换。
- (2) NodeExtensions.swift, 扩展 SCNNNode 类; SCNNNode 类是场景图(Scene Graph) 中的一种结构性元素, 表示 3D 坐标空间中的位置和变换, 开发者可以在 SCNNodes 上添加几何图形、灯光、相机或其它可显示内容。这里的扩展文件中定义了三个函数, allNodes, 获得场景中的所有节点, topmost, 获得树形结构中的根节点, centerPivot, 导航到场景中心。
- (3) UIButtonExtension.swift, 对 UIButton 类的扩展, 增加了设置圆角和更改为悬浮样式的函数。
- (4) UITextFieldExtension.swift, 扩展 UITextField 类, 增加了无边框衬线样式。

5.1.4 Scenes&Models

Scenes 场景文件夹, 存放定义 AR 场景的 HoverScene.swift 文件, 其详细内容将在 5.10 AR 交互一节进行阐述。

Models, AR 模型文件夹, 存放可以放置在 AR 场景中的模型文件。

5.1.5 其它文件

- (1) AppDelegate.swift, 应用代理文件, 由 IDE 自动生成, 负责管理页面的生命周期, 响应外部通知。
- (2) 多个 ViewController 文件, 控制自定义的页面, 详细内容将在各个模块中进行阐述。
- (3) Main.storyboard, 主要故事板; 故事板是 iOS5 推出的一个可视化开发系统, 方便开发者设计和组织页面; 所有模块的页面设计及导航逻辑都存放在主要故事板文件中, 详细内容将在各个模块中进行阐述。
- (4) LaunchScreen.storyboard, 应用启动页故事板。
- (5) Info.plist, 信息属性列表文件, 包含应用的基本配置信息, 如所需权限。
- (6) Eyes.xcdatamodeld, CoreData 模型文件, 定义数据各数据类以在本地存储相关信息。

以上就是基本的目录信息。客户端项目页面控制文件数量较多, 逻辑较为复杂, 而

iOS 开发遵循 MVC 模式，本身有很清晰的组织结构，因此本文也从从 Model, View, Control 三个方面入手，解析开发过程。

5.2 Model

顾名思义，模型(Model)就是数据模型。在客户端，数据模型的定义及数据持久化的实现是基于 CoreData 开发框架的。CoreData 可以让开发人员在可视化操作环境下定义数据模型，声明数据模型之间的关联^[17]。数据模型存储在 Eyes.xcdatamodeld 中。

5.2.1 用户

CoreData 运用了十分成熟的对象模型映射技术，将一个数据模型分解为属性(Attributes)和关系(Relationships)两个部分。用户数据模型由用户基本数据表 User 和个人信息表 Profile、用户设置表 setting 以及三者之间的关系共同定义。

User 表定义了用户基本信息，如图 5-2 所示。

The screenshot shows the Xcode Core Data Model Editor with the User entity selected. It is divided into two sections: 'Attributes' and 'Relationships'.

Attributes:

Attribute	Type
S password	String
S email	String
N id	Integer 32

Relationships:

Relationship	Destination	Inverse
M comments	Comment	◊ from_user ◊
M created_events	Event	◊ creator ◊
M likes	Like	◊ from_user ◊
O profile	Profile	◊ user ◊
O setting	Setting	◊ user ◊

图 5-2 用户基本数据表

Profile 表定义了用户个人信息，如图 5-3 所示。

The screenshot shows the Xcode Core Data Model Editor with the Profile entity selected. It is divided into two sections: 'Attributes' and 'Relationships'.

Attributes:

Attribute	Type
O photo	Binary Data
B sex	Boolean
S saying	String
S name	String
N id	Integer 32
N age	Integer 32

Relationships:

Relationship	Destination	Inverse
O user	User	◊ profile ◊

图 5-3 个人信息表

关系 user 定义了 Profile 与 User 的一对一关系，和 User 表中的 profile 互为反映。

Setting 表定义了用户设置信息，如图 5-4 所示。

▼ Attributes		
Attribute	Type	v
push	Boolean	◇
id	Integer 32	◇
+ -		

▼ Relationships		
Relationship	Destination	Inverse
user	User	◇ setting ◇

图 5-4 用户设置表

Setting 表中也有一个同名关系 user，与 User 表中的 setting 共同定义了 Setting 与 User 之间的一对一关系。

三者之间的关系如图 5-5 所示。

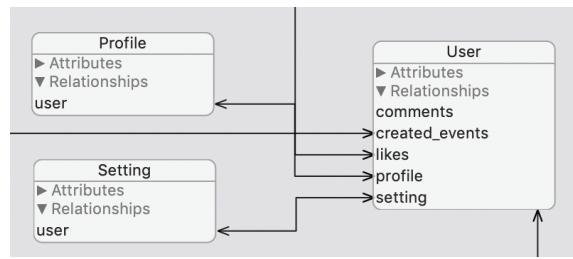


图 5-5 用户数据模型中的包含关系

据图可知实际上 User 是拥有(has one)Profile 和 Setting,之所以建立 Profile 和 Setting 是出于逻辑清晰和优化查询上的考虑，用户基本信息（密码、邮箱号、账号）查询频率较高，而个人信息和设置不然，因此要分离出来单独存储。

5.2.2 内容

内容数据模型由内容表 Event、子任务表 EventStep 与场景表 Location 共同定义。

Event 表是内容基本信息表，其定义如图 5-6 所示。

Attributes		
Attribute	Type	v
image	Binary Data	◇
update_time	Date	◇
start_time	Date	◇
end_time	Date	◇
title	String	◇
summary	String	◇
progress	Float	◇
location_id	Integer 32	◇
id	Integer 32	◇
creator_id	Integer 32	◇
type	Integer 16	◇
+ -		

Relationships		
Relationship	Destination	Inverse
M be_commented	Comment	◇ to_event ◇
M be_liked	Like	◇ to_event ◇
O creator	User	◇ created_events ◇
O location	Location	◇ events ◇
M steps	EventStep	◇ the_event ◇

图 5-6 内容表

由图可知，内容表与其它表有 5 个关系，其中 creator、be_commented、be_liked 是内容表和用户的联系的描述，在 5.2.3 有所阐述，而 location 则定义了 Event 与 Location 的多对一关系，steps 定义了 Event 和 EventStep 的一对多关系。

Location 是场景表，比较简单，存储了场景的地理位置，如图 5-7 所示。

▼ Attributes		
Attribute	Type	v
N lng	Double	◊
N lat	Double	◊
N id	Integer 32	◊
+ -		

▼ Relationships		
Relationship	Destination	Inverse
M events	Event	◊ location ◊

图 5-7 场景表

关系 events 定义了场景表与内容表的一对多关系，是关系 location 表的反射。

EventStep 是子任务表，存储计划类内容的子任务信息，如图 5-8 所示。

▼ Attributes		
Attribute	Type	v
B completed	Boolean	◊
S content	String	◊
N id	Integer 32	◊
N event_id	Integer 32	◊
+ -		

▼ Relationships		
Relationship	Destination	Inverse
O the_event	Event	◊ steps ◊

图 5-8 子任务表

关系 the_event 定义了 EventStep 与 Event 的多对一关系，Event、Location、EventStep 之间的关系如图 5-9 所示。

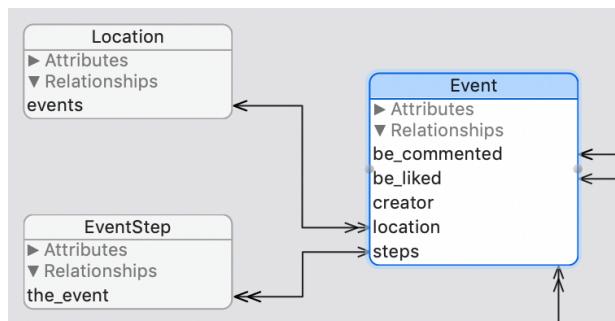


图 5-9 Event Location EventStep 关系图

通过 the_events 和 steps 关系对，可以很方便的从数据库中由内容 id 查询和创建子任务，同样，通过 events 和 location 关系对，可以轻松地访问同一场景下的所有内容。

5.2.3 用户与内容的关系

建立用户及内容的数据模型之后，就可以根据 E-R 图实现两者之间的联系。如图 3-1 所示，用户与内容之间的关系有三种，包括。

- (1) 创建，一个用户可以创建多个内容，一对多关系，由关系对 `created_events/creator` 定义。
- (2) 评论，一个用户可以针对多个内容发表评论，同样，一个内容可以受到多个用户的评论，多对多关系，由 `Comment` 表结合关系对 `comments/from_user/to_event/discussed` 定义。
- (3) 点赞，一个用户可以赞赏多个内容，一个内容也可以受到多个用户的赞赏，多对多关系，由 `Like` 表结合关系对 `likes/from_user/to_event/be_liked` 定义。

关系总图如图 5-10 所示。

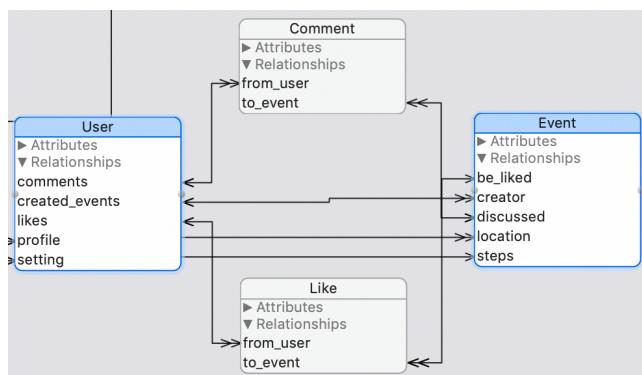


图 5-10 用户与内容关系图

这三个关系是实现内容创建、评论、点赞功能的数据基础。

在 CoreData 中定义各数据模型之后，相应的数据类就可以在各模块的 ViewController 文件中直接使用。

5.3 View, Controller

视图(View)和控制器(Controller)是 MVC 模式的另外两个重要组成部分。在 iOS 开发环境中，视图，也就是用户界面，可以通过可视化工具 storyboard 创建，而控制器则通过在 swift 文件中定义并扩展相应的 UIViewController 类来实现。由于视图与视图控制器(ViewController)关系密切，在这里不再分别论述，而是根据主要功能将各个页面及其控制器分为三个模块，即用户管理、AR 场景、内容管理三大模块。接下来，本文将首先对三大模块的页面组织及导航方式进行说明，而后对各个模块的页面设计与实现进行详细阐述。

5.3.1 页面导航

混乱的页面导航逻辑对客户端应用的使用者来说是一场灾难，同时也会增加开发者的工作量。幸运的是 iOS 开发规范中已经有了很成熟的页面组织模式，足以应对本系统开发过程中遇到的架构问题。

在 UIKit 开发库中，常用的导航控件有两种，Tab Bar Controller 和 Navigation Controller，前者多用于管理一组并行的视图，而后者则用于管理有递进关系的页面，方便前后跳转^[18]。

三大模块的管理应用了 Tab Bar Controller，用户管理模块和内容管理模块中的多个页面则有 Navigation Bar 进行组织，应用首页如图 5-11 所示。



图 5-11 首页导航

其底部的 Tab Bar 包含三个 Tab Bar Item，点击可以跳转至内容管理、AR 场景及用户信息。右下角的橙色按钮则是新建内容的接口。

5.3.2 用户管理

用户管理由用户登录、用户注册、用户信息编辑、用户设置四个模块组成。

(1) 登录

登录页面是未记录用户信息前打开 app 显示的第一个页面，如图 5-12 所示。



图 5-12 登录页面

登录页面为用户输入邮箱号和密码提供接口，同时提供注册和找回密码的按钮，如

图 5-13 所示。



图 5-13 跳转按钮

当用户登录失败或者尚未注册时，即可通过这两个接口进入其它模块。

用户输入邮箱和密码，点击登录按钮，就调用 `sign_in` 接口与服务器通信，登录成功之后，即获得 Session id，保持与服务器的连接状态，获得访问数据库信息的权限。

若用户未注册，便从登录页面进入注册页面。

(2) 注册

本系统以邮箱号为注册依据，为方便用户使用，不设置激活步骤，注册即可使用。

注册模块由三个页面组成，如图 5-14 所示。



图 5-14 注册页面设计

这三个页面是递进关系，用户首先在注册页面输入邮箱号，邮箱号必须是合法存在的，并且是未注册过的；检验符合要求之后，点击下一步即进入验证页面。

在验证页面，用户需要通过浏览器登录邮箱查看服务器发来的验证邮件，输入验证码，即可点击下一步，进行密码设置。

用户密码可以由数字、字母和特殊字符组成；为了保障信息安全，对密码明文进行哈希加密之后，通过注册接口传送给服务器，存储在数据库中。

注册成功之后，页面会自动跳转至登录页面，登录成功之后，即进入应用首页。

5.3.3 AR 场景

点击首页导航条中心的发现控件，即可进入 AR 场景，如图 5-15 所示。

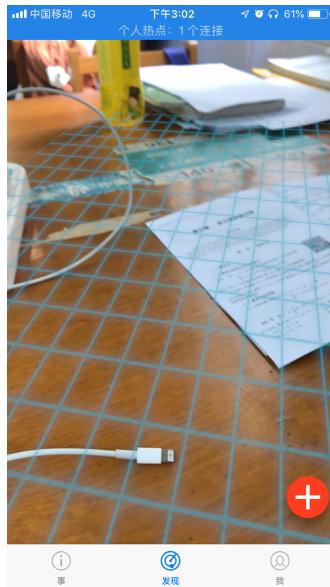


图 5-15 AR 场景初始化

点击蓝色网格标记的虚拟平面，AR 场景中就会按照热度顺序显示场景中已有的内容，如图 5-16 所示。



图 5-16 AR 交互页面

右下角也有橙色的悬浮按钮，点击即可进入内容新建页面。

5.3.4 内容管理

内容管理功能可以分解为以下几个模块：

(1) 内容列表

应用首页即是内容列表，显示用户关注的所有内容，按更新时间排序。

内容列表的每一行都代表了一个内容实体，显示的元素包括内容标题、创建者头像及昵称，内容综述、点赞数、评论数以及关注设置按钮，如图 5-17 所示。



图 5-17 内容列表

点击列表中的某一条内容，即可进入内容详情页，查看内容信息。若用户所在场景即内容发布场景，点击内容可直接进入 AR 界面。页面右下角的圆形按钮为新建按钮，点击之后即可进入新建内容页面。

(2) 新建内容

这一页面有多个入口，可以从内容列表进入，也可以从 AR 场景进入，页面设计如图 5-18 所示。



图 5-18 新建内容页面

新建内容时要求用户填入内容标题、内容综述。系统可以接受三个类型的内容，感言、提问和短期计划。在标题的结尾添加问号即可将内容自动标记为问题。

对于短期计划类的内容，还可以添加子任务，如图 5-19 所示。

Carrier 4:02 PM

添加任务

完成

请在此输入任务详情

计划完成时间

Wed May 29	1	59
Thu May 30	2	00
Fri May 31	3	01 AM
Today	4	02 PM
Sun Jun 2	5	03
Mon Jun 3	6	04
Tue Jun 4	7	05

图 5-19 新建内容页面

添加子任务之后，内容自动标记为短期计划。

(3) 内容详情

点击内容列表上的某一条目，即可查看内容详情，包括内容标题、类型、完成时间、赞数和评论，如图 5-20 所示。



图 5-20 内容详情

本章对系统客户端的设计和实现进行了详细论述，主要介绍了用户注册、登录页面的设计，AR 交互界面的设计与实现。

第六章 服务器设计与实现

在第四章中，已经将服务器架构、功能模块的划分和接口设计进行了阐述，接下来将结合项目文件对服务器系统中各个模块的实现进行详细描述。

6.1 开发目录介绍

服务器基于 Go 语言实现，应用 gorm 对象关系映射框架开发数据库，gin web 框架实现网络接口，目录结构较为简单。vs code 集成开发环境中项目目录如图 6-1 所示。

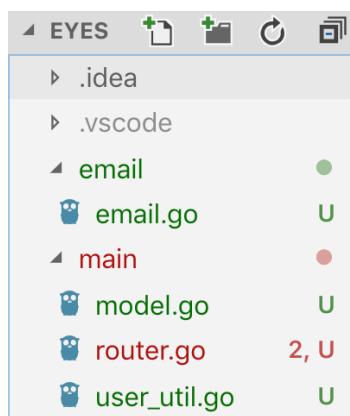


图 6-1 服务器项目目录

以下是各目录文件详细介绍：

- (1) EYES:，服务器项目最外层目录；
- (2) .idea，存储集成开发环境相关文件，自动生成；
- (3) .vscode，存储集成开发环境设置相关文件，自动生成；
- (4) email/email.go，邮件自动发送模块；
- (5) main/model.go，go 文件，实现数据表的定义以及数据库的连接；
- (6) main/router.go，go 文件，实现网络接口函数，并进行路由分发；
- (7) main/user_util.go，身份验证模块，包括 Session 机制、对 email 模块的调用等。

接下来将重点对 model.go 文件与 router.go 文件内容进行分析。

6.2 数据库

本系统基于 Go 语言的 gorm 框架实现数据库的连接和数据表的创建。gorm 是一个全功能的对象关系映射框架，应用 gorm 可以简化数据库查询过程，直接访问期望数据而不必理解数据库的底层结构^[18]。

6.2.1 数据库连接

gorm 提供了连接数据库的函数 gorm.Open，传入数据库引擎类型“mysql”以及连接字符串，即可与数据库进行连接。连接字符串的格式为：

主机地址:密码@/数据库名?编码格式&是否自动解析时间&时区设置

```
var db *gorm.DB
func init() {
    sqlConnection := "root:*****@/eyesDB?charset=utf8&parseTime=True&loc=Local"
    var err error
    db, err = gorm.Open("mysql", sqlConnection)
    if err != nil {
        panic(fmt.Sprintf("failed to connect database: %v", err))
    }
    // 迁移数据模型
    db.AutoMigrate()
}
```

与数据库连接成功之后，调用 gorm 函数 db. AutoMigrate()，自动迁移（migration）数据库。迁移过程中将会自动创建新数据表，增加新属性，但不会删除或更改旧的属性。

6.2.2 数据表定义

在 gorm 框架的帮助下，开发者可以像定义一般结构体一样定义数据模型，并且，gorm 还提供了一个预定义结构体 gorm.Model，代码如下：

```
type Model struct {
    ID          uint      `json:"email" gorm:"primary_key"`
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`
    DeletedAt   *time.Time `json:"deleted_at" sql:"index"`
}
```

包含基本的 ID 属性和三个时间戳，创建时间、更新时间以及删除时间。在多个数据类型中，都用到了这一结构体，如下文中的用户信息表、内容表。

定义数据表的同时，还要定义各表之间的联系。与 CoreData 直接将属性和关系分别定义不同，gorm 通过更直接的结构体组合与外键声明来定义数据表之间的联系，详细做

法在下文有所阐述，实现代码中也对外键的声明进行了清晰的注释。

(1) 用户基本信息表

```
User struct {
    gorm.Model
    Email      string `json:"email" gorm:"not null;unique"`
    Password   string `json:"password"`

    Profile Profile `json:"profile"`
    // 设置外键 与 event 一对多关系
    MyEvents []Event `json:"my_events" gorm:"foreignkey:CreatorID"`
}
```

这里定义了基本信息表中的属性，并指明了基本信息表 User 与个人信息表 Profile、内容表 Event 之间的关联关系，分别为一对一关系与一对多关系。

定义数据模型属性的同时，还通过 json 标注指明了各属性的 json 名，方便在接口函数中通过 json 字符串访问。

(2) 个人信息表

```
Profile struct {
    gorm.Model
    UserID uint   `json:"user_id"` // 默认外键
    Name   string `json:"name" gorm:"not null"`
    Age    bool   `json:"age"`
    Sex    uint   `json:"sex" gorm:"default:'2'" //0 M 1 F 2 UNKNOW
    Saying string `json:"saying"`
    Photo   byte   `json:"photo"`
}
```

个人信息表以 UserID 作为外键；在 gorm 框架中，外键不需要额外声明。

(3) 内容表

```
Event struct {
    gorm.Model
    CreatorID uint        `json:"creator_id"` // 创建者 ID 自定义外键
    LocationID uint        `json:"location_id"` // 默认外键
    Type      uint        `json:"type"`          // 感言 提问 计划
}
```

```

Title      string      `json:"title"`
Summary    string      `json:"summary"`
Progress   float32     `json:"progress"`
Pic        byte        `json:"pic"`
Steps      []EventStep `json:"steps"` // 与子任务一对多关系
}

```

内容表与用户表 User、场景表 Location 都是多对一关系，与 EventSteps 则是一对多关系。

(4) 场景表

```

Location struct {
    ID      uint      `json:"id" gorm:"primary_key"`
    Lat     float64   `json:"lat"`
    Lng     float64   `json:"lng"`
    Events []Event   // 与 Event 一对多关系
}

```

场景表与内容表为一对多关系，通过访问 Location.Events 就可以获取场景中所有内容。

(5) 子任务

```

EventStep struct {
    ID      uint      `json:"id" gorm:"primary_key"`
    EventID uint      `json:"event_id"` // 默认外键
    Content string    `json:"content"`
    Completed bool     `json:"completed"`
    CreatedAt time.Time `json:"created_at"`
    UpdatedAt time.Time `json:"updated_at" sql:"index"`
}

```

子任务与内容表 Event 是多对一关系。代码中的 sql:"index" 标注表示根据属性 UpdatedAt 创建索引。

(6) 点赞表

```

Like struct {
    UserID  uint `json:"user_id" gorm:"primary_key"`
    EventID uint `json:"event_id" gorm:"primary_key"`
}

```

```

IsLike bool `json:"is_like"`

CreatedAt time.Time `json:"created_at"`
UpdatedAt time.Time `json:"updated_at"`
DeletedAt *time.Time `json:"deleted_at" sql:"index"`

}

```

点赞表以 UserID 与 EventID 为联合主键，同时两者也都是外键，表示点赞用户及被点赞内容。

(7) 评论表

```

Comment struct {
    gorm.Model
    UserID uint `json:"user_id"`
    EventID uint `json:"event_id"`
    Content string `json:"content"`
}

```

评论表也有外键 UserID 与 EventID，但由于同一用户可以对同一内容多次评论，因此两者不作为主键，通过 gorm.Model 另行定义主键。

6.3 网络接口

网络接口是基于 go 语言开发微框架 go-gin 实现的，分为路由分发和接口函数实现两个模块。

6.3.1 路由分发

路由是指路由器从一个接口上收到数据包，根据数据包的目的地址进行定向并转发到另一个接口的过程^[19]。go-gin 的路由来自 httprouter 库，但不支持路由正则表达式。以下是 restful 风格的路由实现：

```

func main() {
    router := gin.Default()
    router.GET("/Hi", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "This is Eyes",
        })
    })
}

```

```
v1 := router.Group("eyes/v1")
{
    v1.GET("/people/new/:email", isNew) // 验证是否具有注册资格
    v1.POST("/people/signup", signUp) // 注册

    v1.POST("/user/signin", signIn) // 登录

    v1.GET("/user/:user_id", getProfile) // 查询用户个人信息
    v1.POST("/user/profile", editProfile) // 编辑个人信息
    v1.POST("/user/setting", setUser) // 用户设置

    v1.GET("/user/password/:user_id", resetAsk) // 重设密码申请
    v1.POST("/user/password/verification", resetVerify) // 重设密码验证
    v1.PUT("/user/password", passwordReset) // 重设密码

    v1.POST("/event/creation", createEvent) // 创建内容

    v1.GET("/event/all/:loc_id", fetechAllEvents) // 获得场景中所有内容
    v1.GET("/event/one/:event_id", fetechSingleEvent) // 获得单个内容
    v1.GET("/event/mine", myEvents) // 查询用户创建的所有内容

    v1.PUT("/event/updation", updateEvent) // 更新任务
    v1.PUT("/event/step", addStep) // 添加子任务
    v1.PUT("/event/step/updation", updateStep) // 更新子任务

    v1.DELETE("/event/:loc_id/:event_id", deleteEvent) // 删除内容

    v1.PUT("/event/like/:user_id/:event_id", likeEvent) // 点赞
    v1.PUT("/event/comment/:user_id/:event_id", commentEvent) // 评论
}

router.Run() // 运行路由服务
```

```

    defer db.Close() // 在线程销毁时关闭数据库
}

```

gin 框架可以为我们指定接口的动作类型，具体的接口实现见下文。

6.3.2 接口实现

服务器系统的网络接口基于 go-gin 网络框架实现，编码过程中调用多个 go-gin 框架函数，如表 6-1 所示。

表 6-1 go-gin 框架函数

函数名	功能
gin.Context.ParamsByName(name string)	通过名称获取 URL 中的可变参数
gin.Context.PostForm (key string)	从 POST 提交的 form 表单中提取参数
gin.Context.JSON(code, obj interface{})	返回状态码和结果

返回结果可以是简单的字符串，也可是 JSON 格式的结构体。

服务器接口设计运用了模块化思想，各个模块可以独立实现，而后通过数据通信进行耦合，以下是各个模块的详细实现。

(1) 注册

注册模块有两个接口方法：

① isNew 方法

它的主要功能是验证邮箱是否符合注册要求。首先，通过正则表达式验证客户端发送的邮箱参数是否为标准的邮箱号，代码如下：

```

email := c.ParamsByName("email")
match, _ := regexp.MustCompile(`^([a-zA-Z_\\.-]+)@([\\da-zA-Z\\.-]+)\\.(\\.[a-zA-Z\\.-]{2,6})$`).MatchString(email)
if !match {
    c.JSON(403, "输入邮箱不合法")
    return
}

```

验证通过之后再查询数据库检验邮箱号是否未注册，代码如下：

```

if db.Where("email=?", email).First(&user).RecordNotFound() {
    return false
}

```

检验通过之后即向客户端发送结果，代码如下：

```

if isUserExist(email) {
    c.JSON(403, "账户 "+email+" 已存在")
} else {
    c.JSON(200, "可以注册新账号")
}

```

② signUp 方法

这一方法的功能是创建用户基本信息表和个人信息表，并将之插入到数据库中。

创建用户基本信息表：

```

user := User{Email: c.PostForm("email"), Password: c.PostForm("password")}

if user.Email == "" || user.Password == "" {
    c.JSON(406, "输入错误")
    return
}

```

插入数据库：

```

if db.Create(&user).Error != nil {
    c.JSON(503, "无法创建用户，请稍后再试")
    return
}

```

创建个人信息表：

```

if db.Create(&user.Profile).Error != nil {
    c.JSON(503, gin.H{
        "message": "无法创建个人信息表，但用户已注册",
        "user": user})
    return
}

```

最终注册成功，返回状态码 200 以及 JSON 格式的用户信息。

(2) 登录

登录接口只有一个方法，signIn，首先根据客户端传送的邮箱与密码信息查询数据库，若找到了相关条目，则返回 JSON 格式的用户信息表，否则返回错误代码。代码如下：

```
var user User
```

```

if db.Where(&User{Email: c.PostForm("email"), Password:
c.PostForm("password")}).Find(&user).RecordNotFound() {
    c.JSON(403, "邮箱或密码不匹配")
} else {
    c.JSON(200, user)
}

```

用户登录成功之后，服务器将生产 Session，记录 Session id，以记录连接状态。

(3) 创建内容

创建内容模块只有一个方法，createEvent，接口详细说明见表 4-25，表 4-26 给出了请求参数，表 4-27 给出了响应参数。

在 createEvent 方法被客户端调用时，首先从 form 参数表中获取经度和纬度，根据经纬度查询数据库，获取满足条件的最近的场景，若查询结果为空，则创建新场景。

获取经纬度参数：

```

lat := c.PostForm("lat")
lng := c.PostForm("lng")
if lat == "" || lng == "" {
    c.JSON(403, "无场景经纬度，输入错误")
}

```

查询数据库：

```

queryStr := "select * from location where lat > " +
    lat + "-1 and lat < " + lat + "+1 and " +
    "lng > " + lng + "-1 and lng < " + lng +
    "+1 order by ACOS(SIN((" + lat + " * 3.1415) / 180 )" +
    " * COS((latitude * 3.1415) / 180 ) *COS((" +
    lng + " * 3.1415) / 180 -" +
    " (longitude * 3.1415) / 180 )) * 6380 asc limit 1"
row, _ := db.DB().Query(queryStr)
defer row.Close()

```

若查询接口为空，则根据经纬度创建新场景：

```

if row.Next() {
    if err := row.Scan(&loc); err != nil {
        loc.Lat, err = strconv.ParseFloat(lat, 64)
}

```

```

loc.Lng, err = strconv.ParseFloat{lng, 64)
if db.Create(&loc).Error != nil {
    c.JSON(503, "服务器错误，稍后再试")
    return
}
}
}

```

确定场景之后，创建新的 Event 对象并插入数据库，返回响应码：

```

event := Event{LocationID: loc.ID}
if c.ShouldBindJSON(&event) != nil {
    c.JSON(403, "输入错误")
    return
}
if db.Create(&event).Error != nil {
    c.JSON(503, "服务器错误，稍后再试")
    return
}
c.JSON(200, event)

```

createEvent 函数的性能可以通过优化数据库查询场景的算法而得到进一步提高。

(4) 评论内容

评论内容接口有两个方法，likeEvent 和 commentEvent，接口详细说明见表 4-45，表 4-46 给出请求参数，表 4-47 给出响应参数。

① likeEvent 方法

首先，从 URL 参数中获取评论者账号和被评论内容编号：

```

user_id := c.Param("user_id")
event_id := c.Param("event_id")
if user_id == "" || event_id == "" {
    c.JSON(403, "输入错误")
}

```

接着，查询 like 表中是否已存在该条记录，若存在，则将记录中的 isLike 属性取反，若不存在，则新建 Like 对象并插入数据库中：

```
var like Like
```

```

if db.Where("user_id = ? AND event_id = ?", userID, eventID).
    First(&like).RecordNotFound() {
    var user User
    var event Event
    if db.First(&user, userID).RecordNotFound() ||
        db.First(&event, eventID).RecordNotFound() {
        c.JSON(403, "输入错误,用户或内容不存在")
        return
    } else {
        like = Like{UserID: user.ID, EventID: event.ID, IsLike: true}
        if db.Create(&like).Error != nil {
            c.JSON(503, "服务器错误, 稍后再试")
            return
        }
    }
} else {
    like.IsLike = !like.IsLike
    db.Save(&like)
}
c.JSON(200, like)

```

需要注意的是，由于 user_id 和 event_id 的联合主键及外键，因此当 like 表中已存在记录时无需验证评论者与被评论内容是否存在，而当 like 表中无记录，需要新建 Like 对象时则必须验证两者的存在性。

② commentEvent 方法

commentEvent 方法的算法流程与 likeEvent 相比，无须查询是否已存在记录，直接新建 Comment 对象，并插入数据库中。

获取参数：

```

userID := c.Param("user_id")
eventID := c.Param("event_id")
content := c.Param("content")
if userID == "" || eventID == "" ||
    content == "" {

```

```
c.JSON(403, "输入错误")
return
}
```

新建评论对象：

```
var com Comment
var user User
var event Event
if db.First(&user, userID).RecordNotFound() ||
    db.First(&event, eventID).RecordNotFound() {
    c.JSON(403, "输入错误,用户或内容不存在")
    return
}
com = Comment{UserID: user.ID, EventID: event.ID, Content: content}
if db.Create(&com).Error != nil {
    c.JSON(503, "服务器错误, 稍后再试")
    return
}
c.JSON(200, com)
```

Comment 对象不以用户账号和内容编号为联合主键，因为同一个用户可以对同一内容发布多条评论。

本章详细阐述服务器各个模块接口的实现过程，并给出关键代码，主要包括用户的注册、登录，内容的创建与评论。编码的过程也是测试的过程，接下来将系统地说明测试过程，并给出测试结果。

第七章 系统测试

编码和测试是系统开发必不可少的两个步骤。面对比较复杂的软件系统，开发人员在编码过程中总是会遇到极其错综复杂的问题，而随着开发周期的延长，设计方案可能会因为各种各样的原因而无法实现，研究阶段提出的用户需求可能会无法满足，而软件测试，就是检验软件系统是否满足设计指标，弄清预期需求和最终结果之间的差别的过程。通过不断地发现错误，纠正错误，系统的各个功能才能最大程度上满足用户需求，软件质量才能得到保证^[20]。

软件测试在软件生命周期中横跨两个阶段。在编码阶段，通常编写出每个模块之后，就要对它进行功能测试。编码结束之后，软件系统还要接受各种综合测试。本章主要对系统的各个模块进行单元测试，而后，对整个系统进行性能测试，考察开发成果是否满足设计预期。

7.1 功能测试

功能测试的目的是验证系统的各个功能模块能否正常使用，无需考虑程序内部结构，又称为黑盒测试^[21]。下文将对客户端各个模块和服务器接口分别进行功能测试。

7.1.1 客户端

主要对三个模块进行测试：

(1) 用户登录

在登录模块，系统主要验证用户提交的邮箱号和密码是否能与数据库中存储的信息一一对应。数据库中的部分用户表单如图 7-1 所示。

id	created_at	updated_at	deleted_at	email	password
1	2019-05-14 22:31:28	2019-05-14 22:31:28	NULL	liudxp@eyes.cn	asdfg!@#\$%
3	2019-05-14 22:38:52	2019-05-14 22:38:52	NULL	harry@eyes.cn	poiu!@#\$%
4	2019-05-14 22:38:52	2019-05-14 22:38:52	NULL	people@eyes.cn	098&^%\$xcvbn

图 7-1 服务器用户基本表测试表单

这里构造两个案例对登录页面进行测试。

- ① 邮箱：liudxp@mbt.com；密码：qwert12345，如图 7-2 所示。



图 7-2 登录测试案例 1

- ② 邮箱: liudxp@eyes.cn; 密码: asdfg!@#\$%, 如图 7-3 所示。

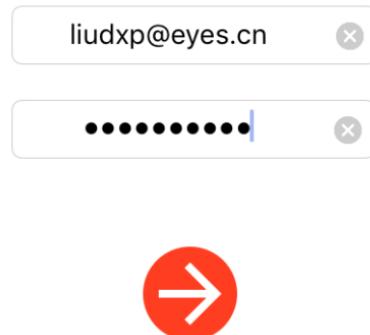


图 7-3 登录测试案例 2

登录成功，进入首页，用户登录模块满足设计要求。

(2) 用户注册

首先是对邮箱输入界面的测试，这一页面的主要功能是验证用户输入的邮箱号是否符合注册要求，这里使用三个案例进行测试：

- ① 不合法的邮箱号，如图 7-4 所示。



图 7-4 邮箱验证测试案例 1

这种情况下邮箱不可用。

- ② 合法但已注册邮箱，如图 7-5 所示。



图 7-5 邮箱输入测试案例 2

提示可以直接登录。

③ 合法未注册邮箱，经测试注册成功，直接进入密码设置页面。

邮箱输入界面符合设计要求，接下来对密码设置页面进行测试，如图 7-6 所示。



图 7-6 注册页面测试

出于信息安全上的考虑，密码输入框的字符是经过圆点加密的。两次输入密码不一致，注册按钮不可用，修改输入，如图 7-7 所示。



图 7-7 注册页面测试

两次输入的密码一致，注册按钮可用，点击注册之后自动登录，进入首页。经测试，注册模块满足设计要求。

按照上文所述方法，针对客户端各个模块设计测试案例，进行黑盒测试，测试结果如表 7-1 所示。

表 7-1 客户端功能测试结果

模块	测试结果
登录	通过
注册	通过
新建内容	通过
修改内容	通过
AR 场景浏览	通过
提交评论	通过
点赞	通过

各个模块通过单元测试，符合设计要求。

7.1.2 服务器

服务器接口负责客户端和服务器之间的数据连接，没有直观的人机交互界面，因此需要专业的图形化程序辅助测试，这里用到的是 Postman。

(1) 注册接口

与注册相关的客户端-服务器接口有两个，`com.mbt.eyes.people.isNew` 接口与 `com.mbt.eyes.people.signUp` 接口，其定义见表 4.5.1；分别进行单元测试：

① isNew 接口测试

接口动作是 GET 类型，通过 URL 传递 `email` 参数，返回判定结果。这里构造两个测试案例。

如图 7-8 所示，测试邮箱 `liudxp@eyes.cn` 为已注册邮箱。

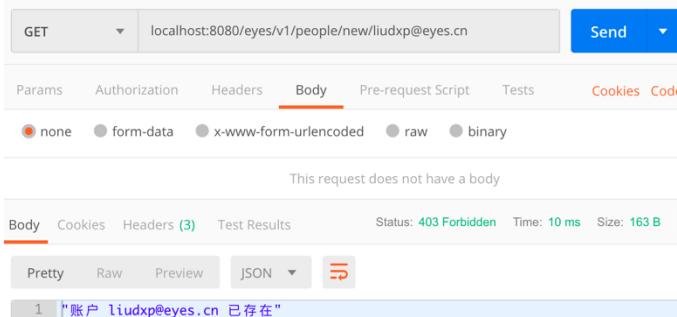


图 7-8 isNew 接口测试案例 1

返回结果状态码为 403，提示邮箱已经被占用。

测试新邮箱 `liudxp@mbt.com`，如图 7-9 所示。

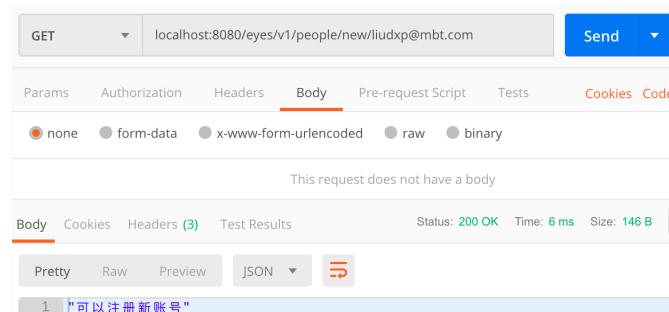


图 7-9 isNew 接口测试案例 2

经测试接口满足设计要求。

② signUp 接口测试

接口动作为 POST 类型，需要通过表单提交邮箱和密码，如图 7-10 所示。

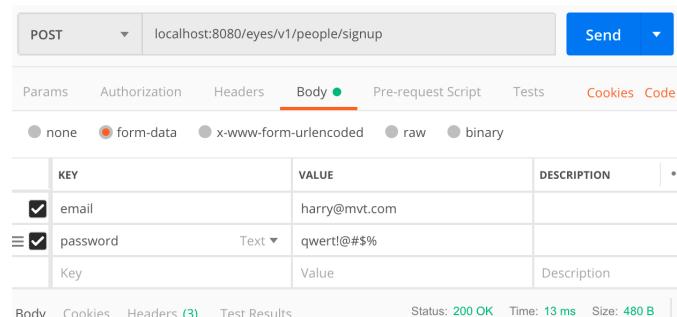


图 7-10 signUp 接口测试案例 1

返回结果如图 7-11 所示。

```

1 {  

2   "created_at": "2019-05-15T23:57:47.469471+08:00",  

3   "updated_at": "2019-05-15T23:57:47.469471+08:00",  

4   "deleted_at": null,  

5   "email": "harry@mvt.com",  

6   "password": "qwert!@#$%",  

7   "profile": {},  

19   "my_events": null  

20 }

```

图 7-11 signUp 测试案例 1 结果

若提交的邮箱号已注册，或因其他原因在数据库中插入数据失败，则返回无法注册的提示，错误代码 503，如图 7-12 所示。

KEY	VALUE	DESCRIPTION
email	harry@mvt.com	
password	qwert!@#\$%	
Key	Value	Description

Status: 503 Service Unavailable Time: 18 ms Size: 178 B

Pretty Raw Preview JSON

1 | "无法创建用户，请稍后再试"

图 7-12 signUp 接口测试案例 2

若不提交 form 表单，如图 7-13 所示，则返回输入错误，代码 406。

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 406 Not Acceptable Time: 7 ms Size: 149 B

Pretty Raw Preview JSON

1 | "输入错误"

图 7-13 signUp 接口测试案例 3

signUp 接口通过测试，满足设计要求。

(2) 登录接口

这一模块只有一个接口方法，signIn，提交邮箱号与密码，返回登录状态；这里设计两个测试案例进行测试，案例 1 提交一对乱码，如图 7-14 所示。

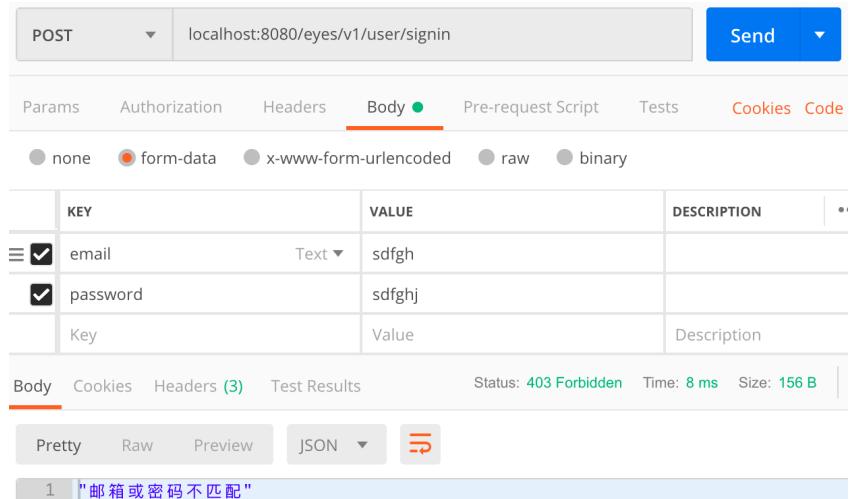


图 7-14 登录接口测试案例 1

显然登录失败。案例 2 提交正确的账号信息，如图 7-15 所示。

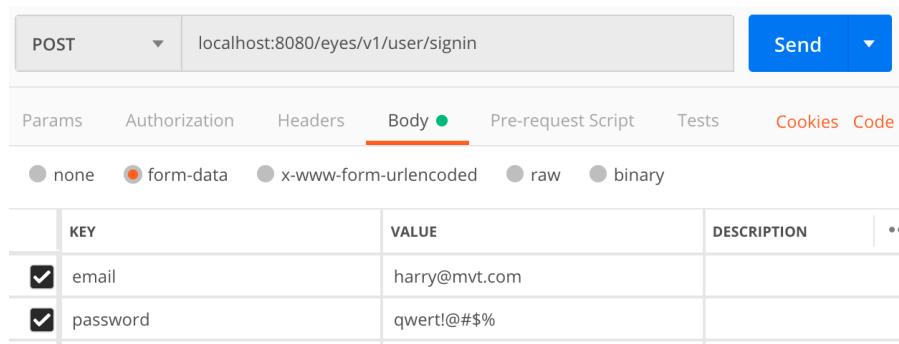


图 7-15 登录测试案例 2

测试结果如图 7-16 所示。

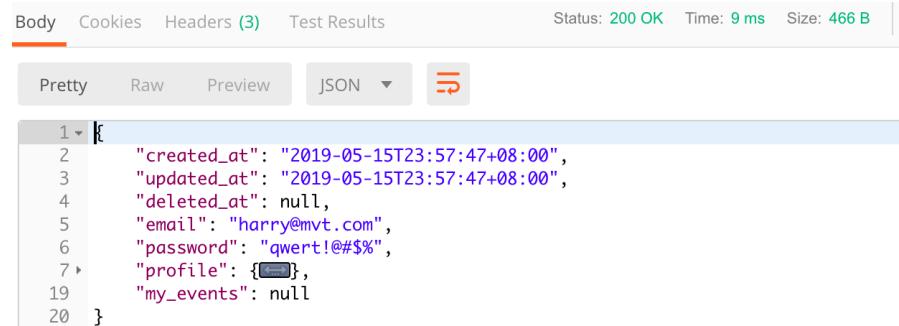


图 7-16 案例 2 结果

登录成功，返回用户信息。

经测试，用户登录接口满足设计要求。

按照上文所述方法，针对服务器各个模块的接口设计测试案例，进行黑盒测试。测试重点为登录、注册模块，内容创建与修改模块，以及评论模块。测试结果如表 7-2 所示。

表 7-2 服务器功能测试结果

模块	接口	测试结果
用户管理	isNew	通过
	signUp	通过
	signIn	通过
	getProfile	通过
	editProfile	通过
	setUser	通过
	resetAsk	通过
	resetVerify	通过
	passwordRset	通过
	userEvents	通过
内容管理	createEvent	通过
	fetechAllEvents	通过
	fetechSingleEvent	通过
	addStep	通过
	updateEvent	通过
	deleteEvent	通过
	likeEvent	通过
	commentEvent	通过

各个模块通过单元测试，符合设计要求。

7.2 性能测试

性能测试是判断软件系统是否能够达到用户要求的性能指标的重要步骤。本系统主要测试项包括服务器接口响应速度及客户端各用户操作响应速度。

(1) 客户端性能测试

客户端性能测试有两个衡量指标，页面载入时间和操作进行时总内存。各操作结果如表 7-3 所示。

从表中可以看到，AR 场景初始化时消耗时间较多，这是因为系统需要根据摄像机得到的图形计算平面节点，并生成虚拟网格平面。

同时，在 AR 场景载入，页面内容载入之后，总内存大小也有所增加，但依然在 200M 的设计要求之内。

设计要求客户端各个操作用户等待时间<1s，经测试各客户端动作耗时基本符合要求。

表 7-3 客户端性能测试

客户端动作	载入时间(ms)	总内存(M)	结果
登录	230	66	通过
注册	260	66	通过
查看个人信息	169	70.01	通过
编辑个人信息	138	70.01	通过
AR 场景初始化	400	104	通过
AR 内容载入	432	104	通过
AR 交互-评论	230	104	通过
AR 交互-点赞	220	104	通过
新建内容	340	69.7	通过
修改内容	320	104	通过

经测试，客户端各交互界面基本满足性能指标。

(2) 服务器性能测试

对服务器接口的性能测试有两个衡量指标，响应时间和内存大小。接下给出各个接口测试结果，如表 7-4 所示。

表 7-4 服务器性能测试

接口	响应时间(ms)	内存(B)	结果
注册	13	480	通过
登录	9	466	通过
新建内容	13	960	通过
修改内容	8	156	通过
提交评论	9	163	通过
点赞	7	108	通过

经测试，各服务器接口满足性能指标。

7.3 兼容性测试

本系统基于 C/S 架构实现，客户端运行在 iOS 系统中，经测试可以在硬件满足 ARKit2.0 要求，系统版本为 12.2 的 iPhone、iPad 设备中正常运行。

7.4 测试结果

各项测试的结果如表 7-5。

表 7-5 测试结果

功能测试	性能测试	兼容性测试	总评
合格	基本合格	合格	基本合格

对系统的功能测试、性能测试及兼容性结果的测试表明本系统已基本满足用户需求，符合设计指标。

结论

本文论述了一个基于增强现实的智能社交系统，该系统以增强现实交互界面为媒介，将用户提交的社交信息以 3D 图文的形式发布在现实中的社交场景中，并提供实时交互接口，使得虚拟的社交信息成为现实的社交场景的一部分，为平台用户提供了一种全新的社交体验。

相比现有的移动社交应用，本系统具有以下特点：

- (1) 摒弃了瀑布流式布局，专注于处理某一现实社交场所的信息，借助社交场所的特殊性梳理过滤社交信息，避免信息超载。
- (2) 有着简洁易用的 AR 交互界面，以社交内容为交互核心，引导用户将注意力转向内容本身。
- (3) 鼓励用户在合适的场合公布自己的短期计划，将社交关注度转化为具有鼓励意义的正反馈。

系统通过了单元测试与综合测试，基本满足设计要求。在实现系统的过程中，总结出以下几条经验：

- (1) Go 语言、gorm 框架、go-gin 框架能够极大地提高服务器开发效率。对象关系映射框架 gorm 使开发者得以将注意力转向数据本身而忽略数据库系统中的具体实现，而 go-gin 框架为 restful api 的实现提供了极大的便利。
- (2) ARKit 2.0 框架为增强现实场景的实现提供了极大的方便。
- (3) 成熟的 MVC 开发模式可以使得图形应用的开发变得简单而高效。

同时，本系统也有着一些遗留问题，如 AR 场景图形元素不够丰富，动态提示算法不够智能，在未来的研究中，需要对 AR 虚拟信息的展示方式进行重新设计，并学习成熟的内容推荐算法，应用到动态提示模块。用户量增加之后，为了保证性能满足要求，服务器带宽也需要进一步升级。

致谢

感谢我的父母，在这四年的大学生活里，在之前十二年的求学生涯里，他们一如既往地为我提供了经济支持。他们总是能理解并支持我的选择，为我骄傲，为我担忧。

感谢我的弟弟刘东坡，他是我在自律方面的老师。他总是督促我锻炼身体，在技术方面为我提供了许多明智建议。在开发系统服务器之前，我曾在 python 和 golang 两种编程语言之间摇摆不定，是我的弟弟根据他的切身经验阐述了 golang 的优越性，坚定了我的选择，很大程度上提高了开发效率。

感谢我的毕业设计导师——景国良老师。论文翻译、系统开发、论文撰写、修改、查重、再修改……我的每一项工作，都离不开景老师的悉心教导。他有着惊人的耐心，一次又一次地指出错误，提出建议，在我消沉的时候鞭策我，在我后进的时候鼓励我。感谢他对我的关系和帮助，与他交往，胜过读许多书，上许多课。

感谢我的朋友江浩，他在学习生活上给了我很多帮助。在撰写论文与毕设翻译的过程中，他在文章的排版格式上给了我很多指导。

最后，感谢我的母校。大学总是令人难忘的，母校给了我更多的人生选择，永远地改变了我的生活。

参考文献

- [1] 崔嵩, 邓威. 基于 MVC 思想的程序设计与管理[J]. 赤子, 2012 (19): 172-174.
- [2] Neuburg M. IOS 8 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics[M]. " O'Reilly Media, Inc.", 2015: 6-9.
- [3] 刘燕. iOS 开发从入门到精通[M]. 清华大学出版社 崧博出版, 2019: 8-11.
- [4] 张金钊, 张金镝, 孙颖. X3D 互动游戏交互设计: 可穿戴式交互技术[M]. 清华大学出版社, 2018: 15-21.
- [5] Clover J. Apple Releases iOS 12 With Faster Performance, Memoji, Siri Shortcuts, Screen Time, Revamped Maps App, ARKit 2.0, and More[J]. 2018: 2-6.
- [6] Harwani B M. Core Data iOS Essentials[M]. Packt Publishing Ltd, 2011: 13-24.
- [7] 蔡媛媛, 赵致琢. 分布式容错计算描述语言及其应用研究[J]. 计算机科学, 2016, 43(5): 146-149.
- [8] 伊恩·萨默维尔. 软件工程[M]. 机械工业出版社, 2018: 41-45.
- [9] 范意宏. 面向用户需求的图书馆移动信息服务[J]. 中国图书馆学报, 2012 (1): 76-86.
- [10] 王立平, 刘详淼, 彭霁. SQL Server 2014 从入门到精通[M]. 清华大学出版社, 2018: 34-42
- [11] 托马斯 M. 康诺利. 数据库系统: 设计、实现与管理[M]. 机械工业出版社, 2017: 141-149.
- [12] 郑文礼, 周红刚, 钟铿光. 管理信息系统原理与应用[M]. 中国图书有限公司, 2016: 43-65.
- [13] 孔军, 孙怡宁, 蒋敏, 等. 基于 UML 的系统需求分析[J]. 2003: 14-21.
- [14] 奥佛甘地, 帕姆奎斯特等, 用例: 模式与蓝图[M]. 清华大学出版社, 2005: 63-67.
- [15] 祝翔, 董启文, 郁可人. 基于 WebSocket 的 PK 答题的设计与实现[J]. 华东师范大学学报 (自然科学版), 2018: 89-100.
- [16] 邝孔武, 王晓敏. 信息系统分析与设计[M]. 清华大学出版社有限公司, 2006: 23-27.
- [17] 张益珲. Swift 从入门到精通[M]. 清华大学出版社, 2018: 12-15.
- [18] Yellavula N. Building RESTful Web services with Go: Learn how to build powerful RESTful APIs with Golang that scale gracefully[M]. Packt Publishing Ltd, 2017: 22-27.
- [19] 苏竞秀, 龙陈锋. Floyd 算法与 RAD 算法性能分析[J]. 计算机应用与软件, 2015: 116-119.
- [20] 尹平. 可复用测试用例研究[J]. 计算机应用, 2010: 1309-1311.
- [21] 张海藩. 软件工程导论[M]. 清华大学出版社, 2013: 56-64.