2023 届本科毕业设计

基于 Uni-app 框架的一端开发多端应用的设计与实现

学　　院：计算机科学与技术学院

专　　业：软件工程

学 生 姓 名：林武特

班　　级：21 软件 3+2　　学号 _____

指导教师姓名：郑毅平　　职称：副教授

最终评定成绩 _____

2023 年 05 月

基于 Uni-app 框架的一端开发多端应用的设计与实现

作者：林武特
导师：郑毅平

摘　要

发展多端应用成为移动互联网当下的趋势，跨平台开发相比于传统一端应用一端开发方式，具有更高的效率和更低的开发成本。本文主要介绍基于 Uni-app 框架开发多端应用，并且结合多线程的 Golang 语言作为后端技术，使得整个应用具有高度的运行效率和多端一致性。

本应用以“中式外语”为实例，前端基于 Uni-app 框架的一端开发多端应用进行实践，后端和爬虫抓取数据采用谷歌开源的多线程 Golang 语言，数据库采用开源的 PostGreSQL 作为数据存储。

Uni-app 是一种基于 Vue.js 的跨应用平台多端开发框架，可以实现一套代码在多个应用终端平台上运行。在 HBuilderX 的编译器上进行 Uni-app 框架一端开发多端应用的设计与实现方式，并通过实例验证了该方法的一致可行性和多端环境优势，来证明了其具有很好的开发效果和应用价值。

关键词：Uni-app，Vue.js，多端，Golang，爬虫

Design and Implementation of Multi end Application Programs at One End of the Uni-app Framework

Author：Wute Lin

Tutor：Yiping Zheng

ABSTRACT

The design of multi-terminal applications has become the trend of rapid development of the mobile Internet, and cross-platform development has higher efficiency and lower development costs than the traditional one-end application and one-end development methods. This article mainly introduces how to develop multi-terminal applications based on the Uni-app framework, and combines the Golang language as a back-end technology to make the whole application highly efficient and secure.

This application takes "Chinese foreign language" as an example, the front-end is based on the practice of developing multi-terminal applications at one end of the Uni-app framework, the back-end and crawler crawling data use Google's open source multi-threaded Golang language, and the database uses open source PostGreSQL as data storage.

Uni-app is a cross-platform design framework based on Vue .js, which enables a single set of code to run on multiple application terminal platforms . The design and implementation of multi-terminal applications are developed on the compiler of HBuilderX, and the feasibility and advantages of this method are verified by examples, which proves that it has good development effect and application value.

Keywords: Uni-app,Vue.js,Multit-erminal,Golang,reptile

目　录

1 绪论

1.1 Uni-app 背景

跨平台开发技术是诞生于当下多应用终端平台，即为满足应用从一个平台移植到另新平台的技术。当前主流的跨平台开发中包括有美国脸书公司开发的 React Native 框架、美国谷歌公司开发的 Flutter 框架、中国公司开发的 Uni-app 框架等。本文选用 Uni-app 进行研究并实现例子。多端开发在各平台小程序端、移动端能够应用，可以减少成本，并让多端用户体验一致。

Uni-app 在于其大幅降低移动应用开发的成本和开发难度。传统的移动应用开发需要编写不同的代码来适配不同的平台，而 Uni-app 可以一份代码同时适配多个平台终端，去除了传统一端应用一端开发的工作，即减少开发者对技术栈的学习成本[1]。

1.1.1 Uni-app 跨平台

跨平台开发技术是诞生于当下多应用终端平台，即为满足应用从一个平台移植到另新平台的技术。当前主流的跨平台开发中包括有美国脸书公司开发的 React Native 框架、美国谷歌公司开发的 Flutter 框架、中国公司开发的 Uni-app 框架等。本文实例应用采用 Uni-app 进行研究并实现例子。多端开发在各平台小程序端、移动端能够应用，可以减少成本，并让多端用户体验一致。

1.1.2 Uni-app 框架

Uni-app 是多端跨平台框架并基于 Vue.js 开发架构，开发一套代码即可实现多平台上运行。目前 Uni-app 支持腾讯微信小程序、支付宝小程序、百度小程序、IOSAPP、安卓 APP、WebH5、腾讯 QQ 小程序、字节头条小程序、华为快应用等多个平台[2]。Uni-app 具有灵活架构和完整库，满足不同应用需求，同时也具有 UI 界面兼容和稳定性。Uni-app 由国内 DCloud 公司开发的多端应用框架，能一端代码在多平台运行。对此 Uni-app 开发多端应用，减少开发人员的工作量，并能快速上线。

1.1.3 Golang 语言

Golang 是多线程的后端业务逻辑开发语言，具有接近 C 语言的优势和高安全等优点。利用 Golang 作为后端技术，能够提高应用程序的效率，并且保证应用的安全性，其次 Golang 打包生成二进制文件可以多服务端部署，强类型语言错误校验力高，同时可做爬虫抓取数据。

1.2 多端适配

1.2.1 多端一致性

Uni-app 可实现多平台上运行，但不同平台存在 UI 差异。在进行一端开发多端应用，需进行多端兼容适配。多端一致性适配需要考虑以下适配：

（1）UI 界面布局的适配：不同终端平台的屏幕尺寸和分辨率存在差异，需要经过 CSS 使用 Flex 布局等方式来实现自适应，珊瑚格来实现不同屏幕分辨率的可视界面；

（2）样式适配：不同终端平台对字体、颜色、边距等样式的处理方式存在差异，需要使用 SCSS 等方式来实现样式的适配；

（3）API 适配：不同终端平台的原生 API 存在差异，需要通过封装或注释条件编译等方式来实现 API 的调取。

1.2.2 跨平台打包

应用项目一旦开发完，据不同终端需求，进行相应端 NPM 打包，以便程序在不同平台上运行。Uni-app 提供多端打包方式，支持小程序和 APP 等平台的应用程序[3]。为实现一端开发多端应用的设计和实现，需将不同平台应用程序打包成相应端的应用。

1.2.3 应用设计

本文所述的应用“中式外语”作为一端开发多端应用实例，本实例采用 Uni-app 作为前端开发技术栈的架构，其中所使用到的翻译 API 使用百度文本翻译接口，以实现单词翻译与单词读音功能；后端采用谷歌 Golang 语言作为后台 POST 和 GET 等接口设计[4]，以实现不同服务端的多端部署需求，并且使用 Golang 语言作为爬虫抓取外国新闻内容；数据库采用 PostgreSQL 作为数据存储，以满足数据分组功能的多模式需求来适应多个书本库存储多个单词表。而部分 UI 设计引入 Uni-ui 来实现多端适配一致性 UI 模块。

1.2.4 实例验证

为了验证一端开发多端应用的设计与实现方式的可行和优势，我选取了一个翻译工具应用作为实例。该应用主要是翻译工具、私有单词库和爬虫抓取外国文，并上架市场各平台应用端。

在这翻译工具应用程序，我使用了一些先进的技术和设计理念。通过技术和理念的应用，可以实现迭代、可维护、可扩展的多端应用。Uni-app 开发应用，这使得可以同时实现多个应用端平台小程序应用、苹果 IOS 和 Android 平台的 APP 应用[5]。此外，以 PsotGreSQL 来存储应用的数据，这使得应用可以快速响应请求。

1.2.5 应用架构

实例应用前端界面层由 Uni-app 的 Vue3.js 语法实现，业务逻辑层和数据层由 JavaScript 代码实现，翻译功能模块的 API 接口采用 Golang 语言去请求，爬虫使用多线程的 Golang 语言采集脚本作为爬虫抓取外国新闻内容，对 PsotgreSQL 数据库数据操作也是通过 Golang 语言实现。业务逻辑层主要实现 API 翻译文本功能和单词库处理，数据层主要实现外文或单词库数据的存取。

前端界面层包括应用程序的 UI 设计和用户交互逻辑，采用 Uni-app 的 Vue3.js 能够让项目快速构建跨平台的应用程序。Vue3.js 具有比 Vue2.js 更快的 Dom 渲染速度。

Uni-app 框架业务逻辑层主要负责应用程序的 Js 交换逻辑处理和业务定义, 包括 API 请求、Date 处理、逻辑计算等。JavaScript 作为主要实现业务逻辑处理，且多端兼容提高应用程序性能，并易于二次维护和二次开发扩展。

2 需求分析

2.1 框架分析

2.1.1 市场需求

在招聘平台出现很多 Uni-app 框架的大量招聘岗位，企业主要招聘岗位原因是以下几个方面的原因：

（1）提高了开发效率。Uni-app 框架可以一次性编写一套代码，生成多个平台上的应用程序，可以大幅度提高开发效率，减少开发成本。

（2）节约了开发资源。使用 Uni-app 框架，开发人员只需熟悉一种开发语言和开发工具，无需学习多种不同的技术，从而节省开发周期。

（3）提高了跨平台应用的质量。Uni-app 框架采用的是基于 Vue.js 的渲染引擎，能够保障跨平台应用程序的界面一致性和兼容性，提高了应用程序的质量和用户体验。

（4）降低了市场开发门槛。由于跨平台开发可以将应用程序更快、更低成本地拓展到多个市场，提高了市场开发的容易程度。

跨平台应用开发已成不可忽视的趋势，在传统的单一平台开发模式已经不能满足市场的需求，跨平台框架的出现成了广大开发者在开发方面的重要选择。跨平台框架能够让开发者使用同一套代码在不同的平台上运行，大大提高了开发效率和可扩展性。

2.1.2 多端性

本实例应用所采用的技术主要有支持多端的 Uni-app 框架、支持多服务端的 Golang 语言、Veu3 和 PostGreSQL 数据库均为开源产品，并引入百度翻译 API 和百度短信 SMS 即时信息服务，官方文档明朗易解决问题。

实例应用采用一端开发多端应用框架，兼容性能强，在多个终端中直接运行，对于腾讯微信小程序、腾讯 QQ 小程序、安卓 APP 和苹果 APP，功能相同界面相同。

2.2 用例分析

对多端应用框架构建项目，本实例应用即以“中式外语”为多端应用实例，本应用实例功能即展示多端环境的一致性和优势，实例实现前端多端适配、多端语言架构、应用市场 API 使用和爬虫抓取数据的实例；对当下应用市场基本技术栈需求和框架的使用，践行框架应用场景下实现功能。

2.2.1 实例用例图

用户实例，用户的注册在所属终端快捷登录或手机注册，在其他终端快捷登录后可绑手机，即在任何终端以手机登录。

本实例应用用户含登录、注册、绑定手机、每日签到、创建自定义书、加入书本、编辑单词和州除书等，如图 2.1 所示。

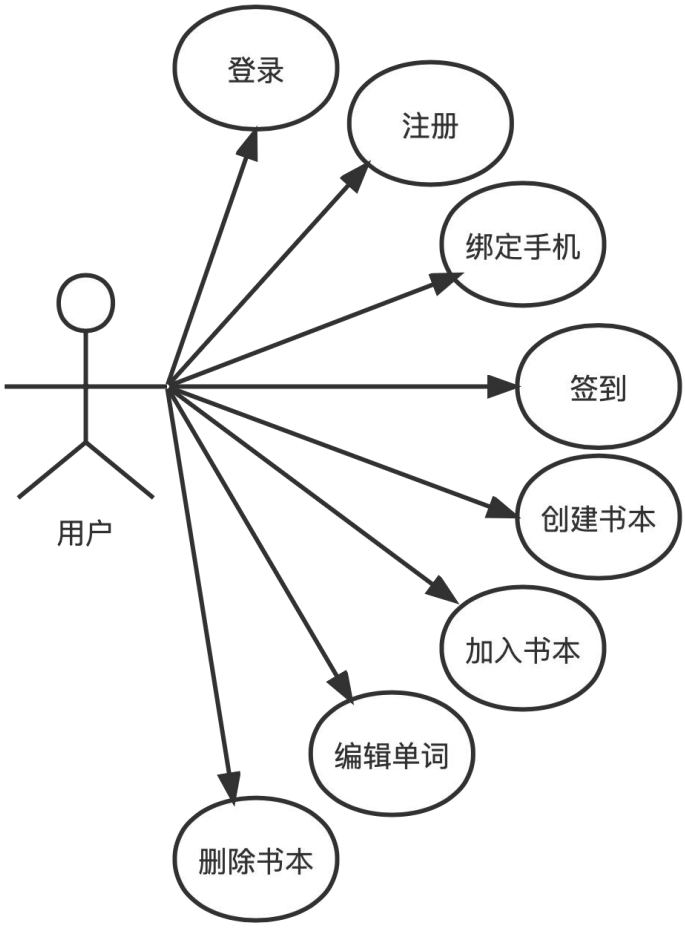

图 2.1 用户用例

(1) 中式外语实例用户登录用例，用户登入后，用户可以更改昵称、手机绑定、更换头像、创建书本、加入书本，修改单词等，如图 2.2 所示。

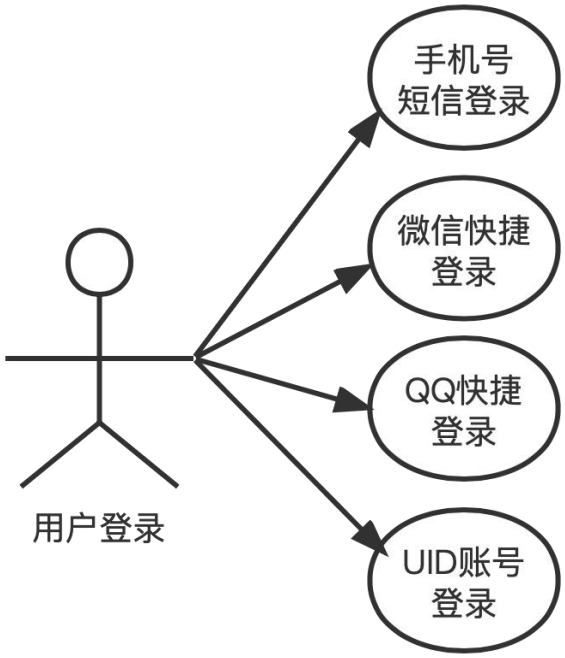

图 2.2 用户登录

(A) 手机短信登录: 短信登录 => 用户输入手机号接收 Code => 提交号码和 Code 登录。

(B) 微信或 QQ 快捷登录：快捷登录 => 用户根据所属平台选择登录。

(C) UID 登录：UID 和输入 Password

(2) 实例用户自动注册过程用例，用户登录后，Golang 后端判断用户是否已存在状态，如果用户不存在则自动创建账号，创建成功后即对数据库新增记录，并返回信息给前端，如图 2.3 所示。

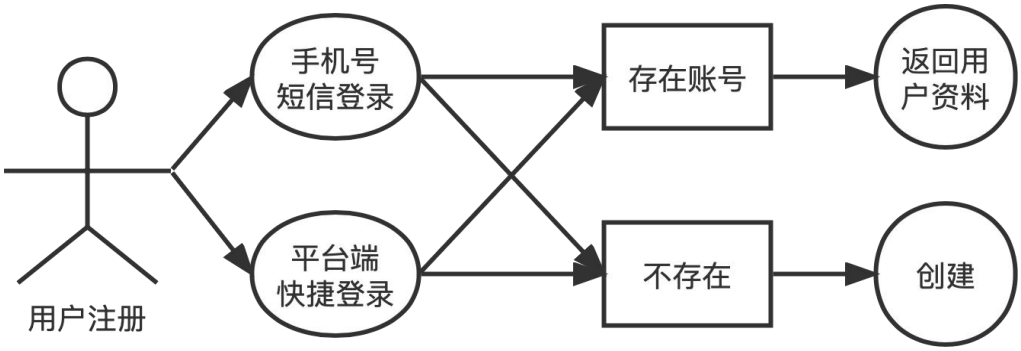

图 2.3 用户注册

(A) 手机号短信登录

(B) 微信快捷登录

(C) QQ 快捷登录

(3) 中式外语实例用户绑手机用例，用户绑手机便捷多端同一账号登录。用例如图 2.4 所示。

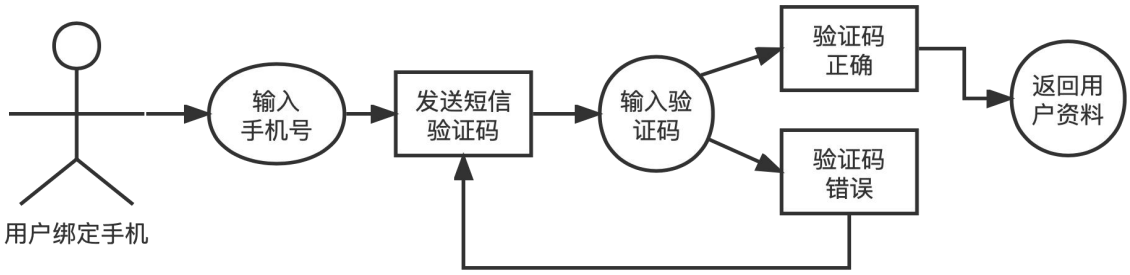

图 2.4 用户绑定手机

(A) 微信或 QQ 快捷登录

(B) 点击用户侧边信息

(C) 选择绑手机

(4) 中式外语实例用户创建书本用例，用户创建书本，模式可选择单词、音标、小说和汉字，以及是否公开，如图 2.5 所示。

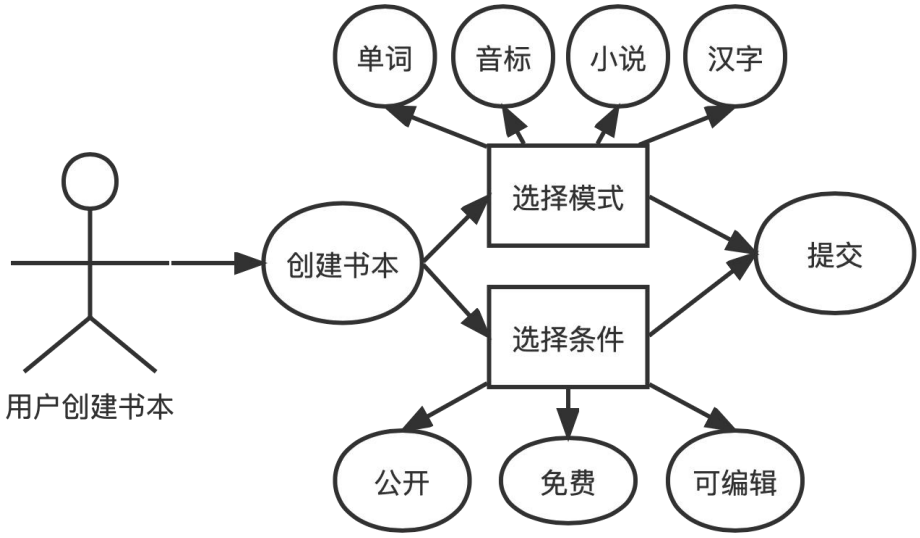

图 2.5 用户创建书本

(A) 用户在书架栏创建自定义书本

(B) 流程：登录成功后 => 书架栏 => 创建自定义书本。

(C) 流程：书本创建成功 => 自动创建单词库表

(5) 中式外语实例用户编辑书本单词用例，作者在学习栏目中编辑书本单词，如图 2.6 所示。

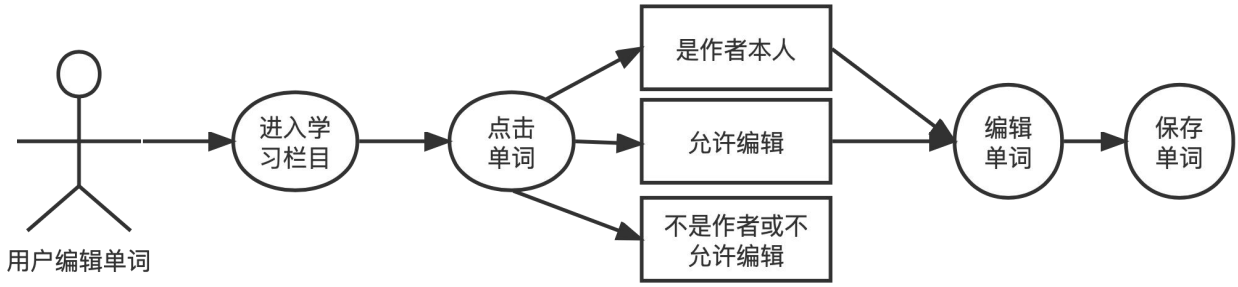

图 2.6 用户编辑单词

(A) 作者在学习栏目编辑现有的单词内容。

(B) 流程：学习栏 => 点击单词 => 修改单词 => 保存单词。

2.2.2 外文模块例图

外文内容根据爬虫定时抓取国外新闻网外国文内容，采用 Golang 语言 Goquery NewDocument 脚本解析 HTML 内容本筛选出所需内容。实例应用即抓取了英文、德文、法文、俄文等，如图 2.7 所示。

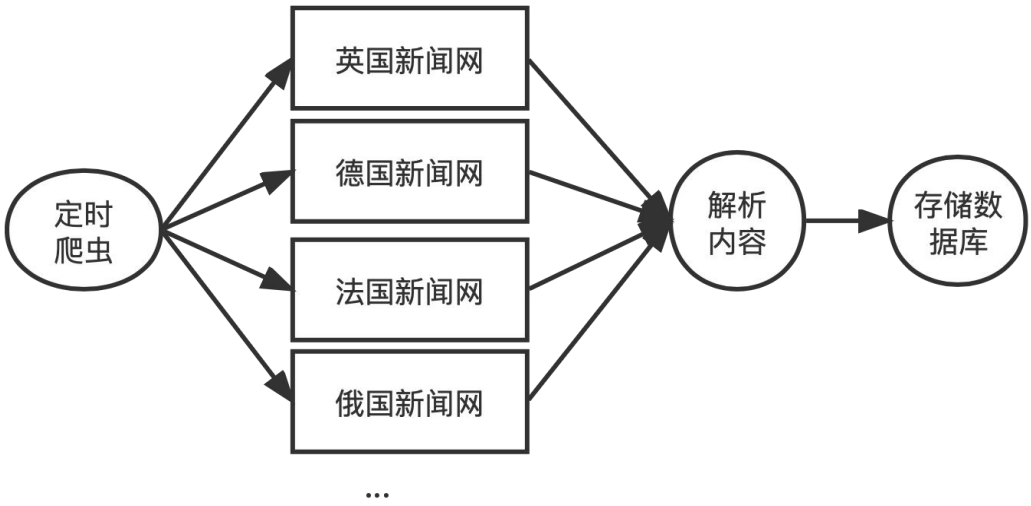

图 2.7 爬取解析流程

(1) 服务器挂载定时爬虫执行采集流程，服务器端设置定时采集任务时间点，时间点时即请求采集接口，如图 2.8 所示。

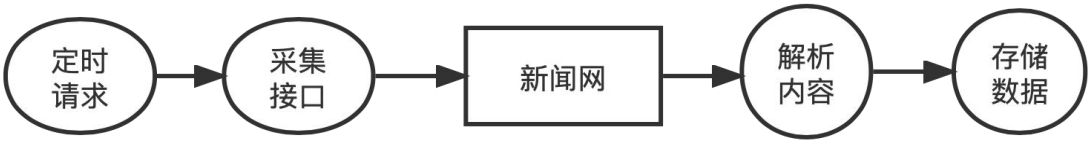

图 2.8 爬虫抓取

流程：定时自动请求 => URL 请求采集接口 => 解析 HTML => 存储采集结果

2.3 可行性分析

在应用平台高速发展产生了一端开发多端应用场景，同时也逐步迭代了单一开发模式的环境。一端开发多端应用以“中式外语”为实例，本应用前端 Uni-app 构建。

后端统一使用一端开发可多端部署的谷歌 Golang 语言创建，服务采用宝塔控制面板便捷管理，实例应用采用模块堆叠式开发，实现应用后期开发堆叠追加，在追加功能时易于二开，在经济上自身全栈技术投入成本低，一次开发多终端使用。

2.4 分析小结

对 Uni-app 实现多端应用是当下市面中小型企业或小项目最佳选择，一个应用开发即一同时间线实现多个应用端。对一端开发多端应用满足更多平台小程序环境，对人力物力实现了经济可行性。本实例应用是一个翻译与自定义单词库的软件作为部分一端开发多端应用的例子，实例应用包含用户注册、用户签到、用户绑定手机、用户加入书本、用户创建书本、用户编辑单词、用户创建单词、用户翻译单词等，在多端环境下功能一致。爬虫抓取多国语言新闻资讯内容，满足多语言学习实时新词等。应用主要体现以应用在多端相同模式相同样式下相同功能，开发一端开发即可实现多端环境使用。

实例应用“中式外语”技术栈中，采用了 Uni-app 框架的 Vue3.js 语法 Setup 模式，快捷构建了应用的实现，翻译模块功能采用了百度翻译 API 接口，外文模块使用 Golang 语言做爬虫采集数据，体现了多线程语言的优越性。

3 概要设计

3.1 概要设计

经过实习了解和学习后，多端应用的实现以“中式外语”实例应用采用 Uni-app 框架开发，分别是用户通过平台快捷登录和 SMS 登录块、用户信息块、书本块、单词块、翻译模块和外文模块等。应用总功能结构图如 3.1 所示。

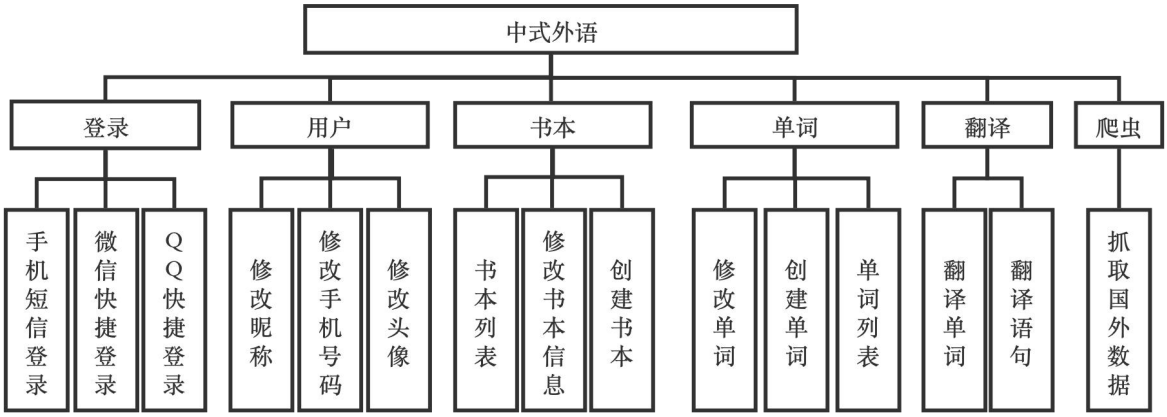

图 3.1 功能架构图

3.1.1 登录设计

登录功能设计了多个平台中的快捷登录是一个运筹帷幄的角色，其核心功能是快捷登录。主要功能有：手机 SMS 短信登录、腾讯微信快捷登录和腾讯 QQ 快捷登录。

（1）手机短信登录

手机短信登录通过百度 SMS 的 API 接口实现短信请求，短信发送前提需要在百度 SMS 中申请短信模板，通过 API 发出短信请求，用户接收服务端发出的短信验证码进行校验，即可返回用户数据。如果没有该用户账号则自动创建新账号。

（2）平台快捷登录

通过所属平台快捷账号登录，微信登录在微信小程序环境下，由用户点击唤起平台快捷登录，获得 code 提交后端获取唯一 openid 信息[6]。用户快捷登录结构图如 3.2 所示。

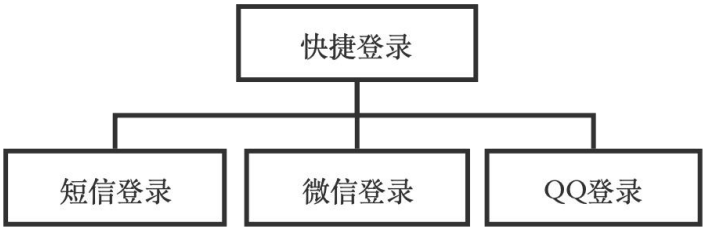

图 3.2 快捷登录

3.1.2 用户设计

用户功能设计了快捷登录、修改昵称、修改手机号码和、绑定手机号码、修改头像和签到，对书本模块有加入书本、管理书本、编辑单词和创建单词等。

(1) 快捷登录：用户通过平台快捷登录或短信模式进行登录，并进入应用页面。

(2) 修改昵称：通过“主页”左侧用户信息展示弹出层，对点击昵称进行修改。

(3) 修改手机号：用户通过个人信息模块短信验证码来更换手机号码。

(4) 绑定手机号：用户未绑定过手机号码可选择绑定手机号码，手机号接收短信验证提交绑定。

(5) 选择书本：用户在“书架”页面选择需要的书，加入书本学习书本内容。

(6) 创建书本：用户创建自主书本，可创建收费书本或个人私用书本，根据书本类型选择书本类型模式，以及书本收费模式和公开模式。

(7) 管理自建书本：用户可以管理自己所创建的书本内容和书本基本信息。

(8) 编辑书本单词：用户加入学习的书本根据权限进行编辑或新建单词。

用户设计结构图如 3.3 所示。

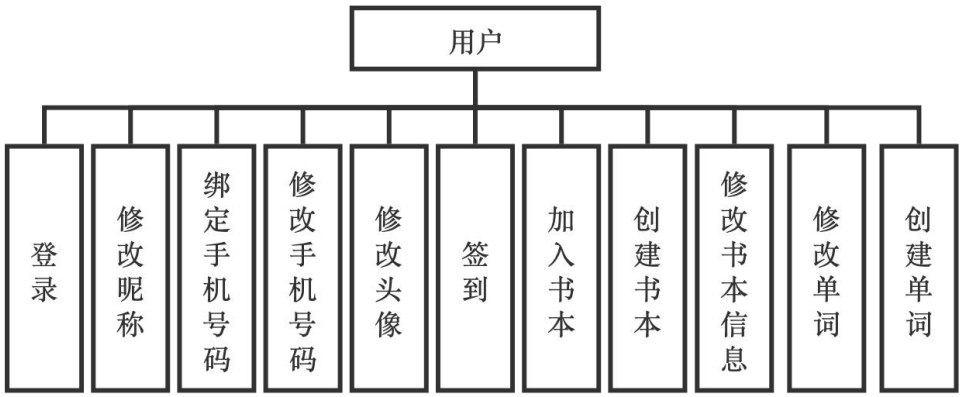

图 3.3 用户结构

3.1.3 外文资讯设计

外文资讯，栏目顶部设计多个文章的类别按钮，实现外文多个不同国家语言进行浏览，对此如下设计。外文数据采用 Golang 语言 Goquery NewDocument 脚本爬虫抓取数据，并对其进行 HTML 标签内容进行解析转码存储。结构图如 3.4 所示。

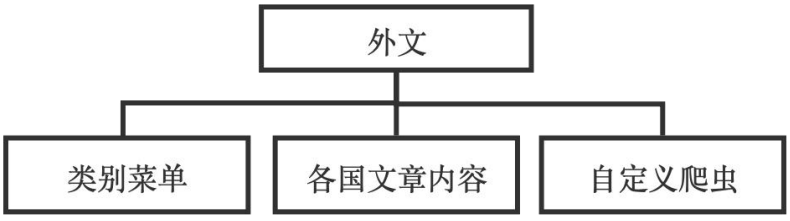

图 3.4 外文架构图

3.1.4 翻译功能设计

翻译功能，翻译采用翻译平台 API 进行 Get 请求，将翻译文本内容通过接口返回结果，支持几十个国家语言翻译。设计蓝图如 3.5 所示，结构图如 3.6 所示。

图 3.5 翻译功能设计蓝图

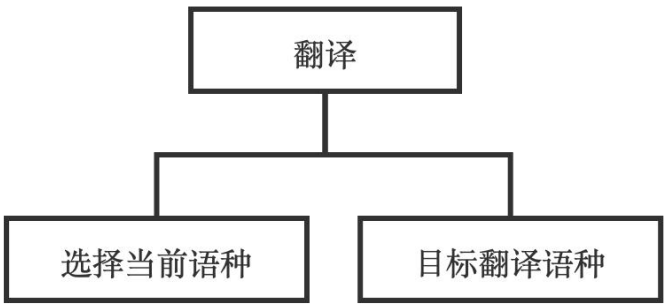

图 3.6 翻译功能结构图

3.1.5 学习模块设计

学习功能，学习单词支持音标、单词、文章、汉字等子组件构造。设计蓝图如 3.7 所示，结构图如 3.8 所示。

图 3.7 单词学习设计蓝图

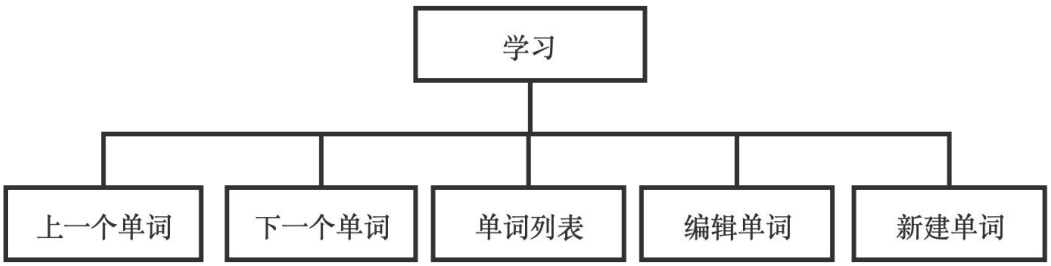

图 3.8 单词学习功能结构

3.1.6 创建单词设计

创建单词功能，学习单词对象组由单词主语、音标、词译、语言标识、词语列表等。设计蓝图如 3.9 所示，结构图如 3.10 所示。

图 3.9 单词学习设计蓝图

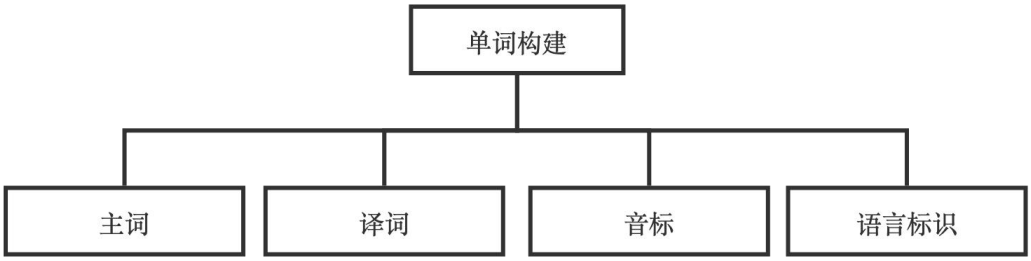

图 3.10 单词构建结构图

3.1.7 书本设计

书本列表，书本列表菜单分别由全部、平台、个人、我的、上传等功能。

创建书本，创建书本支持用户个人私有、免费、其他用户可编辑、书本模式等。

设计蓝图如 3.11 所示，结构图如 3.12 所示。

图 3.11 书本列表设计蓝图

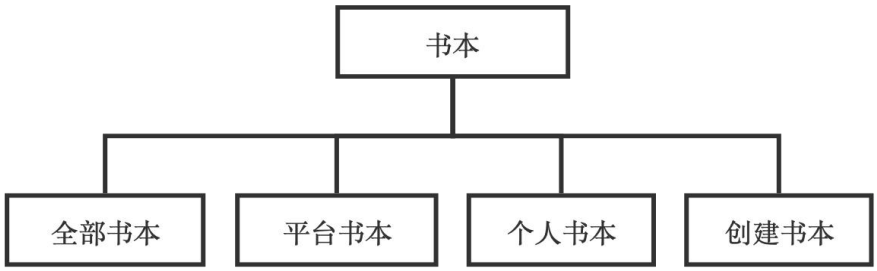

图 3.12 书本功能架构图

3.2 应用类图

实例应用使用类图形象的展示静态结构，为实例应用的各个类关系描述。本应用实例主要类有用户类和书本类等类，实例应用类图描述基本功能和关联模块使开发模块明朗，如图 3.13 所示。

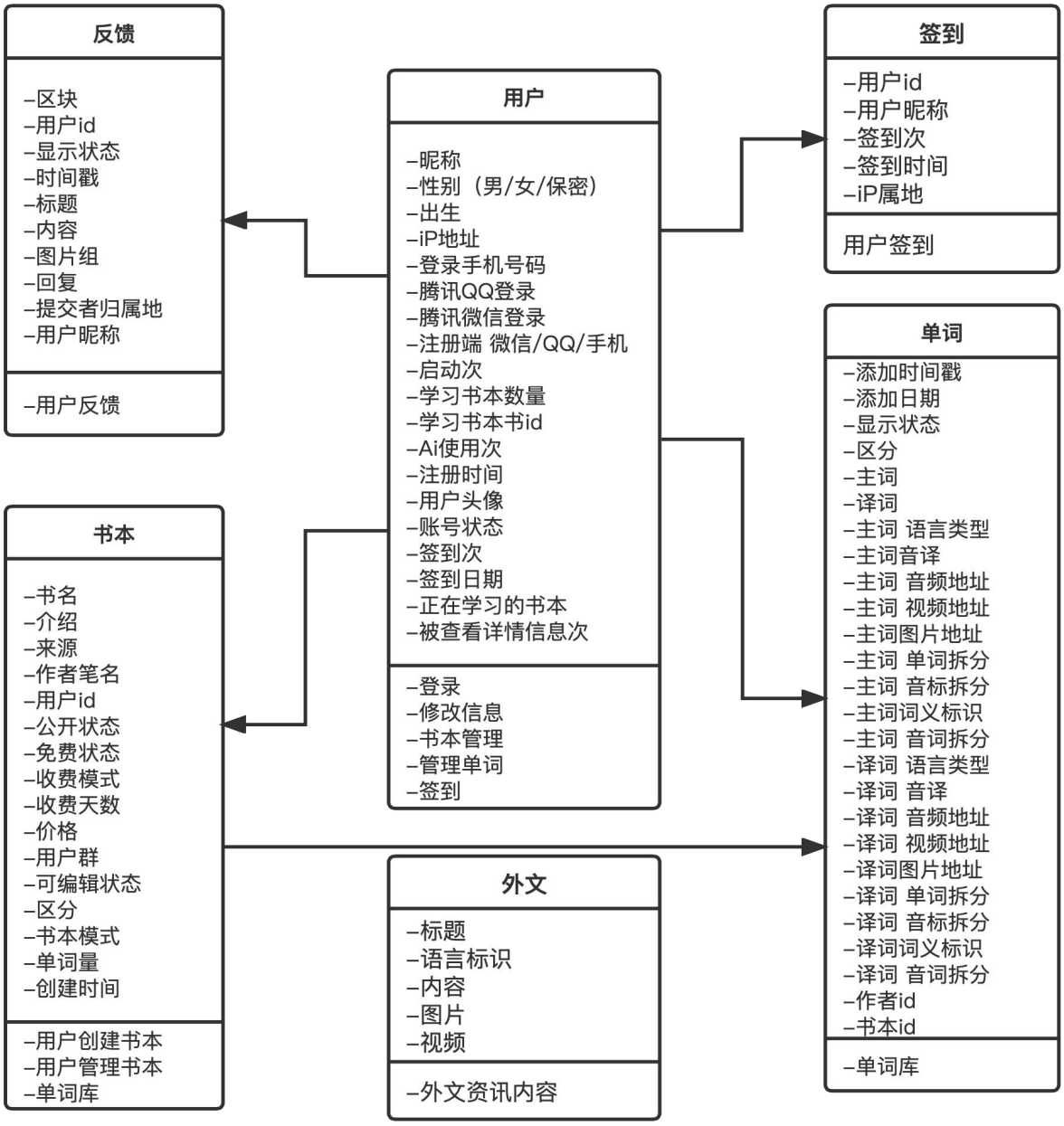

图 3.13 实例应用类图

3.2.1 添加书本序列图

用户在书架页面进行书本创建和管理书本，功能需求设计用户对书本操作实现。时序图如 3.14 所示。

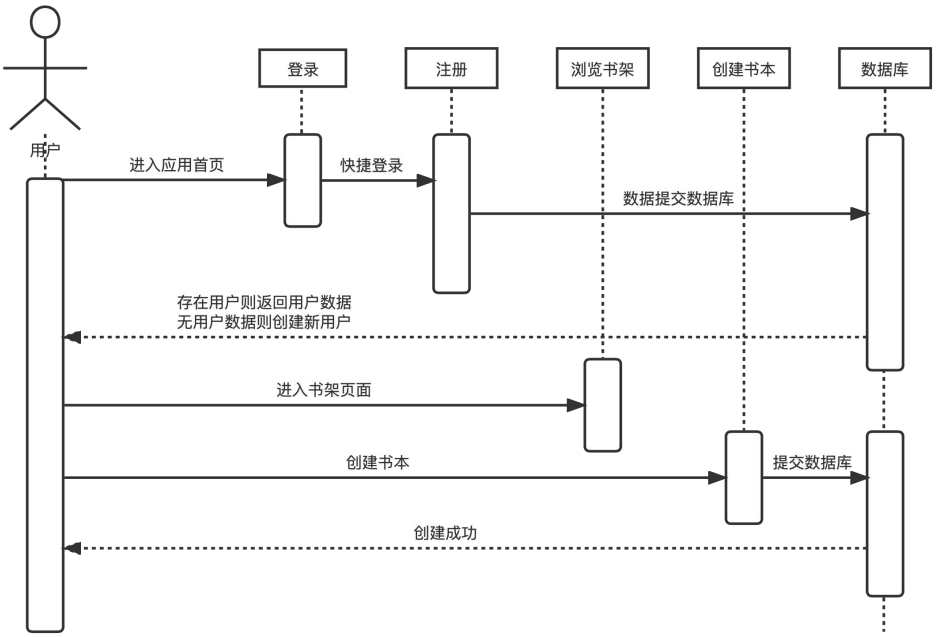

图 3.14 添加书本时序图

3.2.2 加入书本序列图

用户在书架页面进行书本选择，加入需要学习的书本。实例功能设计用户对书本选择加入本书学习的时序图，如图 3.15 所示。

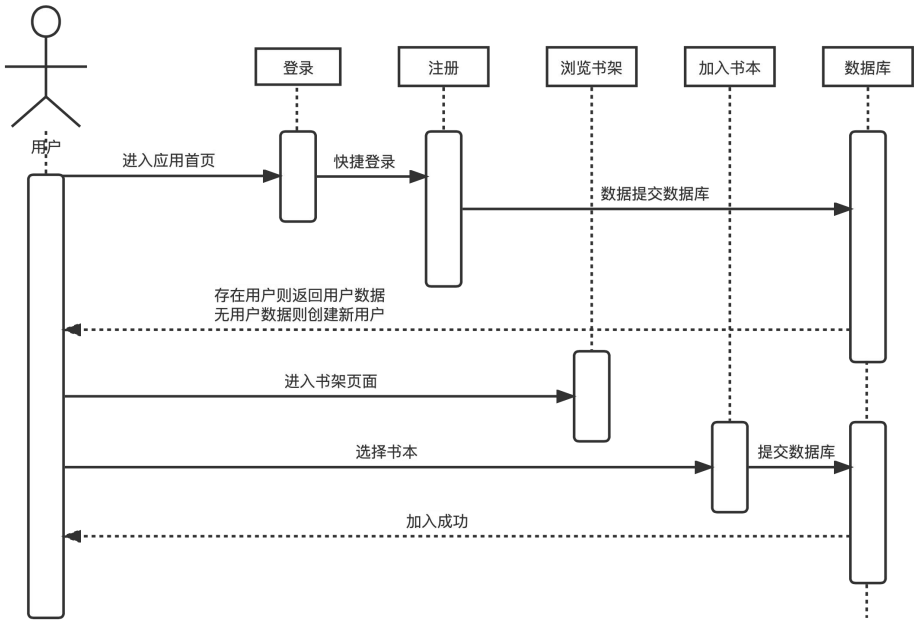

图 3.15 书本时序图

3.2.3 创建单词序列图

单词功能是用户对正在学习的书本所属单词进行编辑和创建新单词，单词功能时序图，如图 3.16 所示。

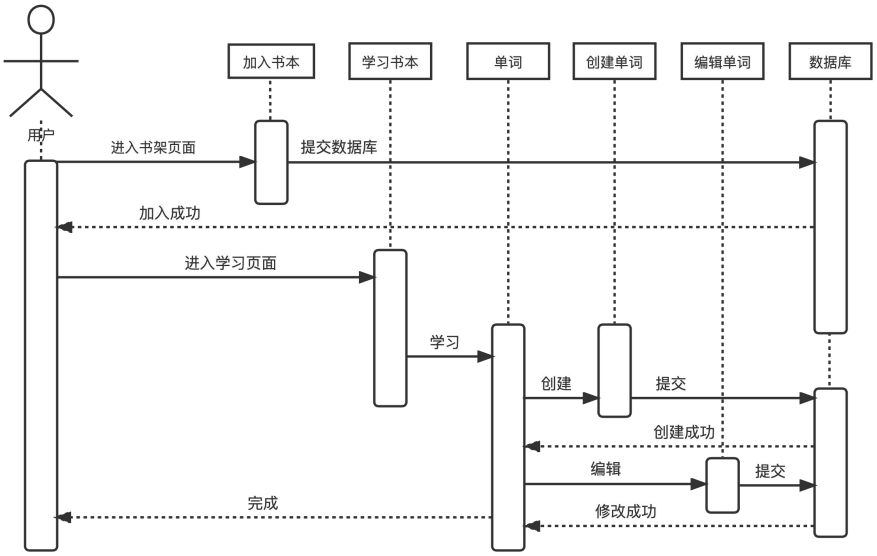

图 3.16 编辑和创建新单词时序图

3.3 活动图

以活动图描述“中式外语”实例功能活动组成，活动的描述和活动执行完了之后，执行下一个进程。在实例应用业务存在逻辑比较复杂时，采用活动图细化实例复杂流程。

（1）实例应用用户修改手机号活动图，如图 3.17 所示。

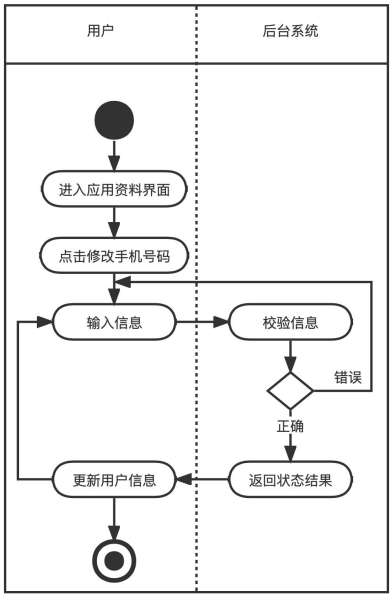

图 3.17 修改手机号活动图

（2）实例应用创建书本活动图，用户在书架页面创建书本，点击创建书本输入书本信息提交请求，后端创建书本后触发创建单词库表的程序活动过程，如图 3.18 所示。

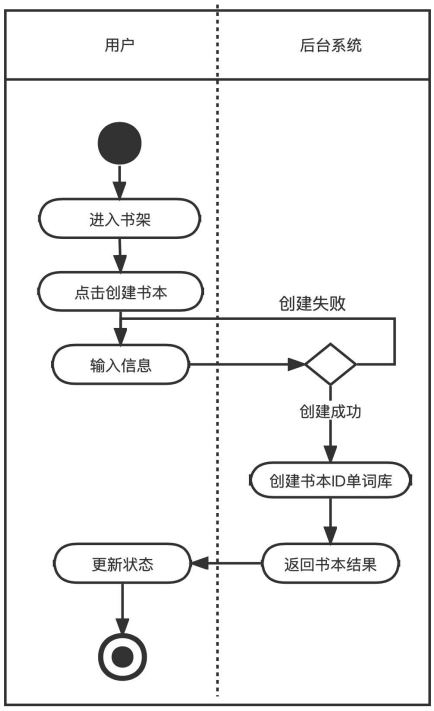

图 3.18 书本创建活动图

（3）实例应用用户发起自定义爬虫采集活动图，用户发起采集请求，程序活动过程，如图 3.19 所示。

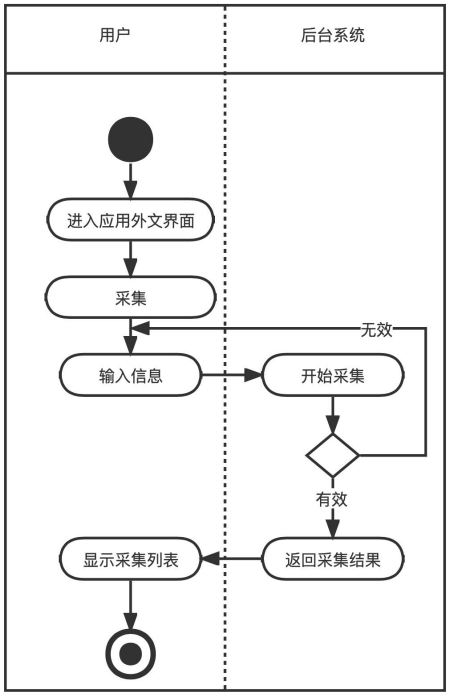

图 3.19 自定义爬虫采集活动图

3.4 状态图

本实例应用程序部分对象过程以状态图展现，描述程序事件转换时的过程。 在实例应用改变对象状态行为，以用户在提交书本创建后进行的行为，只有在提交书本创建k 成功后，后端业务逻辑才会创建单词库表，如果创建书本失败，则不会创建单词库表。

对外国文章资讯内容进行爬取，对爬取的前提需要做采集日志隔离爬虫请求，避免IP 被封锁，爬虫抓取状态功能，当爬虫即将采集指导 url 时，将会检查本地 SQL 存在爬取状态，以保障每次采集时不会过多采集次数，完成采集后即改变爬取状态。

创建书本库表状态图，如图 3.20 所示。

爬虫抓取外文内容状态图，如图 3.21 所示。

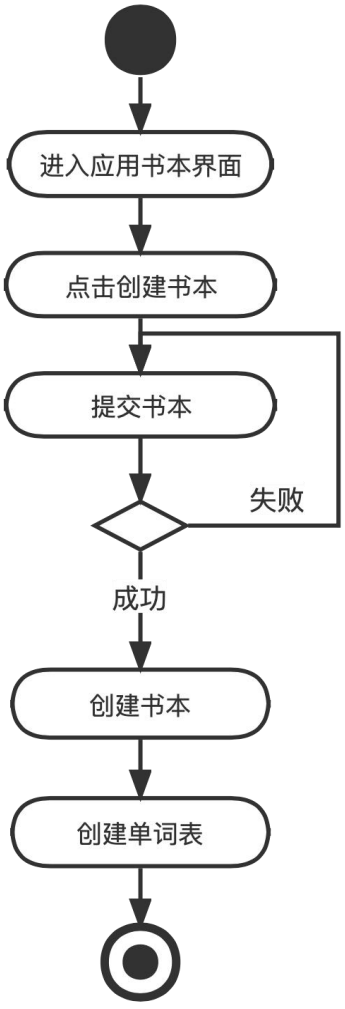

图 3.20 创建书本库表状态图

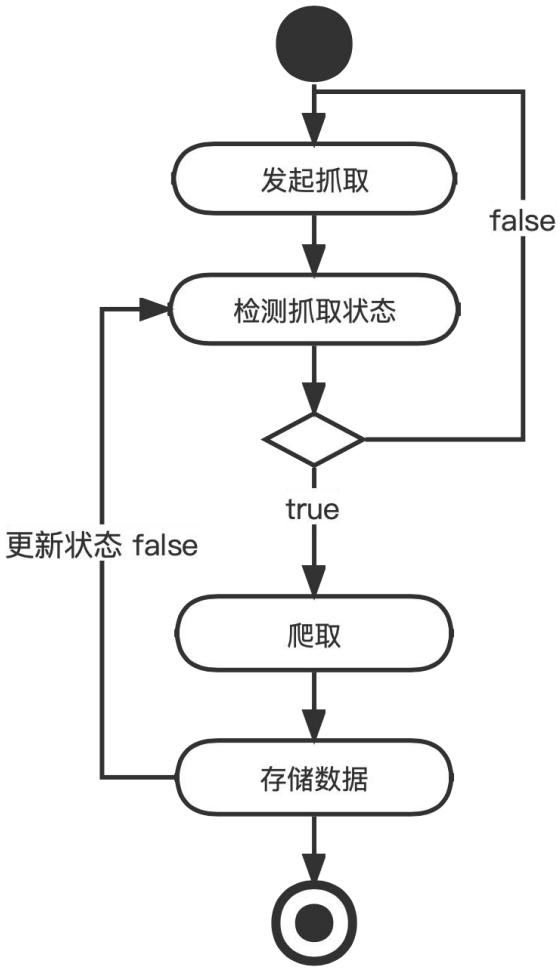

图 3.21 爬虫抓取状态图

3.5 数据库设计

3.5.1 结构设计

实例应用的结构设计使用 E-R 图描述。对实例应用功能需求中存在用户、签到、反馈、书本、翻译等实体，如图 3.22 所示。

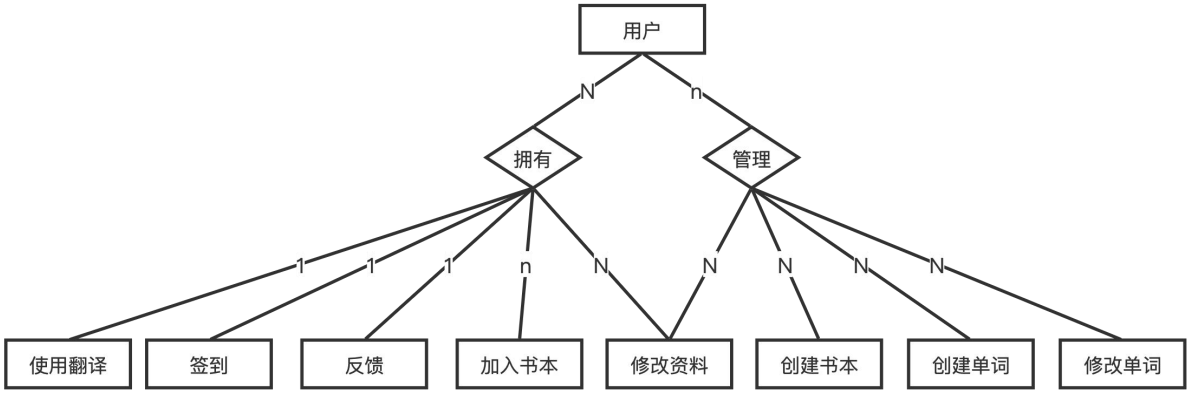

图 3.22 用户实体 ER 图

(1) 用户实体的属性有 ID、头像、昵称、性别、用户注册时间、Ai 使用次、微信号 ID、QQ 号 ID 等，如图 3.23 所示。

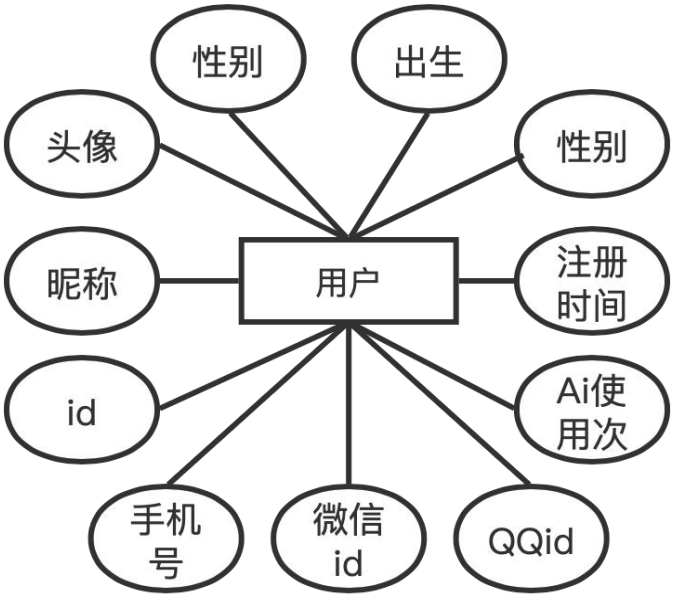

图 3.23 用户实体

(2) 实例应用签到实体的属性有用户 ID、列表 ID、属地 IP 信息等，如图 3.24 所示。

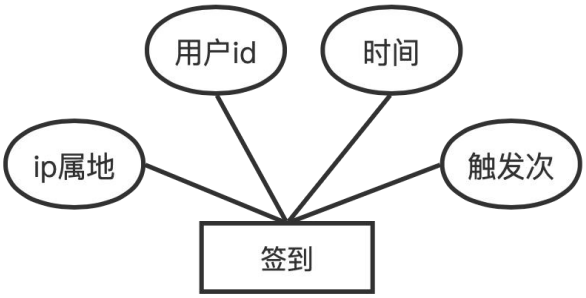

图 3.24 签到实体

(3) 反馈实体的属于有标题、内容、添加时间等等，如图 3.25 所示。

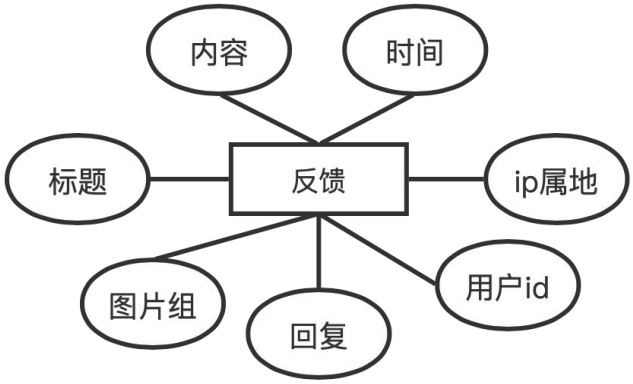

图 3.25 反馈实体

(4) 实例应用书本模块，功能实体的属性有书本名称、书本作者用户 ID、书本介绍作者笔名、书本模式等，如图 3.26 所示。

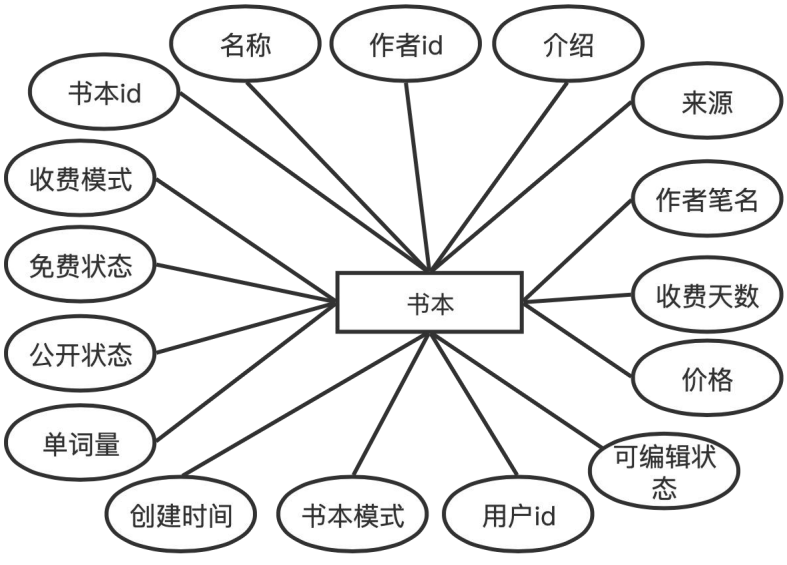

图 3.26 书本实体

（5）单词表实体的属性有创建时间、可用状态、主词和译词等，主词与译词属性相同，例如仅展示主词，如图 3.27 所示。

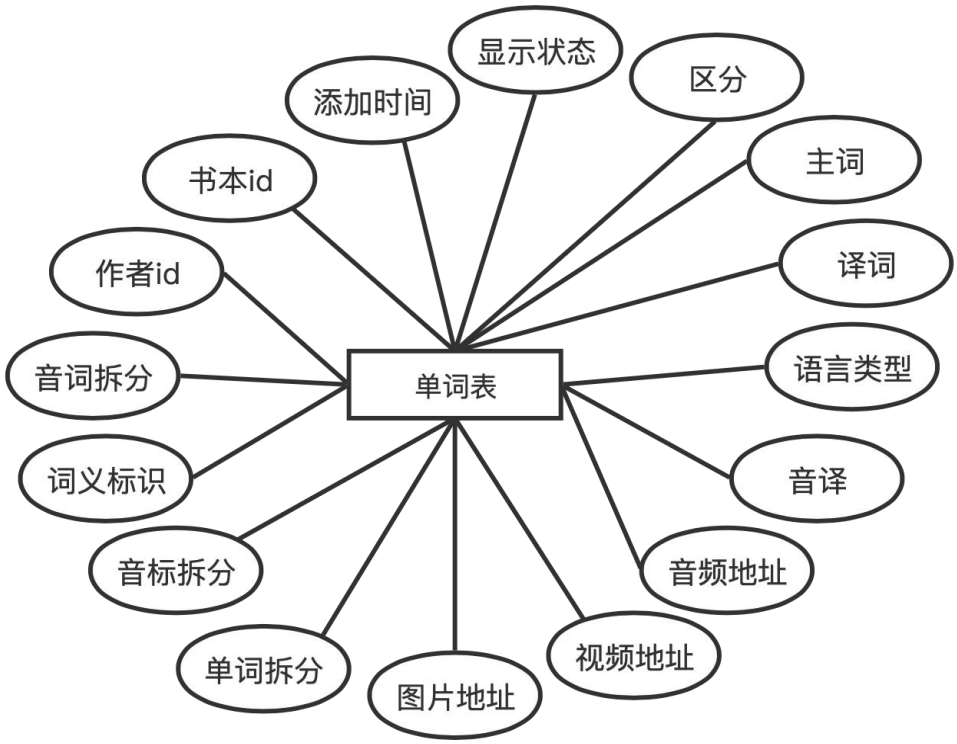

图 3.27 单词表实体

（6）翻译表实体的属性有翻译语言类型即含主词和译词等，如图 3.28 所示。

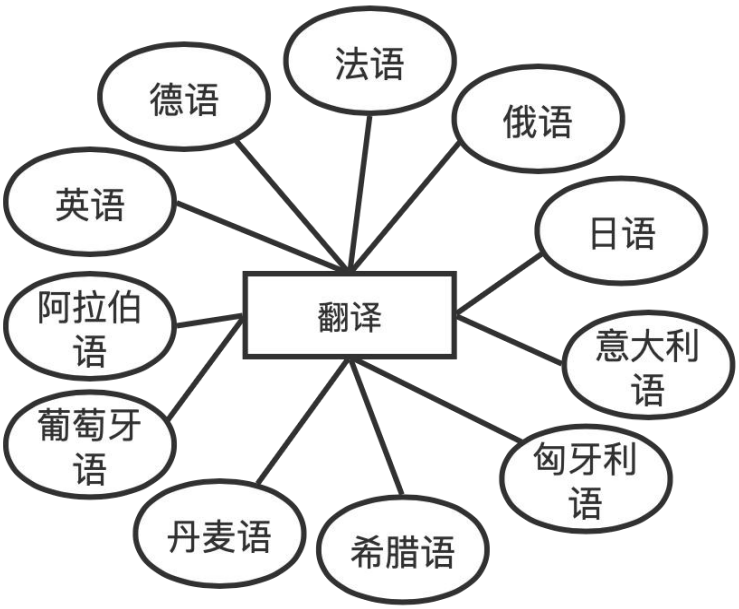

图 3.28 翻译表实体

3.5.2 数据库表设计

1、用户表。存储使用者基本信息，存储用户昵称、ID、性别、性别、出生、IP 地址、腾讯 QQ 登录标识 ID、腾讯微信登录标识 ID 等，如表 3.1 所示。

表 3.1 用户数据库表

字段	数据类型	数据说明	备注
id	int	主键	uid
name	varchar	varying	昵称
gender	varchar	varying	性别
birth	varchar	varying	出生
iPaddress	varchar	varying	iP 地址
phone	varchar	varying	登录手机号码
qqid	varchar	varying	腾讯 QQ 登录
wxid	varchar	varying	腾讯微信登录
register_blok	varchar	varying	注册端标记
intVisit	int2	0	启动次
intBook	int2	0	学习书本数量
arrBook	text	varying	学习书本书 id
intUseAi	int2	0	Ai 使用次
date	varchar	varying	注册时间
img	varchar	varying	用户头像
intShow	int2	0	账号状态
intSignin	int2	0	签到次
signin_date	varchar	varying	签到日期
book_id	int8	0	正在学习的书本
intQuery	int2	0	被查看详情次
login	varchar	varying	上次登录日志
updatelog	varchar	varying	操作日志

2、签到表。用户签到日志信息记录，存储用户每日签到日志，如用户 ID、用户昵称、签到次、签到时间、IP 属地等信息，如表 3.2 所示。

表 3.2 签到数据库表

字段	数据类型	数据说明	备注
id	int	主键	
user_id	int8	0	用户 id
cuser_name	varchar	varying	用户昵称
intSignin	int2	0	签到次
date	varchar	varying	签到时间
iPaddress	varchar	varying	iP 属地

3、反馈表。应用反馈信息存储，如反馈者 ID、反馈者昵称、反馈问题标题、反馈问题内容、反馈者 IP 属地等信息，如表 3.3 所示。

表 3.3 反馈数据库表

字段	数据类型	数据说明	备注
id	int	主键	
blok	varchar	0	区块
user_id	int4	0	提交用户 id
intShow	int2	0	显示状态
date	varchar	varying	时间戳
title	varchar	varying	标题
text	varchar	text	内容
arrlmg	text	text	图片组
reply	varchar	varying	回复
iPaddress	varchar	varying	提交者归属地
ip	varchar	varying	ip
user_name	varchar	varying	用户昵称

4、广告轮播图表。广告轮播表存储标题、图片 URL 和 IP 属地、显示状态、结束时间等内容，如表 3.4 所示。

表 3.4 反馈数据库表

字段	数据类型	数据说明	备注
id	int	主键	
blok	varchar	varying	区分
intShow	int	0	显示状态
date	varchar	varying	时间
date_end	varchar	varying	结束时间
name	varchar	varying	标题
text	varchar	varying	内容
img	varchar	varying	图片
video	varchar	varying	视频
url	varchar	varying	

5、爬虫采集表。抓取数据存储的表前提需要一条校验是否到采集时间或者在已采集过了；对采集后所存储的对应数据字段有：区分类型、模式、标题、内容、时间等信息，如表 3.5 所示。

表 3.5 爬虫采集数据库表

字段	数据类型	数据说明	备注
id	int	主键	
blok	varchar	varying	区分
date	varchar	varying	时间
name	varchar	varying	标题
text	varchar	varying	内容
arrimg	varchar	varying	图片组
video	varchar	varying	视频
sign	varchar	varying	模式

6、书本表。书本表存储用户创建的对应书本内容，如用户 ID、书名、介绍、来源作者笔名、用户 ID、公开状态、用户群等信息，同时关联单词库表，如表 3.6 所示。

表 3.6 书本数据库表

字段	数据类型	数据说明	备注
id	int	主键	
name	varchar	varying	书名
text	varchar	varying	介绍
source	varchar	varying	来源
author	varchar	varying	作者笔名
user_id	int8	0	用户 id
tfOvert	bool	true	公开状态
intPrice	int8	0	价格
arrUser	text	text	用户群
tfEditor	bool	true	可编辑状态
blok	varchar	varying	区分
mode	varchar	varying	模式
icon	varchar	varying	书图标
img	varchar	varying	书封面图
intShow	int2	2	显示状态
intWord	int2	0	单词量
date	varchar	varying	创建时间
intWord	int2	0	关联单词状态
word_id	varchar	varying	关联单词
book_up	varchar	varying	书本过滤
book_no	varchar	varying	禁止 id 群
userlogin	varchar	varying	用户修改日志
updatelog	varchar	varying	操作日志

7、单词表。单词表存储对于书本的单词条，如主词、译词、语言类型、单词拆分、音标拆分、译词语言类型等信息，同时关联书本库表，如表 3.7 所示。

表 3.7 单词数据库表

字段	数据类型	数据说明	备注
id	int4	主键	
tfShow	bool	true	显示状态
blok	varchar	varying	区分
name	varchar	varying	主词
text	varchar	varying	译词
name_blok	varchar	varying	主词语言类型
namesign	varchar	varying	主词音译
name_audio	varchar	varying	主词音频地址
name_video	varchar	varying	主词视频地址
name_image	varchar	varying	主词图片地址
name_arrSplit	varchar	varying	主词单词拆分
name_sign_arrsplit	varchar	varying	主词音标拆分
name_arrExplain	json	null	主词词义
text_blok	varchar	varying	译词语言类型
text_Sign	varchar	varying	译词音译
text_audio	varchar	varying	译词音频地址
text_video	varchar	varying	译词视频地址
text_image	varchar	varying	译词图片地址
text_arrsplit	varchar	varying	译词单词拆分
text_sign_arrSplit	varchar	varying	译词音标拆分
text_arrExplain	json	null	译词词义
user_id	int8	0	作者 id
book_id	int8	0	书本 id

3.6 设计小结

应用设计主要体现一端开发多端应用实现的主要设计。在经过实例的设计中前端效果充分体现多端环境的功能一致性和 UI 兼容一致性[7]。对应用的后端数据处理业务逻辑中采用的谷歌 Golang 语言同样也是实现了多服务端环境的部署运行，后端在一端环境中开发后，可生成在 Windows 中的 exe 格式文件直接运行即可，也可以生成 MacOS 的可执行文件运行, 最常用的 Linux 系统生成的可运行文件更是兼容性最强, 由于 Golang 语言打包后生成的文件是运行系统的二进制文件，则不需要过多的环境配置，即实现了对一端开发多端运行条件。

爬虫设计是采集新闻资讯的必要条件，对实例应用的“外文”模式中所需的外国文章资讯内容所需，继而采用爬虫完成数据采集，爬虫采用 Golang 语言作为 http 请求基本栈，并使用 Goquery 作为标签选择器的脚本进行对 HTML 标签进行筛选和分割内容, 爬虫完成了采集数据进行格式内容转换，并最终将对应内容存储到 PostGreSQL 数据库字段中[8]。

4 应用实现

4.1 主页模块

4.1.1 主页界面

首页包含了顶部轮播图子组件，关于按钮，用户头像，签到按钮，栏目快捷组等；通过组件化层级展示模块，侧边模块采用弹出层组件实现，浮动模块 Flex 布局加 Position 相对定位设计实现，顶部组件横行布局，底部模块 CSS 绝对定位 vh 自适应高度设计，成果图如 4.1 所示。

图 4.1 主页界面成果图

4.1.2 主页代码构造

用户头像侧边弹框和关于窗口组件采用 Uni-popup 组件设计，底部快捷按钮组采用 Flex 布局加 Position 相对定位设计，采用 Vue3.js 语法实现业务逻辑。关键代码如下所示：

```
const popupShowAbout = ref(null)
const tapAbout = (e) => { //显示弹出层
	if (e == 0) {
		popupShowAbout.value.open();
	} else {
		popupShowAbout.value.close();
	}
	if (e == 1) {
		uni.navigateTo({
			url: '/pages/content/HelpList'
		});
	} else if (e == 2) {
		uni.navigateTo({
			url: '/pages/content/Feedback'
		});
	} else if (e == 3) {
		getHTTPAbout();
	}
}

const tapGoto = (e) => {
	if (e == 'xuexi') {
		uni.switchTab({
			url: '/pages/index/xuexi'
		});
	} else if (e == 'fanyi') {
		uni.switchTab({
			url: '/pages/index/fanyi'
		});
	}
}
```

4.2 外文模块

4.2.1 外文界面

外文资讯模块包含了顶部菜单按钮组件，子模块设计了国旗显示、文章标识、语言标识，外文标题和内容，图片轮播组等。模块布局顶部采用 scroll-view 属性组件实现横向菜单滚动，成果图如 4.2 所示。

外文内容图片组采用 swiper 属性组件实现多图片横向滑动展示；菜单栏附带用户自定义采集菜单，即采用 uni-popup 外部组件作为弹出层自定义采集窗口，自定义爬虫采集网页数据经过提交参数后经后端 Golang 语言进行采集内容后 Goquery NewDocument 脚本解析成所需的内容返回，功能成果图如 4.3 所示。

图 4.2 外文资讯界面成果图

图 4.3 自定义爬虫采集图

4.2.2 外文代码构造

外文资讯内容采用爬虫抓取，经过 Golang 语言的 Goquery NewDocument 脚本作为辅助抓取数据定时抓取外国新闻网数据。关键代码如下所示：

```
doc.Find("body").Each(func(i int, s *goquery.Selection) {
```

```
    band := s.Find(divName)                                    //主标签
    band.Each(func(ii int, ss *goquery.Selection) { //获取每一个
        title := ss.Find(divTitle).Text()        //标题
        content := ss.Find(divContent).Text() //内容
        img, _ := ss.Find(divImg).Html()          //为属性名称  采集图片
        title = strings.Replace(title, "\n", "", -1)
        content = strings.Replace(content, "\n", "", -1)

        /*  存储  */
        var texts string
        if divName != "" && title != "" {
            ints++
            texts = "序号[No]：" + strconv.Itoa(ints)
            texts += "\n 标题[title]：" + title
            if divContent != "" {
                texts += "\n 内容[content]：" + content
            }
            if divImg != "" {
                texts += "\n 图片[img]：" + img
            }
        texts += "\n 时间[date]：" + time.Now().Format("2006-01-02 15:04:05")
        texts += "\n 时戳[time]：" + strconv.FormatInt(time.Now().UnixMilli(), 10)
            texts += "\n\n"
        }
        Arr = append(Arr, texts)
    })
})
ret.Info = "请求成功  [" + time.Now().Format("2006-01-02 15:04:05") + "]"
ret.Msg = "请求成功"
ret.List = Arr
return ret
```

4.3 翻译模块

4.3.1 翻译界面

翻译模块包含了图片轮播组件和翻译对象语言，支持多国家语言相互翻译，翻译结果包含了词语、句子和单词音标语音播放等；语言选择使用 uni-popup 弹出层组件实现语言选项列表，单词经过后端 API 接口返回翻译经过后模块化展示，而附带的单词读音语言经过 uni.createInnerAudioContext()内置属性进行触发播放。翻译功能成果图如 4.4 所示，翻译选择功能如图 4.5 所示。

图 4.4 翻译界面成果图

图 4.5 翻译语言选择图

4.3.2 翻译代码构造

翻译采用百度 API 接口处理翻译内容，翻译返回结构将转成前端所属的 Json 渲染展示格式，并返回前端网络请求，关键代码如下所示：

```
resp, err := client.Get(url) //get 请求链接
var Objs Words
```

```
    json.Unmarshal([]byte(body), &Objs) //结果解析
    if Objs.Trans_result != nil {
        str := Objs.Trans_result[0].Dict
        var jsonMap map[string]interface{}      //转 map
        json.Unmarshal([]byte(str), &jsonMap) //map 转 json
        bodyJsons, _ := json.Marshal(jsonMap) //json
        v, _ := UnescapeUnicode(bodyJsons)
        json.Unmarshal(v, &Objs.Trans_result[0].Dicts)
    }
    if Objs.Trans_result != nil && Objs.Trans_result[0].Dicts.Word_result.Simple_means.Symbols != nil
{
        Objs.From_sign = Objs.Trans_result[0].Dicts.Word_result.Simple_means.Symbols[0].Ph_am
        Objs.To_sign = Objs.Trans_result[0].Dicts.Word_result.Simple_means.Symbols[0].Ph_en
    Objs.From_word = Objs.Trans_result[0].Dicts.Word_result.Edict.Item /
        Objs.To_word = Objs.From_word //复制格式
    if Objs.Trans_result != nil {
            str := Objs.Trans_result[0].Dict
            var jsonMap map[string]interface{}
            json.Unmarshal([]byte(str), &jsonMap) //map 转 json
            json.Unmarshal(v, &Objs.Trans_result[0].Dicts) //存结构体
        }
        if Objs.To_word != nil && Objs.To_word[0].Tr_group != nil &&
Objs.To_word[0].Tr_group[0].Similar_word != nil && Objs.To_word[0].Tr_group[0].Similar_word[0] ==
"" {
            if Objs.To_word[0].Tr_group[0].Similar_word[0] == "" {
                Objs.To_word[0].Tr_group[0].Similar_word =
Objs.Trans_result[0].Dicts.Word_result.Simple_means.Word_means
            }
        }
    }
```

4.4 单词学习模块

4.4.1 学习界面

单词学习模块包含单词主词、音标和词义，并内置自动请求翻译端 API 完成音标语言和词译句子查询。单词列表数据使用 JSON.parse(JSON.stringify(arr))内置 JS 属性深度拷贝，实现数据新拷贝也可以使用 MAP 方式，即实现千万条数据流畅的读出，对单词词义列表数据由 API 调取译文返回前端进行渲染，成果图如 4.6 所示。

图 4.6 单词学习模块界面成果图

4.4.2 学习模块构造

学习模块采用组件化，实现不同类型书本内容的展示，通过 v-if 来显示相应的模板，关键代码如下所示：

```
<view v-if="user && user.book_id">
    <letterWord @PutBook="PutBook" />
</view>
```

4.5 创建书本模块

4.5.1 创建书本

实例应用创建书本模块，采用弹出层组件设计，控件包含书本名称输入框、书本介绍输入框、书本来源输入框、作者笔名输入框、书本公开状态按钮、书本可编辑状态按钮、书本免费状态按钮、书本展示内容模式状态按钮等。各功能点开关选项使用 radio 内置组件实现，弹出创建的窗口使用 uni-popup 弹出层组件实现，成果图如 4.7 所示。

图 4.7 创建书本界面成果图

4.5.2 创建单词构造

创建单词由 Golang 语言后端 POST 的 ParseForm 方式接收，并将内容传递给创建书本方法涵提交 PostgreSQL 数据库构建书本。关键代码如下所示：

```
func getAddBook(writer http.ResponseWriter, request *http.Request) {
    request.ParseForm()
```

```
    blok := request.Form.Get("blok")
    mode := request.Form.Get("mode") ...
    obj := Book.AddBook(...)
    msg, _ := json.Marshal(obj)
    writer.Write(msg)
}
func AddBook(blok string, mode string, name string, text string, source string, author string, tfOvert string,
tfFree string,tfEditor string, user_id string) interface{} {
    var ret SQL.Returns
    uinx := strconv.FormatInt(time.Now().UnixMilli(), 10)
    img := "/upload/user/head-img/jpg/" + uinx[len(uinx)-2:] + ".jpg"
    sql := `INSERT INTO "info"."books"
("img","blok","mode","name","text","source","author","tfOvert","tfFree","tfEditor")VALUES `
    sql += `($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12) RETURNING "id";`
    var ids int
    err := PgDB.QueryRow(sql, uinx, img, mode, mode, name, text, source, author, tfOvert, tfFree,
tfEditor, user_id).Scan(&ids)
    if err != nil {
        ret.Code = -400
        ret.Msg = "创建书本失败"
        return ret
    }
    _, err = PgDB.Exec(sql)
    if err != nil {
        ret.Code = -400
        ret.Msg = "创建书本失败"
        return ret
    }
    ret.Code = ids
    ret.Msg = "创建书本成功"
    return ret
}
```

4.6 其他模块

4.6.1 书架模块

书架模块是所有书本的神经中枢，即单词学习栏目的主要构成，如图 4.8 所示。

图 4.8 书本架界面成果图

4.6.2 用户信息

用户信息模块汇聚用户多个信息项列表和用户各类数据印记，成果图如 4.9 所示。

图 4.9 用户信息界面成果图

4.6.3 登录模块

登录模块中根据不同应用端平台展示，所属平台的快捷登录。成果图如 4.10 所示。

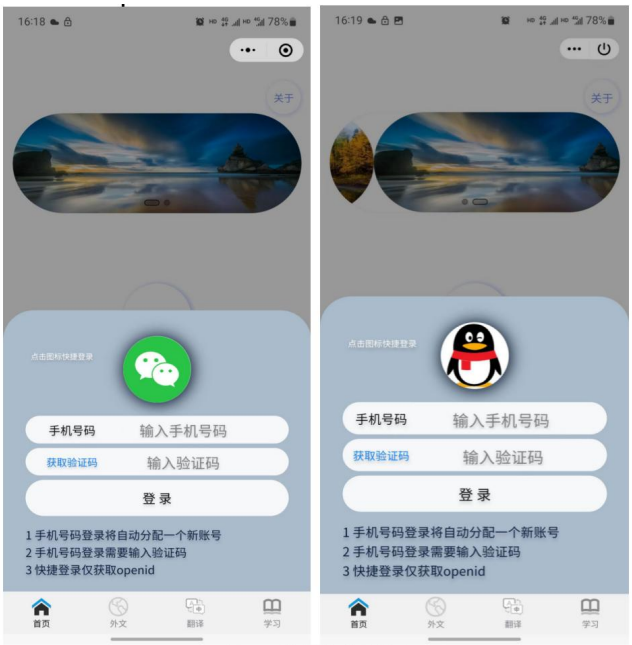

图 4.10 登录界面成果图

4.6.4 主页侧边

主页侧边栏采用 Uni-popup 组件实现多端一致弹出层模式，即展示了用户头像和用户昵称、标用户 UID 和各类信息，并附有绑定手机号的组件块等。成果图如 4.11 所示。

图 4.11 侧边栏用信息界面成果图

4.6.5 反馈模块

反馈模块即采用单独表单，设计了反馈标题、反馈内容、反馈时间和回复内容。成果图如 4.12 所示。

图 4.12 反馈界面成果图

4.7 设计实现小结

实例设计实现了全部模块功能，并实现多端环境应用，同时以实例“中式外语”上架了腾讯微信和腾讯 QQ 小程序平台，也打包生成安卓 APK 文件和 WebH5 端。

对多端应用的设计与实现，在本次实例设计中已完美实现全部效果，多个平台终端功能一致性，UI 界面兼容一致，实现了一端开发多端应用。后端业务对 Golang 语言高并发能力中对 Http HandleFunc 做接口处理并支持 go 异步方法, 查询类接口采用 Request URL Query 方法，对数据更改采用 Request ParseForm 方法，来实现接收对数据业务处理。

5 应用测试

5.1 测试分析

5.1.1 应用说明

针对多端使用，微信小程序端、QQ 小程序端、安卓 APK 和苹果 IOS 等端一键开发，后端技术采用谷歌 Golang 语言完成 http.HandleFunc 的请求入口 POST 和 GET 接口编码设计。本测例以多端环境下的测试，践行一端开发多端应用效果，如表 5.1 所示。

表 5.1 设计架构

模块	架构	工具
前端	Uni-app 框架	HBuilderX
后端	Golang	VS Code
数据库	PostGreSQL	DBeaver
API	Baidu API	Baidu

5.1.2 功能性需求

“中式外语”实例应用功能需求如表 5.2 所示。

表 5.2 功能需求表

功能	子功能
快捷登录	手机短信登录/微信快捷登录/QQ 快捷登录
签到	每日签到
修改账号资料	修改手机号码/ 昵称/性别/出生/头像
书本	加入书本/删除书本/管理书本/新增书本
单词	修改单词/新增单词/删除单词
外文	各国语言文章资讯内容
翻译	单词翻译/中文语句翻译/音标播放

5.1.3 非功能性需求

实例“中式外语”小程序多端部署，本次测试也需要测试小程序在多终端兼容性，后端可在多端系统部署运行。

5.2 实例测试

5.2.1 测试环境

实例项目测试如下：

（1）硬件，如表 5.3 所示。

表 5.3 硬件清单表

硬件设备	处理器型号	内存
PC	2.6 GHz 六核 Intel Core i7	16GB
手机	三星 Note 10	6GB

（2）应用端，如表 5.4 所示。

表 5.4 应用端清单表

应用端	环境	版本
微信小程序端	安卓微信小程序&IOS 微信小程序	8.0.32
QQ 小程序端	安卓 QQ 小程序&IOSQQ 小程序	8.9.38
安卓 APP	三星 Note 10	12.0
WebH5	MS Edge	12

5.2.2 测试例

测试项根据本应用实现多端需要下，满足多端一致的功能块进行测试，多端平台快捷登录实现登录授权权限测试，在多端一致的易用性进行导航和布局模式等功能测试。前端请求 Golang 后端的数据接口正确性测试，多端环境下 UI 兼容一致界面测试，对后端 Golang 语言编写的后端逻辑在服务器端并发承载的性能进行压力等测试。

5.3 测试用例

5.3.1 功能测试

（1）实例应用快捷登录测试，应用授权快捷登录功能测试用例，如表 5.5 所示。

表 5.5 快捷登录测试用例

用例编号	测试数据	执行步骤	预期结果	实际结果
A_001	平台快捷登录	没有权限时点击【快捷登录】按钮	1、弹出授权窗口 2、返回授权结果	用户同意授权，申请权限成功返回用户数据

A_002	手机号码登录	输入手机号码获取验证码，再提交验证码	1、短信接收验证码 2、短信验证成功返回用户信息	验证成功，返回用户数据

(2) 实例应用页面多端兼容测试,多端下各页面模块功能测试用例,如表 5.6 所示。

表 5.6 多端兼容测试用例

用例编号	测试数据	执行步骤	预期结果	实际结果
B_001	书架页面	点击书架菜单	跳转书架页面	进入书架页面
B_002	书架页面返回	在书架页面点击返回	跳转上一级页面	打开上一级页面
B_003	创建书本	在书架页面点击创建书本	弹出创建书本框	弹出创建书本框
B_004	删除书本	在书架页面点击自建书本	弹出删除选择框	弹出删除选择框
B_005	帮助页面	在主页点击帮助菜单	进入帮助列表页面	进入帮助列表页面
B_006	反馈页面	在主页点击反馈菜单	进入反馈页面	进入反馈页面
B_007	关于信息框	在主页点击关于菜单	弹出关于信息窗口	弹出关于信息窗口
B_008	个人信息页面	在主页点击侧边栏头像	进入个人信息页面	进入个人信息页面
B_009	翻译页面	点击底部菜单翻译菜单	进入翻译页面	翻译页面显示正常
B_010	学习页面	点击底部菜单学习菜单	进入学习页面	学习页面显示正常

(3) 实例应用数据请求，每个页面模块的功能考查例题应用，如表 5.7 所示。

表 5.7 应用数据请求测试用例

用例编号	测试数据	执行步骤	预期结果	实际结果
C_001	书架页面	点击书架菜单	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_002	书架页面返回	在书架页面点击返回	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_003	创建书本	在书架页面点击创建书本	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_004	删除书本	在书架页面点击自建书本	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_005	帮助页面	在主页点击帮助菜单	正确请求加载状态，	加载状态，返回状

用例编号	测试数据	执行步骤	预期结果	实际结果
			返回状态正确显示	态正确显示
C_006	反馈页面	在主页点击反馈菜单	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_007	关于信息框	在主页点击关于菜单	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_008	个人信息页面	在主页点击侧边栏头像	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_009	翻译页面	点击底部菜单翻译菜单	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示
C_010	学习页面	点击底部菜单学习菜单	正确请求加载状态，返回状态正确显示	加载状态，返回状态正确显示

(4) 各页面接口模块测试，如表 5.8 所示。

表 5.8 接口测试用例

用例编号	测试数据	执行步骤	预期结果	实际结果
D_001	书架页面	点击书架菜单	Get 请求通过, 数据正确返回	状态 200，数据正确
D_002	创建书本	在书架页面点击创建书本	Post 请求通过，数据正确返回	状态 200，数据正确
D_003	删除书本	在书架页面点击自建书本	Get 请求通过, 数据正确返回	状态 200，数据正确
D_004	帮助页面	在主页点击帮助菜单	Get 请求通过, 数据正确返回	状态 200，数据正确
D_005	反馈页面	在主页点击反馈菜单	Post 请求通过，数据正确返回	状态 200，数据正确
D_006	关于信息框	在主页点击关于菜单	Get 请求通过, 数据正确返回	状态 200，数据正确
D_007	个人信息页面	在主页点击侧边栏头像	Get 请求通过, 数据正确返回	状态 200，数据正确
D_008	翻译页面	点击底部菜单翻译菜单	Post 请求通过	状态 200，数据正确
D_009	学习页面	点击底部菜单学习菜单	Post 请求通过	状态 200，数据正确

(5) 实例各页面 UI 测试，如表 5.9 所示。

表 5.9 UI 测试用例

用例编号	测试数据	执行步骤	预期结果	实际结果
E_001	书架页面	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_002	书架页面返回	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_003	创建书本	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_004	删除书本	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_005	帮助页面	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_006	反馈页面	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_007	关于信息框	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_008	个人信息页面	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_009	翻译页面	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示
E_010	学习页面	微信端、QQ 端、APP 端	Ui 显示一致，布局正确显示	Ui 显示一致, 布局正确显示

5.3.2 性能测试

(1) 性能测试以 Linux 服务器测试，主要对实例进行压力测试，如表 5.10 所示。

表 5.10 性能测试

用例编号	执行步骤	预期结果	实际结果
F_001	1.进入平台测试栏目	压力正常	压力正常
	2.设置测试参数	负载测试正常	负载测试正常
	3.启动测试等待结果	资源使用率测试正常	资源使用率测试正常

（2）实例应用性能测验的各项考核指标，在此分别总结如下。

测试后端负载结果图 5.1、挂载服务端资源使用率测试图 5.2 所示、压力占比图 5.3 所示。

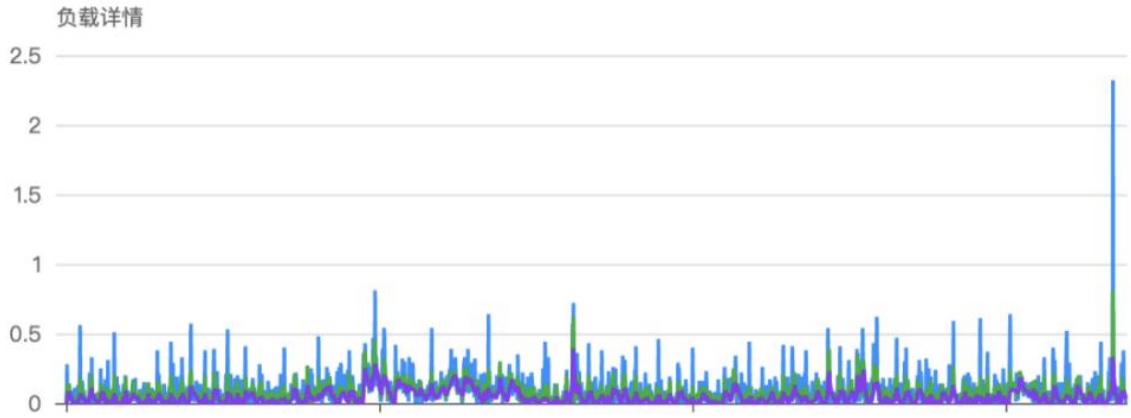

图 5.1 负载测试结果截图

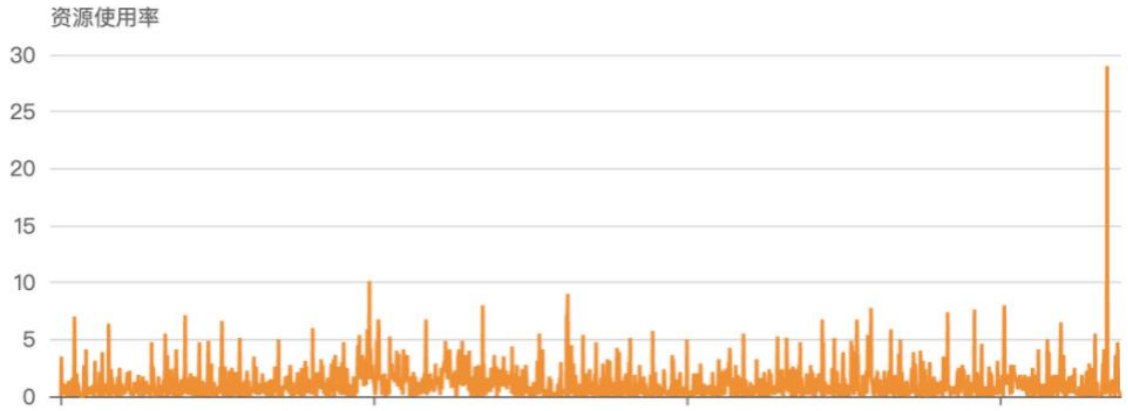

图 5.2 资源使用率结果截图

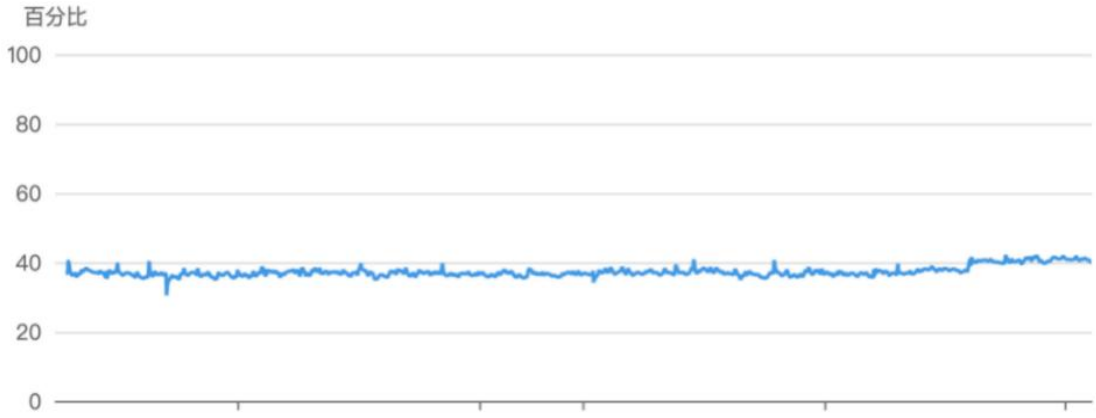

图 5.3 压力测试结果截图

5.4 测试报告

5.4.1 过程分析

(1) 功能测试，如表 5.11 所示。

表 5.11 测试用例

功能模块	测试轮数	执行用例数	用例通过数	用例未通过数	用例通过率
快捷登录	2	3	3	0	100%
创建书本	2	2	2	0	100%
删除书本	2	2	2	0	100%
反馈	2	2	2	0	100%
更新手机号码	2	2	2	0	100%
更新个人信息	2	6	6	0	100%
退出登录	2	1	1	0	100%
翻译	2	2	2	0	100%
学习单词	2	2	1	1	50%

(2) 其他测试，如表 5.12 所示。

表 5.12 测试用例

测试名称	测试轮数	执行用例数	用例通过数	用例未通过数	用例通过率
多端测试	1	10	10	0	100%
接口测试	1	10	10	0	100%
UI 测试	1	10	10	0	100%
性能测试	1	1	1	0	100%

5.4.2 结果分析

实例测试结果从测试多端覆盖率，对多端平台对功能缺陷分析。

(1) 测试覆盖分析，如表 5.13 所示。

表 5.13 测试用例

功能模块	用例总数	用例执行数	测试覆盖率	用例通过率
快捷登录	3	2	100%	100%

功能模块	用例总数	用例执行数	测试覆盖率	用例通过率
创建书本	1	2	100%	100%
删除书本	1	2	100%	100%
反馈	3	2	100%	100%
更新手机号码	1	2	100%	100%
更新个人信息	4	2	100%	100%
退出登录	1	2	100%	100%
性能测试	1	2	100%	100%
绑定手机号码	1	2	100%	100%
翻译选择	2	2	100%	100%
多端一致	1	2	100%	100%
外文全文	1	2	100%	100%
外文独文	1	2	100%	100%

(2) 缺陷分析，如表 5.14 所示。

表 5.14 测试用例

缺陷编号	BUG	BUG 描述	等级
BUG_001	学习栏目	用户未加入书本时，显示错误	严重
BUG_002	退出登录	退出登录未返回主页	一般
BUG_003	书本模块数据请求显示错误	未做状态码判断	一般
BUG_004	个人信息更新数据显示错误	未做状态码判断，未做本地更新	一般
BUG_005	翻译模块数据请求显示错误	未做状态码判断，未去除上次翻译结果	一般

5.4.3 测试总结

多端应用测试总结，多端应用测试的过程包括实例功能、实例多端性、实例接口、UI 一致、实例性能测试[9]。应用的缺陷主要集中在功能测试中，严重缺陷极其少，多端性测试和接口测试中低级缺陷数量低。

实例应用存在低级缺陷和严重缺陷极其少并已修复，符合实例应用运行标准。

结　论

实例基于 Uni-app 框架一端开发多端应用的设计与实现，通过分析研究 Uni-app 多端应用框架，从设计到实现的开发一致性、功能多端适配和跨多应用终端打包等方案。详细介绍了如何利用 Vue 和 Uni-ui 结合 Uni-app 框架开发前端应用，并通过“中式外语”实例应用验证了该方法的可行性和优点。

在实际开发应用程序过程中，采用一端开发多端应用的设计与实现方式，大大提高了开发效率、降低开发成本和维护成本，使用 Golang 语言作为后端技术进行 Get 和 Post 接口开发。通过实践证明，基于 Uni-app 框架一端开发多端应用具有高效的开发效果和应用价值为移动应用开发提供了一种新的思路和方法，相比于传统一端应用一端开发方式，具有更高的效率和更低的开发成本。一端开发多端应用具有很高效的快速开发和应用价值。

通过实例架构设计结果，利用 Uni-app 框架和 Golang 后端实现一端开发多端应用能够显著提升开发效率和代码复用性，同时也大幅度简化了部署和维护的难度，对爬虫抓取数据采用 Golang 的 Goquery NewDocument 脚本作为辅助抓取数据。

在开发快捷方面比起传统的针对不同平台开发的方式，Uni-app 这种开发的方式不仅可以大幅度减少代码的复杂度和维护工作量，还能够为用户提供更好的使用体验和多端环境一致性的前端功能和界面。

参考文献

[1] 陈海涛. 移动应用 UI 辅助开发系统的设计与实现[D]. 北京：北京邮电大学，2021.

[2] 张晓明. 基于 uni-app 和 Android 的学生手机管控系统的设计与实现[D]. 甘肃：兰州大学，2020.

[3] 刘森. 基于 Uni-app 的移动集团专线售前支撑系统的设计与实现[D]. 河南：河南科技大学，2019.

[4] 石晓旭. 基于 MOOC 的图书馆移动学习平台的设计与实现[D]. 江西：南昌大学，2016.

[5] 张琦. 基于 uni-app 的跨平台数字教材系统研发[D]. 北京：北京印刷学院，2022.

[6] 刘庆宇. 微信公众号服务器系统基础组件研发[D]. 武汉：华中科技大学，2016.

[7] 陈海涛. 移动应用 UI 辅助开发系统的设计与实现[D]. 北京：北京邮电大学，2021.

[8] 肖文娟,范梓豪. 防雷检测服务移动终端的研发与应用[G]. 广东：广东气象科学，2020.

[9] 刘楚楚. 基于网络社区挖掘的特殊人群多维特征分析[D]. 湖南：国防科学技术大学，2017.

致　谢

首先是我的指导老师郑毅平，在本文的撰写过程中，他为我提供了许多专业知识和指导意见。指导老师在学术研究上的严格要求和细致的指导，对我的学术研究具有重要的借鉴意义。 感谢您的悉心指导和支持!

其次，我要感谢我的同学们我的舍友们对我的帮助，使我度过了技术领域中前进道路上的缺失，让我有了追逐新技术的动力。

最后，我要感谢本次研究涉及的相关机构和公司，他们提供了丰富的在线资源检索和支持，并为研究提供了必要的条件与保障。

在此，我再次向以上所有的人表示诚挚的感谢和对老师的敬意，是你们的帮助让我顺利完成了本文的撰写。

学生签名:

日　　期：2023 年 05 月 12 日