账易通

项目开发报告（实现、使用说明）

黄亮铭

2022155028

目录

1 项目概述

1.1 项目背景

在当今社会，个人财务管理日益受到重视，人们期望通过便捷的方式记录和管理日常收支，实现财务状况的可视化。然而，现有的记账应用存在诸多问题，如功能复杂、操作繁琐、广告过多、付费限制以及缺乏个性化分析等，导致用户体验不佳。为满足用户对简单高效、无广告、数据安全的记账工具的需求，账易通项目应运而生。

1.2 项目目标

本项目旨在开发一款简洁无广告、功能全面且操作便捷的记账小程序，帮助用户轻松管理个人财务，实现收支记录、预算设置、财务分析等功能，成为用户日常生活中不可或缺的财务管理工具，进而助力用户实现财务自由，提升生活质量。

- **基础功能目标**：提供快速的收支记录功能，允许用户根据日期筛选记录。
- **进阶功能目标**：具备基础的数据统计功能，通过柱状图、饼图等可视化形式呈现月度收支概览。
- **用户体验目标**：设计简洁直观的用户界面，减少操作步骤，确保用户能够快速上手；应用内无广告干扰，让用户专注于记账；提供全面且可靠的财务管理功能。
- **长期目标**：基于匿名化的用户数据分析，为用户提供智能的财务建议，如消费优化提示、储蓄目标提醒等；根据用户的消费习惯，提供个性化的储蓄和支出管理建议。

2 项目实现过程

2.1 开发环境

本项目选择 Uniapp 作为前端开发框架，主要因其基于 Vue.js，支持跨平台快速开发，兼容移动端和小程序平台，拥有丰富的组件库和 API 接口，可大大提高开发效率，适合构建轻量的财务管理工具。Go 语言用于后端开发，其编译快、执行效率高，代码简洁，具备丰富的生态和库支持，能够快速构建高性能、稳定的后端服务。MySQL 被选定为数据库，因其是成熟的关系型数据库，性能稳定，支持事务处理，能保证数据一致性和可靠性，适合存储结构化的财务数据，其索引优化技术还可提升查询性能。

2.1.1 前端环境配置

前端使用集成开发环境 HBuilderX，HBuilderX 内部集成了编译器等一系列工具。我们只需要下载并安装 HBuilderX 即可。

安装完成后，打开 HBuilderX，通过点击菜单栏的“文件”->“新建”->“项目”，选择“uni-app”来创建一个新的 UniApp 项目。

2.1.2 后端环境配置

首先从 Go 官网中下载适用于本机系统的 Go 安装包，双击运行安装包并完成安装。打开终端，输入命令$go\ version$验证 Go 是否安装成功。

默认情况下，Go 会安装到 /usr/local/go。为了正确使用 Go，需要将 Go 的 bin 路径添加到环境变量中。该配置可以在命令行中输入命令完成，也可以在 GoLand 中进行设置。这里选择在 GoLand 中完成配置。

打开 JetBrains 官网，下载并安装适用于本机系统的安装包。启动 GoLand 并创建一个新的 Go 项目。在左侧导航栏选择 Languages & Frameworks > Go > GOROOT，点击 +，添加 /usr/local/go 作为 GOROOT，确保 GOPATH 默认指向 $HOME/go。

在新版本的 Go 中，我们有两种创建项目的方式，一种是传统的类似 Java 的项目，源代码存放在 src 中，可执行文件存放在 bin 中，项目依赖存放在 pkg 中；另一种是使用$go\ mod$对项目进行自动化管理。这里选择后者，通过$go\ mod$创建项目。在项目的根目录下启动终端，输入如下两条命令：$go\ mod\ init\ project_name$和$go\ mod\ tidy$即可。完成以上步骤后，项目的模块化管理便成功配置完成，开发环境也已就绪。

2.1.3 数据库环境配置

在 MySQL 官网上选择适合本机系统的 MySQL Community Server 版本，打开下载的安装包，按照向导进行安装。安装完成后，记下 MySQL 的安装路径。打开终端，编辑$\sim/.zshrc$，在文件中添加内容：$export\ PATH = \$PATH:/usr/local/mysql/bin$。保存并更新环境变量：$source\ \sim/.zshrc$。然后在终端输入命令$mysql\ \ --verson$即可验证 MySQL 是否安装成功。

2.2 架构概述

我们的项目采纳了经典的 B/S（Browser/Server）架构模式，这种模式将系统划分为两个主要部分：客户端和服务器端。在这种架构下，客户端主要承担数据的展示、用户交互的逻辑处理以及与服务器端的通信接口实现。而服务器端则专注于业务逻辑的实现、数据库的管理和为客户端提供必要的服务接口。

接下来，我们将深入探讨项目的前端界面设计、后端业务逻辑处理、数据库设计以及它们之间的交互方式，以展现 B/S 架构在本项目中的具体应用和实现。

2.3 前端

前端采用 Uniapp 进行开发，其项目框架如下图所示。

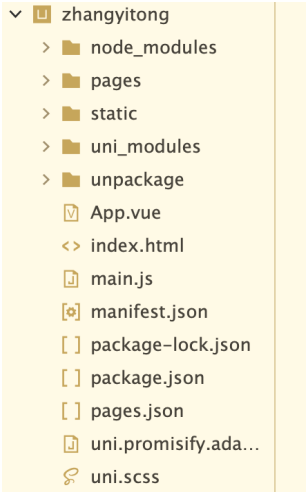

图 1 前端框架

其中一些比较重要的文件（夹）的介绍如下：

- **文件 main.js**：Vue 初始化入口文件。
- **文件 App.vue**：应用配置，用来配置 App 全局样式以及监听。
- **文件 pages.json**：配置页面路由、导航条、选项卡等页面类信息。
- **文件 manifest.json**：配置应用名称、appid、logo、版本等打包信息。
- **文件夹 pages**：业务页面文件存放的目录。
- **文件夹 static**：存放应用引用的本地静态资源（如图片、视频等）的目录。

我们重点关注文件夹 pages 和文件 pages.json。我们完成的所有页面内容均保存在文件夹 pages 中，而页面路由信息、导航栏信息还有选项卡等内容军保存在文件 pages.json 中。文件夹 pages 包含的所有页面均需要在文件 pages.json 里面的页面路由注册才能被正常地使用。

接下来我将依次介绍文件夹 pages 内的主要页面，然后再介绍如何在文件 pages.json 中配置路由信息和导航栏等内容。对于每个页面，我将从 script 即脚本部分和 template 即模版部分介绍前端的实现思路，而 style 即样式部分则直接忽略。

2.3.1“登录“/“注册“页面

因为两个页面在前端的表现基本一致，所以放到同一章节进行说明。“登录”页面和“注册”页面主要分别实现了用户管理模块中的用户登录功能和用户注册功能。

用户点击登录或者点击注册按钮时会发送消息给后端，由后端进行相应的业务逻辑处理。消息内容分为两部分，第一部分为消息类型，第二部分为具体消息内容，然后页面根据后端的响应数据给予用户相应的反馈。

登录界面的脚本部分的各种交互的函数如图 2 所示，为了方便开发人员了解不同函数的功能，我们对每个函数都进行了注释说明。

```
methods: {
    // 处理点击登录按钮之后的逻辑
    handleLogin() { ... }
    // 处理点击注册按钮之后的逻辑
    handleRegister() { ... }
}
```

图 2a“登录”界面交互函数

```
methods: {
    // 点击注册按钮之后的逻辑
    handleRegister() { ...
}
```

图 2b“注册”界面登录函数

登录界面的模版部分的代码如图 3 所示，登录界面和注册界面公用组件，因此这里只展示登录界面的代码。

```
<template>
    <view class="login-page">
        <view class="login-text">
            <view class="head1">{{title}}</view>
            <view class="head2">{{subtitle}}</view>
        </view>
        <view class="login-input">
            <up-text class="input-prompt" text="账号"></up-text>
            <up-input class="input-content" placeholder="请输入账号" type="text" v-model="userName" clearable></up-
            <up-text class="input-prompt" text="密码"></up-text>
            <up-input class="input-content" placeholder="请输入密码" password="true" type="text" v-model="password"
                clearable></up-input>
            <up-button class="login-button" @click="handleLogin" type="success">登录</up-button>
            <up-text class="register-button" @click="handleRegister" decoration="underline" color="#1E90FF" text=
        </view>
        <view class="login-message">{{message}}</view>
    </view>
</template>
```

图 3 模版部分代码

登录界面实现的效果和注册界面实现的效果如图 4 所示。

图 4a“登录”界面

图 4b“注册“界面 6

2.3.2“明细“页面

“明细”页面主要实现了记账管理模块中的新增记录功能、编辑记录功能和删除记录功能。

用户使用上述任意功能时，前端都会向后端发送消息。消息内容分为两部分，第一部分为消息类型，第二部分为具体消息内容，然后页面根据后端的响应数据进行列表展示。

“明细”页面的脚本部分的各种交互的函数如图 5 所示，为了方便开发人员了解不同函数的功能，我们对每个函数都进行了注释说明。

```
// 查询数据记录，从后端数据库获取对应的数据记录
fetchRecords() { ... }
// 获取对应记录类型的图标
getIcon(category) { ... }
// 日期筛选条件变化时重新从数据库获取符合条件的记录
bindDateChange: function(e) { ... }
// 获取日期
getDate(type) { ... }
// 新增记录浮窗显示
showModal() { ... }
// 更新记录浮窗显示，沿用新增记录浮窗显示
showModalFromUpdate(record) { ... }
// 新增记录浮窗隐藏
hideModal() { ... }
// 根据用户输入更改记录类型
onTypeChange(e) { ... }
// 根据用户输入更改记录条目
onCategoryChange(e) { ... }
// 根据用户输入更改日期
OnDateChange(e) { ... }
// 新增记录，像数据库发送新增信息
addRecord() { ... }
// 获取格式化时期（年-月-日）
formatDate(date) { ... }
// 显示更新记录浮窗
showBottomModalForRecord(record) { ... }
// 隐藏更新记录浮窗
hideBottomModal() { ... }
// 删除记录，向数据库发送删除信息
deleteRecord(record) { ... }
// 更新记录，向数据发送更新信息
updateRecord() { ... }
// 编辑记录
editRecord() { ... }
```

图 5“明细“界面交互函数

“明细“页面的模版部分代码如图 6 所示，因为代码过多，我隐藏了一些组件的细节。

```
<template>
    <view class="container">
        <view class="col-content">
            <view class="uni-list">
                <view class="uni-list-cell">
                    <view class="uni-list-cell-db">
                        <picker mode="date" :value="date" @change="bindDateChange" fields="month">
                            <view class="uni-input">{{date}}</view>
                        </picker>
                    </view>
                </view>
            </view>
            <button class="add-btn" @click="showModal"> ...
            <view class="modal" v-if="showModalFlag">
                <view class="modal-content">
                    <view class="modal-header">
                        <text>新增记录</text>
                    </view>
                    <view class="modal-body"> ...
                    <view class="modal-footer">
                        <button class="confirm-btn" @click="editRecord">确定</button>
                        <button class="cancel-btn" @click="hideModal">取消</button>
                    </view>
                </view>
            </view>
        </view>
        <view class="bottom-modal" v-if="showBottomModal" @click="hideBottomModal"> ...
        <view class="list"> ...
    </view>
</template>
```

图 6 模版部分代码

“明细“页面主页面实现的效果如图 7 所示。

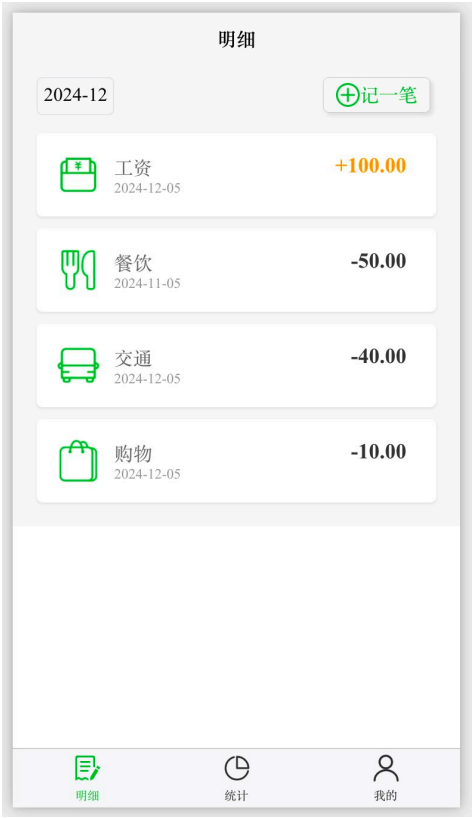

图 7“明细“界面主页面

新增记录子页面实现的效果如图 8 所示。

图 8 子页面-新增记录

编辑记录子页面实现的效果如图 9 所示。

图 9 子页面-编辑记录

2.3.3“统计“页面

“统计页面主要实现了记账管理模块中的可视化功能。在该页面用户可以查看当月各种消费类型的比例和总支出金额。

在加载该页面时，前端会向后端发送消息。消息内容分为两部分，第一部分为消息类型，第二部分为具体消息内容，然后页面根据后端的响应数据进行可视化显示。

“统计”页面的脚本部分的各种函数如图 10 所示，为了方便开发人员了解不同函数的功能，我们对每个函数都进行了注释说明。

```
methods: {
    // 从数据库中获取不同类型支出的金额
    fetchCategories() { ... }
    // 绘制柱状图
    drawChart() { ... }
}
```

图 10”统计“界面交互函数

“统计”页面模版部分的代码如图 11 所示。

```
<template>
    <view class="container">
        <view class="header">
            <text class="year-month">{{current_date}}</text>
            <text style="font-size:32rpx;">共支出: </text>
            <text class="total">¥ {{all_amount}}</text>
        </view>
        <view class="chart">
            <canvas canvas-id="ringChart" style="width: 100%; height: 400px;"></canvas>
        </view>
        <view class="list">
            <view class="list-item" v-for="(item, index) in expenses" :key="index">
                <text class="list-title">{{ item.title }}</text>
                <text class="list-amount">-¥ {{ item.amount }}</text>
            </view>
        </view>
    </view>
</template>
```

图 11 模版部分代码

“统计”页面实现的效果如图 12 所示。

图 12“统计“界面

2.3.4“我的“页面

“我的”页面主要实现了用户管理模块中的个人信息编辑功能和显示用户基本信息功能。

在加载该页面时，前端会向后端发送消息。消息内容分为两部分，第一部分为消息类型，第二部分为具体消息内容，然后页面根据后端的响应数据展示用户信息。

“我的”界面脚本部分的各种函数如图 13 所示，为了方便开发人员了解不同函数的功能，我们对每个函数都进行了注释说明。

```
methods: {
    // 点击“修改密码”按钮触发的函数
    handleModify() { ... }
    // 修改密码逻辑
    modifyPassword() { ... }
    // 点击”退出登录“按钮触发的函数
    handleLogout() { ... }
    // 退出登录逻辑
    logout() { ... }
    // 点击“个人信息”触发函数
    handleModifyInfo() { ... }
    // 点击“关于帐易通”触发函数
    handleAbout() { ... }
}
```

图 13“我的“界面交互函数

“我的“页面模版部分的代码如图 14 所示。

```
<template>
    <view class="container">
        <view class="profile-header">
            <view class="user-info">
                <image class="avatar" src="/static/touxiang.png" mode="aspectFill"></image>
                <view class="info">
                    <text class="username">用户名:{{userName}}</text>
                </view>
            </view>
        </view>
        <view class="menu-list">
            <navigator url="/pages/personalInfo/personalInfo" class="menu-item" @click="handleModifyInfo">
                <text class="menu-text">个人信息</text>
            </navigator>
            <navigator url="/pages/about/about" class="menu-item" @click="handleAbout">
                <text class="menu-text">关于账易通</text>
            </navigator>
            <navigator url="javascript:;" class="menu-item" @click="handleModify">
                <text class="menu-text">修改密码</text>
            </navigator>
            <navigator url="javascript:;" class="menu-item" @click="handleLogout">
                <text class="menu-text">退出登录</text>
            </navigator>
        </view>

    </view>
</template>
```

图 14 模版部分代码

“我的”页面实现的效果如图 15 所示。

图 15“我的“界面

2.4 后端

后端使用 Gin 框架的路由组，将相同页面的不同请求发送到同一个回调函数中进行处理，由回调函数根据消息的类型，将消息分发给对应的业务逻辑处理函数。

通过路由组管理，后端接收前端的请求并进行相应的过程更加合理、清晰。

```
router := gin.Default()
user := router.Group( relativePath: "/user")
{
    user.GET( relativePath: "/login", handleLogin)
    user.GET( relativePath: "/register", handleRegister)
}
record := router.Group( relativePath: "/record")
{
    record.GET( relativePath: "/query", handleQuery)
    record.GET( relativePath: "/add", handleAdd)
    record.GET( relativePath: "/delete", handleDelete)
    record.GET( relativePath: "/update", handleUpdate)
}
count := router.Group( relativePath: "/count")
{
    count.GET( relativePath: "/category", handleCount)
}
router.Run( addr...: ":8080")
```

图 16 路由组

对于前端发起的多种类型的操作，后端实现了精细化的响应机制。通过为每项操作配备专门的处理函数，避免了将所有功能集中在单一函数中，从而确保了代码的简洁性和可维护性。这种模块化的设计方法有助于提高系统的灵活性和扩展性。

```
// 登录操作
func handleLogin(c *gin.Context) {...}

// 注册操作
func handleRegister(c *gin.Context) {...}

// 查询记录操作
func handleQuery(c *gin.Context) {...}

// 添加记录操作
func handleAdd(c *gin.Context) {...}

// 删除记录操作
func handleDelete(c *gin.Context) {...}

// 编辑原有记录操作
func handleUpdate(c *gin.Context) {...}

// 统计操作
func handleCount(c *gin.Context) {...}
```

图 17 处理函数

业务逻辑处理函数需要查询数据库相关数据时，我们使用 gorm 框架进行查询操作。在使用 gorm 时，我们不需要显示地编写 SQL 语句。Gorm 为我们提供了一些列的 SQL 方法，如 create、first、model 和 delete 等，这些方法在内部实现了对应的 SQL 操作。Gorm 就是使用 Go 编写的 ORM（对象关系映射）技术，它将数据库中的表映射为 Go 中的对象，将 SQL 操作映射为 Go 中的方法。

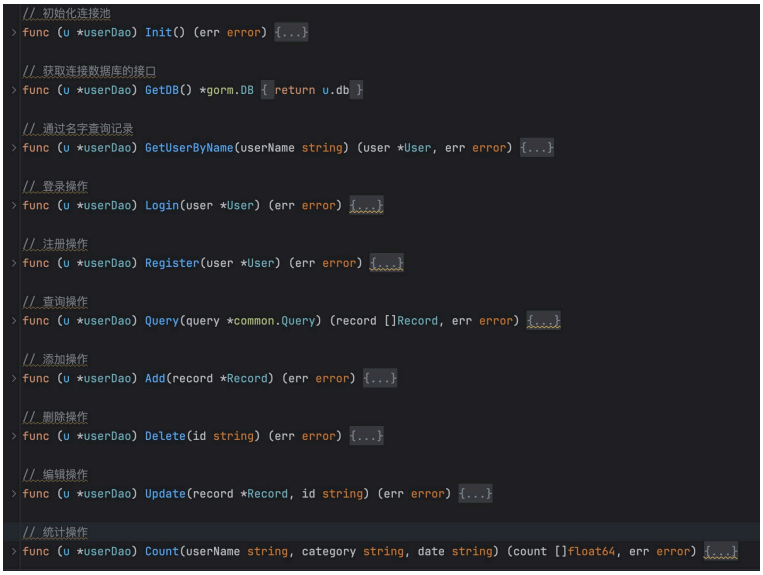

图 18 操作数据库的函数

此外，后端与数据库部分的交互使用了工厂模式通过将对象的创建和使用分离，减少了业务逻辑与具体组件类之间的依赖。并且工厂模式使得添加新的类实现变得简单，只需添加相应的工厂方法，无需修改现有代码，符合开闭原则，提高了系统的灵活性和可拓展性。

```go
type userDao struct {  12 usages
    userName string
    userPwd  string
    host     string
    port     int
    dbName   string
    timeout  string

    dsn string
    db  *gorm.DB
}

// 工厂模式
func NewUserDao() *userDao {  1 usage
    userName := "root"
    userPwd := "12345678"
    host := "127.0.0.1"
    port := 3306
    dbName := "zhangyitong"
    timeout := "10s"
    return &userDao{
        userName: userName,
        userPwd:  userPwd,
        host:     host,
        port:     port,
        dbName:   dbName,
        timeout:  timeout,
    }
}
```

图 19 工厂模式的结构体和实例化函数

2.5 数据库

2.5.1E-R 图

根据目前的项目进度，我们对项目设计报告中给出的 E-R 图进一步优化，得到下面的 E-R 图。

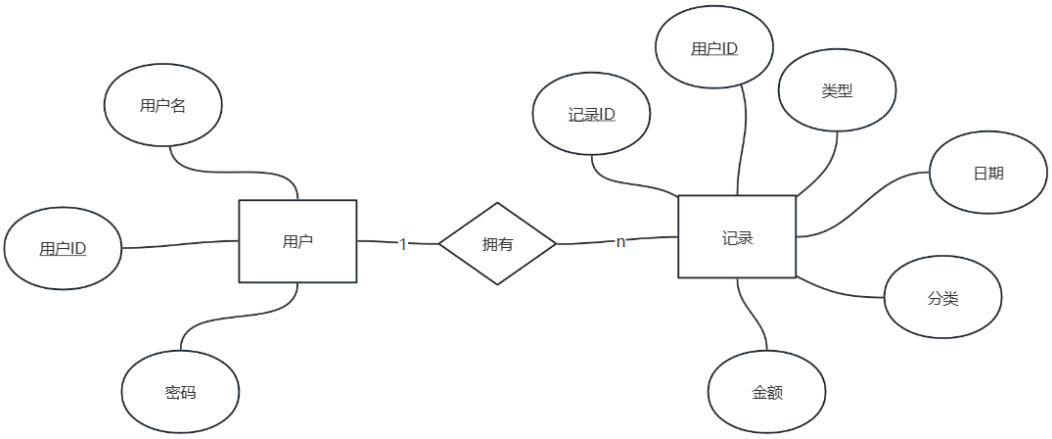

图 20 数据库 E-R 图

2.5.2 数据库创建

我们根据之前的项目设计报告中对数据库的设计和上述 E-R 图创建两张数据库表如图 21 和图 22 所示，并使用外键将两张表联系在一起。

```
mysql> create table user(
    -> id INT NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(20) NOT NULL,
    -> pwd VARCHAR(20) NOT NULL,
    -> PRIMARY KEY(id)
    -> )ENGINE=InnoDB DEFAULT CHARSET=utf8;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

图 21 用户表创建

```
mysql> create table record (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     userName VARCHAR(20) NOT NULL,
    ->     type VARCHAR(64) NOT NULL,
    ->     category VARCHAR(64) NOT NULL,
    ->     amount DECIMAL(12,2) NOT NULL,
    ->     time DATE NOT NULL,
    ->     PRIMARY KEY(id),
    ->     FOREIGN KEY(userName) REFERENCES user(name)
    ->     )ENGINE=InnoDB DEFAULT CHARSET=utf8;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

图 22 记录表创建

3 系统使用说明

1. 记录
   - **功能描述**：该模块是整个程序的首页，根据之前的系统原型报告，该模块为用户提供了添加收支记录、查看收支记录列表和根据日期筛选收支记录的功能。
   - **操作方式**：1）用户点击右上角的“记一笔”按钮即可快速记录收支；2）用户点击做右上角的日期按钮即可根据日期对记录进行筛选操作，方便用户查看自己需要的信息/
2. 统计
   - **功能描述**：该模块主要以饼状图+列表的方式展示用户当月的各种消费类型所占的比例，让用户更加容易分析当月的支出。
   - **操作方式**：用户点击下方导航栏的“统计”按钮即可跳转至该页面。
3. 我
   - **功能描述**：该模块主要是用户的个人中心，包含个人信息、设置等相关功能，用户可以通过该模块查看个人信息。
   - **操作方式**：用户可以在该模块中查看个人信息、更改密码。
4. 登录/注册
   - **功能描述**：登录和注册模块为用户提供了账户创建和访问个人数据的入口。用户可以通过这些功能创建新账户或登录已有账户。
   - **操作方式**：对于新用户，填写必要信息进行注册；对于已有账户的用户，输入账户凭证进行登录。

用户反馈：

我们鼓励用户在使用过程中提供反馈。如果在使用账易通过程中遇到任何问题或有任何建议，用户可以通过应用内的反馈功能与我们联系。我们致力于不断改进产品，以提供更优质的服务。

安全提示：

账易通承诺保护用户的财务数据安全。我们采用了加密技术来保护用户数据，并建议用户定期更新密码，不要在公共设备上保存敏感信息。

通过上述功能模块，账易通旨在为用户提供一个全面、安全、易用的个人财务管理工具，帮助用户实现财务自由，提升生活质量。我们将持续更新和优化应用，以满足用户日益增长的需求。