蚂蚁热帮（AntHelp）生活服务系统的设计与实现

——基于 Token 认证的用户、权限管理

摘　要

通过对市场深入而充分地调研，得出准确的用户需求，结合后续的概要设计、详细设计、测试、部署和实施来完成整个系统。

热帮服务系统旨在服务大众，并且是以大众帮助大众的形式来进行服务，提高社会生产力，资源利用率，使得人们的生活更加地方便快捷。蚂蚁热帮（AntHelp）更是融合了社区模块，拉近了人与人之间的距离，让人们在轻松愉悦的交谈环境中互帮互助,让人们在互相交流的轻松氛围中获取资源。

Token 认证是安全、有效的对用户登录信息进行验证以及对接口的权限验证的授权管理方式。通过 Token 签名，可以对用户登录信息的有效性进行验证，同时可以利用此技术实现单点登录，重复请求接口的处理。在 Token 签名的使用中，还可以借此对所有的接口进行权限的验证处理，给不同的用户角色进行不同的 api 接口开放，增加了系统的健壮性和安全性。

本系统采用的 C#进行编码，同时采用最新的开发环境 Microsoft Visual Studio 2017 和 Microsoft SQL Server 2016 开发环境进行开发。在强有力的开发环境的支持下，采用了自行开发的 QX_Frame 前后端开发框架进行高效快速的开发。技术上使用了 ASP.NET WebAPI 构建后端服务，OWIN 进行 IIS 解耦，Autofac 作为 IOC 容器，EntityFrameWork6.0 作为 ORM 框架，Angular2 进行前端框架的搭建。

关键词： 热帮服务　　社交　　C#　　QX_Frame

The Design and Implementation of AntHelp Life Service System

——Token authentication based on user, rights management

ABSTRACT

The depth of the market and fully research, to obtain accurate user needs, combined with the follow-up design, detailed design, testing, deployment and implementation to complete the entire system.

The hot service system is designed to serve the public and to serve the public in the form of public help, to improve social productivity and resource utilization, making people's lives more convenient. Ants (AntHelp) is the integration of the community module, closer to the distance between people, so that people in a relaxed and pleasant conversation environment mutual help, let people in the exchange of relaxed atmosphere to obtain resources.

Token authentication is a secure and effective way to authenticate user login information and authorize authorization management of interface authority. Through the Token signature, you can verify the validity of the user login information, and can use this technology to achieve single sign-on, repeat the request interface processing. In the use of Token signature, you can also use this authority to verify the processing of all the different user roles to different api interface open, increasing the robustness and security of the system.

The system uses C # encoding, while using the latest development environment Microsoft Visual Studio 2017 and Microsoft SQL Server 2016 development environment for development. In a strong development environment with the support of the use of self-developed QX_Frame front and back development framework for efficient and rapid development. Technically using the ASP.NET WebAPI to build back-end services, OWIN IIS decoupling, Autofac as an IOC container, EntityFrameWork6.0 as the ORM framework, Angular2 front-end frame structures.

Key words: Help Service Social C# QX_Frame

目　　录

第一章 绪论

1.1 项目背景与现状

1.1.1 项目背景简介

蚂蚁热帮（Ant Help）是以生产第三方服务平台为目标，致力于更方便，更快捷的，更有效的 O2O 生活服务平台。突出了“你出钱，我出力！”的互帮互助主题。更方便，更快捷，更有效的生活方式，改变人们生活节奏，提高人们办事效率。继滴滴、美团、饿了么之后的更进一步的良好的生活服务平台。

1.1.2 当前国内外现状

本系统采用的是社交+O2O 的方式，纵观国内外社会的发展，如今的人们更加趋向于社交平台的使用，而传统的电话邮件的方式已经逐步落后于社交，但是并不能完全取缔。O2O 的浪潮也随着各支付平台的兴起一浪接过一浪。但这并不能代表所有前景都是非常美好的。未来仍需要良好快速的发展，使得互联网更加接地气！我们的社交+O2O，充分地实现了在社交中 O2O，又在 O2O 的线上线下交互中可以有效的利用社交来增加用户粘性，使得系统可用性大大增加。

GlobalWeb Index 在 2014 年年底发布了一份社交网络研究报告，报告显示，在过去的半年内，Twitter 在过去半年用户和活跃用户增长率只有 18%和 26%，而 Facebook 这个全球最大的社交应用的用户和活跃用户增长率则只有 6%和 2%，马上就要探底。在 Facebook 网罗了全球超过 30%人口的情形下，除非能够进入中国（这在短期内显然是不可能的），否则它的增长基本已达极限，在 2015 年 Facebook 的用户增长率很可能会低于 0%。作为国内较早的社交网络之一，微博“活跃度下降”已不是一个月两个月的事了，今年微博的第三季度财报显示：微博月活跃用户数（MAU）为 1.67 亿，较上年同期增长 36%。这个数据还算让人欣慰，但考虑到 7 月世界杯的淘汰赛为微博贡献了巨大的活跃度，这个数据并不那么让人乐观。可以看到，国内外的互联网社交发展并非一帆风顺，虽然客户仍然增多，但是增长速度确是异常缓慢，如果继续保持这样的发展现状，而未能融入新的元素，那么将不会再有当初爆炸时候的发展劲头。

1.2 选题意义与系统主要工作

1.2.1 选题目的及意义

为什么大家要用我们的平台？我们平台有什么样的优势呢？且不说点餐需要使用点餐软件进行点餐，打车需要运行打车软件进行打车，而且很多司机并未注册打车软件进行打车服务。况且点到的餐由餐馆随便找个“爆粗口”的大妈配送外卖引起的各种食欲的下降、开车师傅因为长相太“魁梧”而阻碍了社会的沟通交流…种种可能想到的各种问题，在我们这里都就能找到很好的解决办法！

我们同时旨在以社交为基础扩展开来的服务，这就意味着帮助你的对象可能是你心仪已久的同校音乐系帅哥，你帮助的对象可能是你心动多年的美术系美女，凡是你平时能接触到的社交群体，都可能作为服务被服务的对象，如果不幸匹配到了校外彪悍大叔，同时你还不想让他帮助你，那么你可以一键 cut 掉他。

该产品以大学生为前期对象，以各类大学为前期试点，通过试点的实际实施情况进行分析优化改进，逐步向全面市场推广，争取服务到广大人民群众中去。

1.2.2 本系统主要工作

如果你想要别人帮助你，不管是什么样的要求，你都可以进行下单，下单的结果就好比是在朋友圈发了一条动态“x 号宿舍楼 有 x 美女 需要找人代取一下快递 ，报酬 xx 。恳切希望帅哥来拯救” 当然如果你有足够的魅力的话，完全可以写报酬 0 哦。然后就有人在圈内进行不断地刷刷刷，是的，下拉刷新，然后看到了你发的这条动态，刚好顺路（当然也可以不顺路的），进行接单。接下来我们下单接单双方可以随时进行对对方信息的查看，如果你觉的对方不太靠谱，大可以取消掉订单，然后选择重新发布。如此可以直到选中您满意的服务对象。服务完毕，双方可以在聊天中约定交付方式，然后进行交付，系统就可以按照原先的约定进行交付啦。除了快递，托熟人捎饭，打车…不限信息种类，只要你能想到的任务，都可以进行发布。只要你有足够的勇气完成，都可以接受。通过我们的“蚂蚁热帮”社交生活服务平台，我们更加扩宽了我们日常生活的圈子，更进一步提升了交友的范围，使我们的日常交流更加丰富，同时能提高我们的生活质量和节约时间、提升生产效率。“蚂蚁热帮”，让我们的生活更精彩！

本人在系统的设计实现中的主要任务为系统后台框架的研发以及系统环境的搭建、系统服务的开发，除此之外，本人主要研究的内容为基于 QX_Frame 框架的通用用户账户管理、用户信息管理、Token 认证、系统权限管理等业务实现。本人研究的内容在任何系统的实现中都具有相当重要的意义，任何系统都离不开用户，有用户就有权限。本人解耦了用户管理模

块和业务逻辑系统，使得用户模块得以通用化，在未来的各系统的使用中，有深远的意义。

1.3 本章小结

本章对本课题的选题意义和国内外发展情况进行了调查，明确了项目的背景信息，然后对本系统所涉及的核心业务进行了大概的介绍。在完善的调查结果以及对项目背景信息明确的前提下继而进行项目的正式开展。

第二章 系统可行性分析

2.1 背景简介

该项目是以生产第三方服务平台为目标，致力于更方便，更快捷的，更有效的 020 生活服务平台。是在当前人们越来越快的生活节奏以及越来越方便的生活行为方式的大环境下进行启动的一个良好的生活服务平台项目。如果项目能达到预期的效果，一定是继滴滴、美团、饿了么之后的齐肩并进的良好生活服务平台。

2.2 可行性分析

2.2.1 社会可行性

项目针对当前人们日益加快的生活节奏以及日益方便的生活方式研发，在社会上的认可程度以及社会接受程度上有比较大的先天优势。除了在系统业务上的用户接受度的优势外，项目中还有人们生活中常见的社区模块，社区丰富了人们日常生活中的沟通方式，以一种更方便快捷的方式带给人们日常的交流的同时也让人们互帮互助。

法律因素方面。开发中所有商业软件都采用购买的方式使用正版软件。所有源代码为自主开发，自有知识产权。涉及到第三方控件和支付接口，也通过正规渠道进行商务购买。开发中使用的开源技术不存在法律纠纷。在用户使用可行性方面，用户对系统的熟悉和使用能再很短的周期适应，同时管理员也能迅速接受系统的管理方式，能很好的融入系统，乐在其中。

综上所述，本系统没有违背法律和道德，所以在社会上是可行的。

2.2.2 经济可行性

1. **人员费用**

   本系统开发期 2 个月，开发人员 3 人，作为毕业设计，开发人员都为毕业设计相关人员，故而不需要开发人员费用。

2. **硬件设备费用**

   系统的开发使用均由毕业设计相关人员自备，故而不需要硬件设备费用。

3. **软件费用**

   系统所需软件费用为 0 元，其中包括：

   (1)Microsoft Visual Studio 2017 community（免费）

   (2)Microsoft SQL Server 2014 Express（免费）

   (3)Microsoft Visual Code （免费）

   (4)Google Chrome（免费）

4. **耗材费用**

   由于文档等需求，各种耗材综合费用为 250 元。

5. **其他费用**

   毕业设计相关文档查重费用 200 元。

6. **不可预见费用**

   由于开发过程中遇到的事件不可知，拟不可预见费用约 200 元。

7. **总开发费用**

   综上分析，该系统总的开发费用为 650 元。

2.2.3 技术可行性

该系统采用微软的 Asp.net webApi2 以及市场上较为流行的 Angular2，自行研发的 QX_Frame，大大减轻了系统开发的难度，在有成熟框架作为依托的前提下，结合自行开发的代码生成器，加之团队成员对开发环境的适应以及对开发技术的悉心学习，具备了足够的专业基础知识。因此，在技术上是不存在困难的。

2.2.4 可行性分析结论

通过从经济、技术、社会可行性三方面的分析，得知系统的设计实现在经济上、技术上、社会上都具有很高的可行性，故而得出结论，项目可行。

2.3 本章小结

在本章中，我们对系统的可行性进行了一系列的分析，包括经济可行性、技术可行性、社会可行性等。经过严密的可行性分析，我们最终得出了我们的系统完全可行的结论，保证了后续的系统继续分析设计有了良好的可行性基础，有利于系统的逐步完成。

第三章 系统开发环境及相关技术简介

3.1 Microsoft Visual Studio 2017 简介

微软借着 Visual Studio 品牌 20 周年之际，于美国太平洋时间 2017 年 3 月 7 日 9 点召开发布会议，宣布正式发布新一代开发利器 Visual Studio 2017。作为 Microsoft 公司的主力集成开发工具（IDE），历经了 20 年的革新，今天终于走到了 vs2017 这个有历史意义的节点，很有幸，我能在第一时间安装体验了 visual studio 2017，并充分体验并利用了他带来的各项好处。这里我列举 Visual Studio 2017 的新增功能[1]：

1. 与云集成，内置工具可让所有的 NET Core、Azure 应用程序、服务、Docker 容器等全面集成。 无缝体验让你感觉如同在 Azure 数据中心工作一样[2]。
2. 有效协作，直接管理任意提供程序托管的团队项目，包括 Visual Studio Team Services、Team Foundation Server 或 GitHub。 或者，使用新的“打开任意文件夹”功能，无需通过正式项目或解决方案即可快速打开并处理几乎所有代码文件。
3. 交付优质移动应用，借助 Xamarin 的高级调试、分析工具以及单元测试生成功能，与以往相比你可以更快、更轻松地构建、连接和调整适用于 Android、iOS 和 Windows 的本机移动应用。 你还可以选择使用 Apache Cordova 开发移动应用，或构建 C++ 跨平台库。
4. 提高语言水平，Visual Studio 继续加强对最新编程语言功能的支持。 无论使用 C#、Visual Basic、C++、TypeScript、F# 还是使用第三方语言（例如 JavaScript），在整个开发体验中你都能获得一流的功能支持[3]。
5. 性能得以优化，Visual Studio 包含了对日常使用的核心功能的大量性能改进。 你还会发现在整个开发生命周期中，启动速度显著加快，内存占用大大降低，并且响应能力明显提高。

3.2 Microsoft SQL Server 2016 简介

作为当前最新版本的 Microsoft SQL Server 数据库，Microsoft SQL Server 2016 在延续了历史版本的各项优势以外，更是添加了诸多的新特性[4]。这些新的特性使我们的使用更加的方便快捷，提高了我们的开发效率。下面我展示一些 Microsoft SQL Server 2016 的新特性：

1. 全程加密技术(Always Encrypted)，全程加密技术(Always Encrypted)支持在 SQL

Server 中保持数据加密，只有调用 SQL Server 的应用才能访问加密数据。

2. 动态数据屏蔽(Dynamic Data Masking)，利用动态数据屏蔽功能，你可以将 SQL Server 数据库表中待加密数据列混淆，那些未授权用户看不到这部分数据。利用动态数据屏蔽功能，你还可以定义数据的混淆方式。
3. JSON 支持，在 SQL Server 2016 中，你现在可以在应用和 SQL Server 数据库引擎之间用 JSON 格式交互。微软公司在 SQL Server 中增加了对 JSON 的支持，可以解析 JSON 格式数据然后以关系格式存储[5]。
4. SQL SERVER 支持 R 语言，SQL Server 支持 R 语言处理以后，数据科学家们可以直接利用现有的 R 代码并在 SQL Server 数据库引擎上运行。这样我们就不用为了执行 R 语言处理数据而把 SQL Server 数据导出来处理。该功能把 R 语言处理带给了数据。
5. Query Store，如果你经常使用执行计划，你就会喜欢新版的 Query Store 功能。在 2016 之前的版本中，你可以使用动态管理试图(DMV)来查看现有执行计划。
6. 行级安全(Row Level Security)，SQL 数据库引擎具备了行级安全特性以后，就可以根据 SQL Server 登录权限限制对行数据的访问。限制行是通过内联表值函数过滤谓词定义实现的。安全策略将确保过滤器谓词获取每次“SELECT”或者 “DELETE”操作的执行。

3.3 Angular2 前端框架和 Typescript 语言简介

AngularJS 当初是提供给设计人员用来快速构建 HTML 表单的一个内部工具。随着时间的推移，各种特性 被加入进去以适应不同场景下的应用开发。然而由于最初的架构限制（比如绑定和模板机制），性能的提升已经非常困难了。由此为引，Angualr2 便这样诞生了。

Angular2 的开发语言采用 Microsoft 的 Typescript 语言，当年的浏览器大战，让人记忆犹新，Chrome 的出品商 Google 和 IE 的出品商微软正是浏览器大战的两大主角。俗话说：没有永远的朋友，也没有永远的敌人，只有永远的精益求精。Google 与微软相逢一笑泯恩仇，进行了具有历史意义的合作，这将是 IT 互联网领域的一次历史性的牵手。

Angular2 由八个主要部分构成，正是这八大件的密切配合，相互协作，才有了 Angular2 的高效、快速、方便等先天优势[6]：

1. 模块 (module)
2. 组件 (component)
3. 模板 (template)
4. 元数据 (metadata)
5. 数据绑定 (data binding)
6. 指令 (directive)

7. 服务（service）
8. 依赖注入（dependency injection）

Typescript 是微软开发的自由的，开源的编程语言，语言中包含了后端开发人员熟悉的静态类型和基于类的面向对象编程特点。使得前后端的开发人员在语法的使用上更近了一步。TypeScript 是 JavaScript 类型的超集，它可以编译成纯 JavaScript。TypeScript 可以在任何浏览器、任何计算机和任何操作系统上运行，并且是开源的。TypeScript 扩展了 JavaScript 的语法，所以任何现有的 JavaScript 程序可以不加改变的在 TypeScript 下工作。TypeScript 是为大型应用之开发而设计，而编译时它产生 JavaScript 以确保兼容性。Typescript 有几大特性：

1. 类型批注和编译时类型检查
2. 类
3. 接口
4. 模块
5. lambda 函数

3.4 QX_Frame 开发框架介绍

3.4.1 框架简介

QX_Frame.FrameWork4.6（版本号：2.0.0）为本人亲自开发的开源项目，该项目于 2016 年 11 月开始启动，至今已投入使用。框架采用最新的.NetFramework4.6 和 AngularJS2 平台进行搭建的前后台通用开发框架，框架包含了 Console 控制台应用程序，WindowsForm 应用程序，Web 应用程序，项目间采用插拔式的方式进行配置，仅仅需要使用需要的项目进行使用即可，大大降低了应用程序间的耦合性[7]。目前该项目已部开源于 GitHub 社区->https://github.com/dong666/QX_Frame.FrameWork4.6。

3.4.2 框架技术要点

1. 封装了通用的数据访问层基类 WcfService、Entity<T>，并且采用 Wcf<T>方法创建自定义数据层管道，传入查询条件对象，通用管道查询方法 QueryAll()、QuerySingle()进行对象查询,返回一致的查询结果 WcfQueryResult；支持分页条件查询；极大提高了编码效率。
2. 框架层的数据库使用 db_qxframe，和业务数据库分离，充分解耦了后期框架和业务逻辑，使得框架重用性得到提高。
3. Ioc 容器采用高效的依赖注入框架 Autofac 框架，提高了程序的运行效率。

4. 框架采用 AOP 技术进行通用的异常处理，并且加入了系统日志，提高了项目的编码效率和后期的可维护性。
5. 框架内使用缓存，提升查询效率。
6. 框架的数据库处理采用 ORM+Sql 的兼容方式，丰富了开发人员的开发使用方式。
7. 插拔式的项目结构使得开发更加丰富且随心所欲，极大的给开发人员带来了便捷。同时也大大增加了框架的兼容性和后续可维护性。

3.4.3 框架展望

1. 增加多种数据库的支持，不依赖于.net 环境下的 Microsoft SQL Server，新增对 MongoDB 的非关系型数据库的支持。
2. 采用 Memcached 实现分布式集群服务器缓存。
3. 增加提示信息国际化的配置方式，使得框架更加人性化。
4. 升级采用.net Core 跨平台的开发方案，增加框架健壮性。

3.5 Token 认证简介

Token 认证是当下流行的身份验证机制，用于保证服务端接口的安全性和时限性。其原理初始时用户提交账号数据给服务端，服务端采用一定的策略生成一个字符串（token），token 字符串中包含了少量的用户信息，并且有一定的期限。服务端会把 token 字符串传给客户端，客户端保存 token 字符串，并在接下来的请求中带上这个字符串。使用基于 Token 的身份验证方法，在服务端不需要存储用户的登录记录[8]。大概的流程是这样的：

1. 客户端使用用户名跟密码请求登录；
2. 服务端收到请求，去验证用户名与密码；
3. 验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端；
4. 客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里；
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 Token；
6. 服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据；

通过 Token 认证的机制，保障了服务器用户数据的安全性，同时也使服务器的安全性得到了提高。

下面以图形方式展示了传统的认证方式以及 Token 认证方式的差异，在 Token 认证中，客户端请求资源的时候，服务端会验证 Token 的有效性来确定是否有获取资源的权限，来达到提升 api 接口安全性的目的，如图 3.1 所示。

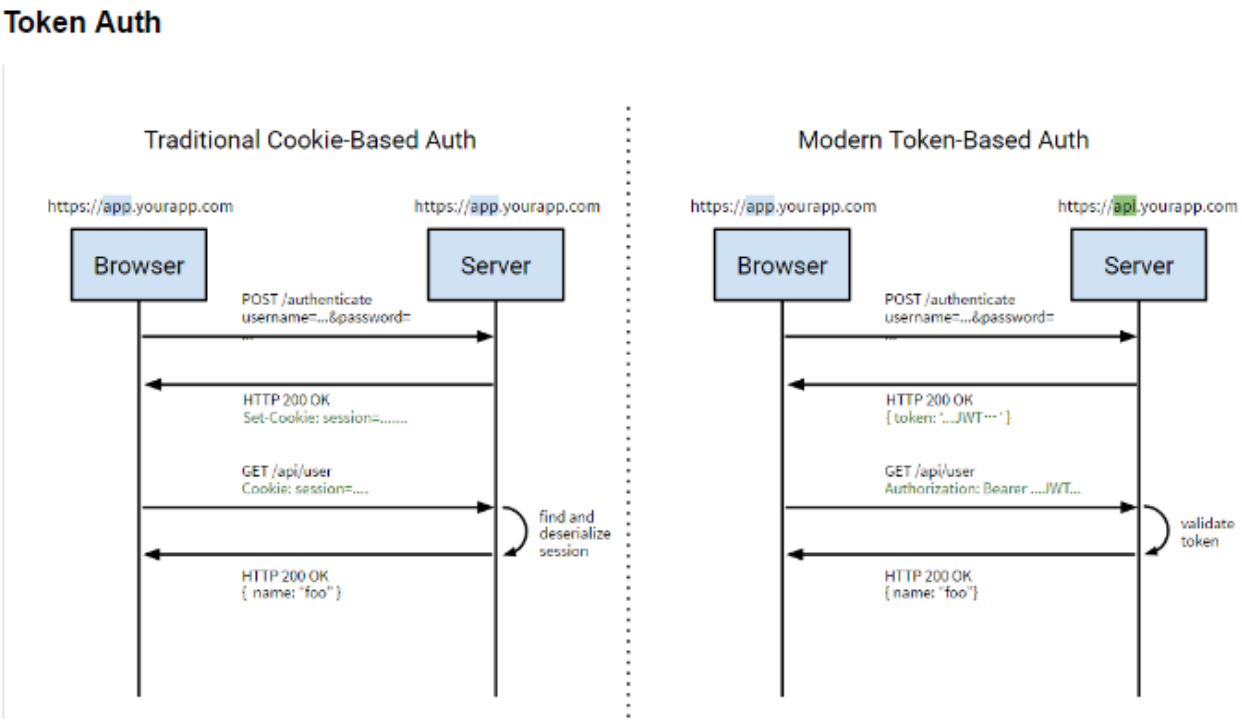

图 3.1 Token 认证和传统验证对比图

Fig. 3.1 Token authentication and verification contrast figure

3.6 本章小结

在本章中，我们对系统的开发环境以及所用到的技术点进行了分析，在开发环境的选用上，我们选择的都是当前最前沿的框架，包括 2016 年刚出不久的 Angular2 前端开发框架，以及 C# .NETFramework4.6.1，利用最新的技术，不仅可以跟随时代前沿，保持自己在编程领域的新鲜度，而且可以得到很多新特性的支持，极大的提高了代码编写的效率。除此之外，我们还对我们利用到的技术进行了提前的认真学习，保证在项目实现过程中能有丰富的知识储备，以便顺利地完成系统的设计。

第四章 需求分析

4.1 功能划分

4.1.1 总功能模块划分

1. 用户管理：在用户管理模块中，设计系统相关用户的角色以及对各种角色的用户信息进行管理。在对用户信息进行管理的同时还涉及到了用户的注册登录等功能，该模块有涉及到系统的 Token 认证相关概念。
2. 权限管理：该模块主要用于系统的各项功能的访问权限的管理，对不同的用户开放不同的权限进行操作，使得系统的稳定性和安全性得到了提高，加强了管理员对人员的管理以及系统的维护。
3. 订单管理：订单管理模块用户接收用户的下单请求，并且由系统分发订单信息，其他用户抢单接单进行对订单的后续操作。不同的用户角色对订单有不同的操作策略。
4. 消息管理：消息管理模块主要用于管理员对用户发送系统消息，或者是用户对系统的意见或建议，以及用户对其他用户的投诉信息，可以向系统投诉信箱发送信息，系统管理员登录后可以查看用户发送来的投诉信息，并进行对投诉信息的处理。
5. 优惠活动：优惠活动模块用户可以在规定期间内进行系统活动的参与。
6. 社区模块：社区模块类似于社交媒体的沟通平台，用户可以在平台内进行交流分享。

我们以图示方式展示系统总功能模块的划分，如图 4.1 所示。

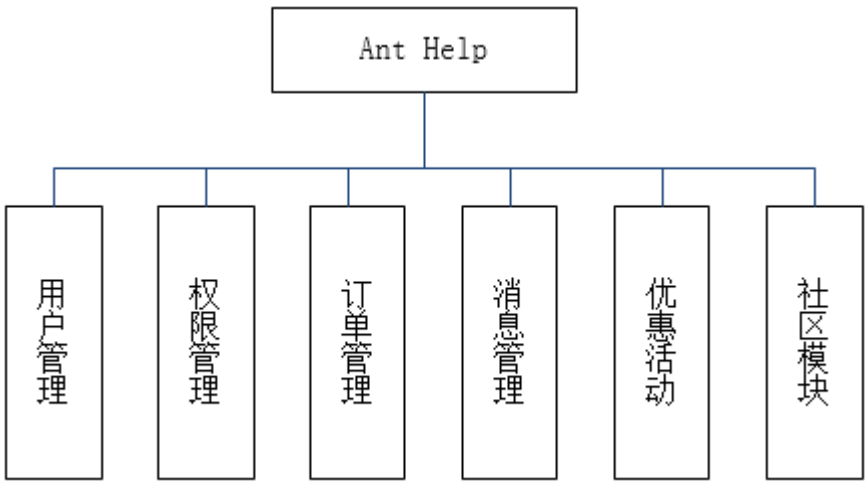

图 4.1 AntHelp 系统总功能模块图

Fig. 4.1 AntHelp function module diagram of system

4.1.2 用户管理模块

用户管理模块主要有对用户的登录注册相关信息的管理以及对用户信息的管理，同时还有对用户的安全、用户的角色、用户的状态信息进行管理。该模块涉及到用户登录后对 Token 签名的生成以及对签名的维护，以及单点登录的实现。用户注册采用邮箱验证的方式进行验证注册。用户角色以及用户状态由管理员角色进行对普通用户的授权，角色和状态的改变会直接影响到后续的接口访问权限。用户管理功能模块如图 4.2 所示。

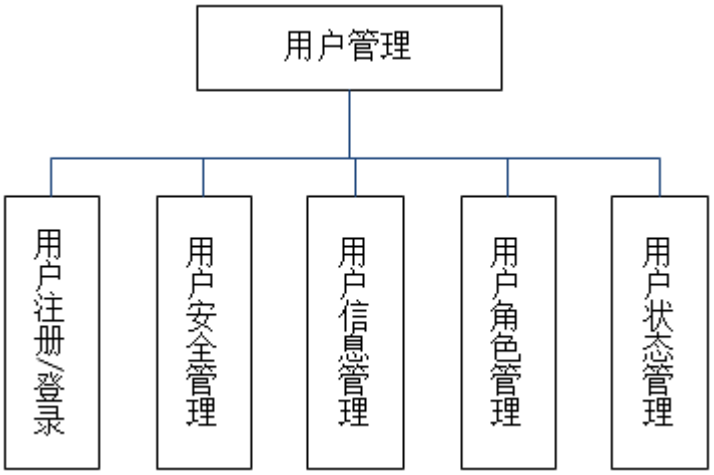

图 4.2 用户管理功能模块图

Fig. 4.2 User management function module diagram

4.1.3 权限管理模块

权限管理模块，主要是利用了用户管理模块中，用户登录之后生成的 Token 签名进行对用户信息的有效性进行验证，并对用户角色、用户状态进行验证，所有的 api 接口都会涉及到 Token 值的获取以及通过对 Token 签名值解密得到的用户信息进行权限认证。用户权限管理功能模块如图 4.3 所示。

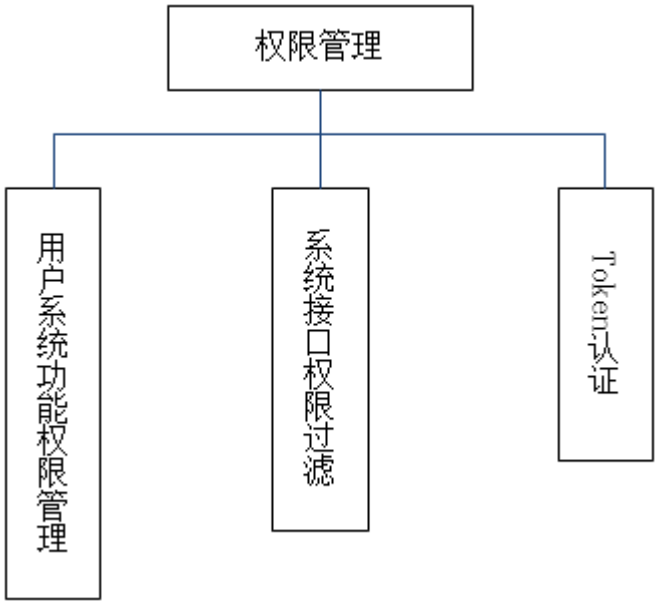

图 4.3 用户权限管理功能模块图

Fig. 4.3 User rights management function module diagram

4.2 用例分析

1. 管理员

(1)用户信息管理

A.查看用户信息，包含对用户状态，用户角色的查看。

B.修改用户信息，包含对用户状态的修改，以及对用户角色的修改。用户角色的修改直接关系到对接口访问的授权控制。

(2)管理员注册登录

A. 管理员的身份注册，可以由其他管理员或者系统管理员直接任命。

B. 管理员登录，登录完成会自动保存角色信息，以便在访问接口时判断是否为管理员以及对接口权限的控制。

管理员的用例图如图 4.4 所示。

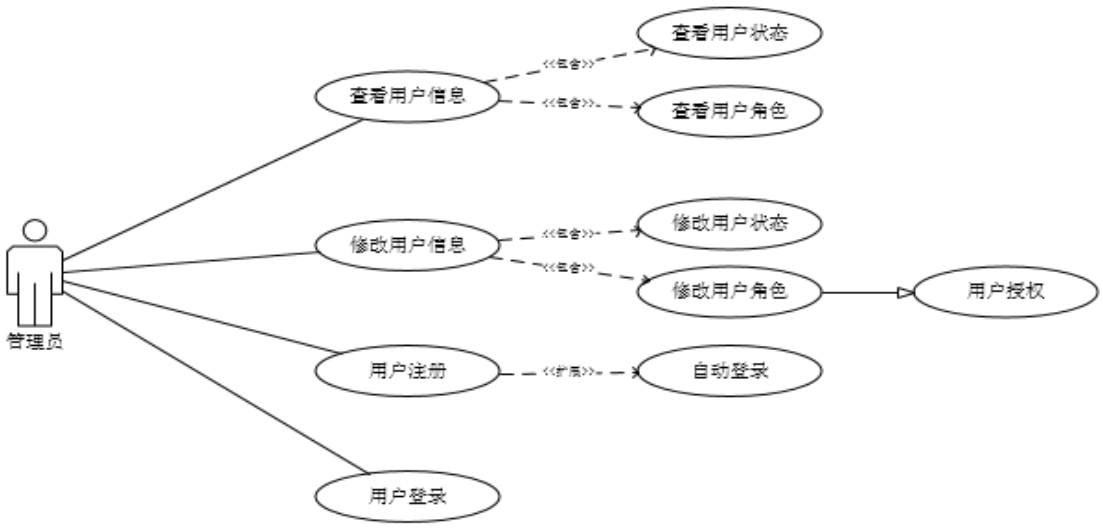

图 4.4 管理员用例图

Fig. 4.4 Administrators use case diagram

2. 用户

(1)用户信息管理

A.查看用户信息，包含对用户状态，用户角色的查看。只能查看个人的全部信息。

B.修改用户信息，包含对个人用户的信息修改。

(2)用户注册登录

A. 用户注册，以邮箱认证的方式注册用户，并且在注册完成后实现自动登录。

B. 用户登录，登录完成会自动保存角色信息，以便在访问接口时判断是否具有足够的权限。

用户的用例图如图 4.5 所示。

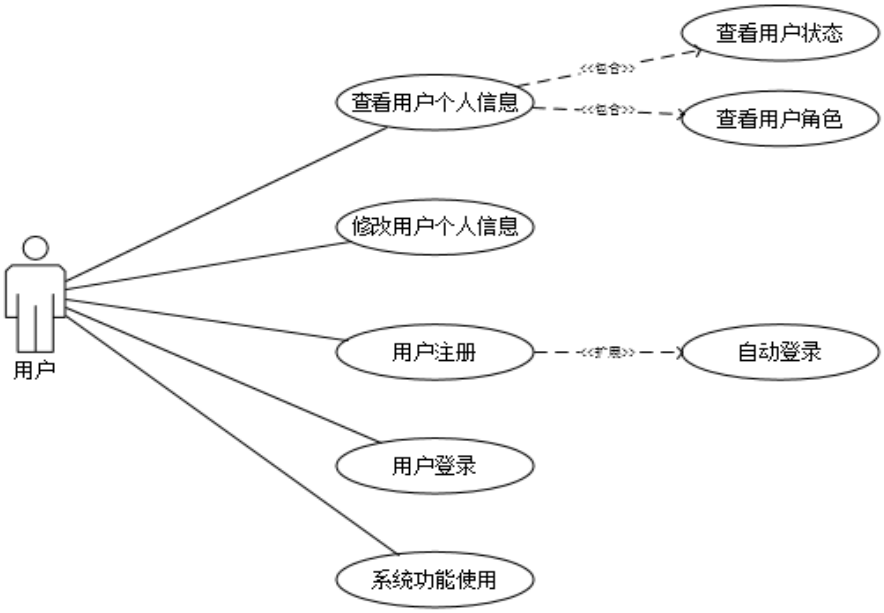

图 4.5 普通用户用例图

Fig. 4.5 User use case diagram

4.2.1 用户（管理员）注册

1. 用例说明

用户（管理员）注册用例的明细说明如下表 4.1 所示。

表 4.1 用户（管理员）注册用例明细表

Table 4.1 Search case detail table

用例名	用户（管理员）注册
参与者	用户（管理员）
前置条件	用户输入合法的邮箱、账号、密码并提交
事件流程	1. 用户输入邮箱、账号、密码进行注册提交； 2. 客户端对账号密码进行验证； 3. 客户端将请求发送到服务器； 4. 服务器对邮箱账号密码进行验证； 5. 服务器处理账号密码，存入临时空间，并向用户邮箱发送注册验证链接； 6. 用户接收邮件并点击其中的注册链接进行跳转注册； 7. 客户端自动跳转提示用户注册成功； 8. 注册成功自动进行登录；

2. 用例流程

用户注册处理流程如图 4.6 所示。

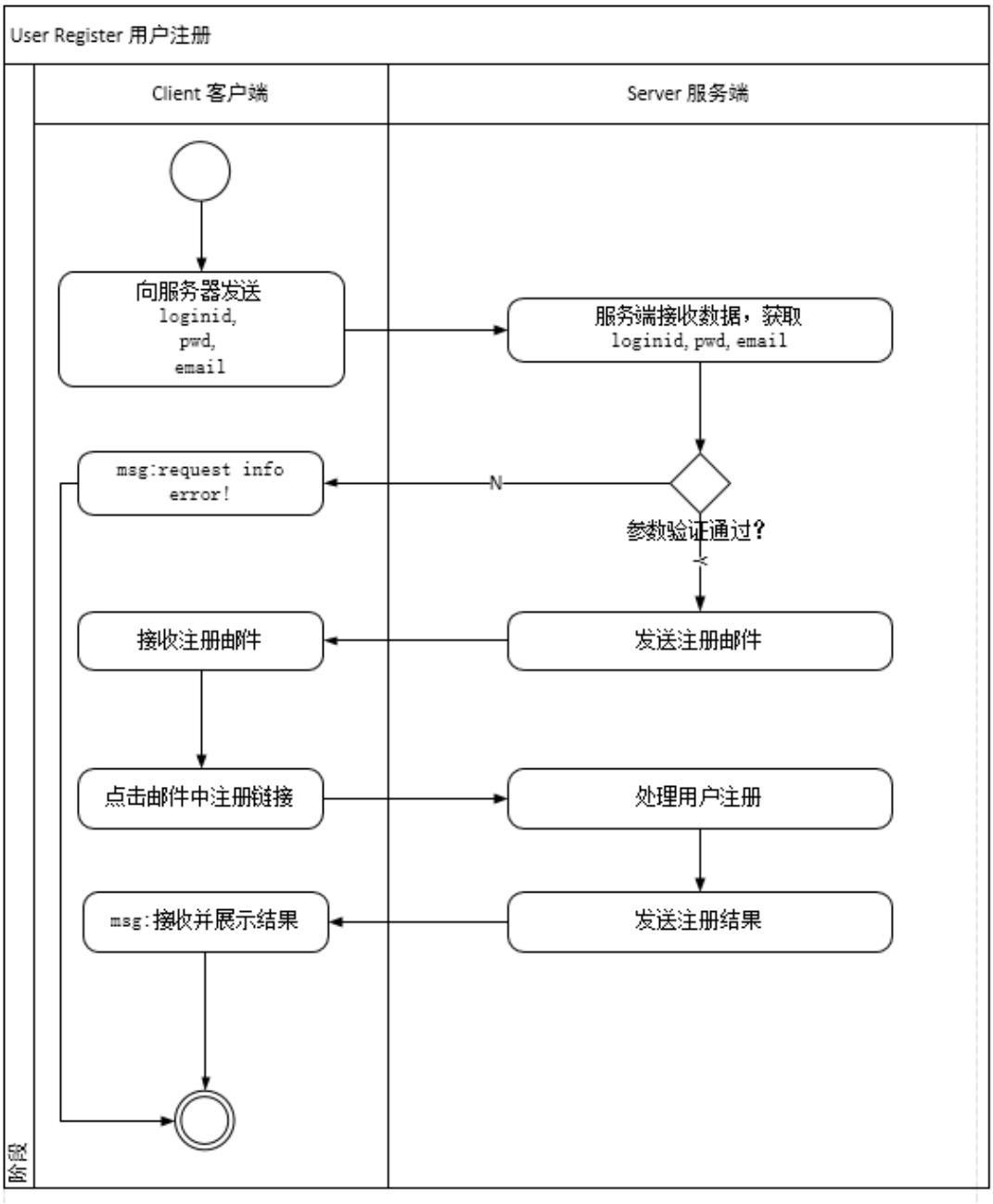

图 4.6 普通用户注册活动图

Fig. 4.6 User register activity diagram

3. 人机交互

用户注册用例人机交互图如下图 4.7 所示。

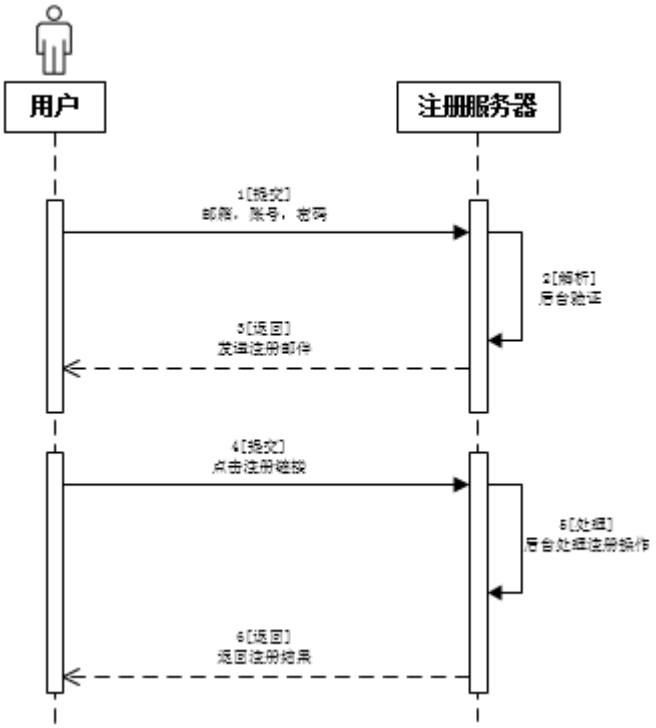

图 4.7 普通用户注册人机交互图

Fig. 4.7 User register interactive diagram

4.2.2 用户（管理员）登录

1. 用例说明

用户（管理员）登录用例的明细说明如下表 4.2 所示。

表 4.2 用户（管理员）登录用例明细表

Table 4.2 User login case diagram

用例名	用户（管理员）登录
参与者	用户（管理员）
前置条件	用户输入合法的账号、密码并提交
事件流程	1. 用户输入合法账号密码进行提交； 2. 客户端生成时间戳和随机数，并进行对发送参数的加密，加密格式如下 MD5[logiId+MD5[Pwd]+random+timestamp] 然后将 loginId+random+timestamp+MD5_String 发送到服务器； 3. 服务器对时间戳进行校验判断是否过期；对时间戳和随机数进行校验判断是否重复请求；获取服务器密码并进行同样格式的加密校验，判断是否账号密码错误； 4. 服务器生成公私钥、服务器密钥 secretKey=MD5[loginid+MD5[pwd]+timestamp]、token 签名

	tokenSign=MD5[loginid+logintimestamp+random]并将生成的结果存储到服务器端； 5. 服务器生成 token 令牌 token=RSA_publicKey[uid+loginid+expiretimestamp+tokensign]； 6. 服务器返回 appKey，secretKey，token，用户信息； 7. 客户端保存 secretKey，token，loginId；’

2. 用例流程

用户登录处理流程如下图 4.8 所示。

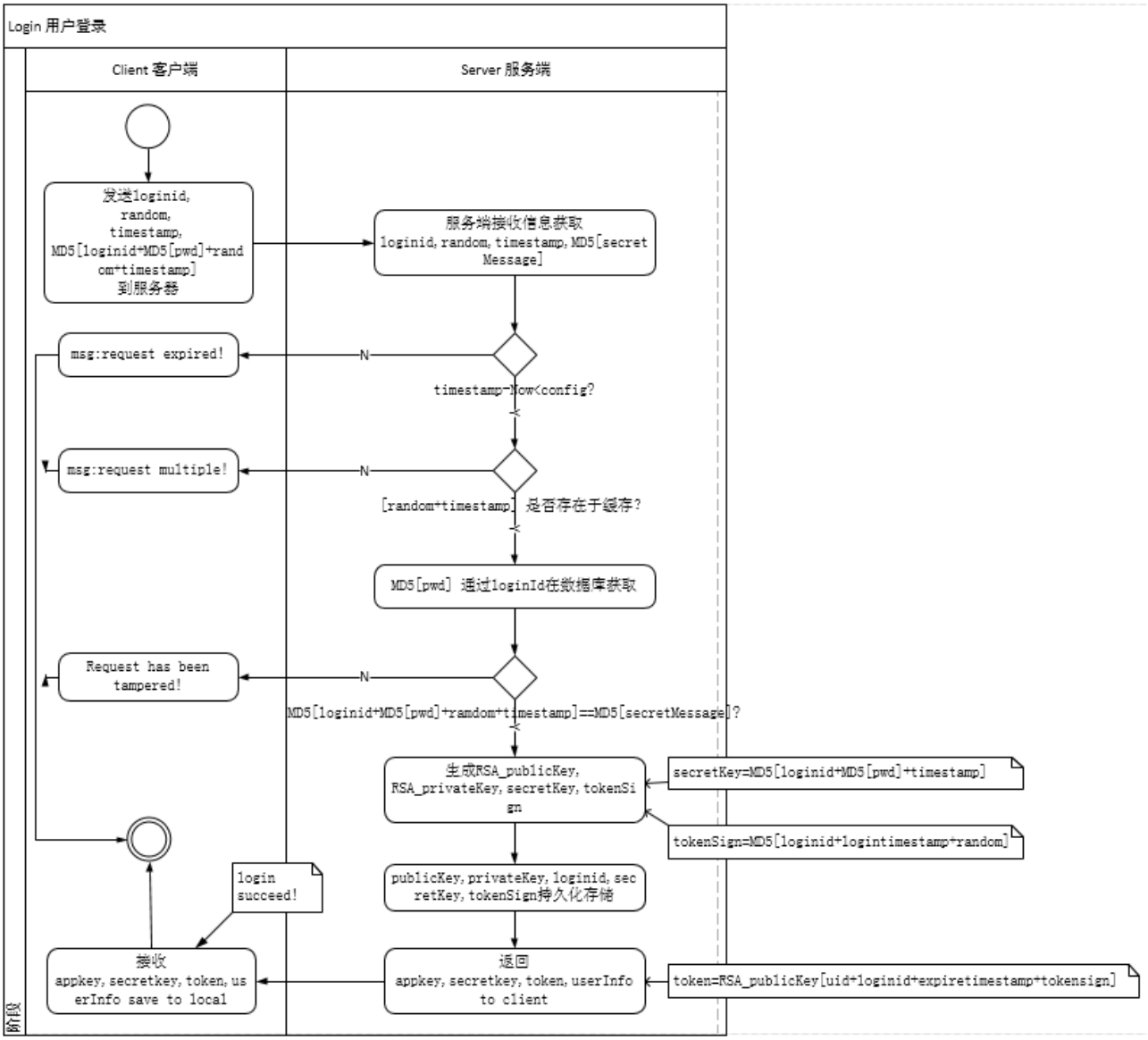

图 4.8 用户登录活动图

Fig. 4.8 User login activity diagram

3. 人机交互

用户登录用例人机交互图如下图 4.9 所示。

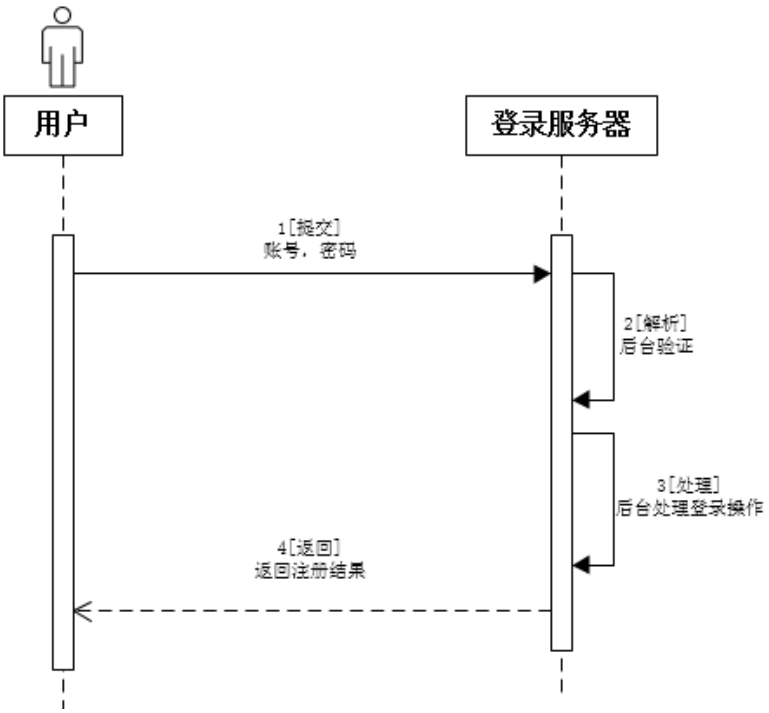

图 4.9 普通用户登录人机交互图

Fig. 4.9 User login interactive diagram

4.2.3 系统功能使用（Token 认证、权限认证、以查询用户信息为例）

1. 用例说明

用户信息查询用例的明细说明如下表 4.3 所示。

表 4.3 用户信息查询用例说明明细表

Table 4.3 User information query case diagram

用例名	系统功能使用（Token 认证，权限认证，用户信息查询为例）
参与者	用户（管理员）
前置条件	用户点击某用户的信息链接查询
事件流程	1. 用户点击某用户信息链接进行查询； 2. 客户端从本地存储空间获取 appKey、token、sign，生成 random，timestamp，请求参数（分页参数、查询条件）。Sign 的生成算法为 sign=MD5[random+timestamp+a+a.value+b+b.value+c+c.value+secretkey]； 3. 服务器对时间戳进行校验判断是否过期；对时间戳和随机数进行校验判断是否重复请求； 4. 服务端用相同的签名算法对接口参数进行验证 sign=MD5[random+timestamp+a+a.value+b+b.value+c+c.value+secretke

	y]，用客户端发送来的签名和服务端生成的签名进行匹配，如果不能匹配，即认为请求参数被篡改； 5. 服务端获取用户端的 appKey 和 Token 值，用 appKey 得到服务端的 RSA 私钥，并用私钥进行对 Token 解密，得到 uid,loginid,expiretimestamp,tokensign，通过 uid 获取用户的权限，并和当前接口权限进行对比验证，expireTimeStamp 和当前时间进行比较校验，判断该 Token 是否过期（登录信息过期，需要重新登录），tokenSign 和通过 appKey 获取的 tokenSign 进行匹配，校验 token 是否正确； 6. 验证全部通过，返回请求资源；

2. 用例流程

用户信息查询流程如下图 4.10 所示。

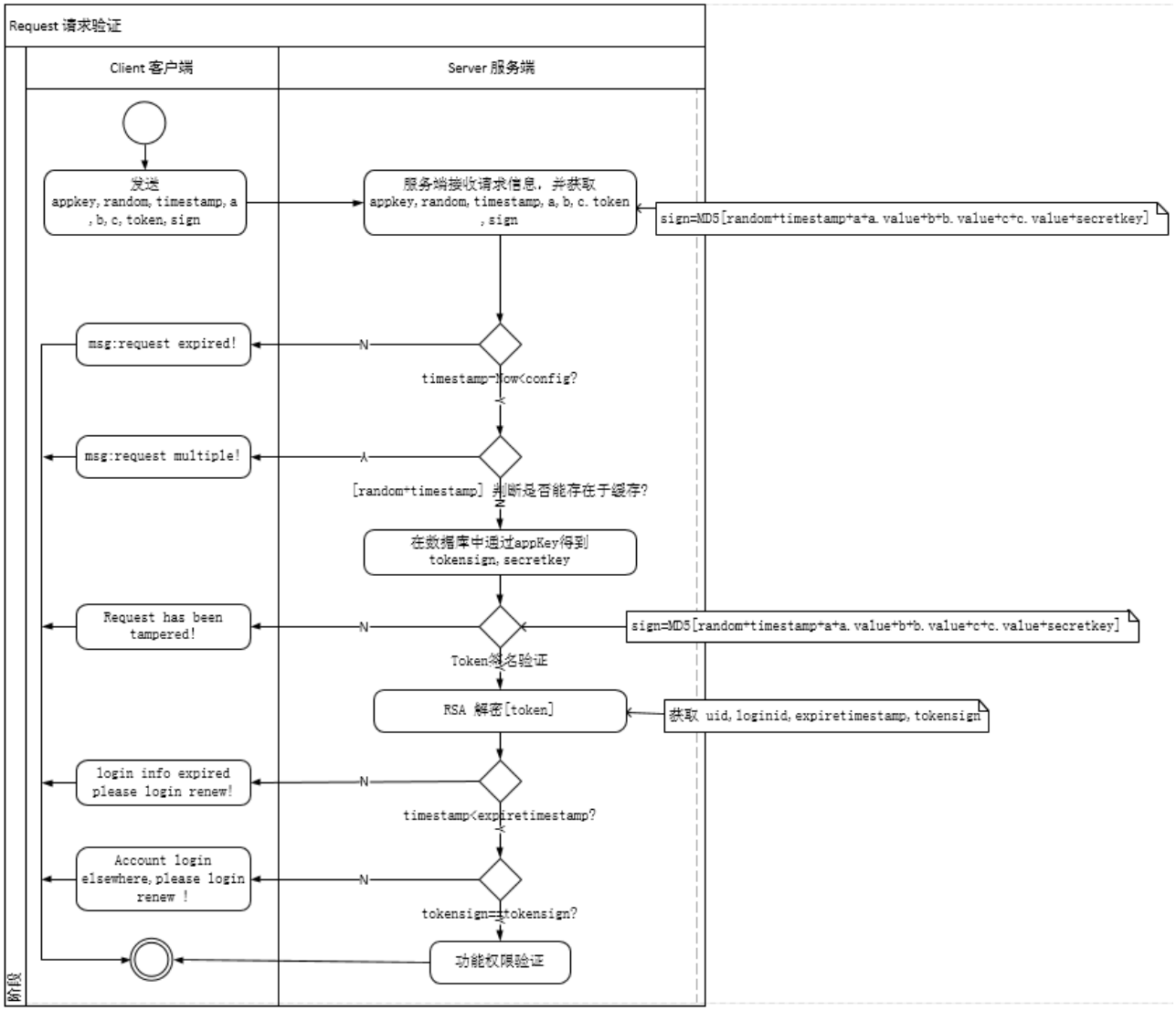

图 4.10 用户信息查询活动图

Fig. 4.10 User information query activity diagram

3. 人机交互

用户信息查询用例人机交互图如下图 4.11 所示。

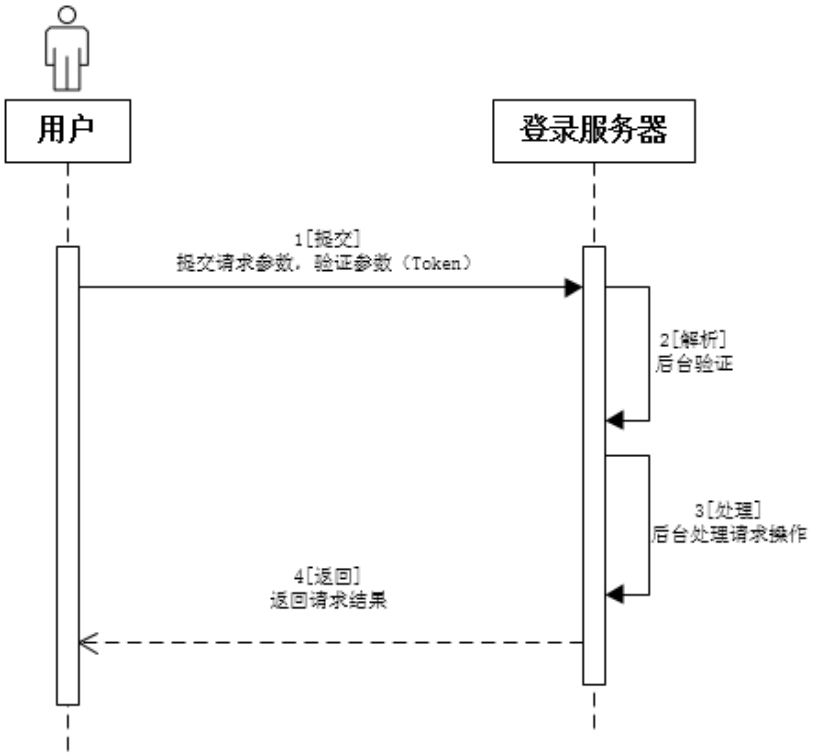

图 4.11 用户信息查询人机交互图

Fig. 4.11 User information query interactive diagram

4.3 本章小结

本章对系统用例进行了分析，并在详尽的分析的基础上得出了用例图。在对用例的详细分析时，我们用到了用例业务活动图以及用例的业务分析人机交互图，分别对用例的详细流程以及业务的人机实现过程进行了详细的描述，对以后的系统详细分析打下了良好的基础。

第五章 系统详细设计

5.1 对象模型设计

5.1.1 业务对象

根据对本系统的业务模块进行分析以及对 Token 认证的流程的掌握，再通过对当前业务的内容扩展，可以逐步提取出系统的业务实体对象以及 Token 认证的实体对象。在本系统中，将用户角色、用户状态、用户信息、Token 认证相关的对象从 AntHelp 中抽离出来，便于后续在其他系统上的重用以及扩展，故而当前框架级的业务对象有：用户账户（密保）实体、用户角色（角色描述）实体、用户状态（状态描述）实体、用户信息实体（用户账单、账单记录）、Token 认证实体等，从内容上来说，每一个实体都会有相应的扩展实体存在，而每一个扩展实体对于主实体而言，又都是至关重要的，密不可分的[9]。

5.1.2 对象属性

通过业务对象结合业务的分析，得到了初步的对象以及其属性。对初步的业务对象和框架对象进行进一步抽取、合并、扩展，降低了各对象之间的耦合性以及提升了对象的扩展性。

在当前的业务对象模型中，用户是主体，所有的业务对象模型都是为用户服务，因此，用户账号是所有业务对象的核心。这里采用用户账号和用户信息分离的模式，便于系统的设计使用，充分考虑到后续系统的扩展问题，将用户角色、用户状态、用户血型、用户性别、用户账目等抽取成单个的用户信息对象实体，既降低系统之间的耦合性又能充分考虑到后续的系统扩展。Token 认证实体则是专门用来进行 Token 认证的功能实体，当然 Token 认证实体的使用又必然会涉及到用户账户的实体，因此说，用户账户是所有系统实体的核心，所有的实体都为用户账户实体服务。最终我们得到了优化以后的业务对象模型如下图 5.1 所示。

图 5.1 业务对象模型图

Fig. 5.1 The business object model diagram

5.1.3 对象关系

通过对业务对象的分析、提取、优化，我们已然得到了我们系统所需的业务实体，并且我们的业务实体充分考虑了系统适用性及系统扩展性，因此我们所有的工作都完美完成，下一步我们需要进行对上一步抽取出来的实体对象进行对其对象之间的关系进行分析。当然，在设计实体的同时我们已经充分考虑了实体对象之间的关系，这里我们进行更详细，更具体的分析，并用图示的形式展示出来，实体对象之间的关系如图 5.2 所示。

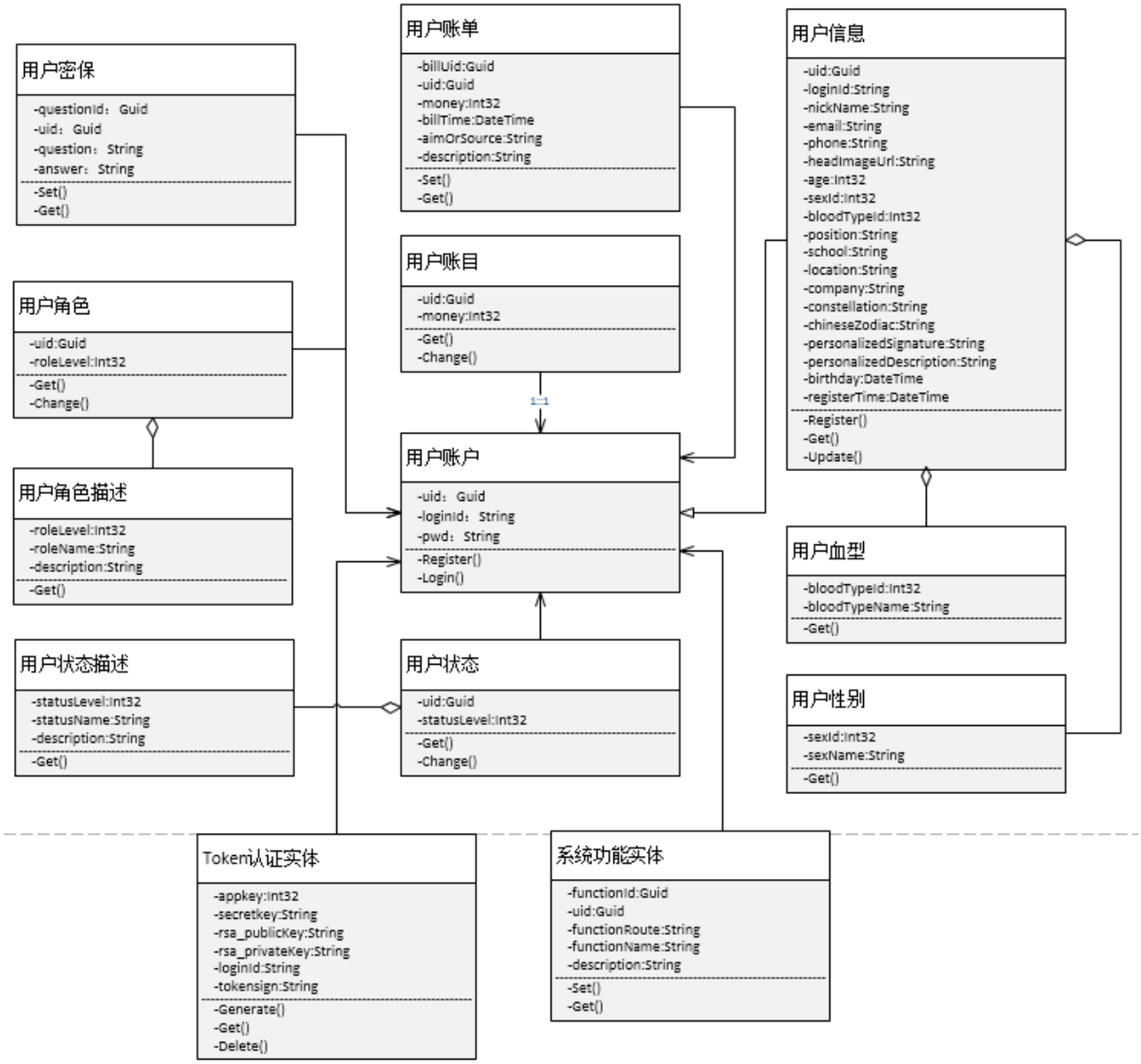

图 5.2 业务对象关系模型图

Fig. 5.2 The business object model relation diagram

5.2 业务逻辑设计

5.2.1 提取业务逻辑类

本系统的架构严格按照 QX_Frame 的标准格式进行类的创建及命名，QX_Frame 规定的项目结构为：Data、Data.Contract、Data.Service、WebApi.Controllers 因此，涉及到的类分为实体类 Data、服务数据处理接口 Data.Contract、服务数据处理类（逻辑类）、控制器

类 WebApi.Controllers。下面对本系统用到的业务逻辑类进行一一的介绍。

1. 实体类，根据本系统前面进行的业务对象模型设计，可以获取本系统所需要的业务对象，也就是本系统涉及到的业务逻辑实体类 Entities。包括用户账户类、用户信息类（用户血型类、用户性别类）、用户角色类（角色描述类）、用户状态类（状态描述类）、用户密保类、用户账目类、用户账单类、Token 认证实体类、系统功能权限实体类。
2. 实体业务逻辑类，根据框架的标准格式，实体业务逻辑类封装了每一个业务对象实体的基础增删改查操作以及对查询操作的大力扩充。在此基础上，实体业务逻辑类还包括了一些复杂的对实体的操作流程，并提供相应的方法供控制器调用。综上所述，我们的实体业务逻辑类应包括：用户账户逻辑类、用户信息逻辑类（用户血型逻辑类、用户性别逻辑类）、用户角色逻辑类（角色描述逻辑类）、用户状态逻辑类（状态描述逻辑类）、用户密保逻辑类、用户账目逻辑类、用户账单逻辑类、Token 认证实体逻辑类、系统功能权限实体逻辑类。
3. 控制器类，控制器类是提供一个对外的接口供客户端调用，除了实现对应的操作方法外，控制器类还同时肩负着对用户权限的控制，保证了系统的安全性。系统控制器类的实现采用 Restful 风格的 Action 接口，这里约定了控制类的作用是对对应资源的各种有效操作。在系统的业务逻辑分析详尽的基础上，我们提取出了必要的需要操作的资源，并且把现有资源实现到控制器中：账号控制类、用户控制类、权限控制类、登录控制类、用户金额控制类、用户角色控制类（角色描述控制类）、用户状态控制类（状态描述控制类）、文件控制类。
4. 边界类，在控制器类实现的过程中，系统的权限控制是所有的接口都需要使用的公共部分，因此将系统权限的认证类抽离出来进行封装成一个统一的过滤器类来进行使用，这里的 Token 认证、权限控制类属于各业务控制器类的边界类，由控制器类实现的过程中统一进行调用。除此之外，系统涉及到的边界类还应该有统一的异常处理控制，实现功能必不可少的 Helper 帮助，这都是在系统的实现中和系统业务逻辑类密不可分的功能类。

5.2.2 业务逻辑类分析

1. 用户（管理员）注册

在用户（管理员）注册用例的逻辑关系中，需要涉及到两个控制器类（UserAccount 控制类和 User 控制类），两个控制类的先后调用完成了整个的用户注册模块用例逻辑。在控制类的实现中，分别用事务的方式对 UserAccountService、UserAccountInfoService、UserRoleService、UserStatusService、UserMoneyService 服务类进行了调用，服务类又通过统一的数据上下文完成对实体的操作。在各层类的调用中，不可避免的要使用到边界类，这里我们主要使用的边界类由 Helper_DG，由 qx_Frame 框架提供的帮助类库还有 Exception_DG 由 qx_Frame 提供的通用异常处理类，简化了业务逻辑的实现。用户（管理员）注册用例逻辑类图如下图 5.3 所示。

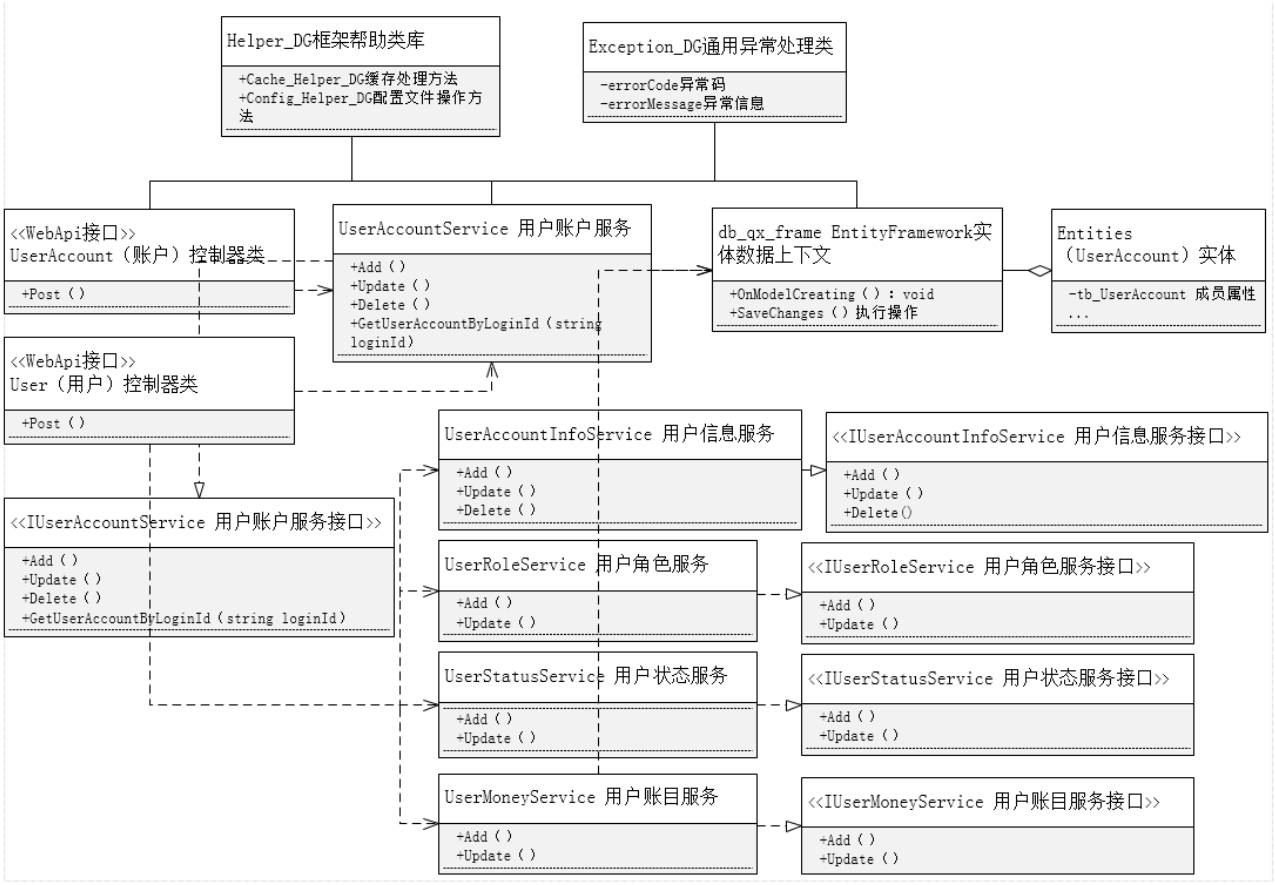

图 5.3 注册用例业务逻辑类图

Fig. 5.3 Registered business logic class diagram

2. 用户（管理员）登录

在用户（管理员）登录用例逻辑类中，实现了 Login 控制类。Login 控制类主要用到了 UserAccountService 服务类和 AuthenticationService Token 认证服务类，这两个服务类分别实现了对用户的账户信息进行查询以便于登录、对用户登录后 Token 值的生成以及对 appKey 等 Token 认证相关信息进行存储，这里调用了 db_qx_frame Entities 实体数据上下文对想关实体进行持久化存储。在控制类以及服务类的实现过程中，不可避免的要使用到相关的边界类，这里我们主要使用的边界类由 Helper_DG，由 qx_Frame 框架提供的帮助类库还有 Exception_DG 由 qx_Frame 提供的通用异常处理类，再 Helper_DG 中主要用到了加密解密的算法方法，在系统安全方面有了加密解密的支持，可以使得安全性进一步提升，同时也简化了业务逻辑的实现。用户（管理员）登录用例逻辑类图如下图 5.4 所示。

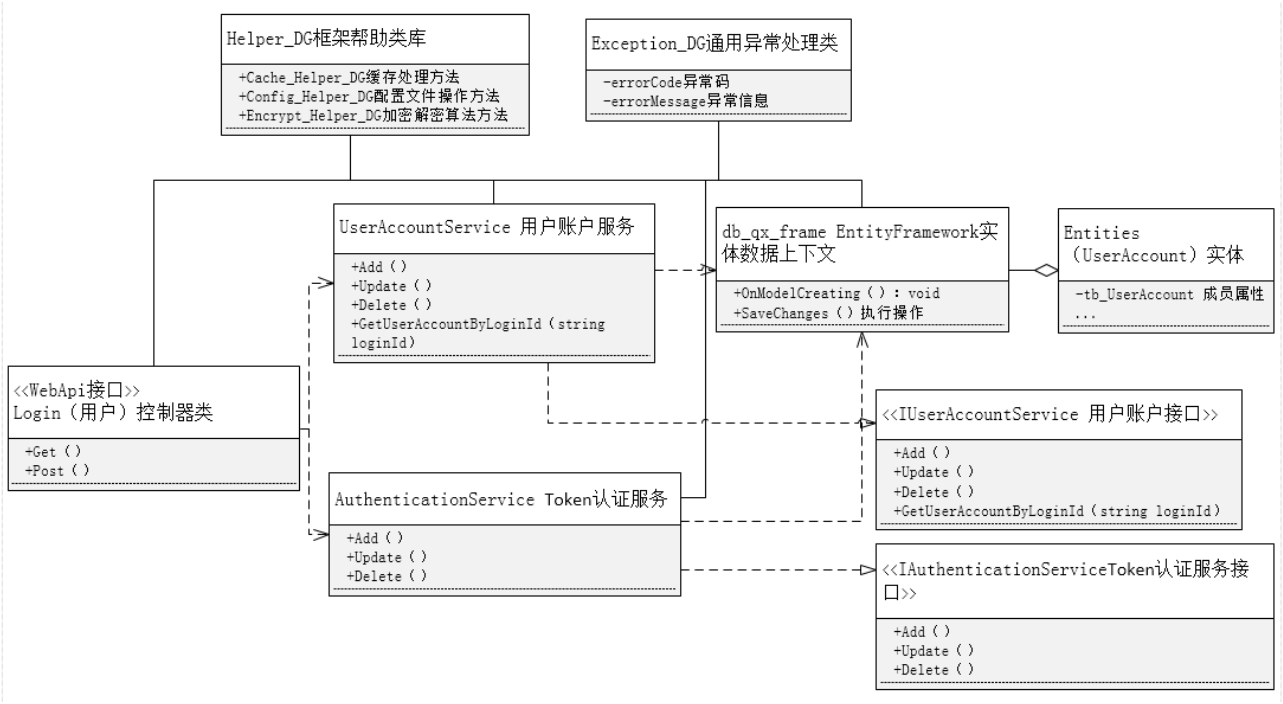

图 5.4 登录用例业务逻辑类图

Fig. 5.4 Login business logic class diagram

3. 接口请求 Token 认证（以用户信息查询接口为例）

在整个系统架构中，我们以过滤器的实现方式实现了对用户角色权限的认证，当然我们无法对所有的请求都进行一一描述，这里我们仅对用户信息的查询接口进行用户角色权限认证流程的分析展示，并且该方式以过滤器方式实现，通过配置可以轻松地进行类的复用。User 控制类，调用了 UserAccountService 服务类进行对用户信息的查询，在对用户详细信息的查询中还会涉及到对 UserAccountInfoService 服务类的调用，以及对 UserRoleService、UserStatusService、UserMoneyService、BloodTypeService 等服务类的调用，这里仅以获取用户的账户信息来进行展示。除此之外，我们还利用了 LimitsAttribute_DG 的特性标签过滤器类，该类主要用户给 User 控制类添加特性标签，以便在调用接口时会同时进行用户权限以及 Token 签名的认证，该权限验证类调用了 AuthenticationService Token 认证服务以及 UserRoleService 用户角色服务、UserStatusService 用户状态服务。所有的服务类又都依赖于 db_qx_frame Entities 实体数据上下文类来进行对持久化层实体类的属性值的获取，又反向提供到服务类，继而转到控制层来对客户端信息的反馈。接口请求 Token 认证用例逻辑类图如下图 5.5 所示。

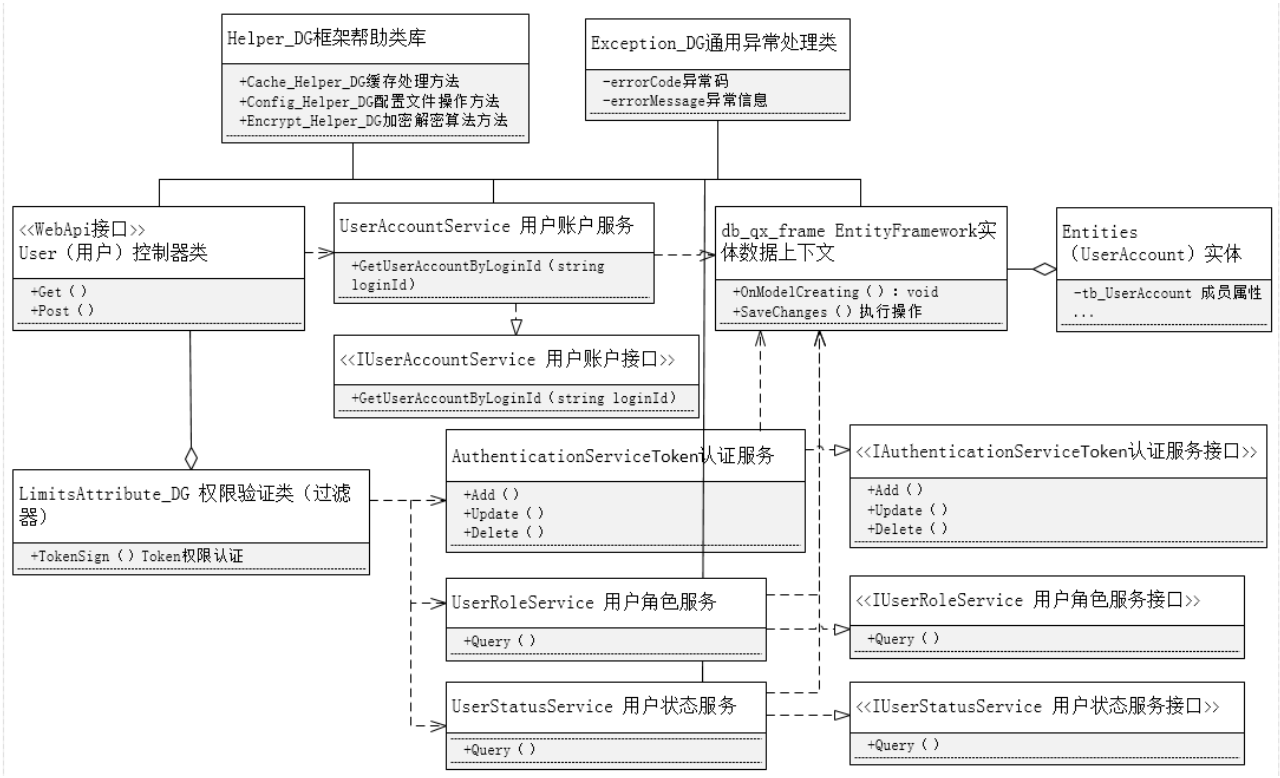

图 5.5 用户信息查询用例业务逻辑类图

Fig. 5.5 User information query business logic class diagram

5.2.3 业务逻辑交互分析

实现一个用例涉及到边界类、控制类和实体类三种逻辑要素，这三种类通过消息交互完成用例功能。下面我们将针对前步骤中的用例分析以及逻辑类流程分析，进而在本节中对逻辑类之间的交互关系进行更加详细的分析，这将有助于系统代码的编写实现，使得系统的完成更具有条理性、逻辑性，对系统的实现起到了很大的帮助作用。

1. 用户（管理员）注册

用户（管理员）的注册主要需要两大控制类以及一个用户账户服务类，实体类和数据库上下文类完成，当然在系统的实现过程中免不了对边界类诸如 QX_Frame.Helper_DG 类的使用以及统一异常处理类 Exception_DG 的使用。在用户注册的功能实现流程中，我们以客户端作为用户的代言，在用户操作客户端进行注册的时候，客户端会向账户控制类发送请求，意为意图注册，并在缓存中进行处理，向用户填写的邮箱中发送一个真实注册的链接。用户通过接收邮件并点击邮件中的链接导向二次注册的界面，这个界面也将作为其后进行显示注册结果的界面。

在用户跳转到二次注册的界面后，系统客户端会向 UserController 发送一个二次注册的请求，控制类调用 UserAccountService 服务类，进行对真实信息注册的处理，包括在事务条件下对用户信息的初始化以及对用户权限的初始化。进而更新到实体模型，数据上下文进行将实体对象模型持久化，最终由 UserAccountService 服务类返回处理结果。用户注册的逻辑

模型如下图 5.6 所示。

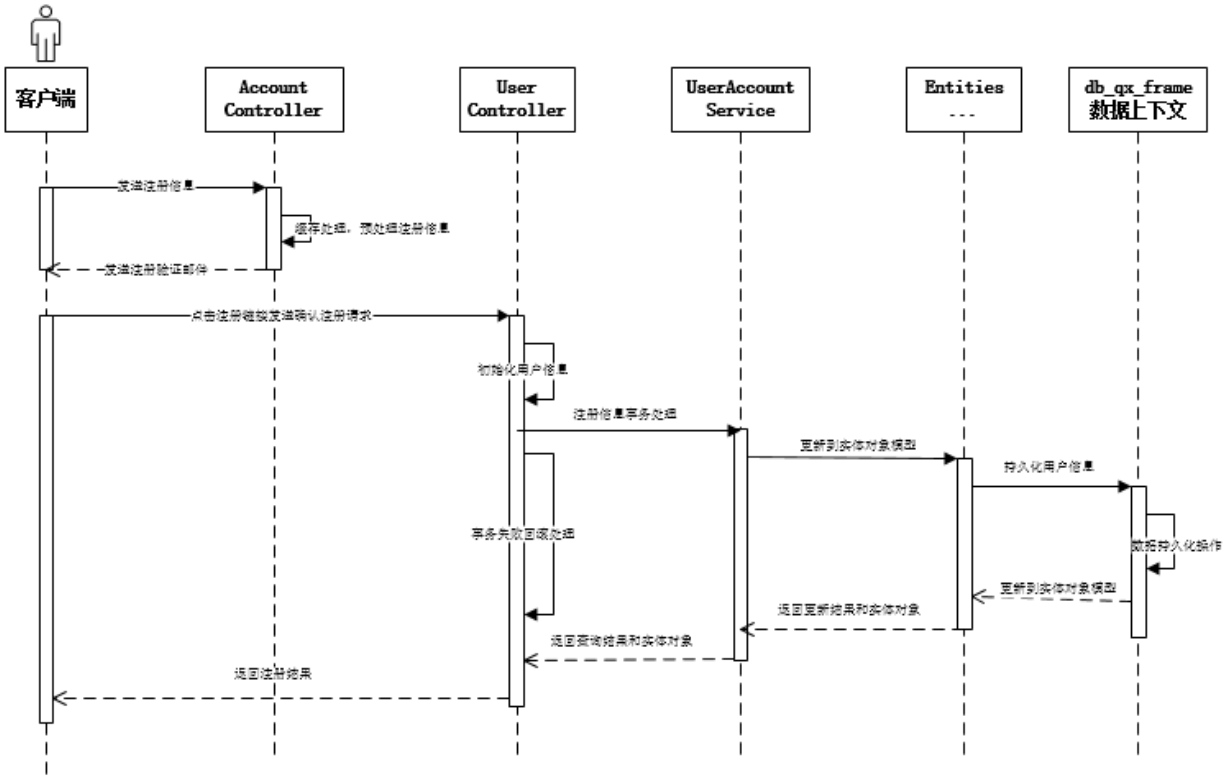

图 5.6 注册用例业务逻辑类顺序图

Fig. 5.6 Register business logic class sequence diagram

对功能逻辑的说明如下表 5.1 所示。

表 5.1 注册用例功能逻辑说明表

Table 5.1 Registration case logic function specification table

名称	类型	说明	资源
客户端	参与者	表示任何请求发起者	-
AccountController	控制类	用户账户的对外控制接口	AccountController.cs
UserController	控制类	用户信息对外控制接口	UserController.cs
UserAccountService	实体类	用户账户服务类	UserAccountService.cs
Entities	实体类	数据对象实体	QX_Frame.Data.Entities.cs
db_qx_frame 数据上下文	实体类	持久化操作数据上下文	db_qx_frame.cs

2. 用户（管理员）登录

用户登录用例的实现主要由 LoginController 控制类来完成的，该业务逻辑的流程为用户在客户端操作任何接口的时候都会触发对登录状态的判断，这里的判断会涉及到下一个业务逻辑即 Token 认证的业务逻辑，在本个业务逻辑分析中，先简单进行分析。LoginController 登录控制器在接收到用户的判断是否登录的请求后，先进行基本的判断，继而将客户端发送来的 Token 签名利用 AuthenticationService 服务类进行处理，在这个服务类中，通过获取持久化反馈的实体对象信息，进行查询解密秘钥，并进行对 Token 签名的解密操作，解密后

对 Token 的进一步判断来确定用户是否算作已经登录。

如果在上一步的用户登录状态判断中，结果为用户没有登录也就是用户登录信息过期或者从未登录，那么，用户就需要跳转到登录界面进行登录。这里客户端在登录界面向 LoginController 发送登录请求，登录控制类接收到登录请求后，首先对登录信息进行有效性的验证（使用相同的加密算法对持久化层的信息进行加密，然后进行匹配），如果有效性验证成功，下一步会由登录信息使用通过 Authentication 逻辑类调用一系列加解密算法生成 Token 签名值和 Token 值，进而进行对象信息持久化操作，最终向客户端返回 Token 值可，并由客户端进行本地化存储。其逻辑模型如下图 5.7 所示。

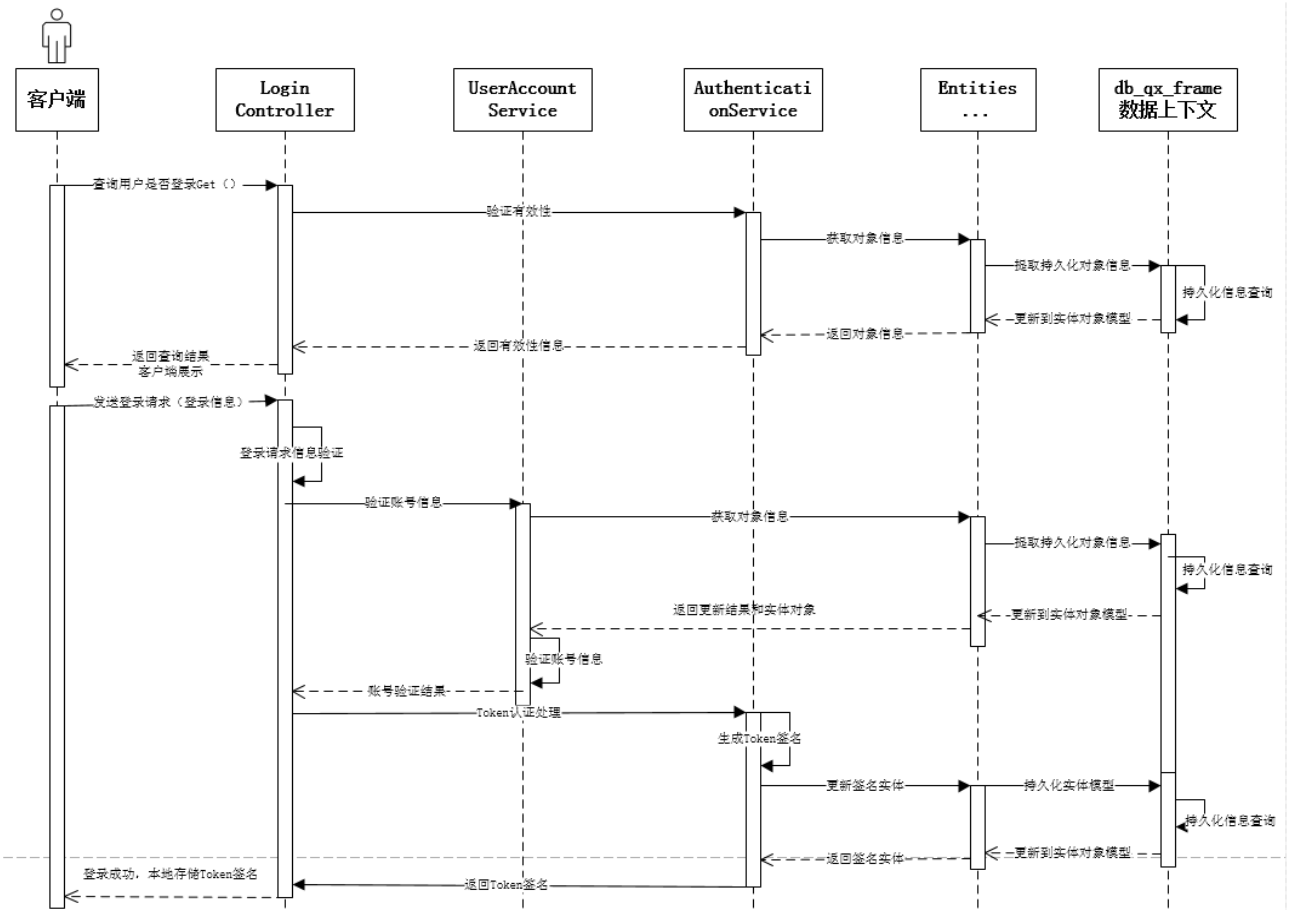

图 5.7 登录用例业务逻辑类顺序图

Fig. 5.7 Login business logic class sequence diagram

对功能逻辑的说明如下表 5.2 所示。

表 5.2 登录用例功能逻辑说明表

Table 5.2 Login case logic function specification table

名称	类型	说明	资源
客户端	参与者	表示任何请求发起者	-
LoginController	控制类	用户登录的对外控制接口	LoginController.cs
AuthenticationService	实体类	权限控制服务类	AuthenticationService.cs
UserAccountService	实体类	用户账户服务类	UserAccountService.cs
Entities	实体类	数据对象实体	QX_Frame.Data.Entities.cs
db_qx_frame 数据上下文	实体类	持久化操作数据上下文	db_qx_frame.cs

3. 用户信息查询（Token 认证、权限认证）

在本节业务逻辑分析中，我们欲分析 Token 认证以及权限认证，但是我们这里无法泛泛地对 Token 认证以及权限认证进行描述，因为这里的认证应该是面向所有的业务逻辑接口的，因此，我们以用户信息查询这一接口为例对 Token 认证以及权限认证来进行描述。在用户信息查询中，我们以客户端代表用户的所有操作，主要功能靠 UserController 用户控制器以及 LimitAttribute_DG 权限验证类来进行业务操作。

当用户或管理员欲查询用户信息时，由客户端向 UserController 控制类发送查询请求，控制类的执行会先被 LimitAttribute_DG 权限验证类进行拦截，在权限验证类中，会直接调用 AuthenticationService 服务类进行权限实体对象信息的获取以及对权限信息分析，最终反馈到权限验证类中，在权限验证类中，会对用户的权限进行和参数进行匹配验证，如果不能达到相应的权限等级，服务端会拒绝请求，并直接向客户端返回异常信息。在权限验证类中，我们同时会对 Token 值和 Token 签名的分析，同样要使用到 AuthenticationService 服务类来对对象实体的获取以及对信息的有效性进行检查判断。如果上一步的判断全部生效，那么权限控制类会将控制权交由 UserController 用户控制类，由用户控制类进行下一步的用户信息的处理操作。

在上一步中，权限控制类已经对请求进行了过滤以及对权限和 Token 进行了验证，如果验证通过，那么系统的控制权又便会交由 UserController，我们在 UserController 用户控制类中，继续对调用 UserAccountService 用户账户服务类来进行对用户信息的实体对象持久化信息的获取。获取的用户信息会以一定的格式展示到客户端供用户参考使用，其逻辑模型如下图 5.8 所示。

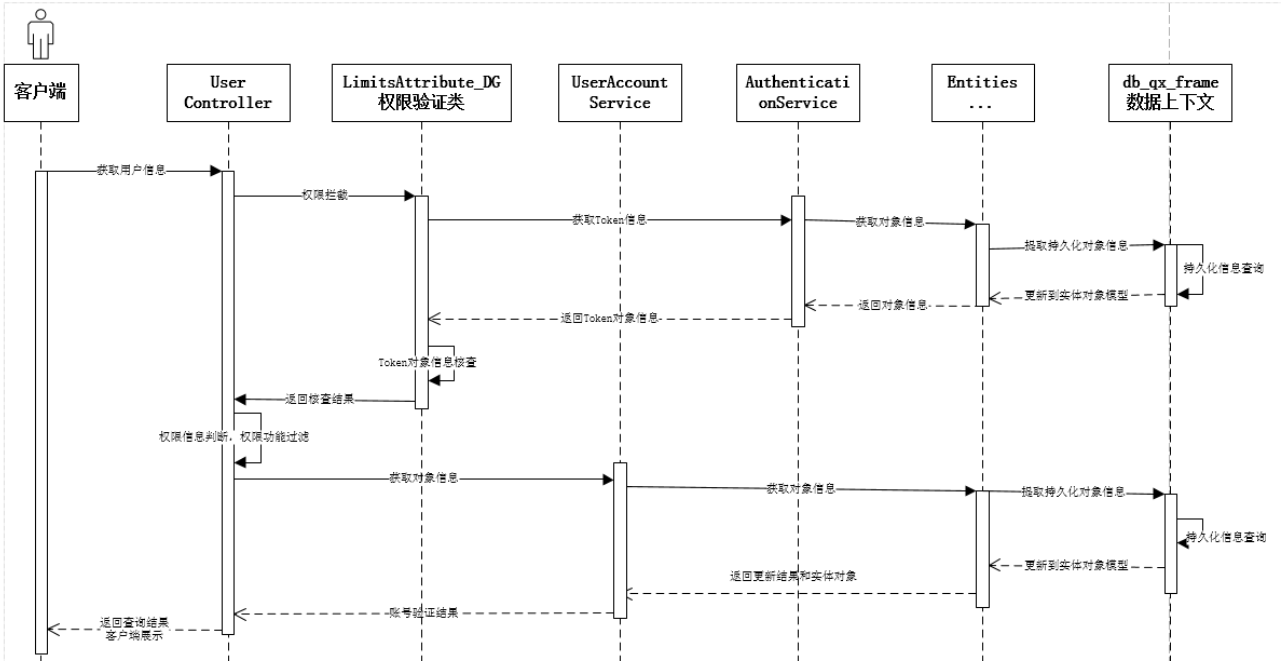

图 5.8 用户信息查询用例业务逻辑类顺序图

Fig. 5.8 User information query business logic class sequence diagram

对功能逻辑的说明如下表 5.3 所示。

表 5.3 注册用例功能逻辑说明表

Table 5.3 Registration case logic function specification table

名称	类型	说明	资源
客户端	参与者	表示任何请求发起者	-
UserController	控制类	用户登录的对外控制接口	UserController.cs
LimitAttribute_DG	实体类	用户权限过滤器类	LimitAttribute.cs
AuthenticationService	实体类	权限控制服务类	AuthenticationService.cs
UserAccountService	实体类	用户账户服务类	UserAccountService.cs
Entities	实体类	数据对象实体	QX_Frame.Data.Entities.cs
db_qx_frame 数据上下文	实体类	持久化操作数据上下文	db_qx_frame.cs

5.3 数据库设计

5.3.1 系统数据库表分析

根据前面的业务对象模型设计接口，可以很轻松的分析出本系统所需要数据库表以及表之间的主外键关联关系。最终的出所需要设计的数据表为：用户账户表、用户角色表、角色描述表、用户状态表、状态描述表、用户信息表、用户血型表、用户性别表、用户账目表、账单记录表、Token 认证表、密保问题表、系统功能权限表。在系统表的主外键设计过程中，由于系统的初衷是通用的用户管理模板，基于插拔式的用户管理系统，在后续的系统架构中对系统的扩展性要求较高，因此系统在主外键关联关系上并不是很紧密的设计，都是为了能在后续便于扩展[10]。通过舍弃系统表之间的范式约束，得到了系统的松耦合、高复用性、高扩展性的优势，凡事有舍必有得，由此可以看出。

本系统设计的数据库都是以用户账号表为核心，在系统的实现中也是先实现用户账号的业务流程，再实现其他后续的业务流程，因为所有和用户账号相关联的表都是以用户账号表的 UID 字段为基础进行后续的实现扩展的，这里在对表的操作需要用事务进行。

下面先以图示的方式将系统用户管理、Token 认证部分的数据库表展示出来，如图 5.9 所示。

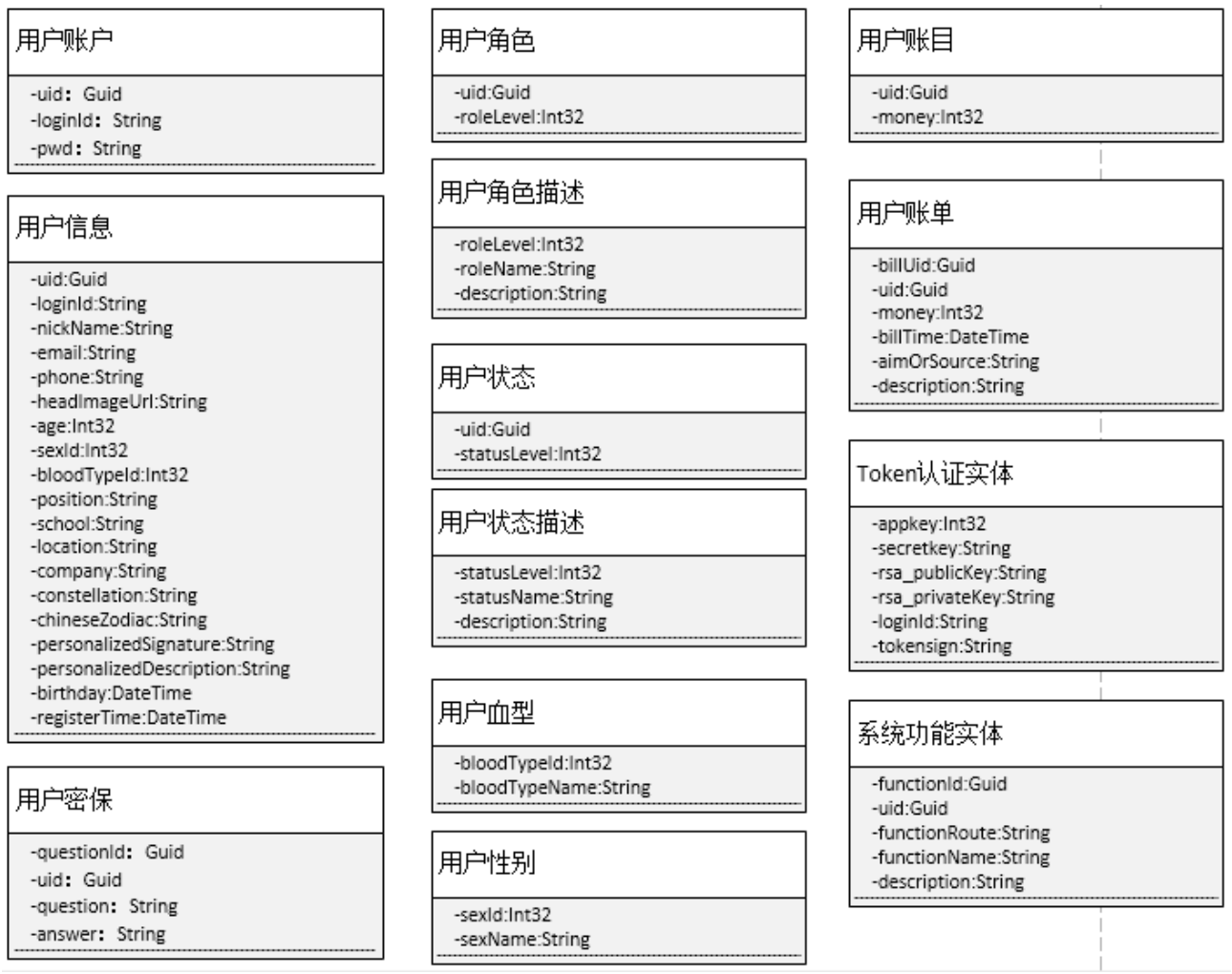

图 5.9 数据库表结构设计图

Fig. 5.9 The database table structure design

5.3.2 系统数据库表关系分析

在上面的数据库表结构展示中，我们可以清晰地看到我们业务逻辑实现所要依赖的数据库表，在上述的表结构中缺少了对主外键的关联展示，我们用 Microsoft SQL Server 自带的数据库表关联图示来进行对表之间的关联关系进行展示，如图 5.10 所示。

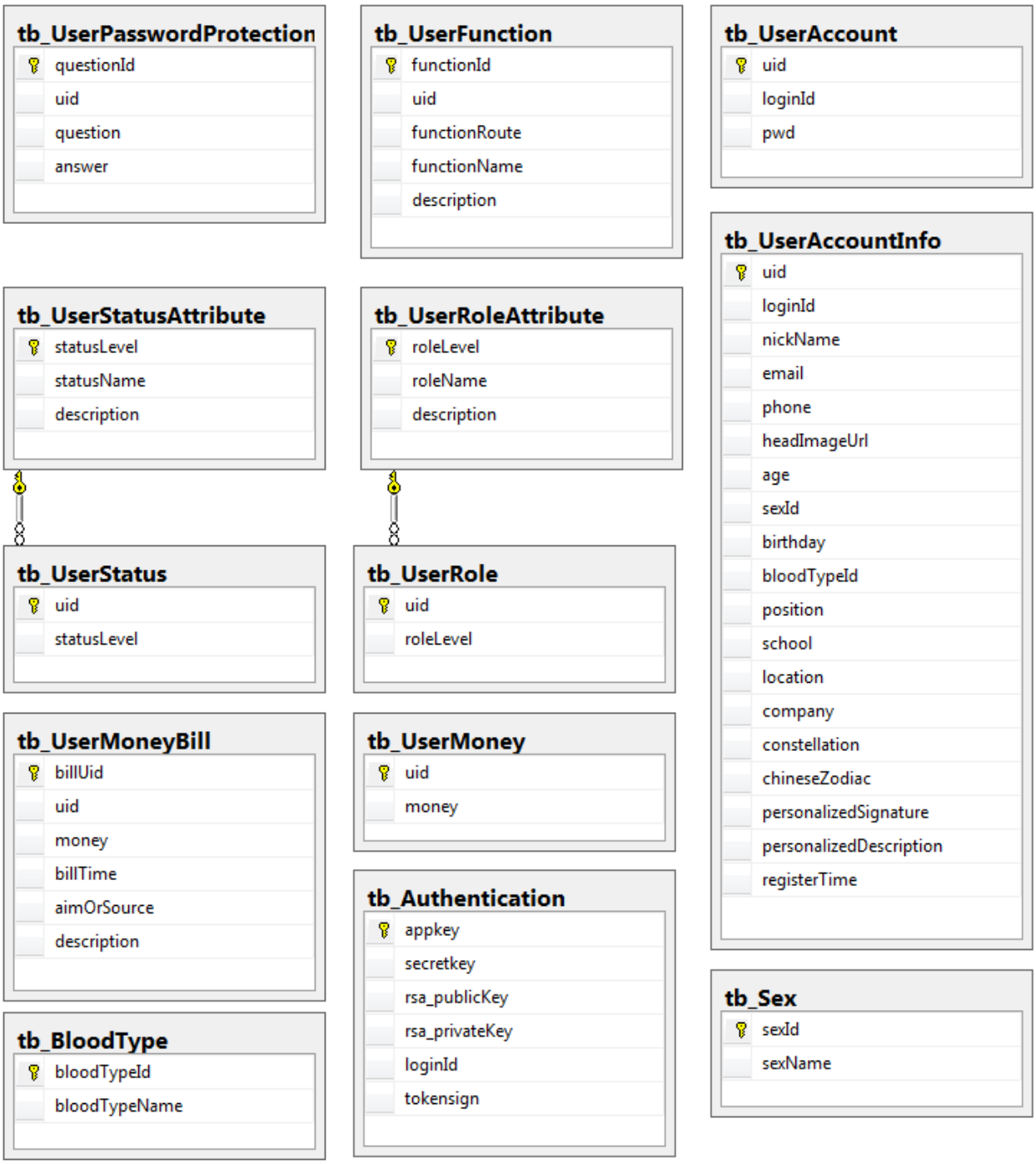

图 5.10 数据库表结构关系设计图

Fig. 5.10 The database table structure relation design

依据设计，我们的用户账号表和用户信息表等其他表之间的关联不主张主外键的强关联形式，而用统一的 Uid 来进行表之间的关联，在对数据库的操作中，如果涉及到对表之间关系冲击操作，我们会在代码段中用事务的方法来进行统一的管理。这样设计的优势在于表之间的联系适当的降低，范式约束的消除，但是我们得到的优势却在于后续扩展的更加方便得当。[11]

5.3.3 系统数据库表描述

我们在分析完数据库的表以及数据库表之间的关系之后，应当进而对每一张数据库表进行剖析，逐个分析每个字段的定义类型以及字段对应的说明，并形成相应的表格，这样不仅有利于我们对系统的结构一目了然，同时也会在后续对系统设计中对表的查阅起了很关键的作用。下面我们对我们系统中所用到的表进行逐一的分析。

1. 用户账户表 tb_UserAccount 说明如下表 5.4 所示。

表 5.4 用户账户表结构说明表

Table 5.4 User Account structure specification table

键名	类型	说明
uid	uniqueidentifier	用户 uid
loginId	nvarchar	用户登录 id
pwd	nvarchar	密码（Md5 加密）

2. 用户信息表 tb_UserAccountInfo 说明如下表 5.5 所示。

表 5.5 用户信息表结构说明表

Table 5.5 User information structure specification table

键名	类型	说明
uid	uniqueidentifier	用户 uid
loginId	nvarchar	用户登录 id
nickName	nvarchar	昵称
email	nvarchar	邮箱
phone	nvarchar	电话号码
headImageUrl	nvarchar	头像链接地址
age	int	年龄
sexId	int	性别 id
birthday	datetime	出生日期
bloodTypeId	int	血型 id
position	nvarchar	职位
school	nvarchar	学校
location	nvarchar	地址
company	nvarchar	公司
constellation	nchar	星座
chineseZodiac	nchar	生肖
personalizedSignature	nvarchar	个性签名
personalizedDescription	nvarchar	个人说明
registerTime	datetime	注册时间

3. 用户密保表 tb_UserPasswordProtectionQuestion 说明如下表 5.6 所示。

表 5.6 用户密保表结构说明表

Table 5.6 User secret protect structure specification table

键名	类型	说明
questionId	uniqueidentifier	问题 id
uid	uniqueidentifier	用户 uid
question	nvarchar	问题
answer	nvarchar	答案

4. 用户角色表 tb_UserRole 说明如下表 5.7 所示。

表 5.7 用户角色表结构说明表

Table 5.7 User role structure specification table

键名	类型	说明
uid	uniqueidentifier	用户 uid
roleLevel	int	角色等级

5. 用户角色描述 tb_UserRoleAttribute 说明如下表 5.8 所示。

表 5.8 用户角色描述表结构说明表

Table 5.8 User role description structure specification table

键名	类型	说明
roleLevel	int	角色等级
roleName	nvarchar	角色名称
description	nvarchar	角色描述

6. 用户状态 tb_UserStatus 说明如下表 5.9 所示。

表 5.9 用户状态表结构说明表

Table 5.9 User status structure specification table

键名	类型	说明
uid	uniqueidentifier	用户 uid
statusLevel	int	状态等级

7. 用户状态描述 tb_UserStatusAttribute 说明如下表 5.10 所示。

表 5.10 用户状态描述表结构说明表

Table 5.10 User status description structure specification table

键名	类型	说明
statusLevel	int	状态等级
statusName	nvarchar	状态名称
description	nvarchar	状态描述

8. 用户血型 tb_BloodType 说明如下表 5.11 所示。

表 5.11 用户血型表结构说明表

Table 5.11 User blood type structure specification table

键名	类型	说明
bloodTypeId	int	血型 id

bloodTypeName	nchar	血型名称

9. 用户性别 tb_Sex 说明如下表 5.12 所示。

表 5.12 用户性别表结构说明表

Table 5.12 User sex structure specification table

键名	类型	说明
sexId	int	性别 id
sexName	nchar	性别名称

10. 用户账目 tb_UserMoney 说明如下表 5.13 所示。

表 5.13 用户账目表结构说明表

Table 5.13 User money structure specification table

键名	类型	说明
uid	uniqueidentifier	用户 uid
money	Int	余额

11. 用户账单 tb_UserMoneyBill 说明如下表 5.14 所示。

表 5.14 用户订单表结构说明表

Table 5.14 User order structure specification table

键名	类型	说明
billUid	uniqueidentifier	账单 id
uid	uniqueidentifier	用户 uid
money	int	金额（+-）
billTime	datetime	交易时间
aimOrSource	nvarchar	目标或来源
description	nvarchar	描述

12. Token 认证实体 tb_Authentication 说明如下表 5.15 所示。

表 5.15 Token 认证表结构说明表

Table 5.15 Token sign structure specification table

键名	类型	说明
appkey	int	appKey
secretkey	nvarchar	secretKey
rsa_publicKey	nvarchar	RSA 公钥
rsa_privateKey	nvarchar	RSA 私钥
loginId	nvarchar	用户登录 id
tokensign	nvarchar	Token 签名

13. 系统功能实体 tb_UserFunction 说明如下表 5.16 所示。

表 5.16 系统功能实体表结构说明表

Table 5.16 System functin structure specification table

键名	类型	说明
functionId	uniqueidentifier	功能权限 id

uid	uniqueidentifier	用户 uid
functionRoute	nvarchar	功能路由
functionName	nvarchar	功能名称
description	nvarchar	描述

5.4 本章小结

在本章中，我们对系统的业务对象进行了详细的分析，明确了系统的实现中需要设计的业务对象，并且对对象之间的关系进行了适当的分析，用图示的方式展示出来。而后我们进行了对业务逻辑类的分析以及类之间的关系的描述，在类逻辑关系的分析中，我们还对类之间的调用关系用顺序图进行表述。类之间调用关系的顺序图直接对应了系统中代码的实现，使得我们系统代码的实现由了图例的参考，起到了工欲善其事，必先利利器的效果。最后，我们对系统的所用到的数据库表进行了详细的分析，得出了数据库表的详细描述表，不管是在以后项目的维护还是如今系统的搭建，对于系统功能的实现都是有极大的好处的，使得我们的系统实现根据有条理性。

第六章 系统实现与代码编写

6.1 系统实现

通过为期两个月的项目编码周期，本项目编码工作如期完成，在项目的实现过程中，不可避免的要出现各类的 Bug，但我们不畏艰险，敢于直面困难，每次遇到困难都是先个人分析，不能解决再进行小组讨论，最后总能很好的解决问题。经过我们的不懈努力，项目最终的结果很令人满意，同时对于自己也是一次不错的收获。

项目开发的过程总是缓慢而又快速的，好在我们有良好的系统架构以及有丰富经验的编码成员，保证了系统的正常实现。下面我们对我们的系统架构进行简单地介绍。

6.1.1 系统架构简介

我们的项目框架采用柒小前后台通用开发框架，由本人独立研发于 2016 年，进行为期半年的系统架构,终于成功承载了各类大中小型项目,为各类系统的研发有了成熟的框架基础。我们的蚂蚁热帮系统就是建立在 QX_Frame 的基础之上的。我们的系统主要分为 QX_Frame.Data 、 QX_Frame.Data.Contract 、 QX_Frame.Data.Service 、 QX_Frame.Web 、 QX_Frame.Web.Srv、QX_Frame.WebApi、QX_Frame.WebApi.Test 七层。下面我们对每一层起到的作用进行逐一的介绍。

1. QX_Frame.Data，Data 层是数据的核心，我们系统用的实体类 Entities、数据传输对象 DTO、枚举类 Enum、选项类 Options、QueryObject 查询对象甚至封装的底层数据持久化 ORM db_qx_frame、db_AntHelp 都是搭建于 Data 层的。Data 层是大部分层所依赖的数据层。
2. QX_Frame.Data.Contract，Contract 层是 Service 服务层的接口，这里我们存放的都是服务层的规范说明，在后续的维护中，我们需要及时参考 Contract 层来进行系统的重构以及扩展，这里对系统的维护性起到了很关键的作用。有利于系统的长久健康发展。
3. QX_Frame.Data.Service，如果说 Data 层是数据的核心，那么 Service 层就是业务的核心，Service 层包含了系统中的主要业务，也可以说是大部分业务，因为简单地业务也会在控制器类中得以体现。Service 层还继承了封装的框架数据持久化操作接口，也就是说 Service 层完成了数据的持久化，但是并没有明显的代码去操作数据持久化操作，这里完美的体现了框架的独特性。Service 层也承担了系统的业务逻辑，我们复杂的业务逻辑都会在 Service 层得以实现，明确的层次感使得开发会更加得心应手。
4. QX_Frame.Web，无需多言，Web 层肯定是对于客户端的描述，框架实现中客户端不拘泥于

Web。App、Winform、Console 应用程序都是系统的常见架构，但是我们本次系统使用的是 Web 应用程序，因此这里用 Angular2 构建了 Web 层。

5. QX_Frame.Web.Srv，Web.Srv 准确描述应该是 WebApi.Srv。这层是用 OWIN 技术来解耦了 WebApi 与 IIS 之间的千丝万缕的联系，我们可以让我们的接口部署于更丰富的环境，不必拘泥于 Windows IIS。
6. QX_Frame.WebApi，WebApi 是数据访问接口，这里是和 Web 层直接交互的地方，也是由这一层作为 Service 层和外界的桥梁，Service 诡秘丰富的系统服务才得以实现它的价值。
7. QX_Frame.WebApi.Test，不需要过多介绍，这一层是对 WebApi 控制器接口的测试层。

6.1.2 框架层级类描述

在上一节我们对框架的整体架构进行了介绍，同时对每一层的作用进行了简要的说明，这一节我们对每一层框架中实现的类进行简单地介绍，我们用图表的形式展示出来，便于我们直观地进行查看。这里只显示了部分关键的类进行展示，项目的层级类描述如下表 6.1 所示。

表 6.1 系统层级类分析表

Table 6.1 The system hierarchy analysis table

系统层级	类名	说明
QX_Frame.Data	Db_qx_frame.cs	数据库操作上下文
	Tb_Authenticatin.cs	账户权限实体类
	Tb_UserAccount.cs	用户账户实体类
	Tb_UserAccountInfo.cs	用户信息实体类
	Tb_UserRole.cs	用户角色实体类
	Tb_UserStatus.cs	用户状态实体类
	…	…
QX_Frame.Data.Contract	IAuthenticationService.cs	账户权限服务接口类
	IUserAccountService.cs	用户账户服务接口类
	IUserAccountInfoService.cs	用户信息服务接口类
	IUserRoleService.cs	用户角色服务接口类
	IUserStatusService.cs	用户状态服务接口类
	…	…
QX_Frame.Data.Service	AuthenticationService.cs	账户权限服务类
	UserAccountService.cs	用户账户服务类
	UserAccountInfoService.cs	用户信息服务类
	UserRoleService.cs	用户角色服务类
	UserStatusService.cs	用户状态服务类
	…	…
QX_Frame.WebApi	UserAccountController.cs	用户账户控制器类
	UserController.cs	用户控制器类
	LoginController.cs	用户登录控制器类

	UserRoleController.cs	用户角色控制器类
	UserStatusController.cs	用户状态控制器类
	…	…

6.2 功能代码

本系统功能代码详见附录。

6.3 本章小结

在本章中，我们对系统构建的框架进行了简单地介绍，也对系统在框架的基础上的搭建步骤，搭建的方式进行了简单地介绍。在第二节中，我们对系统每一层实现的类进行了简单地说明，让我们能够清晰地了解到每一层具体实现的类的形式。最后我们对系统实现的关键代码进行了补充展示。

第七章 系统测试

7.1 测试的目的

本新闻网站旨在构建一个公众及时了解互帮信息的平台，在系统的界面上保持整洁，简单的风格。在系统的功能上采用简约易上手的原则，简化了客户对系统的熟悉的过程，缩短了用户适应时间。本网站的测试工作就是基于此要求来进行网站的测试，在对网站的界面功能进行改进的同时尽最大努力来发现软件的缺陷，以便给公众提供良好的产品，为社会大众负责。

7.2 测试的过程

在本项目主要用到白盒测试和黑盒测试两大部分，白盒测试又主要用于单元测试和集成测试中，单元测试在系统项目编码过程中进行测试完成，集成测试在项目正式发布过程中进行测试。黑盒测试用于在项目的系统测试以及项目验收阶段进行整体的测试[12]。本项目中，按照系统需求规格说明书和设计文档进行了相关的测试，以便测试项目是否达到了需求规格说明书中提到的项目需求以及项目的整体健壮性是否达到标准。

7.2.1 测试参考文档

《软件项目计划》
《软件测试》
《软件确认测试用例》

7.2.2 测试环境与配置

系统服务器：腾讯云服务器 182.254.210.156，服务器界面如下图 7.1 所示。

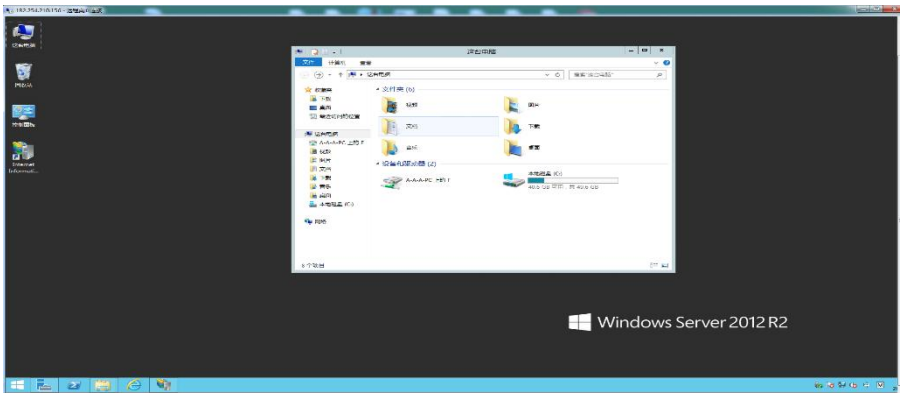

图 7.1 系统部署服务器桌面截图

Fig. 7.1 The system publish server screenshots

数据库：sqlserver2014 在服务器中安装完成

客户端：windows7 旗舰版 IE9.0

网络：校园网

7.2.3 测试方法

本系统的测试方法通过对系统的初步模拟使用来进行系统 Bug 的发现以及系统的各项优化。

7.2.4 测试实施

在系统的测试实施过程中，我们采用测试表的方式对测试用例进行描述，这样使得测试更加条理，思路更加清晰。下面我们对我们系统主要实现的功能进行逐一的测试。

1. 用户注册测试的过程如表 7.1 所示。

表 7.1 注册用例测试过程分析表

Table 7.1 Register case test process analysis table

项目/软件	AntHelp 蚂蚁热帮服务系统		**程序版本**	1.0.0
功能模块	用户注册		**编制人**	王东
用例编号	Test_1		**编制时间**	2017-05-12
功能特性	对新用户的注册，以及对已经注册用户的注册拒绝			
测试目的	验证是否输入合法的信息，确保数据完整性，以及确保数据写入数据库			
预置条件	用户输入注册信息提交注册请求		**特殊规程说明**	需要有效的邮箱地址
参考信息	需求分析中关于“用户注册”的说明			
测试数据	1	Qixiao	123456	Wd8622088@foxmail.com
	2	Qixiao	123456	Wd8622088@foxmail.com
	3	Bantina	123	bantina@foxmail.com

	4	Bantina	123456	bantina@foxmail.com
	5	Bantina	123456	Bantina2@foxmail.com
	6	Zhangyuqing	123456	zhangyuqing@qq.com

操作步骤	操作描述	数据	期望结果	实际结果	测试状态（P/F）
1	直接点击提交	空	显示警告信息“用户名不能空！”、“密码不能为空！”、“邮箱地址不正确！”	（符合）	P
2	测试数据 1，提交	测试数据 1	正确接收到邮件	（符合）	P
3	测试数据 2，提交	测试数据 2	显示警告信息“该用户已注册！”	（符合）	P
4	测试数据 3，提交	测试数据 3	显示警告信息“密码长度太短！”	（符合）	P
5	测试数据 4，提交	测试数据 4	正确接收邮件	（符合）	P
6	测试数据 5，提交	测试数据 5	未能收到邮件	（符合）	P
7	测试数据 6，提交	测试数据 6	未能收到邮件	（不符合）	P
8	点击操作步骤中 2 邮件中链接	无	提示注册成功，3 秒后自动登录	（符合）	P

2. 用户登录测试的过程如表 7.2 所示。

表 7.2 登录用例测试过程分析表

Table 7.2 Login case test process analysis table

项目/软件	AntHelp 蚂蚁热帮服务系统		程序版本	1.0.0
功能模块	用户登录		编制人	王东
用例编号	Test_2		编制时间	2017-05-12
功能特性	用户登录的功能验证			
测试目的	验证是否输入合法的信息，确保数据完整性，以及确保数据写入数据库			
预置条件	用户输入登录信息提交注册请求		特殊规程说明	规定时间内无需输入登录信息
参考信息	需求分析中关于“用户登录”的说明			
测试数据	1	Qixiao	123456	
	2	Qixiao2	123456	
	3	Bantina	123	
	4	Bantina	123456789	
	5	Bantina	123456	
	6	Zhangyuqing	123456	

操作步骤	操作描述	数据	期望结果	实际结果	测试状态（P/F）
1	直接点击提交	空	显示警告信息“用户名不能空！”、“密码不能为空！”	（符合）	P
2	测试数据 1，提交	测试数据 1	登录成功并跳转	（符合）	P
3	测试数据 2，提交	测试数据 2	显示警告信息“该用户不存在！”	（符合）	P
4	测试数据 3，提交	测试数据 3	显示警告信息“密码长度太短！”	（符合）	P
5	测试数据 4，提交	测试数据 4	显示警告信息“账号或密码错误！”	（符合）	P
6	测试数据 5，提交	测试数据 5	登录成功并跳转	（符合）	P
7	测试数据 6，提交	测试数据 6	登录成功并跳转	（符合）	P
8	直接打开页面	空	显示登录状态	（符合）	P
9	直接打开页面	空	未登陆，跳转到登录页	（符合）	P

3. 用户信息查询（Token 认证）测试的过程如表 7.3 所示。

表 7.3 用户信息查询分析表

Table 7.3 User information query case test process analysis table

项目/软件	AntHelp 蚂蚁热帮服务系统		程序版本	1.0.0
功能模块	用户信息查询（Token 认证）		编制人	王东
用例编号	Test_3		编制时间	2017-05-12
功能特性	用户点击用户信息链接			
测试目的	验证是否能从数据库成功查询到信息			
预置条件	用户点击用户信息链接		特殊规程说明	无
参考信息	需求分析中关于“用户查询”的说明			
测试数据	1	Qixiao 点自己	普通用户	
	2	Qixiao 点他人	普通用户	
	3	Bantina 点自己	管理员	
	4	Bantina 点他人	管理员	
	5	Zhangyuqing 点自己	用户被停用	
	6	Zhangyuqing 点自己	用户被冻结	

操作步骤	操作描述	数据	期望结果	实际结果	测试状态（P/F）
1	测试数据 1，提交	测试数据 1	展示个人信息	（符合）	P

2	测试数据 2，提交	测试数据 2	提示没有权限	（符合）	P
3	测试数据 3，提交	测试数据 3	展示个人信息	（符合）	P
4	测试数据 4，提交	测试数据 4	展示用户信息	（符合）	P
5	测试数据 5，提交	测试数据 5	显示警告信息“账号被停用！”	（符合）	P
6	测试数据 6，提交	测试数据 6	显示警告信息“账号被冻结！”	（符合）	P
7	测试数据 1，提交	测试数据 1	显示警告信息“账号登录信息过期，请重新登录！”跳转到登录页面	（符合）	P

7.2.5 系统缺陷分析

通过新闻网站的黑盒测试，发现了诸多系统问题，在下一部分缺陷统计及缺陷分析部分会详细对测试用例的测试情况作为总结。通过系统黑白盒的测试，系统基本完成了需求规格说明书中的需求说明，达到了预期的目标，系统存在一些细节上的问题，不影响系统的整体使用。

在系统的黑盒测试中，通过设计响应的测试用例，发现了在用户注册模块中，除了对用户输入的信息限制不够严谨外（前后台同时验证），还有用户注册邮件发送的不稳定性。我们的邮箱使用的网易 163 邮箱作为系统邮箱进行发送，在邮箱发送的过程中，我们发现了在 Foxmail 的邮箱服务器接收邮件是完全没有问题的，但是在 QQ 邮箱的使用上，我们仍然是一块短板，我们的邮箱服务器性能有待提升。

7.3 测试结论

这是测试报告的核心，主要汇总测试各种数据并进行度量，度量包括对测试过程的度量和能力评估、对软件产品的质量度量和产品评估。其中需求覆盖分析以及测试覆盖分析如表 7.4 以及表 7.5 所示。

表 7.4 需求覆盖分析

Table 7.4 Test result analysis table

功能	测试点描述	是否测试	重要等级	是否通过	备注
用户注册	注册邮件的接收，注册信息的入库	是	A 级	通过	无
用户登录	规定时间内无需重复登录，登录成功与否	是	A 级	通过	无

用户信息查询	用户角色状态的判断，用户权限的限制，用户登录信息过期的处理	是	S 级	通过	该部分测试进行多次测试

说明：这里的重要等级分为 S 级、A 级、B 级其中 S 级是最重要的。由表格可知，需求总数为 3 部分，测试通过的需求总数为 3 部分，由需求覆盖率=测试通过需求点/需求总数*100%可得需求覆盖率为 100%。

表 7.5 测试覆盖分析

Table 7.5 Test coverage analysis

功能	测试用例数	执行数	未执行数	通过数	失败数
用户注册	8	8	0	7	1
用户登录	9	9	0	9	0
用户信息查询	7	7	0	7	0

说明：由表格可知，用户注册模块测试覆盖率为 100%，用户登录模块测试覆盖率为 100%，用户信息查询模块测试覆盖率为 100%，系统总的测试覆盖率为 100%，系统的测试通过率为 95.8%。

由测试用例以及最后的分析可以得出，本次测试执行不够充分，但是测试的预期目标都已达成，所有经过测试的功能模块全部通过，没有发现重大的功能缺陷，系统较为安全可靠。但系统可能存在边界值输入产生异常的隐患，需要进行进一步测试。

7.4 本章小结

在本章中，我们对系统进行了初步的单元测试和集成测试，在测试中我们也发现了诸多的问题，有的及时改正，有的有待攻克技术难点后改正，总之我们的测试结果也是对我们的系统的一次功能上或者稳定性的一次考核，每一次考核对自己来说都是一次进步。

在测试中发现的一系列问题，对于今后的工作学习生活都是一份难得的经验，对我们自身来说也是一种提升。当然测试只是尽可能地发现系统的漏洞，不可能百分百覆盖到系统漏洞，因此，在测试阶段我们努力吸收前人的教训的同时更应该注重创新，我们应该打破固有的思维模式，才能在测试过程中发现更多的有价值的漏洞。

第八章 总结

8.1 结论

在这次毕业设计中，不仅让我学习到了丰富的专业知识，更多的是让我学习到了去探索知识的能力和团队成员之间合作的能力，这些知识都让我在今后的人生道路上受益匪浅。

在设计的进行中，我对曾经熟悉又陌生的 UML 系统设计模式知识进行了巩固，能快速很好地利用所学知识进行对系统的详细分析与图例的设计绘制。乐在其中的同时让我充分体会到了丰富扎实的专业知识对于系统的分析设计及实现是多么地重要，在有充足的理论知识的指导下进行的工作往往会令我们达到事半功倍的效果。

在系统的实现过程中，我对不久前独自开发的 QX_Frame 后台通用开发框架进行了第一次真正意义上的项目实践，在实践的过程中，我找到了很多表面的以及潜在的问题，让我对自己的开发框架一次又一次的进行了 Bug 的修复，使得框架更加地成熟与完善。除了 Bug 上的修改，我也对框架进行了升级，让代码的编写上更加地简洁明了，语法更加地清晰，结构更加地简单，对今后的框架基础上的开发都是不菲的帮助。

在编码的学习和使用上，我个人更加倾向于对新语法，新特性的使用。在我看来，一个 IT 工作人员最重要的思维就是去接纳新的事物，使用新的产品，我们才能始终走在时代前沿，不被日新月异的时代潮流所抛弃。因此我的架构采用的都是当今最流行的技术产品，包括.NetFramework4.6.1、Angular2 框架的采用，都是对新事物的接纳。并且让自己体会到了当前编码世界的新思想，新能力。

总的来说，我在这次系统设计实现上的最大的收获就是在系统设计能力上的提高，对数据库设计的能力提高，还有在框架上的不断完善。当然，万千技术上的能力提高都不能比得上我们团队合作上的收获，我在团队合作能力上的提高都是我日后进步的助力，对我的今后的职业生涯产生了深远的影响。

8.2 展望

紧张而又有序的毕业设计终于接近了尾声，在系统的设计和实现的完成基础上，总会有些许的不尽人意的地方值得我们深思，不管是改过的未改过的 Bug 还是在设计上的差强人意，总留给我们以后进步的空间，让我们在今后的道路上脚踏实地地成长。

我们的系统在设计思想上，我认为社区模块和我们的系统主要业务功能联系还不是很紧密，至少在设计功能和业务功能完成的基础上来宏观地看，这两块的功能应该更加紧密地结

合在一起，这样才能更加地接近我们的设计初衷，也更加让我们的社区 020 得到良好的实践。

在数据库的设计方面，我认为还有很多差强人意的地方，我们对于数据库的改动太多频繁都是对我们数据库设计不能很友好的具体直观体现，尽管我们对数据库的设计进行了很多次的改动，然而对我们的数据库而言，我仍然不是十分的乐观和满意，我认为在数据库的设计上，范式的使用还是比较少的。尽管当初为了程序的扩展性我们减少了很多的范式的使用，但是在后续的开发中，我发现有很多的范式实现还是必不可少的，在今后的设计中，我会格外地看中这一点，尽可能让最初的数据库设计更加完善，更加合理。

除了设计方面，代码上的希冀便是对代码结构上的统筹规划，我认为有一个良好的代码结构规范对于后期的维护是极有好处的，在这方面，我们做的是很好，但是还有一点不好的地方就是一开始对于代码的分类并没有做的很好，导致后面的代码文件存储路径比较混乱，对于文件的引用和使用都是极其不方便的。

我希望在今后的系统设计和实现中，以上提出的问题都能得到很高的关注点，并且付诸到项目实施中去，才能在今后的道路上走的更加长远。

参考文献

[1] 明日科技.C#从入门到精通.清华大学出版社,2012.9.
[2] Christian Nagel.C#高级编程（第 9 版）.清华大学出版社
[3] Andrew Troelsen.C#与.NET 4 高级程序设计.人民邮电出版社,2011.4.
[4] 张越廷,顾彦玲.ASP.NET 从入门到精通.清华大学出版社,2008.9.
[5] 马伟.ASP.NET 4 权威指南.机械工业出版社,2011.1.
[6] 张昌龙,辛永平.ASP.NET 4.0 从入门到精通.机械工业出版社,2011.1.
[7] 李彦,高博,唐继强,许惠彬.ASP.NET 4.0 MVC 敏捷开发给力起飞.电子工业出版社,2011.9.
[8] 侯志荣. WEB 应用程序开发-算法分析与应用 [M].北京：人民邮电出版社,2000.1
[9] Robert Vieria.SQL Server2008 编程入门经典.清华大学出版社
[10] 徐孝凯,贺佳英.数据库基础与 SQL Server 应用开发. 清华大学出版社,2008.4.
[11] 萨师煊,王小珊.数据库系统概论（第三版）[M],北京：高等教育出版社,2008.6.
[12] 詹姆斯 •R •埃文斯 •威廉 •M •林赛.质量管理与质量控制.中国人民大学出版社,2010.5.

附　录

附录 1：注册接口代码

```
// POST: api/User/loginId  mail click register api
public IHttpActionResult Post(string loginId)
{
    if (string.IsNullOrEmpty(loginId))
    {
        throw new Exception_DG("loginId", "loginId must be provide.", 1002);
    }

    bool addSuccess = true;

    tb_UserAccount userAccount = tb_UserAccount.Build();
    userAccount.uid = Guid.NewGuid();
    userAccount.loginId = loginId;

    object pwd_mail_cache = Cache_Helper_DG.Cache_Get(loginId);//get pwd and mail from cache

    if (pwd_mail_cache == null)
    {
        throw new Exception_DG("register info expired,please register renew.", 3003);
    }

    using (var fact = Wcf<UserAccountService>())
    {
        var channel = fact.CreateChannel();
        int userAccountCountByloginId = channel.QueryCount(new tb_UserAccountQueryObject
{ QueryCondition = t => t.loginId.Equals(loginId) });
        if (userAccountCountByloginId > 0)
        {
            throw new Exception_DG("the loginId has been exist!", 3002);
        }
```

```
    }
    userAccount.pwd = pwd_mail_cache.ToString().Split(',')[0];
    Transaction_Helper_DG.Transaction((Action)(() =>
    {
        //add tb_UserAccount
        using (var fact = Wcf<UserAccountService>())
        {
            var channel = fact.CreateChannel();
            addSuccess = addSuccess && channel.Add(userAccount);
        }
        //add tb_UserAccountInfo
        using (var fact = Wcf<UserAccountInfoService>())
        {
            var channel = fact.CreateChannel();
            addSuccess = addSuccess && channel.Add(new tb_UserAccountInfo { uid = userAccount.uid,
loginId = userAccount.loginId, nickName = loginId, email = pwd_mail_cache.ToString().Split(',')[1],
registerTime = DateTime.Now, birthday = DateTime.Now });
        }
        //add tb_UserStatus
        using (var fact = Wcf<UserRoleService>())
        {
            var channel = fact.CreateChannel();
            addSuccess = addSuccess && channel.Add(new tb_UserRole { uid = userAccount.uid, roleLevel
= 0 });
        }
        //add tb_UserStatus
        using (var fact = Wcf<UserStatusService>())
        {
            var channel = fact.CreateChannel();
            addSuccess = addSuccess && channel.Add(new tb_UserStatus { uid = userAccount.uid,
statusLevel = 0 });
        }
    }));
    if (!addSuccess)
    {
        throw new Exception_DG("register error.", 3004);
```

```
    }
    long timeStamp = DateTime_Helper_DG.GetCurrentTimeStamp() * 1000;
    int random = new Random().Next(100, 999);
    //get rsa keys
    RSA_Keys rsa_Keys = RSA_GetKeys();
    tb_Authentication authentication;
    using (var fact = Wcf<AuthenticationService>())
    {
        var channel = fact.CreateChannel();
        authentication = channel.QuerySingle(new tb_AuthenticationQueryObject { QueryCondition = t
=> t.loginId.Equals(loginId) }).Cast<tb_Authentication>();
        if (authentication == null)
        {
            authentication = tb_Authentication.Build();
            //secretKey=MD5[loginid+MD5[pwd]+timestamp]
            authentication.secretkey = MD5_Encrypt($"{loginId}{userAccount.pwd}{timeStamp}");
            authentication.rsa_publicKey = rsa_Keys.publicKey;
            authentication.rsa_privateKey = rsa_Keys.privateKey;
            authentication.loginId = loginId;
            //tokenSign=MD5[loginid+logintimestamp+random]
            authentication.tokensign = MD5_Encrypt($"{loginId}{timeStamp}{random}");
            if (channel.Add(authentication))
            {
                authentication = channel.QuerySingle(new tb_AuthenticationQueryObject
{ QueryCondition = t => t.loginId.Equals(loginId) }).Cast<tb_Authentication>();
            }
        }
        else
        {
            //secretKey=MD5[loginid+MD5[pwd]+timestamp]
            authentication.secretkey = MD5_Encrypt($"{loginId}{userAccount.pwd}{timeStamp}");
            authentication.rsa_publicKey = rsa_Keys.publicKey;
            authentication.rsa_privateKey = rsa_Keys.privateKey;
            authentication.loginId = loginId;
            //tokenSign=MD5[loginid+logintimestamp+random]
            authentication.tokensign = MD5_Encrypt($"{loginId}{timeStamp}{random}");
```

```
            channel.Update(authentication);
        }
    }
    Int Auth_Token_ExpireTime_days =
Config_Helper_DG.AppSetting_Get("Auth_Token_ExpireTime_days").ToInt();
    int Auth_Token_ExpireTime_hours =
Config_Helper_DG.AppSetting_Get("Auth_Token_ExpireTime_hours").ToInt();
    int Auth_Token_ExpireTime_minutes =
Config_Helper_DG.AppSetting_Get("Auth_Token_ExpireTime_minutes").ToInt();
    long expireTimeStamp =
DateTime_Helper_DG.GetTimeStampByDateTimeUtc(DateTime.UtcNow.AddDays(Auth_Token_ExpireTime_days)
.AddHours(Auth_Token_ExpireTime_hours).AddMinutes(Auth_Token_ExpireTime_minutes));
    //token=RSA_publicKey[uid+loginid+expiretimestamp+tokensign]
    string token =
RSA_Encrypt($"{userAccount.uid}&{userAccount.loginId}&{expireTimeStamp}&{authentication.tokensig
n}", authentication.rsa_publicKey);
    //send appkey,secretkey,token,userInfo to client
    return Json(Return_Helper_DG.Success_Msg_Data_DCount_HttpCode("register user succeed , login
succeed", new { loginId = userAccount.loginId, appKey = authentication.appkey, secretKey =
authentication.secretkey, token = token }, 1));
}
```

附录 2：登录接口控制器类代码

```
using QX_Frame.App.WebApi;
using QX_Frame.Data.Entities.QX_Frame;
using QX_Frame.Data.QueryObject;
using QX_Frame.Data.Service.QX_Frame;
using QX_Frame.Helper_DG;
using QX_Frame.Helper_DG.Extends;
using QX_Frame.WebAPI.Filters;
using System;
using System.Web.Http;
using static QX_Frame.Helper_DG.Encrypt_Helper_DG;
```

```
namespace QX_Frame.WebAPI.Controllers
{
    /// <summary>
    /// copyright qixiao code builder ->
    /// version:4.2.0
    /// author:qixiao(柒小)
    /// time:2017-04-10 11:08:30
    /// </summary>

    /// <summary>
    ///class LoginController
    /// </summary>
    public class LoginController : WebApiControllerBase
    {
        // GET: api/Login
        public IHttpActionResult Get(int appKey,int random,long timeStamp,string token)
        {
            //timeStamp verification
            bool isTimeStampValid = (DateTime_Helper_DG.GetCurrentTimeStamp() - timeStamp / 1000)
<= Config_Helper_DG.AppSetting_Get("RequestExpireTime").ToInt() * 60;
            if (!isTimeStampValid)
            {
                throw new Exception_DG("request expired", 3006);
            }
            //[random+timestamp] can be find in cache?
            if (Cache_Helper_DG.Cache_Get($"{random}{timeStamp}") != null)
            {
                throw new Exception_DG("request multiple", 3007);
            }
            Cache_Helper_DG.Cache_Add($"{random}{timeStamp}",  1);//add  [random+timestamp]  into
cache

            tb_Authentication                              authentication                              =
AuthenticationController.GetAuthenticationByAppKey(appKey);

            //get token array from decrypt token string
```

```
            string[]          tokenArray          =          Encrypt_Helper_DG.RSA_Decrypt(token,
authentication.rsa_privateKey).Split('&');

//$"{userAccount.uid}&{userAccount.loginId}&{expireTimeStamp}&{authentication.tokensign}"
            long expireTimeStamp = tokenArray[2].ToInt64();
            string tokenSign = tokenArray[3];

            if (expireTimeStamp < DateTime_Helper_DG.GetCurrentTimeStamp())
            {
                throw new Exception_DG("login info expired please login renew", 3011);
            }

            if (!tokenSign.Equals(authentication.tokensign))
            {
                throw new Exception_DG("account login elsewhere,please login renew", 3012);
            }

            return   Json(Return_Helper_DG.Success_Msg_Data_DCount_HttpCode("account   has   been
login",new { loginId=authentication.loginId},1));
        }

        // POST: api/Login
        public IHttpActionResult Post([FromBody]dynamic query)
        {
            if (query == null)
            {
                throw new Exception_DG("arguments must be provide", 1001);
            }
            if (query.loginId == null)
            {
                throw new Exception_DG("loginId must be provide", 1002);
            }
            if (query.random == null)
            {
                throw new Exception_DG("random must be provide", 1006);
            }
```

```
            if (query.timeStamp == null)
            {
                throw new Exception_DG("timeStamp must be provide", 1007);
            }
            if (query.secretString == null)
            {
                throw new Exception_DG("secretString must be provide", 1009);
            }

            string loginId = query.loginId;
            int random = query.random;
            long timeStamp = query.timeStamp;
            string secretString = query.secretString;

            //timeStamp verification
            bool isTimeStampValid =( DateTime_Helper_DG.GetCurrentTimeStamp() - timeStamp / 1000)<=
Config_Helper_DG.AppSetting_Get("RequestExpireTime").ToInt() * 60;
            if (!isTimeStampValid)
            {
                throw new Exception_DG("request expired", 3006);
            }
            //[random+timestamp] can be find in cache?
            if (Cache_Helper_DG.Cache_Get($"{random}{timeStamp}") != null)
            {
                throw new Exception_DG("request multiple", 3007);
            }
            Cache_Helper_DG.Cache_Add($"{random}{timeStamp}",  1);//add  [random+timestamp]  into
cache

            //get MD5[pwd] from database
            tb_UserAccount userAccount;
            using (var fact = Wcf<UserAccountService>())
            {
                var channel = fact.CreateChannel();
                userAccount = channel.QuerySingle(new tb_UserAccountQueryObject { QueryCondition =
t => t.loginId.Equals(loginId.Trim()) }).Cast<tb_UserAccount>();
```

```
            }

            if (userAccount == null)
            {
                throw new Exception_DG("no user account found by loginId", 3001);
            }

            //MD5[loginid+MD5[pwd]+ramdom+timestamp]==MD5[secretMessage]?
            bool secretStringMatched =
MD5_Encrypt($"{loginId}{userAccount.pwd}{random}{timeStamp}").Equals(secretString);
            if (!secretStringMatched)
            {
                throw new Exception_DG("the request has been tampered also mains account or pwd error",
3008);
            }

            //get rsa keys
            RSA_Keys rsa_Keys = RSA_GetKeys();

            tb_Authentication authentication;
            using (var fact = Wcf<AuthenticationService>())
            {
                var channel = fact.CreateChannel();

                authentication = channel.QuerySingle(new tb_AuthenticationQueryObject
{ QueryCondition = t => t.loginId.Equals(loginId) }).Cast<tb_Authentication>();

                if (authentication==null)
                {
                    authentication =tb_Authentication.Build();
                    //secretKey=MD5[loginid+MD5[pwd]+timestamp]
                    authentication.secretkey =
MD5_Encrypt($"{loginId}{userAccount.pwd}{timeStamp}");
                    authentication.rsa_publicKey = rsa_Keys.publicKey;
                    authentication.rsa_privateKey = rsa_Keys.privateKey;
                    authentication.loginId = loginId;
```

```
                    //tokenSign=MD5[loginid+logintimestamp+random]
                    authentication.tokensign = MD5_Encrypt($"{loginId}{timeStamp}{random}");
                    if (channel.Add(authentication))
                    {
                        authentication = channel.QuerySingle(new tb_AuthenticationQueryObject
{ QueryCondition = t => t.loginId.Equals(loginId) }).Cast<tb_Authentication>();
                    }
                }
                else
                {
                    //secretKey=MD5[loginid+MD5[pwd]+timestamp]
                    authentication.secretkey =
MD5_Encrypt($"{loginId}{userAccount.pwd}{timeStamp}");
                    authentication.rsa_publicKey = rsa_Keys.publicKey;
                    authentication.rsa_privateKey = rsa_Keys.privateKey;
                    authentication.loginId = loginId;
                    //tokenSign=MD5[loginid+logintimestamp+random]
                    authentication.tokensign = MD5_Encrypt($"{loginId}{timeStamp}{random}");
                    channel.Update(authentication);
                }
            }

            int Auth_Token_ExpireTime_days =
Config_Helper_DG.AppSetting_Get("Auth_Token_ExpireTime_days").ToInt();
            int Auth_Token_ExpireTime_hours =
Config_Helper_DG.AppSetting_Get("Auth_Token_ExpireTime_hours").ToInt();
            int Auth_Token_ExpireTime_minutes =
Config_Helper_DG.AppSetting_Get("Auth_Token_ExpireTime_minutes").ToInt();
            long expireTimeStamp =
DateTime_Helper_DG.GetTimeStampByDateTimeUtc(DateTime.UtcNow.AddDays(Auth_Token_ExpireTime_days)
.AddHours(Auth_Token_ExpireTime_hours).AddMinutes(Auth_Token_ExpireTime_minutes));
            //token=RSA_publicKey[uid+loginid+expiretimestamp+tokensign]
            string token =
RSA_Encrypt($"{userAccount.uid}&{userAccount.loginId}&{expireTimeStamp}&{authentication.tokensig
n}", authentication.rsa_publicKey);
            //send appkey,secretkey,token,userInfo to client
```

```
            return  Json(Return_Helper_DG.Success_Msg_Data_DCount_HttpCode("login  succeed",  new
{ loginId = loginId, appKey = authentication.appkey, secretKey = authentication.secretkey, token =
token }, 1));
        }

        // PUT: api/Login
        public IHttpActionResult Put([FromBody]dynamic query)
        {
            throw new Exception_DG("The interface is not available", 9999);
        }

        // DELETE: api/Login
        public IHttpActionResult Delete([FromBody]dynamic query)
        {
            throw new Exception_DG("The interface is not available", 9999);
        }
    }
}
```

附录 3：Token 认证过滤器类代码

```
using QX_Frame.Data.Entities.QX_Frame;
using QX_Frame.Data.Options;
using QX_Frame.Helper_DG;
using QX_Frame.Helper_DG.Extends;
using QX_Frame.WebAPI.Controllers;
using System;
using System.Net.Http;
using System.Web;
using System.Web.Http.Controllers;
using System.Web.Http.Filters;

namespace QX_Frame.WebAPI.Filters
{
    /**
```

```
     * author:qixiao
     * create:2017-4-13 10:46:17
     * */
    [AttributeUsage(AttributeTargets.All, AllowMultiple = true)]//多次调用
    public class LimitsAttribute_DG : ActionFilterAttribute
    {
        public int RoleLevel { get; set; } = 0;
        public override void OnActionExecuting(HttpActionContext actionContext)
        {
            int appKey = 0;
            string token = string.Empty;
            Microsoft.Owin.OwinContext                          httpContext                          =
actionContext.Request.Properties["MS_OwinContext"] as Microsoft.Owin.OwinContext;
            if (actionContext.Request.Method == HttpMethod.Get)
            {
                if (!string.IsNullOrEmpty(httpContext.Request.Query["token"]))
                {
                    appKey = httpContext.Request.Query["appKey"].ToInt();
                    token = httpContext.Request.Query["token"];
                }
                else
                {
                    throw new Exception_DG("appKey and token must be provide", 1013);
                }
            }
            else
            {
                dynamic query = actionContext.ActionArguments["query"];
                token = query.token;
                appKey = query.appKey;
            }

            if (appKey == 0)
            {
                throw new Exception_DG("appKey", "appKey must be provide", 1010);
            }
```

```
if (string.IsNullOrEmpty(token))
{
    throw new Exception_DG("token", "token must be provide", 1011);
}

var tokenInfo = AuthenticationController.GetTokenInfoByAppKeyToken(appKey, token);
Guid uid = tokenInfo.Item1;
string loginId = tokenInfo.Item2;
long expireTimeStamp = tokenInfo.Item3;
string tokenSign = tokenInfo.Item4;

if (expireTimeStamp < DateTime_Helper_DG.GetCurrentTimeStamp())
{
    throw new Exception_DG("login info expired please login renew", 3011);
}
//token->tokenSign match db->tokenSign
if (!tokenSign.Equals(tokenInfo.Item5.tokensign))
{
    throw new Exception_DG("account login elsewhere,please login renew", 3012);
}

//Limit validate
tb_UserRole userRole = UserRoleController.GetUserRoleByUid(uid);
if (userRole.roleLevel < this.RoleLevel)
{
    throw new Exception_DG("do not have enough of permission", 3020);
}

tb_UserStatus userStatus = UserStatusController.GetUserStatusByUid(uid);
if (userStatus.statusLevel == opt_AccountStatusLevel.NORMAL.ToInt())
{
    // continue
}
else if (userStatus.statusLevel == opt_AccountStatusLevel.ABANDON.ToInt())
{
```

```
                throw new Exception_DG("account abandoned", 3017);
            }
            else if (userStatus.statusLevel == opt_AccountStatusLevel.FREEZE.ToInt())
            {
                throw new Exception_DG("account has been frozen", 3018);
            }
            else if (userStatus.statusLevel == opt_AccountStatusLevel.STOP.ToInt())
            {
                throw new Exception_DG("account disabled", 3019);
            }
        }

        public override void OnActionExecuted(HttpActionExecutedContext actionExecutedContext)
        {
            base.OnActionExecuted(actionExecutedContext);
        }
    }
}
```

致　谢

在蚂蚁热帮（AntHelp）系统的研发中，我首先要感谢我的老师对我的帮助支持和鼓励，是我的老师从我的选题到整个毕业设计过程中孜孜不倦的教诲让我有充足的信心去完成我的毕业设计。并且在我遇到困难和问题时候能及时对我施以帮助，在我有做的不够完好的地方能及时纠正，保证了我毕业设计项目的完美完成。其次我要感谢我的团队成员，也是我很好的朋友们，是他们始终与我同仇敌忾，以一个共同的目标坚持不懈，同时正因为有了他们，才能在毕业设计中融入很多优秀的想法，也正因如此，我们的毕业设计才能优秀地完成。在团队开发中或许也会出现一些摩擦，但这都不是问题，每一次的小问题都促使了我们的成长，同时也是人生路上一段宝贵的经验，让我在以后遇到相同的问题时候波澜不惊，能冷静地去思考、处理。

除了项目中的参与人员外，我还要感谢我的父母和我亲爱的同学们，是他们一直在背后默默地鼓励我，支持我，才有了今天毕业设计很好的完成，是你们减轻了我背后的重担，我由衷地感谢你们。

在毕业设计完成之际，我还感谢我自己，能以一百二十分的精力去调研、分析、设计、实现整个系统，并且在整个过程中学到了很多在学校的课堂上学习不到的知识，在系统架构上的进步便是我对自己开发的框架日益完善的过程，经历了这段毕业设计，我所开发的 QX_Frame 也日趋成熟，越来越能应对各种各样的项目需求，我很感激这次毕业设计，也很感激参与这次毕业设计的我。

虽然我们的项目接近尾声，但我始终相信，山外青山楼外楼，强中更有强中手。我们项目完成的很好，并不代表我们掌握了必备的知识，在漫漫人生路上，我们还有更多的知识需要我们去学习，还有更多的辉煌等着我们去创造！生命不息，奋斗不止！