南京大学本科生毕业论文（设计）中文摘要

毕业论文题目：个人事务管理系统的设计与实现

软件学院 院系 软件工程 专业 2006 级本科生姓名：翟羿

指导教师：贝佳

摘要：

随着社会的发展和信息化程度的提高，人们的工作节奏不断加快，工作任务不断加重，所处理的事项越来越多，面越来越广，量越来越大，在这样的工作状态下，大脑会经常处于紧张状态，有时会感觉大脑一片空白或杂乱无章，所以单凭个人能力要高效优质的完成任务是很困难的。如何使每个人在有效工作时间内发挥最大的工作效能成为人们面临的一个重大课题。GTD（Getting Things Done）行为管理方法的提出有效的解决了这一问题。GTD 行为管理方法通过将人们头脑中的各种事项以记录的方式从人们的头脑中迁移出来，从而使人们可以集中精力和资源于正在执行的事项。为了让 GTD 行为管理方法的执行更加方便有效，本文介绍了一款自主设计和开发的 GTD 个人事务管理系统——SharpIt。

SharpIt 个人事务管理系统是一个基于 GTD 行为管理方法实现的个人事务管理系统。它可以帮助用户高效地规划各项任务以集中时间和资源完成重要事项。同时调整用户的工作节奏，使用户轻松处理各项庞大繁杂的工作，提高用户的工作效率。SharpIt 个人事务管理系统为 C/S 架构，采用 MVC 模式进行设计，基于 Java SE 和 Java EE 技术实现，使用 XML 技术进行数据的存储和管理，通过 HTTP 协议进行网络传输，采用 TSP 小组软件开发过程进行项目的开发和管理。

本文首先探讨了 GTD 行为管理方法，介绍了 MVC，Java SE，Java EE，XML，HTTP 等技术和开发环境。然后介绍了一个基于 GTD 行为管理方法实现的个人事务管理系统——SharpIt。它具有用户管理，事务管理，联系人管理等功能，可以根据 GTD 行为管理方法帮助用户管理待办事项，并且支持多用户协同工作，提供多语言支持。SharpIt 个人事务管理系统的设计与实现以及在实现系统时衍生的轻量级 MVC 框架 Sharp MVC 和轻量级显示框架 Sharp UI 的设计与实现是本文讨论的重点。最后对 SharpIt 个人事务管理系统的发展做了个人展望，并对 SharpIt 个人事务管理系统的可改进之处提出了建议。

关键词：GTD, MVC, Java SE, Java EE,XML

南京大学本科生毕业论文（设计）英文摘要

THESIS：The design and implementation of the affairs management module of the personal affairs management system
DEPARTMENT: Software Institute
SPECIALIZATION: Software Engineering
UNDERGRADUATE: Zhai Yi
MENTOR: Bei Jia

ABSTRACT:

SharpIt is a personal affairs management system which is based on the GTD (Getting Things Done) behavior management theory. It can help users to plan the tasks efficiently to let them focus the time and resources on the important issues. SharpIt is implemented with C/S structure and MVC design pattern. Java SE and Java EE are invoked to the system. XML is used for data storage and management. HTTP protocol is used for the network transportation. The whole project is developed and managed using TSP process.

In this paper, first of all, the GTD behavior management theory, MVC pattern, Java SE, Java EE, HTTP technologies and the development environment of the system are introduced. Then the design and implementation of the SharpIt System was described in details. Finally my personal view to the development of the SharpIt System and the how to improve the SharpIt system are illustrated at the end of the paper.

KEY WORDS: GTD, MVC, Java SE, Java EE, XML

目　录

第一章 概述

1.1 项目背景

随着社会的发展和信息化程度的提高，人们的工作节奏不断加快，工作任务愈加繁杂，如何高效有序地完成工作成为人们研究和探讨的一项重要课题。

2001 年 Allen David 出版了《Getting Things Done: The Art of Stress-Free Productivity》，提出了 GTD（Getting Things Done）行为管理方法，旨在“指导工作者在有需要或期望之时，如何才能尽快去做而达到高效和轻松的最佳境界”[1]。GTD 的主要原则在于通过记录的方式将工作者头脑中的各种任务迁移出来，从而让工作者可以集中精力于正在完成的事情。GTD 行为管理方法和其他行为管理方法不同之处在于 GTD 并不把重点放在设置任务的优先级上，而是着重于制定出在各种环境下的任务列表。例如，制定一个需要打电话的列表，或者在市区才能完成的事情的列表。GTD 行为管理方法的实施包括收集、处理、组织、检查和执行五个核心活动[1]。

传统的记录方式（如纸笔，记事本软件等）虽然可以实现将工作者头脑中的各种任务迁移出来的功能，但记录方式复杂，记录内容繁琐混乱，记录格式无法统一，记录文件不易于查找和管理，同时也缺少必要的提醒功能，从而降低了 GTD 的实际工作效率。而一些传统事务管理工具（如 Outlook，Google Calendar 等）并没有原生支持 GTD 的收集、处理、组织、检查和执行的核心活动，而是需要通过一些变通的方式间接的实现 GTD 行为管理方法，无法让用户真正体验到 GTD 行为管理方法的高效和优越性。

本系统正是针对传统记录方式和传统事务管理工具的缺点，遵照 GTD 行为管理方法的主要原则和核心活动，秉承跨平台和易扩展的设计思路进行设计和实现的。通过使用 SharpIt 个人事务管理系统，用户可以免受头脑中各种无关想法的干扰，有条不紊地组织和规划各项任务，集中时间和资源完成重要事项，将工作节奏牢牢把握在手中，轻松高效地应对各项庞大繁杂的工作。让用户在享受无压工作乐趣的同时畅想高效工作的成就感。

1.2 国内外 GTD 技术现状

1.2.1 国内外 GTD 工具简介

国内外许多团队和个人都针对 GTD 行为管理方法理论开发了相应的事务管理工具，比较著名的有 Doit.im,，Remember the milk，Monkey GTD， Stella GTD，Thinking Rock 等。

Doit.im:一款随时在线的 GTD 工具。它支持 Windows, Mac, Linux, iPhone, Android, Windows Mobile，Symbian 等平台。

图 1.1 Doit.im 系统

Remember the milk:一款优秀的在线 GTD 系统。它重造了任务列表，可扩展性强，支持 Google Calendar，Twitter，iPhone/iPod Touch，Gmail 和 BlackBerry 等平台。

图 1.2 Remember the milk 系统

Stella GTD:一款 iPhone 平台上的 GTD 任务管理工具。

图 1.3 Stella GTD 系统

Thinking Rock: 一款免费的 GTD 工具，支持 Windows, Mac, Linux 等平台。

其他 GTD 工具还包括 ActiontasticGTD、 Inbox for Gmail、Chandler、Tooledo 等。

1.2.2 国内外 GTD 工具特点

分析上述国内外 GTD 事务管理工具可发现除了采用 GTD 管理方法以外还有如下特点：

网络支持：随着网络技术的不断发展，互联网（Internet）已经成为连接世界的纽带。为了让用户能够随时随地使用 GTD 工具管理日常工作和事务，GTD 工具均提供良好的网络支持，这样用户在任何可以连接互联网的地方均可以使用工具对日常事务进行管理。

提醒功能：GTD 行为管理方法只是规定了用户在管理个人事务时的原则、方法和基本活动。但用户在根据 GTD 制定完成任务计划后由于记忆、记录方式等各种原因很难严格按照制定的计划执行事项。通过自动提醒功能，用户在制定计划时就可以设定提醒时间，大大方便 GTD 的使用性，提高了用户的工作效率。

与成熟系统的集成：为了方便用户的使用，很多 GTD 工具都与传统的邮件系、事务管理系统和即时聊天系统等进行集成，提供了历史数据导入、联系人导入、事项的转发等功能。

1.2.3 国内外 GTD 工具发展趋势

通过对大部分国内外 GTD 工具的使用和研究，目前国内外大多数 GTD 事务管理工具都朝着跨平台和协同工作这两个方向发展。

跨平台：随着科技的发展，各种各样的嵌入式设备正融入人们的日常生活中，人们对软件的使用也不再局限于 PC 平台。对于日常事务的管理，人们更加倾向于在更加方便，更加触手可及的智能手机、PDA，Web 等平台上进行操作。通过使用移动设备和网络平台，用户可以随时随地的创建，编辑，处理事务，真正将 GTD 事务管理方法融入日常生活中。

协同工作：随着网络技术的不断发展，基于互联网的协同工作技术越来越受到人们的重视，多人协同工作的创建，修改，同步等问题成为 GTD 工具需要解决的问题之一。目前大部分 GTD 工具对协同工作功能缺乏良好的支持。

1.3 论文组织结构

本文组织结构如下：

第一章 绪论：介绍项目背景，说明系统所解决的现实问题。

第二章 技术概述：介绍实现系统所需要的关键技术。

第三章 系统分析与设计：分析系统需求，对系统进行概要设计，划分模块结构并对各模块进行详细设计。

第四章 系统实现：描述系统实现细节和技术难点。

第五章 总结与展望：总结整篇论文，提出改进之处，展望技术发展。

第二章 技术概述

2.1 GTD

GTD[1]即 Getting Things Done 的缩写，2001 年 Allen David 出版了《Getting Things Done: The Art of Stress-Free Productivity》，在这本书中 GTD 行为管理方法首次被提出。GTD 行为管理方法的理念在于只有将工作者心中所想的所有的事情都记录下来并且安排好下一步的计划，工作者才能够心无挂念，全力以赴地做好目前的工作，提高效率。而当工作者总是有些事萦绕在心头，悬而未决的时候，要么就是会不时地想起它而影响现在的工作，要么就是会忘记了去做。而 GTD 行为管理方法通过将所有的这些事都罗列出来再进行分类，确定下一步的处理方法，将所有这些悬而未决之事都纳入一个可控制的管理体系中。[1]

GTD 的具体实践可以分为收集、整理、组织、回顾与执行五个基本活动。执行流程如图 2.1。

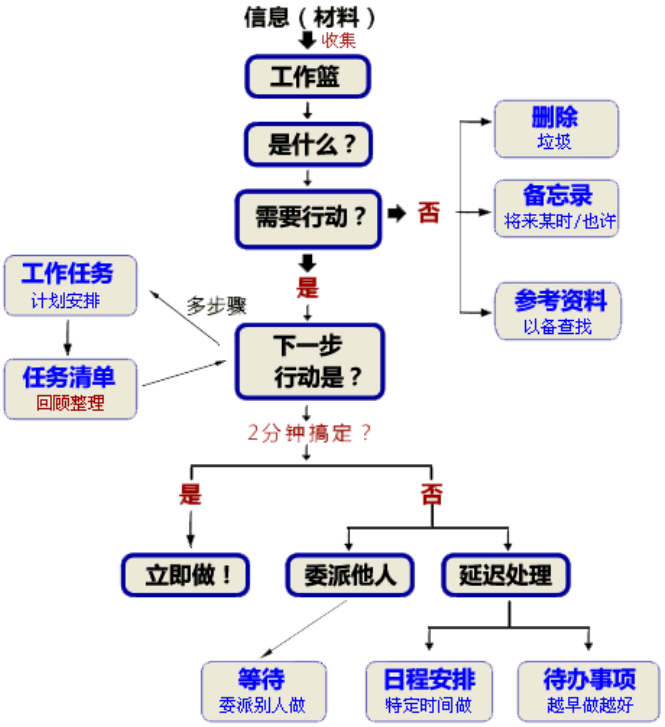

图 2.1 GTD 行为管理方法流程图

搜集：GTD 行为管理方法要求把任何用户需要跟踪，记录或者执行的事情整理搜集在一个称为“收集箱”的地方，“收集箱”可以是收件箱，电子邮箱，磁带，笔记本，PDA，或者它们的组合。把你脑子里的任何东西都拿出来放到你的搜集设备里，准备好做下一步的处理。每星期所有的“收集箱”都应该被至少清空一次。

整理：GTD 行为管理方法要求遵循一个严格的工作流程整理用户收集箱：

1）从最上面开始。
2）一次处理一项。
3）不把任何东西放回收集箱。
4）如果任何一个事项需要执行：
a.如果执行该事项所耗费的时间少于两分钟则执行；
b.委托别人完成；
c.将其延期。
5）否则
a.把它存档以便查询；
b.把它扔掉；
c.使它成熟以便下一步的处理 。

在 GTD 处理工作流程中需要遵守两分钟原则，即：任何事项如果执行花费的时间少于两分钟，那么立即执行该事项。两分钟是一个分水岭，这样的时间和正式地推迟一个动作所花的时间差不多。

组织：GTD 行为管理方法要求建立一个列表集合，使得用户可以跟踪需要关注的项目：

1）下一步行动(Next actions)：对于每个需要用户关注的事项，GTD 要求用户制定好什么是可以实际采取的下一步行动。例如，如果事项为“写项目报告”，则下一步行动可能会是“给 Fred 发邮件开个简短会议”，或者“给 Jim 打电话问报告的要求”，或者类似的事情。虽然要完成这个事项，可能会有很多的步骤和行动，但是其中一定会有用户需要首先去做的事情，这样的事情就应该被记录在“下一步行动”列表上。GTD 推荐的做法是把这些事项根据能够被完成的“环境”整理分类，例如“在办公室”,“用电话”,“在商场”等。

2）项目(Projects)：每个需要多于一个实际的行动才能达到的生活或者工作中的“开放式回路”就是一个“项目”。GTD 要求使用跟踪及周期性的回顾来确保每个项目都有一个“下一步行动”进行下去。

3）等待(Waiting for)：当用户已经指派了一个事项给其他用户执行或者在项目进行下去之前需要等待外部的事件时，GTD 要求用户在系统中跟踪以及定期检查是否已经可以采取行动或者需要发出一个提醒。

4）将来／可能(Someday/Maybe)：这些事情为用户需要在某个点去做，但是不是马上。例如：“学习中文”，或者“进行一个潜水假期”。

5）归档：GTD 的最后一个关键组织模块是归档系统。推荐的方式是用户可以维护一个按照字母顺序组织的归档系统，这样可以比较容易快速的存储和提取你所想要的信息。

检查：如果用户不是至少每天回顾检查，那么用户的行动和提醒的列表将会变的毫无用处。推荐的做法是只要用户有时间就进行回顾检查。以用户当时拥有的精力，资源和时间，决定什么是当前重要的事情，然后执行。如果用户倾向于拖延，那么用户可能会总是做最容易的事情而避免那些难的。为了解决这个问题，用户可以按事项在列表上的顺序一个接一个执行事项。

GTD 要求用户至少以星期为周期回顾所有比较主要的“行动”,“项目”和“等待”的事项，确保所有的新任务或者即将到来的事件都进入用户的系统，而且所有的事情都更新到最新的情况。GTD 推荐制作一个难题档案来帮助用户更新主要行动的记忆。

执行：用户按照列表开始行动，在具体行动中可能会需要根据所处的环境，时间的多少和精力的情况以及重要性来选择事项执行。

2.2 MVC

MVC[2]模式是一个由多种设计模式复合而成的复合设计模式，它强制性的使应用程序的输入、处理和输出分开。使用 MVC 应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。MVC 结构图如图 2.2。

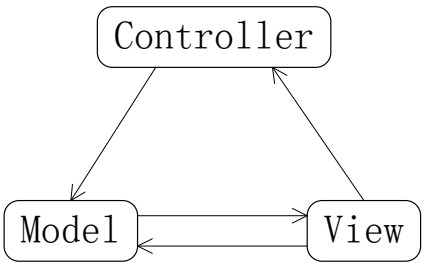

图 2.2 MVC 结构图

模型：模型表示数据和业务逻辑。在 MVC 的三个部件中，模型处理最多的任务。模型返回的数据是中立的，即模型与数据格式无关。同一个模型能为多个视图提供数据，减少了代码的重复性。

视图：视图是用户看到并与之交互的界面。对于桌面应用程序来说，视图就是由语言的显示对象构建出的程序界面。对于传统的 Web 应用程序来说，视图就是由 HTML 元素组成的页面。而在新式的 Web 应用程序中，HTML 依旧在视图中扮演着重要的角色，但会配合一些新式的技术以其构成视图，新式技术包括 Adobe Flash，XHTML，XML/XSL，WML 等一些标识语言和 Web services 等。

MVC 一大好处是它能为你的应用程序处理很多不同的视图。视图中只负责显示数据而不是处理数据，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它的职责只是种输出数据并和用户进行操作。

控制器：控制器接受用户的输入并调用模型和视图的接口去完成用户的需求。例如当单击程序中的按钮或者提交 Web 表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后确定用哪个视图来显示模型处理返回的数据。

在 MVC 的处理过程中，首先控制器接收用户的请求，并决定应该调用哪个模型来进行处理，然后模型用业务逻辑来处理用户的请求并返回数据，最后控制器应用相应的视图格式化模型返回的数据，并通过表示层呈现给用户。

MVC 的使用可以降低系统各模块之间的耦合度；提高系统的重用性；降低开发生命周期的成本；使系统能够快速部署；提高系统的可维护性和可管理性。

当然 MVC 模式并不是适合所有的系统开发，使用 MVC 模式也会给设计和开发带来一些弊端，如 MVC 难于理解，会增加测试的难度，会增加工作量。MVC 模式不适合中小项目的开发。

2.3 Java

Java[4]是由 Sun Microsystems 公司于 1995 年 5 月推出的 Java 程序设计语言和 Java 平台的总称。Java 平台由 Java 虚拟机和 Java 应用编程接口构成。Java 应用编程接口为 Java 应

用提供了一个独立于操作系统的标准接口，可分为基本部分和扩展部分。Java 分为三个体系 Java SE(Java Platform Standard Edition，Java 平台标准版)，Java EE(Java Platform Enterprise Edition，Java 平台企业版)和 Java ME(Java Platform Micro Edition，Java 平台微型版)。Java 平台体系结构图如图 2.3

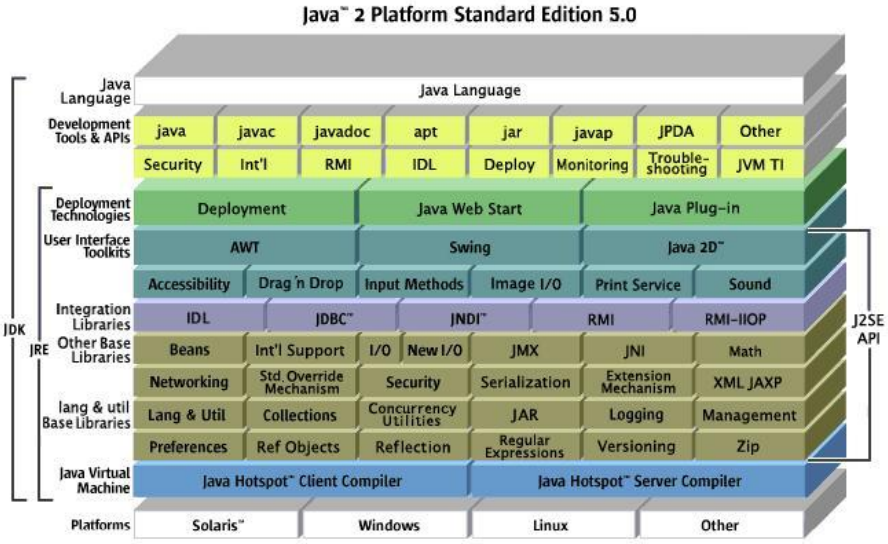

图 2.3 Java 平台体系结构图

Java 语言的语法与 C 语言和 C++语言很接近，使得大多数程序员很容易学习和使用 Java。另一方面，Java 丢弃了 C++ 中很少使用的、很难理解的、令人迷惑的那些特性，如操作符重载、多继承、自动的强制类型转换。特别地，Java 语言不使用指针，并提供了垃圾自动回收机制，使得程序员不必自己进行内存管理，大大降低了编程难度。

Java 语言是一个面向对象程序设计语言。Java 语言提供类、接口和继承等原语，降低开发难度，Java 只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制。Java 语言全面支持动态绑定。总之，Java 语言是一个纯的面向对象程序设计语言。

Java 程序（.java 文件）被编译为体系结构中立的字节码文件（.class 的文件），然后可以在 Java 虚拟机中运行。这样实现了 Java 的跨平台特性

Java 语言的优良特性使得 Java 应用具有无比的健壮性和可靠性，这也减少了应用系统的维护费用。Java 对对象技术的全面支持和 Java 平台内嵌的 API 能缩短应用系统的开发时间并降低成本。Java 的编译一次，到处可运行的特性使得它能够提供一个随处可用的开放结构和在多平台之间传递信息的低成本方式。特别是 Java 企业应用编程接口（Java Enterprise APIs）为企业计算及电子商务应用系统提供了强大的支持。

Java SE（Java Platform Standard Edition，Java 平台标准版）。Java SE 允许开发和部署在桌面、服务器、嵌入式环境和实时环境中使用的 Java 应用程序。Java SE 包含了支持 Java Web 服务开发的类，并为 Java Platform，Enterprise Edition（Java EE）提供基础。在本项目中主要使用到了 Java SE 中的基本类库和 Swing 框架来实现客户端程序。

Java EE（Java Platform Enterprise Edition，Java 平台企业版）。Java EE 帮助开发和部署可移植、健壮、可伸缩且安全的服务器端 Java 应用程序。Java EE 是在 Java SE 的基础上构建的，它提供 Web 服务、组件模型、管理和通信 API，可以用来实现企业级的面向服务体系结构（service-oriented architecture，SOA）和 Web 2.0 应用程序。

Java EE 出了包含 Java SE 的基本应用程序开发接口外，还针对企业级开发提供了开发

支持，Java EE 平台由一整套服务（Services）、应用程序接口（APIs）和协议构成，它对开发基于 Web 的多层应用提供了功能支持。Java EE 的核心技术包括：JDBC(Java Database Connectivity)，JNDI(Java Name and Directory Interface)，EJB(Enterprise JavaBean)，RMI(Remote Method Invoke)，Java IDL/CORBA，JSP(Java Server Pages)，Java Servlet，XML(Extensible Markup Language)，JMS(Java Message Service)，JTA(Java Transaction Architecture)，JTS(Java Transaction Service)，Java Mail，JAF(JavaBeans Activation Framework)。由于篇幅有限，此处指介绍与项目相关的 Java Servlet，XML，和 Java Mail 技术。

Java Servlet[5]: Servlet 是一种小型的 Java 程序，它扩展了 Web 服务器的功能。作为一种服务器端的应用，当被请求时开始执行，这和 CGI Perl 脚本很相似。Servlet 提供的功能大多与 JSP 类似，不过实现的方式不同。JSP 通常是大多数 HTML 代码中嵌入少量的 Java 代码，而 servlets 全部由 Java 写成并且生成 HTML。

XML(Extensible Markup Language)[5]: XML 是一种可以用来定义其它标记语言的语言。它被用来在不同的商务过程中共享数据。XML 的发展和 Java 是相互独立的，但是，它和 Java 具有的相同目标正是平台独立性。通过将 Java 和 XML 的组合，您可以得到一个完美的具有平台独立性的解决方案。

JavaMail[5]: JavaMail 是用于存取邮件服务器的 API，它提供了一套邮件服务器的抽象类。不仅支持 SMTP 服务器，也支持 IMAP 服务器。

2.4 C/S

C/S[8]架构是一种典型的两层软件系统体系架构，其全称是 Client/Server 架构，即客户端服务器端架构。通过它可以充分利用客户端和服务器端硬件环境的优势，将任务合理分配到 Client 端和 Server 端来实现，降低了系统的通讯开销。其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是 Socket 服务器端，服务器端的程序通过 Socket 与客户端的程序通信。C/S 架构图如图 2.4。

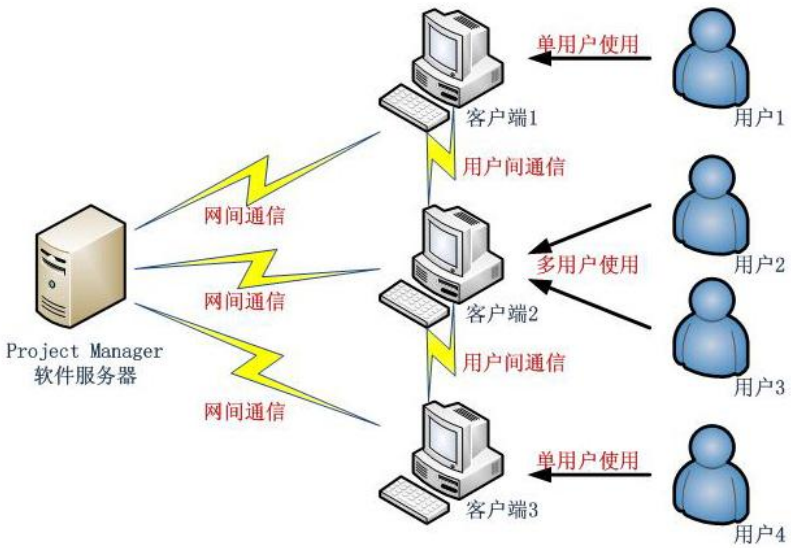

图 2.4 C/S 架构图

C/S 架构的优点如下：

界面和操作丰富：由于 Client（客户端）为运行在用户电脑上的程序，所有基于用户电脑操作系统的操作均可实现。

安全性能高：在 C/S 架构中，Client（客户端）和 Server（服务器）的通信多为基于 Socket 的自定义协议的通信，可以实现多层认证，具有较高的安全性。同时在 C/S 架构中，用户的数据可以存储在服务器上，保证了数据的安全性和可靠性。

响应速度较快：在 C/S 架构中由于只有一层交互，同时大量的计算可以在客户端实现，减轻了服务器的计算压力，保证了系统的稳定性和响应时间。

C/S 架构的缺点如下：

适用面窄：一般一个客户端程序只能运行在一个平台上。使用跨平台技术虽然可以做到多平台的支持，但是会降低程序的性能。

用户群固定：由于客户端程序需要安装才可使用，因此不适合面向一些不可知的用户。

维护成本高：系统每次升级，所有的客户端程序都要改变。

2.5 XML

XML[9]（Extensible Markup Language），可扩展标记语言是一种平台无关的表示数据的方法。由于 XML 是文本格式，故使用 XML 创建的数据可以被任何应用程序在任何平台上读取，甚至可以通过手动编码来编辑和创建 XML 文档。XML 通常由文档（Document）、声明（Declaration）、标签（Tag）、文本（Text）、元素（Element）、属性（Attribute）、注释（Comments）和处理指令（Processing Instruction）等 8 部分组成。

XML 文档可以通过多种方式进行解析，本系统中主要使用了 DOM 和 XPath 这两种常用的方法对 XML 文档进行解析。

XPath 是一门在 XML 文档中查找信息的语言，用于在 XML 文档中通过元素和属性进行导航，通过 XPath 我们可以在 XML 中准确定位。XPath 是许多高级 XML 应用如 XQuery 和 XPointer 等的基础。

DOM[10]（Document Object Model），文件对象模型是 W3C 组织推荐的处理 XML 的标准编程接口。DOM 的结构如图 2.5 所示。DOM 将整个 XML 文件映射为一个由层次节点组成的文件，通过在内存中生成树的结构来存储数据。目前 DOM 有 1 级，2 级，3 级共 3 个级别。

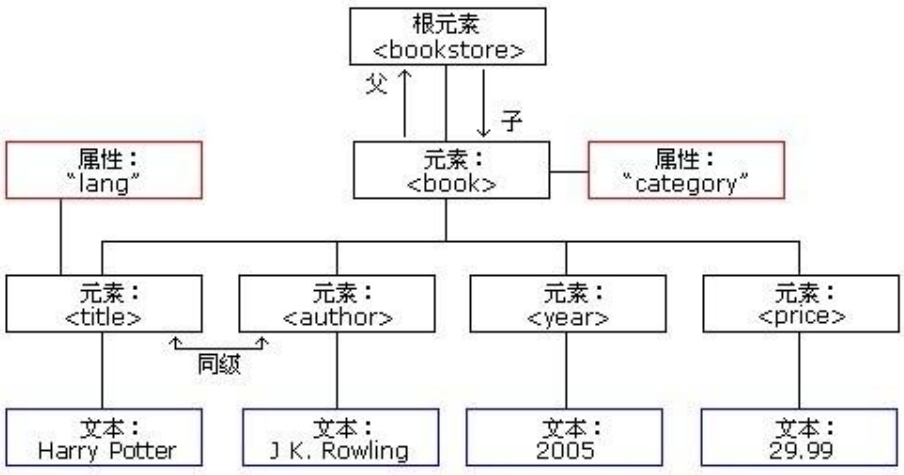

图 2.5 DOM 节点树实例

DOM 使用方便，可被重复使用。但是 DOM 由于自身设计的原因，备受解析速度慢，内存占用多的诟病。尤其当解析的 XML 文件较大时，这个问题特别突出。对于 1-4G 的内存，如果 XML 文件大小大于 10M，则很有可能会造成内存溢出。其次，DOM 不是线程安全的。若要并发访问 DOM，必须考虑并发控制。

第三章 系统分析与设计

3.1 系统需求分析

3.1.1 客户端需求分析

采用用例驱动对系统客户端功能需求分析如下：

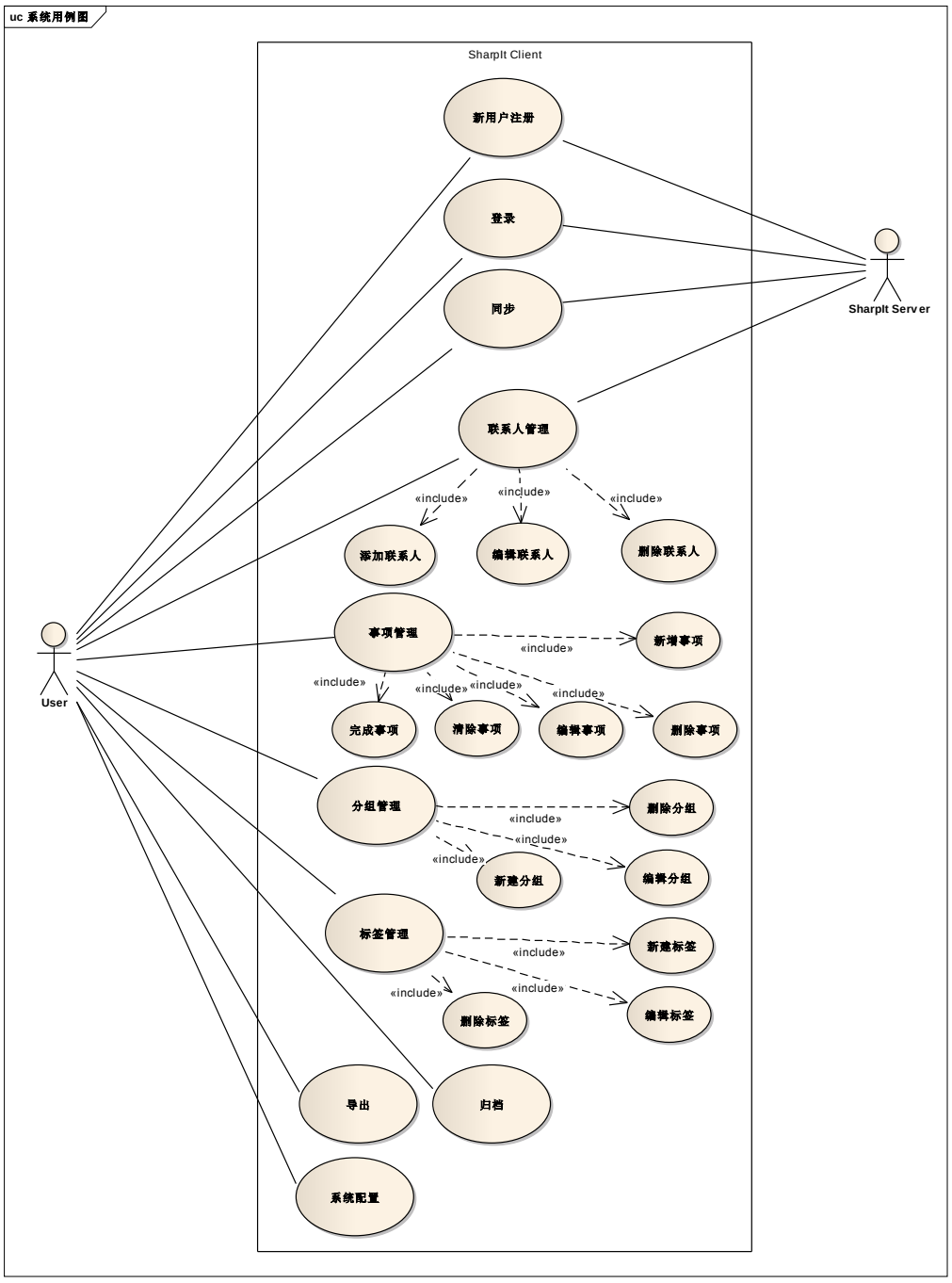

图 3.1 SharpIt 客户端用例图

对用例采用事件流分析如下：

表 3.1 新用户注册用例

用例名称	新用户注册
用例描述	新用户注册用例需要用户使用一个常用邮箱作为登录系统的标识，并设置自己的昵称和日期显示格式，提交后系统会向用户的邮箱发送密码，用户登录邮箱获得密码后即可激活账号登录系统。
主事件流	1）前置条件，用户进入注册面板，用例开始 2）系统提示用户输入常用邮箱 3）用户输入常用邮箱 4）系统检查邮箱有效性 A1：邮箱已被注册或者邮箱格式不正确 5）系统提示用户输入昵称 6）系统提示用户选择日期显示格式 7）系统提示用户勾选同意软件服务条款 8）用户点击注册 A2：注册失败 9）系统提示密码已发送到邮箱 10）系统返回登录界面 11）用例结束
其他事件流	**A1:** 1）系统提示用户邮箱已存在或格式错误，请重新输入邮箱 2）返回主事件流第 3 步 **A2：** 1）系统提示用户注册失败，请重新注册 2）返回主事件流第 2 步

表 3.2 登录用例

用例名称	登录
用例描述	登录用例需要用户输入在系统注册过的邮箱和密码登录系统。
主事件流	1）前置条件，用户进入登录面板，用例开始 2）系统提示用户输入邮箱和密码 3）用户输入邮箱和密码 4）系统验证用户输入的邮箱和密码 A1：用户名不存在或者密码错误 5）系统进入主界面，用例结束
其他事件流	**A1:** 1）系统提示用户邮箱或密码有误，请重新输入 2）返回主事件流第 3 步

表 3.3 同步用例

用例名称	同步
用例描述	同步用例中客户端和服务器端存储的用户数据将同步一致。
主事件流	1）前置条件，用户进入同步面板，用例开始

	3）用户点击立刻同步按钮 4）系统客户端开始和服务器端同步数据 A1：连接服务器失败 5）体统提示用户同步完毕，用例结束
其他事件流	**A1:** 1）系统提示用户无法连接服务器，请检查网络或联系客服 2）返回主事件流第 1 步

表 3.4 联系人管理用例

用例名称	联系人管理
用例描述	联系人管理用例包括添加联系人，删除联系人，编辑联系人信息三个子用例
子用例名称	添加联系人
子用例描述	添加联系人用例中首先系统需要用户输入联系人在系统注册的邮箱，然后系统向联系人发送添加联系人的请求，最后得到联系人的同意后完成添加联系人的操作。
主事件流	1）前置条件，用户进入邀请联系人面板，用例开始 2）系统提示用户输入联系人的邮箱 3）用户输入联系人的邮箱 A1：邮箱不存在或者邮箱格式错误 4）系统提示用户输入联系人的昵称 5）用户输入联系人的昵称 6）用户点击添加按钮 7）系统向联系人的账号发送请求 8）联系人同意请求 A2：联系人拒绝请求 9）用户添加联系人完成，用例结束
其他事件流	**A1:** 1）系统提示该邮箱不存在或邮箱格式错误，请用户重新输入。 2）返回主事件流第 2 步 **A2:** 1）系统向用户发送系统消息，联系人拒绝添加请求。用例结束
子用例名称	删除联系人
子用例描述	删除联系人用例中用户选择要删除的联系人后点击删除按钮即可完成删除联系人的操作。
主事件流	1）前置条件，用户进入联系人列表用例开始 2）用户点击要删除的联系人条目 3）用户点击删除按钮 4）系统提示用户是否确定要删除 5）用户点击确定 A1：用户点击取消 6）系统删除联系人，用例完成
其他事件流	A1: 1）系统返回联系人列表，用例结束
子用例名称	编辑联系人

子用例描述	编辑联系人用例中用户在联系人编辑面板中编辑联系人的昵称和电话号码后即可完成编辑联系人的操作。
主事件流	1）前置条件，用户进入联系人编辑，用例开始 2）用户编辑联系人的昵称和电话 3）用户编辑联系人完成，用例结束
其他事件流	无

表 3.5 事项管理用例

用例名称	事项管理
用例描述	事项管理用例包括新建事项，删除事项，清除事项，编辑事项，完成事项五个子用例。
子用例名称	新建事项
子用例描述	新建事项用例中用户可以通过点击新建按钮打开新建面板或者收集箱中的快速新建栏来新建一个事项。
主事件流	1）前置条件，用户进入新建事项面板，用例开始 2）用户编辑事项的各项属性，包括名称，标签，备注，过期时间，转发，执行时间，是否重复等内容。 3）用户点击确定按钮 4）系统按照用户的设置新建一个事项并将其放置到指定的事项列表，用例结束
其他事件流	无
子用例名称	删除事项
子用例描述	删除事项用例中用户可以通过点击删除按钮来删除选中的事项。删除的事项会被系统移至垃圾箱，用户可以在垃圾箱的事项列表中对已被删除的事项进行编辑。
主事件流	1）前置条件，用户进入事项列表，用例开始 2）用户选中想要删除的一个或多个事项 3）用户点击删除按钮 4）系统将用户删除的事项移至垃圾箱，用例结束
其他事件流	无
子用例名称	清除事项
子用例描述	清除事项用例中用户可以通过点击垃圾箱列表的清除按钮来清除选中的事项。清除的事项将不可恢复
主事件流	1）前置条件，用户进入垃圾箱事项列表，用例开始 2）用户选中想要清除的一个或多个事项 3）用户点击清除按钮 4）系统将选中的事项清除，用例结束
其他事件流	无
子用例名称	编辑事项
子用例描述	编辑事项用例中用户可以通过点击编辑按钮或者双击事项列表中的事项条目打开事项编辑面板来编辑选中的事项。
主事件流	1）前置条件，用户进入事项编辑面板，用例开始 2）用户编辑事项的各项属性，包括名称，标签，备注，过期时间，转

	发，执行时间，是否重复等内容。 3）用户提交编辑完成的事项 4）系统对用户编辑的事项进行修改并根据事项的执行时间将事项分类到相应的事项列表，用例结束
其他事件流	无
子用例名称	完成事项
子用例描述	完成事项用例中用户可以通过点击完成按钮或者点击事项条目中的完成复选框来完成选中的事项。
主事件流	1）前置条件，用户进入事项列表，用例开始 2）用户选中已经完成的一个或多个事项。 3）用户点击完成按钮或点击事项条目中的完成复选框。 4）系统将用户完成的事项移至已完成事项列表，用例结束
其他事件流	无

表 3.6 标签管理用例

用例名称	标签管理
用例描述	标签管理用例包括新建标签，删除事项，编辑标签三个子用例。
子用例名称	新建标签
子用例描述	添加联系人用例中首先系统需要用户输入联系人在系统注册的邮箱，然后系统向联系人发送添加联系人的请求，最后得到联系人的同意后完成添加联系人的操作。
主事件流	1）前置条件，用户进入新建标签面板，用例开始 2）系统提示用户输入所要创建标签的标签名 3）用户输入标签名 A1：标签名已存在 A2：用户输入为空 4）用户点击确定按钮 5）系统创建新的标签，用例结束
其他事件流	**A1:** 1）系统提示标签名已存在，请用户重新输入。 2）返回主事件流第 2 步 **A2:** 1）系统提示标签名不能为空，请用户重新输入。 2）返回主事件流第 2 步
子用例名称	删除标签
子用例描述	删除标签用例中用户可以通过在标签列表中选中想要删除的标签，然后点击删除按钮删除标签。
主事件流	1）前置条件，用户进入标签列表，用例开始 2）用户选中想要删除的标签 3）用户点击删除按钮 A1：被删除的标签被事项引用 4）系统删除标签，用例结束
其他事件流	**A1:**

	1）系统提示已有事项引用该标签，询问用户是否要删除标签 2）用户点击确定按钮，返回主事件流第 4 步 A2 用户点击取消按钮 **A2:** 1）系统返回标签列表，用例结束
子用例名称	编辑标签
子用例描述	编辑标签用例中用户可以通过在标签列表中双击想要编辑的标签进入标签编辑模式编辑标签的名称。
主事件流	1）前置条件，用户进入标签列表，用例开始 2）用户双击想要编辑的标签进入标签编辑模式 3）系统提示用户输入新的标签名 3）用户输入新的标签名 A1：用户输入的标签名和已有标签名重复 4）系统修改标签，用例结束
其他事件流	**A1:** 1）系统提示该标签名已存在，请用户重新输入 2）返回主事件流第 3 步

表 3.7 分组管理用例

用例名称	分组管理
用例描述	分组管理用例包括新建分组，删除分组，编辑分组，完成分组四个子用例。
子用例名称	新建分组
子用例描述	新建分组用例中用户可以通过点击新建分组按钮新建一个默认名称的分组。
主事件流	1）前置条件，用户进入分组管理面板，用例开始 2）用户点击新建分组按钮 3）系统新建一个默认名称的分组，用例结束。
其他事件流	无
子用例名称	删除分组
子用例描述	删除分组用例中用户可以通过点击删除分组按钮删除选中的分组及分组内的所有事物。
主事件流	1）前置条件，用户进入分组管理面板，用例开始 2）用户选中需要删除的分组 3）用户点击删除分组按钮 4）系统提示用户是否确定要删除分组及分组内的事项 5）用户点击确定按钮 A1 用户点击取消按钮 6）系统删除分组，并将分组内的事项移至垃圾箱列表，用例结束
其他事件流	**A1:** 1）用户点击取消 2）系统返回分组管理面板，用例结束
子用例名称	编辑分组
子用例描述	编辑分组用例中用户可以通过双击分组管理面板中的想要编辑的分组条目进入分组编辑模式对分组的名称进行编辑。
主事件流	1）前置条件，用户进入分组管理面板，用例开始

	2）用户双击想要编辑的分组条目进入分组编辑模式 3）系统提示用户输入新的分组名称 4）用户输入新的分组名称 5）系统修改分组的名称，用例结束
其他事件流	无
子用例名称	完成分组
子用例描述	完成分组用例中用户可以通过点击完成分组按钮批量完成分组中的事项。
主事件流	1）前置条件，用户进入分组管理面板，用例开始 2）用户点击完成分组按钮 3）系统将分组内的所有事物移至已完成列表，用例结束
其他事件流	无

表 3.8 归档用例

用例名称	归档
用例描述	归档用例中用户通过点击完成列表中的归档按钮将已完成的事项进行归档整理。
主事件流	1）前置条件，用户进入事项完成列表，用例开始 2）用户点击归档按钮 3）系统询问用户需要归档事项的特征 4）用户选定事项归档的条件 5）系统按照用户指定条件对已完成事项进行归档，用例结束
其他事件流	无

表 3.9 导出用例

用例名称	导出
用例描述	导出用例中用户通过点击导出按钮可以将系统中的事项导出为 CVS 文件以方便其他系统和工具使用。
主事件流	1）前置条件，用户进入事项列表，用例开始 2）用户点击导出按钮 3）系统打开路径选择面板，让用户选择 CVS 文件保存的路径 4）用户选择路径 5）用户点击保存按钮 6）系统将事项列表中的事项按照用户指定的路径保存为 CVS 文件，用例结束
其他事件流	无

表 3.10 系统配置用例

用例名称	系统配置
用例描述	系统配置用例中用户可以配置系统的基本信息，包括用户昵称，日期的显示格式，是否开机自动启动，是否自动登录等内容。
主事件流	1）前置条件，用户进系统设置面板，用例开始 2）用户编辑用户昵称，日期显示格式，是否开机启动，是否自动登录等内容 3）系统根据的用户的选项进行配置，用例结束
其他事件流	无

3.1.2 服务器端需求分析

采用用例驱动对系统服务器端功能需求分析如下：

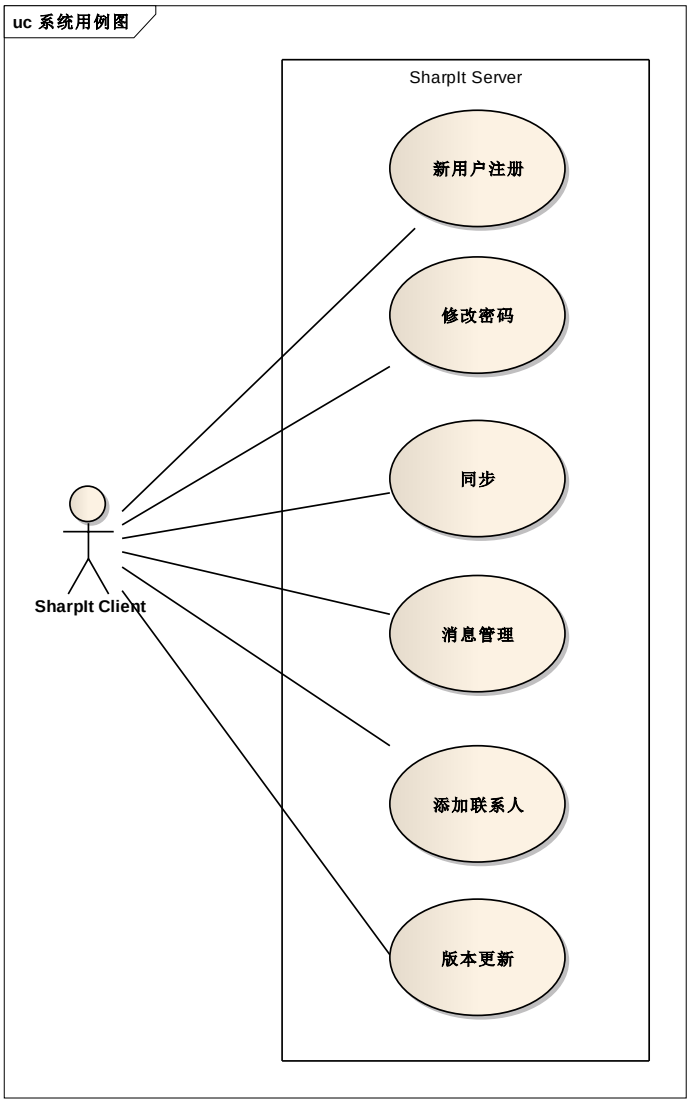

图 3.2 SharpIt 服务器端用例图

表 3.11 新用户注册用例

用例名称	新用户注册
用例描述	新用户注册用例中客户端向服务器端提交用户填写的个人信息，服务器对用户的个人信息进行验证后向用户的邮箱发送登录系统的密码，用户完成注册。
主事件流	1）前置条件，客户端向服务器端提交新用户注册请求，用例开始 2）服务器系统验证用户的邮箱 A：用户的邮箱已被注册 3）系统向用户的邮箱发送密码 4）系统返回给客户端注册成功信息，用例结束
其他事件流	**A1:** 1）系统返回给客户端该邮箱已被注册，用例结束

表 3.12 修改密码用例

用例名称	修改密码
用例描述	修改密码用例中客户端向服务器端提交用户原有的密码和新的密码，系统验证原有密码后修改用户的密码为新密码。
主事件流	1）前置条件，客户端向服务器端提交修改密码请求，用例开始 2）服务器系统验证用户的原有密码 A1 用户提交的原有密码和实际密码不符 3）系统返回给客户密码修改成功信息，用例结束
其他事件流	**A1:** 1）系统返回给客户端密码输入错误信息，用例结束

表 3.13 同步用例

用例名称	同步
用例描述	同步用例中客户端向服务器端提交同步客户数据请求，系统将客户端数据和服务器端数据同步后将最新的数据发送给客户端。
主事件流	1）前置条件，客户端向服务器端提交同步请求，用例开始 2）服务器同步客户端和服务器端的数据，用例结束
其他事件流	无

表 3.14 消息管理用例

用例名称	消息管理
用例描述	消息管理用例中系统有了新的消息就会发送给客户端。
主事件流	1）前置条件，新的系统消息产生，用例开始 2）服务器向客户端发送消息，用例结束
其他事件流	无

表 3.15 添加联系人用例

用例名称	添加联系人
用例描述	添加联系人用例中客户端向服务器端提交添加联系人请求，系统将请求转发给被请求的联系人，等待被请求的联系人回复后完成添加联系人用例。
主事件流	1）前置条件，客户端向服务器端提交添加联系人请求，用例开始 2）服务器向被添加的联系人转发请求 3）被请求的联系人同意请求 A1 被请求的联系人拒绝请求 4）系统分别向两方的客户端发送联系人添加成功信息，用例结束
其他事件流	**A1:** 1）系统向发出请求的客户端发送拒绝添加信息，用例结束

表 3.16 版本更新用例

用例名称	版本更新
用例描述	版本更新用例中客户端向服务器端提交检查更新请求，系统比较客户端的版本号和最新版本号，如果版本号不同则对客户端进行更新。
主事件流	1）前置条件，客户端向服务器端提交检查更新请求，用例开始 2）服务器对比客户端的版本号和最新版本号 3）版本号不同，服务器向客户端发送更新命令 A1 版本号相同 4）客户端下载更新程序进行更新，用例结束
其他事件流	**A1:** 1） 系统向客户端发出不需要更新的消息，用例结束

3.1.3 系统性能需求分析

对系统的性能需求分析如下：

精度：产品对于精度要求不高，系统对所有的用户输入均做出检查和判断以确保用户输入正确。

时间特性要求：

1 响应时间：界面跳转响应时间≤1s。

2 更新处理时间：数据修改响应时间≤2s。

3 数据传送时间：远程数据同步时间≤10s，如超过 10s 则提示无法创建于服务器的连接。

灵活性：

1 同时支持鼠标操作和键盘操作。

2 能适应不同操作系统。

3 在用到其他软件工具时，封装接口，如外部工具有变，可随时更换外部工具。

4 能在尽量不改变程序整体的基础上对系统的功能进行添加和修改。

3.2 系统体系结构的设计

系统为 C/S 架构，客户端采用 MVC 模式进行设计，服务器端采用分层结构进行设计，系统体系结构如图 3.3 所示。

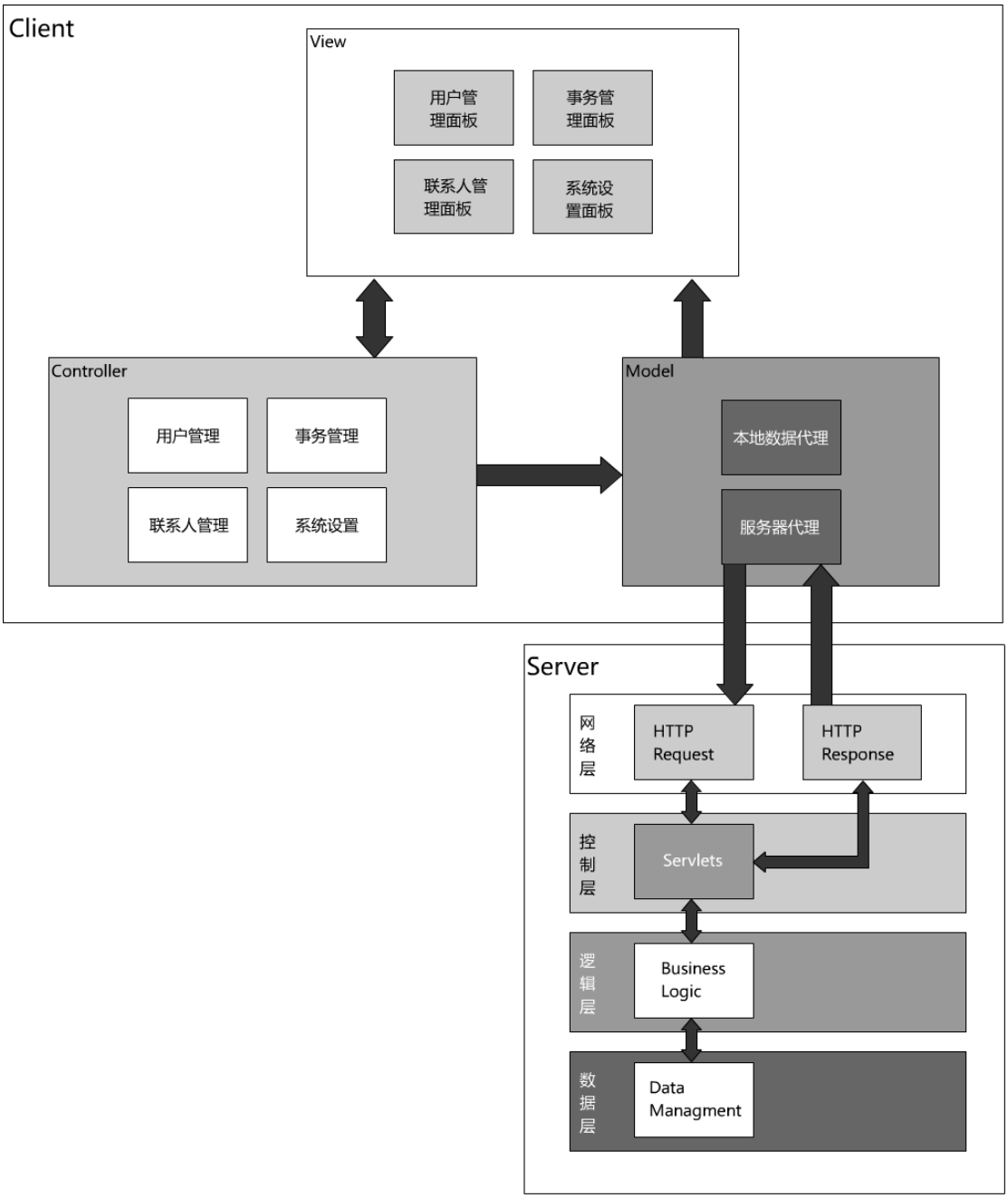

图 3.3 系统体系结构图

客户端按照 MVC 设计模式的要求分成 Model,View 和 Controller 三个大模块。Model 为数据模块，负责提供本地数据代理和服务器数据代理，本地数据代理主要负责本地 XML 数据的管理，服务器代理负责和服务器端的交互。View 为显示模块，负责与用户的交互。Controller 为控制模块，负责系统的业务逻辑，也是系统的核心部分，主要分为以下几个子模块：

事务管理模块：事务管理模块是系统的核心模块，包含对事项管理，标签管理，分组管

理等功能。

用户管理模块：用户管理模块负责用户基本信息的管理，包括设置用户昵称，修改密码等功能。

联系人管理模块：联系人管理模块负责管理用户的联系人，包括联系人的昵称，邮箱及用户与联系人之间互相转发的事项等。

系统设置模块：系统设置模块包括系统的语言设置，日期显示格式设置，是否自动登录及开机是否自动启动等设置。

服务器端采用分层结构，对于客户端的每个请求会执行每一层的方法对客户端进行操作。

网络层：负责与客户端的交互，通过接受客户端的 HTTPRequest 接收客户端的请求，通过 HTTPResponse 返回对客户端请求的处理结果。

控制层：负责将 HTTP 请求转交给相应的业务逻辑进行处理，并将业务逻辑层的处理结果封装为 XML 返回给 HTTPResponse。具体实现为一些 Sevrlet。

逻辑层：负责处理具体的业务逻辑，如注册，登录，同步等操作。

数据层：负责管理用户数据，并维持一个简单的缓存，以减少 I/O 操作，减少响应时间，提高系统的执行效率。

3.3 Sharp MVC 框架的设计

SharpIt 客户端采用 MVC 模式进行设计。通过对传统 MVC 模式进行扩充和改进自主开发了一个轻量级 MVC 框架——Sharp MVC，进一步对 MVC 模式中的各个模块进行解耦，方便团队开发和协作。Sharp MVC 架构图如图 3.4。

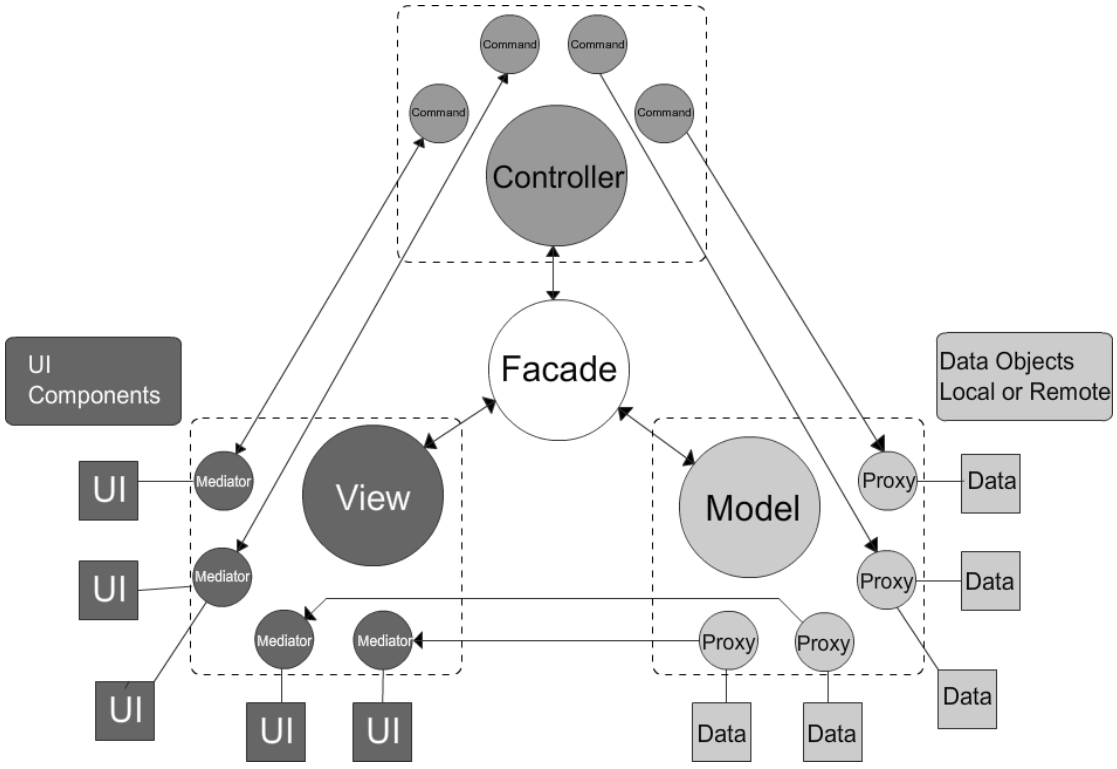

图 3.4Sharp MVC 架构图

Sharp MVC 在传统 MVC 结构的基础上进一步对 MVC 的数据，视图，控制器模块进行解耦。采用了观察者模式，单例模式，命令模式，代理模式，中介者模式进行复合实现。

Sharp MVC 框架的目标很明确，即把程序分为低耦合的三层：Model、View 和 Controller。降低模块间的耦合性，各模块如何结合在一起工作对于创建易扩展，易维护的应用程序是非常重要的。

在 Sharp MVC 实现的经典 MVC 元设计模式中，这三部分由三个单例模式类管理，分别是 Model、View 和 Controller。三者合称为核心层或核心角色。

Sharp MVC 中还有另外一个单例模式类——Fa çade，Fa çade 提供了与核心层通信的唯一接口，以简化开发复杂度。

Sharp MVC 中 Model 保存对 Proxy 对象的引用，Proxy 负责操作数据模型，与远程服务通信存取数据。 通过这样的方式保证了 Model 层的可移植性。

Sharp MVC 中 View 保存对 Mediator 对象的引用 。由 Mediator 对象来操作具体的视图组件，包括：添加事件监听器 ，发送或接收 Notification ，直接改变视图组件的状态。 这样做实现了把视图和控制它的逻辑分离开来。

Sharp MVC 中 Controller 保存所有 Command 的映射。Command 类是无状态的，只在需要时才被创建。 Command 可以获取 Proxy 对象并与之交互，发送 Notification，执行其他的 Command。

MVC 的主要类设计如下：

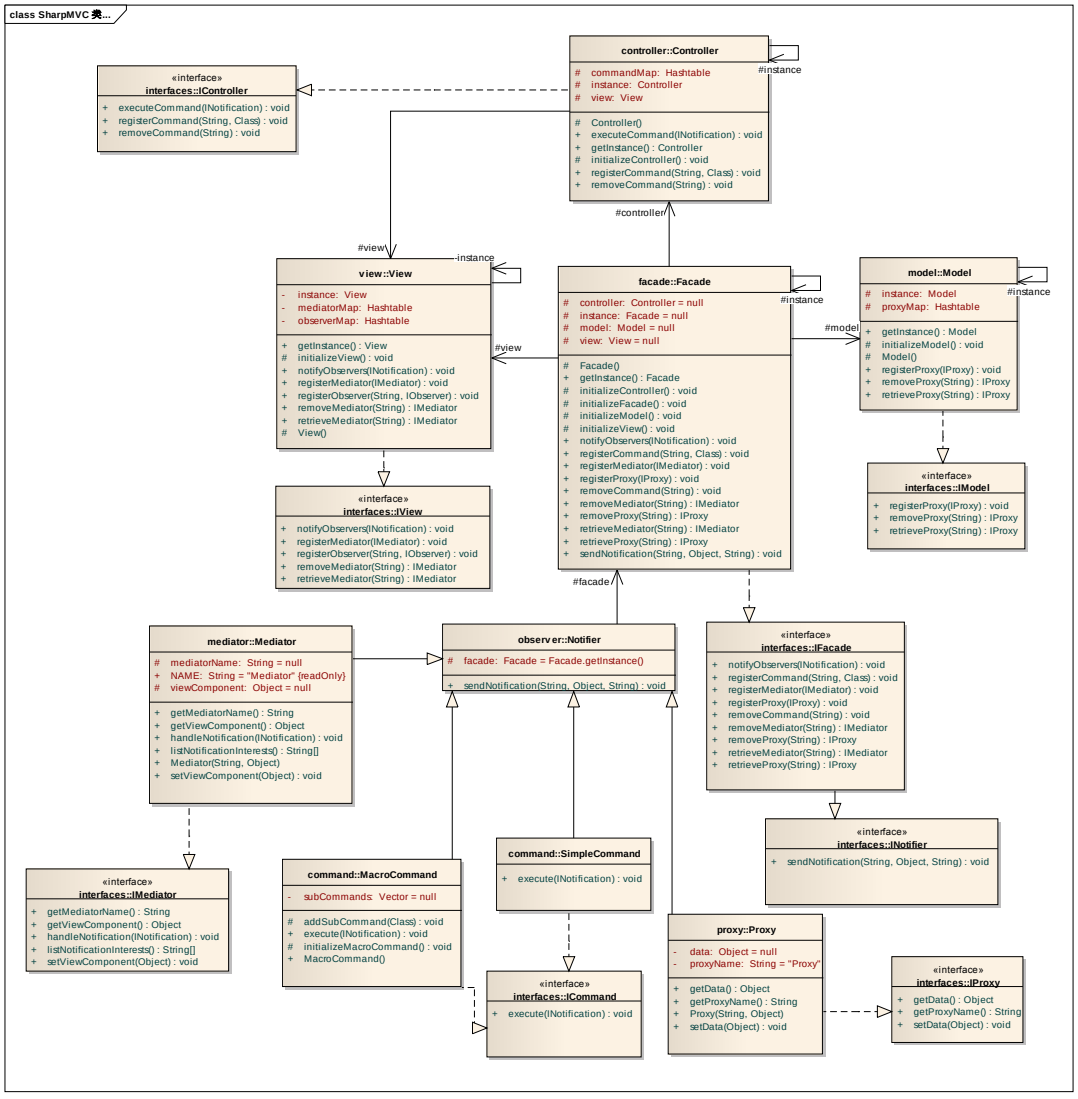

图 3.4Sharp MVC 类图

Facade：Facade 类采用单例模式，它负责初始化核心层（Model，View 和 Controller），并能访问它们的 Public 方法。

这样，在实际的应用中，开发者只需继承 Facade 类创建一个具体的 Facade 类就可以实现整个系统并不需要在代码中导入编写 Model，View 和 Controller 类。

Proxy、Mediator 和 Command 就可以通过创建的 Facade 类来相互访问通信。

Model：Model 类采用单例模式，它负责管理系统的数据对象及与逻辑即 Proxy 类，通过 registerProxy 方法向系统注册一个 Proxy，通过 removeProxy 方法从系统注销一个 Proxy，通过 retrieveProxy 方法可以获取一个 Proxy。

View：View 类采用单例模式，它负责管理系统的显示对象的中介者即 Mediator 类，通过 registerMediator 方法向系统注册一个 Mediator，通过 removeMediator 方法从系统注销一个 Mediator，通过 retrieveMediator 方法可以获取一个 Mediator。

Controller：Controller 类采用单例模式，它负责管理系统的业务逻辑对象即 Command 类，通过 registerCommand 方法向系统注册一个 Command，通过 removeCommand 方法从系统注销一个 Command。

Proxy：Model 保存对 Proxy 对象的引用。Proxy 对象负责操作数据模型和执行系统的域逻辑。Proxy 对象中保存的数据对象可以使本地数据也可以使远程数据。通过扩展 Proxy 对象向系统增加域逻辑代码，通过 getData 方法获取 Proxy 对象保存的数据，通过 setData 方法更改 Proxy 对象中的数据。

Mediator：View 保存对 Mediator 对象的引用 。Mediator 对象负责操作具体的视图组件，包括：添加事件监听器 ，发送或接收 Notification ，直接改变视图组件的状态等。一个 Mediator 对象可以操作一个或多个视图组件。通过改写 Mediator 类的 listNotificationInterests 方法向系统注册 Mediator 关心的 Notification，通过改写 handleNotification 方法对 Mediator 关心的 Notification 进行处理。通过 getViewComponent 方法获取 Mediator 负责操作的具体的视图组件。

Command： Controller 保存所有 Command 的映射。Command 类是无状态的，只在需要时才被创建。Command 类负责执行系统的业务逻辑，通过改写 excute 方法向系统添加业务逻辑。Command 可以获取 Proxy 对象并与之交互，发送 Notification，执行其他的 Command。经常用于复杂的或系统范围的操作，如应用程序的“启动”和“关闭”。

Sharp MVC 通过使用观察者模式以传递 Notificaton 对象的方式来实现一个松耦合的通信机制。 Observer 是整个 Sharp MVC 的基础。Sharp MVC 观察者模式如图

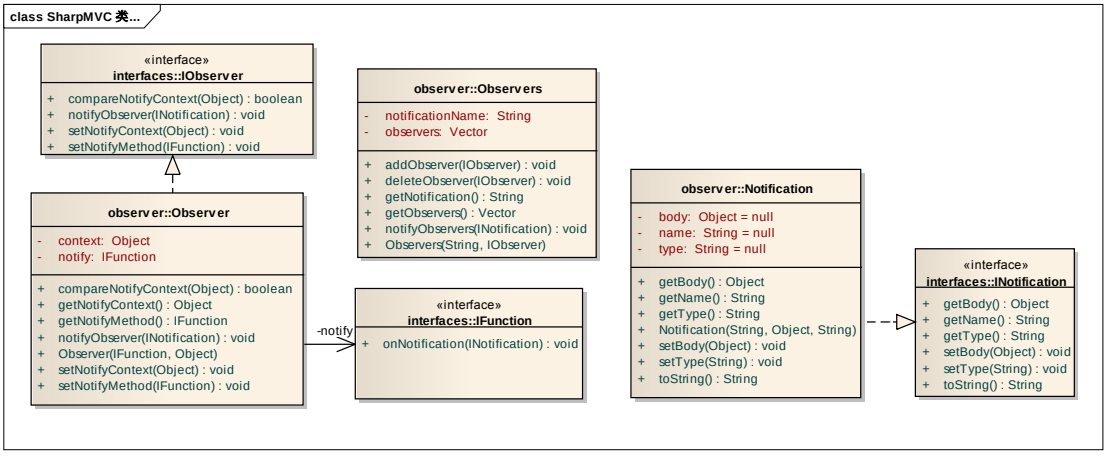

图 3.5Sharp MVC 观察者模式类图

Notification 可以被用来触发 Command 的执行，Facade 保存了 Command 与 Notification 之间的映射。当 Notification（通知）被发出时，对应的 Command（命令）就会自动地由 Controller 执行。Command 实现复杂的交互，降低 View 和 Model 之间的耦合性。

当用 View 注册 Mediator 时，Mediator 的 listNotificationInterests 方法会被调用，以数组形式返回该 Mediator 对象所关心的所有 Notification。 之后，当系统其它角色发出同名的 Notification（通知）时，关心这个通知的 Mediator 都会调用 handleNotification 方法处理 Notification（通知）。

在很多场合下 Proxy 需要发送 Notification（通知），比如：Proxy 从远程服务接收到数据时，发送 Notification 告诉系统；或当 Proxy 的数据被更新时，发送 Notification 告诉系统。 如果让 Proxy 也侦听 Notification（通知）会导致它和 View（视图）层、Controller（控制）层的耦合度太高。 View 和 Controller 必须监听 Proxy 发送的 Notification，因为它们的职责是通过可视化的界面使用户能与 Proxy 持有的数据交互。

3.4 Sharp UI 框架设计

由于 Java SE 中的 Swing 框架不能很好的满足系统客户端界面的开发，通过对 Swing 框架的扩展，自主设计和开发了一个轻量级的显示框架——Sharp UI。Sharp UI 通过对 Swing 框架的 JFrame，JPanel 两个容器进行扩展，采用事件机制进行模块间的通信，可以方便的跟换组件的皮肤和显示效果。在大大降低了使用复杂性的同时可以提供更加良好的用户体验。

Sharp UI 采用自定义的 SharpEvent 事件进行模块间的通信，实现的 SharpFrame, ImagePanel, CheckBox, ScrollPanel, ComboBox, SimpleButton, Calendar, ProgressBar 等常用组件，并提供相应的皮肤接口从而支持换肤的功能。

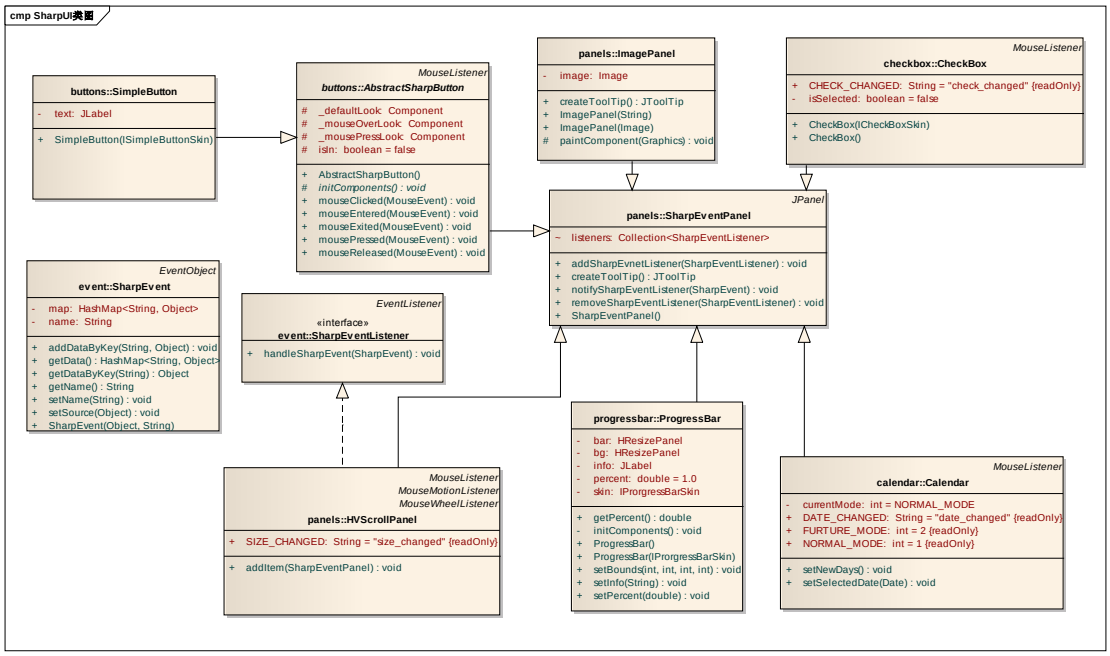

图 3.6SharpUI 类图

SharpEvent：Sharp UI 框架的自定义事件，包含 Name 和 Data 两个属性，Name 为事件的名称，Data 为事件携带的数据。

SharpEventListener：侦听 SharpEvent 事件的接口，所有侦听 SharpEvent 事件的类均需实现 SharpEventListener 接口并实现 handleSharpEvent 方法。

SharpEventPanel：SharpUI 所有组件的父类，继承自 JPanel，提供了对 SharpEvent 事件的支持。实现了 addSharpEventListener,removeSharpEventListner 和 notifyListener 三个方法从而实现了事件机制。

ImagePanel：显示指定路径图片的组件。

SimpleButton：按钮组件，提供了针对不同的鼠标动作而进行状态切换的功能。提供 setText 方法改变按钮上的文字。

CheckBox：复选框组件，可以进行复选操作。提供个 isSelected 方法获得该复选框是否被选中。

ScrollPanel：滚动面板组件，对较大的现实对象提供滚动条以便在较小的窗口内显示。提供 addItem 方法向面板中添加显示的对象。

ComboBox：下拉框组件，通过弹出下拉框为用户提供选择功能。提供 addItem 方法添加条目，提供个 getSelectedItem 方法获得选中的条目。

Calendar：日历组件，为用户提供日期选择功能。提供 getDate 方法获取选中的日期。

ProgressBar：进度条组件，提供进度显示功能。提供 setPrecent 方法改变进度条显示进度百分比。

3.5 客户端设计

3.5.1 事务管理模块设计

事务管理模块包括事项管理，分组管理，标签管理，归档管理四个子模块。所有的操作均由用户与用户界面的交互开始，用户执行某项操作后用户界面会发出相应的事件，管理界面的 Mediator 侦听事件后将需要的操作封装为 Notification 发送给系统，系统根据 Notification 和 Command 的映射执行相应的 Command，Command 最终对本地数据代理 ContextProxy 和 Mediator 执行最终的操作。

事项管理模块中主要包含新建事项，修改事项，删除事项，完成事项，转移事项等操作。标签管理模块中主要包含新建标签，修改标签，删除标签等操作。分组管理模块中主要包含新建分组，修改分组，删除分组，完成分组等操作。归档管理模块主要包括归档事项和查看归档等操作。每个操作都有相应的 Command 类调用 ContexyProxy 执行完成。

事务管理主要通知设计：

表 3.17 事务管理主要通知设计

事项管理通知	
NEW_EVENT	新建事项通知，由 NewEventDialogMediator 类发出，NewEventCommand 类执行
MODIFY_EVENT	修改事项通知，由 InboxPanelMediator 类发出，ModifyEventCommand 类执行
TRANSPORT_EVENT	转移事项通知，由 InboxPanelMediator 类发出，TransportEventCommand 类执行
COMPLETE_EVENT	完成事项通知，由 InboxPanelMediator 类发出，CompleteEventCommand 类执行
DELETE_EVENT	删除事项通知，由 InboxPanelMediator 类发出，DeleteEventCommand 类执行

标签管理通知	
NEW_TAG	新建事项通知，由 TagManagementMediator 类发出，NewTagCommand 类执行
MODIFY_TAG	修改事项通知，由 TagManagementMediator 类发出，ModifyTagCommand 类执行
DELETE_TAG	删除事项通知，由 TagManagementMediator 类发出，DeleteTagCommand 类执行
分组管理通知	
NEW_GROUP	新建分组通知，由 ProjectsPanelMediator 类发出，NewGroupCommand 类执行
MODIFY_ GROUP	修改分组通知，由 ProjectsPanelMediator 类发出，ModifyGroupCommand 类执行
DELETE_ GROUP	删除分组通知，由 ProjectsPanelMediator 类发出，DeleteGroupCommand 类执行
COMPLETE_ GROUP	删除分组通知，由 ProjectsPanelMediator 类发出，DeleteGroupCommand 类执行
归档管理通知	
ACHIEVE_EVENT	归档通知，由 CompletePanelMediator 类发出，ArchieveEventCommand 类执行

事务管理主要类说明

View：

InboxPanelMediator，操纵收集箱列表的中介者类，处理 InboxPanel 抛出的新建事项，修改事项，完成事项，删除事项，转移事项等事件并更改 InboxPanel 中的显示。

TodayPanelMediator，操纵今日待办列表的中介者类，处理 TodayPanel 抛出的修改事项，完成事项，删除事项，转移事项等事件并更改 TodayPanel 中的显示。

NextActionPanelMediator，操纵行动列表的中介者类，处理 NextActionPanel 抛出的修改事项，完成事项，删除事项，转移事项等事件并更改 NextActionPanel 中的显示。

ScheduledPanelMediator，操纵日程列表的中介者类，处理 ScheduledPanel 抛出的修改事项，完成事项，删除事项，转移事项等事件并更改 ScheduledPanel 中的显示。

SomedayPanelMediator，操纵择日待办列表的中介者类，处理 SomedayPanel 抛出的修改事项，完成事项，删除事项，转移事项等事件并更改 SomedayPanel 中的显示。

SubProjectPanelMediator，操纵分组事项列表的中介者类，处理 SubProjectPanel 抛出的修改事项，完成事项，删除事项，转移事项等事件并更改 SubProjectPanel 中的显示。

CompletedPanelMediator，操纵已完成事项列表的中介者类，处理 CompletedPanel 抛出的修改事项，删除事项，转移事项等事件并更改 CompletedPanel 中的显示。

TrashPanelMediator，操纵垃圾箱列表的中介者类，处理 TrashPanel 抛出的修改事项，完成事项，删除事项，转移事项等事件并更改 TrashPanel 中的显示。

NewEventDialogMediator，新建面板的中介者类，处理新建版面抛出的新建事项事件并封装为 Notification 交由系统处理。

TagManagementMediator，操纵标签管理列表和新建标签面板的中介者类， 处理标

签管理列表和新建标签面板抛出的新建标签，修改标签和删除标签的事件，并更改分组列表的显示。

ProjectPanelMediator，操纵分组列表的中介者类，处理分组列表抛出的新建分组，修改分组，删除分组，完成分组等事件，并更改分组列表的显示。

AchieveFrameMediator，操纵已归档面板的中介者类，处理归档面板抛出的刷新，搜索等事件并更改已归档面板的显示。

Controller：

NewEventCommand，新建事项命令，通过调用 Model 新建一个待办事项。

NewRepeatEventCommand，新建重复事项命令，通过调用 Model 新建一个重复事项。

ModifyEventCommand，修改事项命令，通过调用 Model 修改指定的待办事项。

ModifyRepeatEventCommand，修改重复事项命令，通过调用 Model 修改指定的重复事项。

CompleteEventCommand，完成事项命令，通过调用 Model 将指定的事项移至完成列表。

TransportEventCommand，转移事项命令，通过调用 Model 将指定的事项移动到指定的列表。

DeleteEventCommand，删除事项命令，通过调用 Model 将指定的事项移动到垃圾箱列表。

DeleteRepeatCommand，删除重复事项命令，通过调用 Model 将指定的重复事项移动到垃圾箱列表。

NewTagCommand 新建分组命令，通过调用 Model 新建一个标签。

ModifyTagCommand 修改分组命令，通过调用 Model 修改指定的标签。

DeleteTagCommand 删除分组命令，通过调用 Model 删除指定的标签及有此标签中

NewGroupCommand 新建分组命令，通过调用 Model 新建一个分组。

ModifyGroupCommand 修改分组命令，通过调用 Model 修改指定的分组。

DeleteGroupCommand 删除分组命令，通过调用 Model 删除指定的分组及分组内的。

CompleteGroupCommand 完成分组命令，通过调用 Model 完成分组内的所有待办事项。

AchieveEventCommand，归档事项命令，通过调用 Model 将指定事项进行归档。

OpenAchieveFrameCommand，打开归档面板的命令。

Model：

ContexyProxy，本地数据代理，提供 newEvent，newRepeatEvent ，modifyEvent，ModifyRepeatEvent，completeEvent，transportEvent，deleteEvent，newTag，modifyTag，deleteTag，newGroup，modifyGroup， completeGroup， deleteGroup，achieveEvent 方法供 Command 调用。

事务管理主要顺序图：

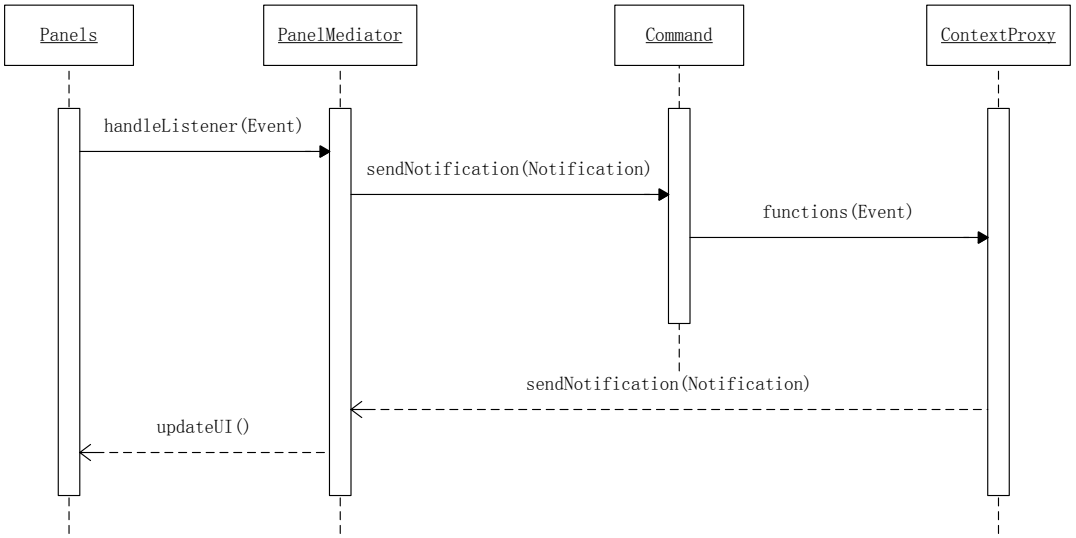

图 3.7 事务管理主要顺序图

3.5.2 用户管理模块设计

用户管理模块包括用户注册，用户登录，找回密码，修改密码，语言设置等操作。所有的操作均由用户与用户界面的交互开始，用户执行某项操作后用户界面会发出相应的事件，管理界面的 Mediator 侦听事件后将需要的操作封装为 Notification 发送给系统，系统根据 Notification 和 Command 的映射执行相应的 Command，Command 通过调用服务器数据代理 HttpServerProxy 与服务器进行交互，服务器根据接收到的客户端请求调用相应的 Servlet，Servlet 通过业务逻辑层对数据的处理返回 XML 文档给客户端，客户端根据服务器返回的 XML 生成相应的 Notification 由系统进行处理。

用户管理主要通知设计：

表 3.18 用户管理主要通知设计

新用户注册通知	
REGISTER_CHECK	检查是否已被注册通知，由 RegisterMediator 类发出，RegisterCheckCommand 类执行
REGISTER	新用户注册通知，由 RegisterMediator 类发出，RegisterCommand 类执行
REISTER_RESULT	注册结果通知，由 HttpServerProxy 类发出，RegisterMediator 处理
用户登录通知	
LOGIN	用户登录通知，由 LoginPanelMediator 类发出，LoginCommand 类执行
LOGIN_RESULT	用户登录结果，由 HttpServerProxy 类发出，LoginPanelMedaitor 处理

找回密码通知	
FIND_PASSWORD	找回密码通知，由 FindPWDPanelMediator 类发出，FindPWDCommand 类执行
SEND_IDENTIFYING_CODE	发送验证码通知，由 FindPWDPanelMediator 类发出，SendIdentifyingCommand 类执行
修改密码通知	
SET_PASSWORD	设置新密码通知，由 ConfigPanelMediator 类发出，SetPasswordCommand 类执行
更改语言通知	
CHANGE_LANGUAGE	由 LoginPanelMedaitor 发出，ChangeLanguageCommand 类执行

用户管理主要类说明：

View：

LoginFrameMediator：操作用户登录的中介者类，处理用户登录，注册面板打开，语言切换，找回密码面板打开及更新注册面板的显示。

RegisterFrameMediator：操作用户注册的中介者类。处理用户邮箱验证，用户注册以及用户注册面板的更新显示。

RegisterSuccessMediator：操作用户注册成功的中介者类。处理重发密码及注册成功面板更新显示。

FindPWDStep1FrameMediator：操作找回密码的中介者类，处理用户邮箱验证和找回密码面板的更新显示。

FindPWDStep2FrameMediator：操作找回密码的中介者类，处理验证码验证、用户密码修改以及找回密码面板的更新显示。

Controller：

LoginCommand：用户登录命令。

RegisterCommand: 用户注册命令。

ValidateUserCommand：验证邮箱是否存在命令。

SendPasswordAgainCommand：重发密码命令。

SendIdentifyingCodeCommand：发送验证码命令。

SetPasswordCommand：重设密码命令。

ValidateUserCommand：验证邮箱是否存在命令。

Model：

HttpServerProxy：处理与服务器交互的数据代理。提供 login，isRegister，sendPasswordAgain，register，sendIdentifyingCode，changePWD 方法供 command 调用。

LanguageProxy：处理语言切换的数据代理。提供 loadLanguage 方法供 command 调用

用户管理主要顺序图：

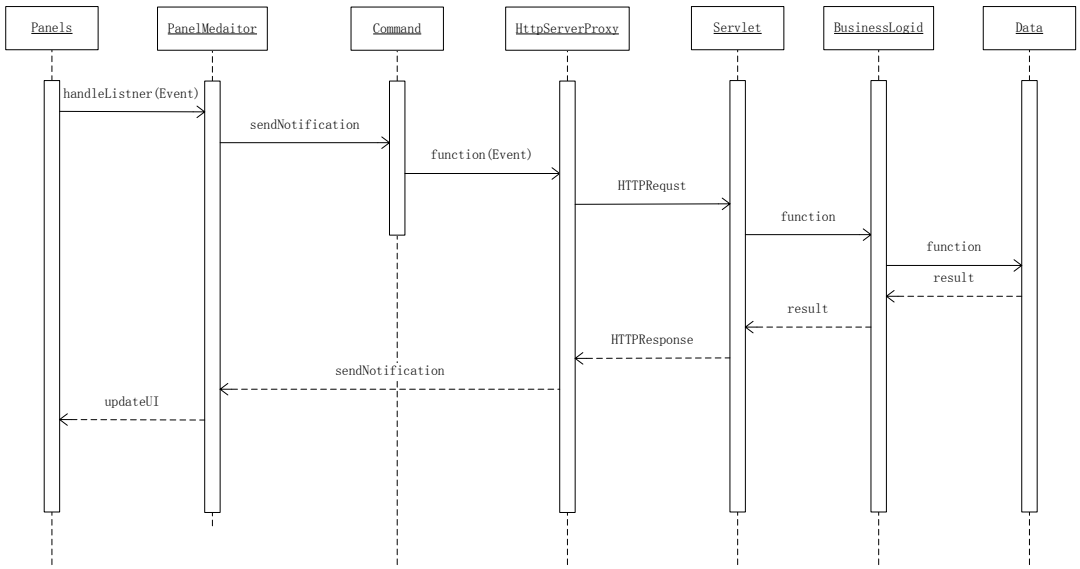

图 3.8 用户管理主要顺序图

3.5.3 联系人管理模块设计

联系人管理模块包括添加联系人，删除联系人，编辑联系人，事项转发等操作。所有的操作均由用户与用户界面的交互开始，用户执行某项操作后用户界面会发出相应的事件，管理界面的 Mediator 侦听事件后将需要的操作封装为 Notification 发送给系统，系统根据 Notification 和 Command 的映射执行相应的 Command，Command 通过调用服务器数据代理 HttpServerProxy 与服务器进行交互，服务器根据接收到的客户端请求调用相应的 Servlet，Servlet 通过调用业务逻辑层的方法与其他用户进行交互从而在用户之间建立相应的联系，最后通过同步操作向各用户的客户端同步数据，客户端根据服务器返回的 XML 生成相应的 Notification 由系统进行处理。

联系人管理主要通知设计：

表 3.19 联系人管理主要通知设计

ADD_FRIEND	添加联系人通知，由 AddFriendPanelMediator 类发出，AddFriendCommand 类执行
DELETE_FRIEND	删除联系人通知，由 FriendListMediator 类发出，DeleteFriendCommand 类执行
MODIFY_FRIEND	修改联系人信息通知，由 FriendPanelMediator 类发出，ModifyFriendMediator 类执行

联系人管理主要类说明：

View：

AddFriendPanelMediator：添加联系人面板中介者类。发送添加联系人通知。

FriendListMediator：联系人列表中介者类。发送删除联系人和打开联系人编辑面板的通知。

FriendPanelMediator：联系人面板中介者类，可以发送修改联系人信息的通知。

Controller：

AddFriendCommand:添加联系人命令。

DeleteFriendCommand:删除联系人命令。

ModifyFriendCommand:修改联系人信息命令。

Model：

ContexyProxy：本地数据代理。提供 addFriend，deleteFriend，modifyFriend 方法供 command 调用

HttpServerProxy：处理与服务器交互的数据代理。提供 sync 方法进行与服务器的同步，从而完成对联系人的操作和联系人之间事件转发的操作。

联系人管理主要顺序图：

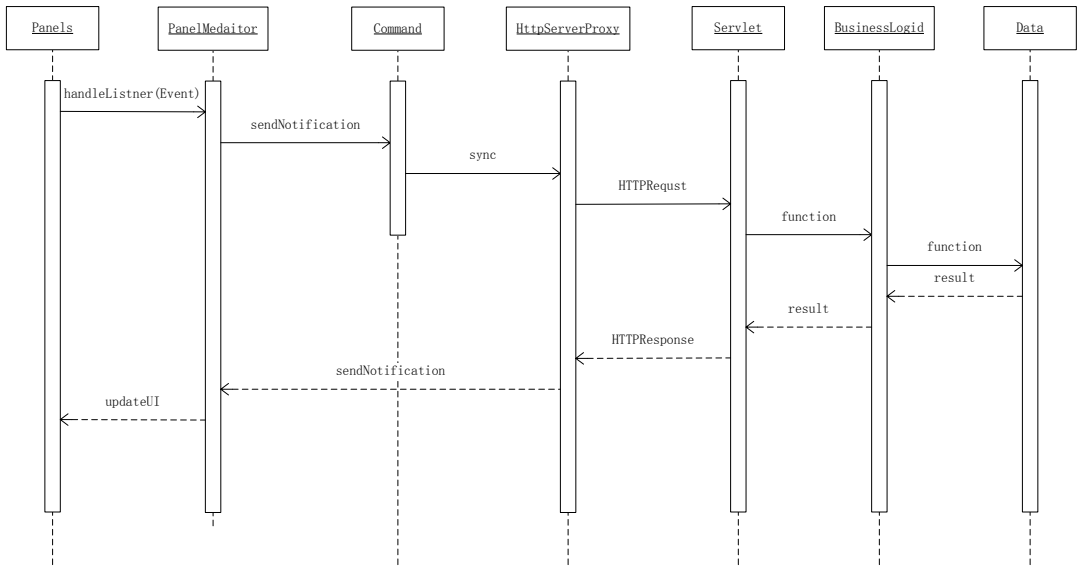

图 3.9 联系人管理主要顺序图

3.5.4 本地数据管理模块设计

在客户端，各数据采用 XML 文件进行存储，客户端与服务器端也通过 XML 文件进行数据的传递与同步。客户端存储的数据实体包括：事务、标签、分组、好友和重复事务。另外，加上在与服务器同步时需要更新文件，共需要六种数据实体。其中除了更新文件外，其他的五个数据类型均有相应的类与之对应。

主要 XML 文件及对应 Schema 文件描述：

事项 event.xml 文件：event.xml 存储了所有的事项，其 Schema 设置如图 3.10 所示：event_list 为 root 节点，下面包含了所有的事务，每个事务表示为一个 event 节点，如果该事务没有标签信息，则 event 节点的子节点 tag_list 为空，如有标签，则所有的标签 id 均会罗列在 tag_list 子节点下。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="event_list">
    <xs:complexType>
      <xs:element name="event" type="eventType" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="eventType">
    <xs:attribute name="id" type="xs:string"/>
    <xs:attribute name="status" type="xs:string" default=""/>
    <xs:attribute name="name" type="xs:string" default=""/>
    <xs:attribute name="create_date" type="xs:date" default=""/>
    <xs:attribute name="plan_date" type="xs:date" default=""/>
    <xs:attribute name="finish_date" type="xs:date" default=""/>
    <xs:attribute name="deadline" type="xs:date" default=""/>
    <xs:attribute name="group_id" type="xs:string" default=""/>
    <xs:attribute name="forward_to" type="xs:string" default=""/>
    <xs:attribute name="forward_from" type="xs:string" default=""/>
    <xs:attribute name="remind_time" type="xs:string" default=""/>
    <xs:attribute name="location" type="xs:string" default=""/>
    <xs:attribute name="note" type="xs:string" default=""/>
    <xs:attribute name="is_profiled" type="xs:string" default="false"/>
    <xs:element name="tag_list">
      <xs:complexType>
        <xs:element name="tag" type="xs:string" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:complexType>
    </xs:element>
  </xs:complexType>
</xs:schema>
```

图 3.10：event.xml 文件 Schema

标签 tag.xml 文件：tag.xml 文件存储了所有的标签对象，标签数据只含有 id、name、status 这三个属性，其 Schema 设置如图 3.11 所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="tag_list">
    <xs:complexType>
      <xs:element name="tag" type="tagType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="tagType">
    <xs:attribute name="id" type="xs:string"/>
    <xs:attribute name="status" type="xs:string" default=""/>
    <xs:attribute name="name" type="xs:string" default=""/>
  </xs:complexType>
</xs:schema>
```

图 3.11：tag.xml 文件 Schema

分组 group.xml 文件：group.xml 文件存储了所有的分组信息，其 Schema 设置如图 3.12 所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="group_list">
        <xs:complexType>
            <xs:element name="group" type="groupType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="groupType">
        <xs:attribute name="id" type="xs:string"/>
        <xs:attribute name="status" type="xs:string" default=""/>
        <xs:attribute name="name" type="xs:string" default=""/>
        <xs:attribute name="is_finish" type="xs:string" default="false"/>
    </xs:complexType>
</xs:schema>
```

图 3.12：group.xml 文件 Schema

重复事项 repeatEvent.xml 文件：repeatEvent.xml 文件存储了所有重复事项对象，其 Schema 设置如图 3.13 所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="repeat_event_list">
        <xs:complexType>
            <xs:element name="repeat_event" type="repeatEventType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="repeatEventType">
        <xs:attribute name="id" type="xs:string"/>
        <xs:attribute name="status" type="xs:string" default=""/>
        <xs:attribute name="name" type="xs:string" default=""/>
        <xs:attribute name="group_id" type="xs:string" default=""/>
        <xs:attribute name="create_date" type="xs:date" default=""/>
        <xs:attribute name="note" type="xs:string" default=""/>
        <xs:attribute name="last_repeat_date" type="xs:date" default=""/>
        <xs:attribute name="end_date" type="xs:date" default=""/>
        <xs:attribute name="repeat_times" type="xs:integer" default=""/>
        <xs:element name="tag_list">
            <xs:complexType>
                <xs:element name="tag" type="xs:string" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="repeat" type="repeatType"/>
    </xs:complexType>
    <xs:complexType name="repeatType">
        <xs:attribute name="type" type="xs:string"/>
        <xs:element name="start_time" type="xs:string"/>
        <xs:element name="range">
        <xs:element name="endMode">
            <xs:complexType>
                <xs:element name="arg" type="argType" maxOccurs="2"/>
            </xs:complexType>
        </xs:element>
    </xs:complexType>
    <xs:complexType name="argType">
        <xs:element name="name" type="xs:string"/>
        <xs:element name="value" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

图 3.13：repeatEvent.xml 文件 Schema

联系人 friend.xml 文件：friend.xml 文件存储了所有联系人对象。包含 email、status、name、type 及 tele 属性，其 Schema 设置如图 3.14 所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="friend_list">
        <xs:complexType>
            <xs:element name="friend" type="friendType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="friendType">
        <xs:attribute name="email" type="xs:string"/>
        <xs:attribute name="status" type="xs:string" default=""/>
        <xs:attribute name="name" type="xs:string" default=""/>
        <xs:attribute name="type" type="xs:string" default=""/>
        <xs:attribute name="tele" type="xs:integer" default=""/>
    </xs:complexType>
</xs:schema>
```

图 3.14：friend.xml 文件 Schema

更新文件 update.xml 文件：update.xml 文件存储了同步操作时所需要的信息。每隔一段时间，客户端需要提取各个文件中修改过的部分上传给服务器。而上文各数据 status 属性即是用来标记这条记录的状态。更新文件即是将自从上次更新一来改变过的数据写入，上传给服务器端。由于这里我们有五种数据实体，则在更新文件中分别用五个列表存储，其 Schema 设置如图 3.15 所示：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="client">
        <xs:complexType>
            <xs:attribute name="email" type="xs:string"/>
            <xs:element name="event_list">
                <xs:complexType>
                    <xs:element name="event" type="eventType" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xs:complexType>
            <xs:element>
            <xs:element name="tag_list">
                <xs:complexType>
                    <xs:element name="tag" type="tagType" minOccurs="0"
                      maxOccurs="unbounded"/>
                </xs:complexType>
            <xs:element>
            <xs:element name="group_list">
                <xs:complexType>
                    <xs:element name="group" type="groupType" minOccurs="0"
                      maxOccurs="unbounded"/>
                </xs:complexType>
            <xs:element>
            <xs:element name="friend_list">
                <xs:complexType>
                    <xs:element name="friend" type="friendType" minOccurs="0"
                      maxOccurs="unbounded"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="repeat_event_list">
                <xs:complexType>
                    <xs:element name="repeat_event" type="repeatEventType" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xs:complexType>
            <xs:element>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

图 3.15：update.xml 文件 Schema

3.6 服务器端设计

3.6.1 通信层设计

客户端和服务器端采用 HttpRequest 和 HttpResponse 进行通行，在服务器的通信层中，系统需要使用 HttpServletRequest 接口来进行对 Http 请求的解析，同时使用 HttpServletResponse 对 Http 响应进行封装。

3.6.2 控制层设计

主要 Servlet 说明：

ChangePWDServlet：ChangePWDServlet 负责接受用户的修改密码请求。通过将请求中的头进行解析，需要得到用户的注册邮箱和新密码。将这两个信息作为参数，调用逻辑层的 changePWD 方法来改变用户的密码。最后将这个方法的返回值写到响应实体里面返回给客户端。

ForwardIDServlet：ForwardIdSerlvet 负责生成转发事项的标识符。通过调用逻辑层的 getForwardID 方法得到一个新的转发任务的标识符。然后将该标识符写入到响应实体里返回给客户端。

MailServlet：MailServlet 负责发送验证码邮件。通过接受客户端的请求对指定的邮箱地址发送邮件。

RegisterServlet：RegisterServlet 负责新用户注册。通过接受客户端的注册请求进行解析，通过调用逻辑层的 register 方法注册新用户。最后将返回值写入到响应实体里面返回给客户端。

ValidateUserServlet：ValidateUserServlet 负责检验邮箱是否已经注册。通过接受请求中的头进行解析，获取邮箱地址。然后调用逻辑层的方法检验邮箱是否已经被注册。最后将返回值写入到响应实体里面返回给客户端。

SyncServlet：SyncServlet 负责客户端和服务器端数据的同步。首先它解析同步请求中的请求实体，里面包含客户端的更新数据。然后调用逻辑层中的 sync 方法将客户端和服务器端的 XML 数据进行同步。最后将同步的结果写回到响应实体中返回给客户端。

3.6.3 逻辑层设计

逻辑层包含两个子模块：用户账户管理模块和数据同步模块。用户账户管理模块负责提供用户验证，用户注册，密码修改和登录功能。数据同步模块负责提供数据同步功能。

用户账户管理设计：

用户帐户数据管理主要处理数据对象为 User.xml。由 UserInfo 提供具体的操作接口。UserInfo 及相关类的类图如图 3.16 所示。UserInfo 的具体方法描述如下：

getUserDoc()：用来获得解析过 User.xml 的 DOM。

AddSyncTimes()：用来将更改用户的同步次数。当用户同步次数到达某个关键值时，将该用户升级为高级用户，并把其 GTD 数据转移到高级用户的数据存储文件中。

changePWD()：用来更改用户的密码，为密码修改功能提供服务。

changeName()：用来修改用户昵称。

addUser()：用来添加新用户资料。当用户资料已存在时，则不作为。这个方法为用户注册提供服务。

findUser()：用来查找用户资料。当用户不存在是，返回未被 UserInfo 设置的 User 对象。该方法为用户验证，用户登录提供服务。

getUserFileName()：用来查询指定用户数据的存储位置。

setUserFileName()：用来修改指定用户数据的存储位置。

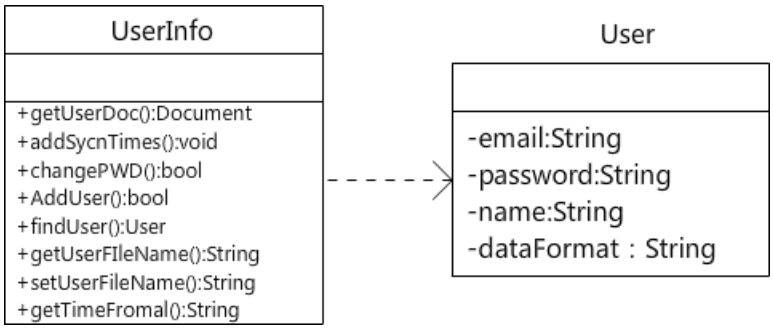

图 3.16 UserInfo 及相关类图

用户账户管理主要功能顺序图：

用户验证：

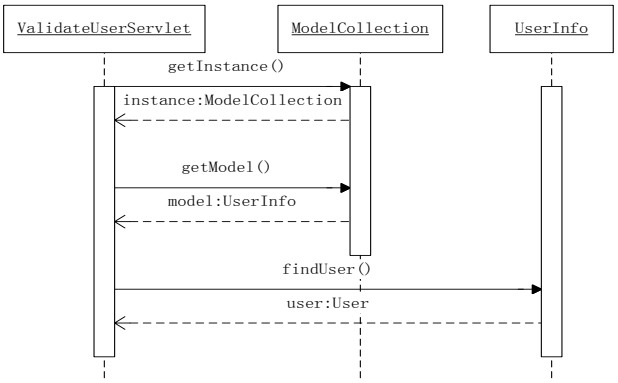

图 3.17 用户验证顺序图

修改密码：

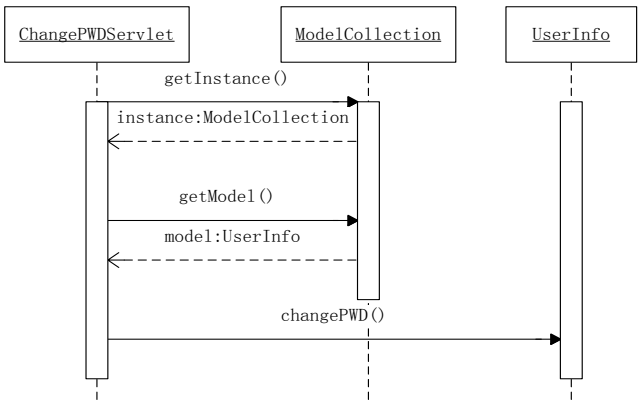

图 3.18 修改密码顺序图

用户注册：

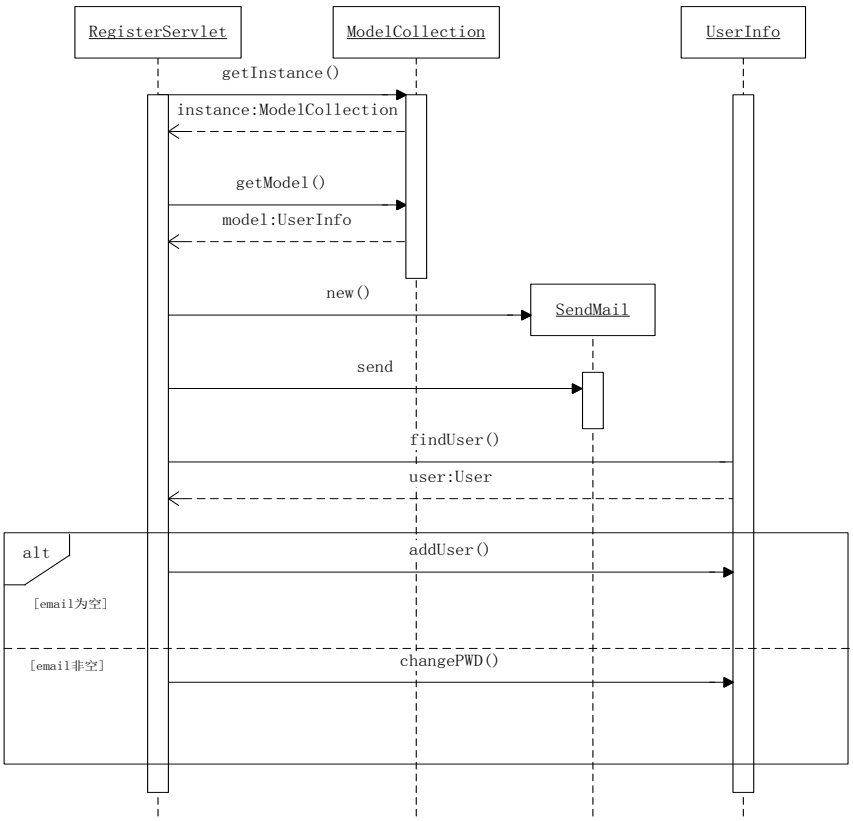

图 3.19 用户注册顺序图

用户登录：

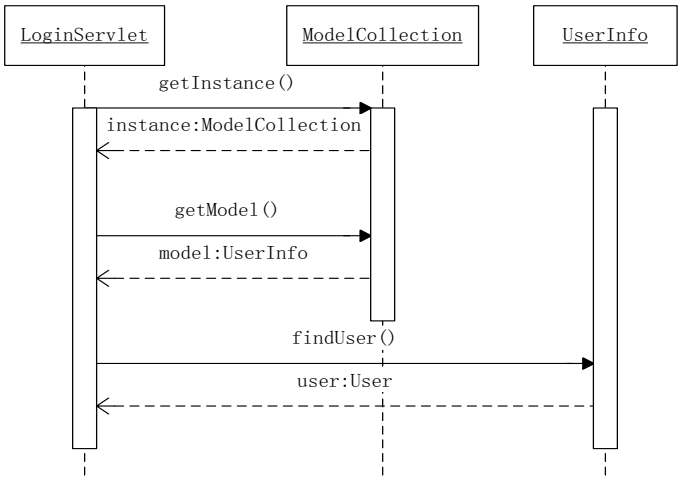

图 3.20 用户登录顺序图

数据同步模块的设计：

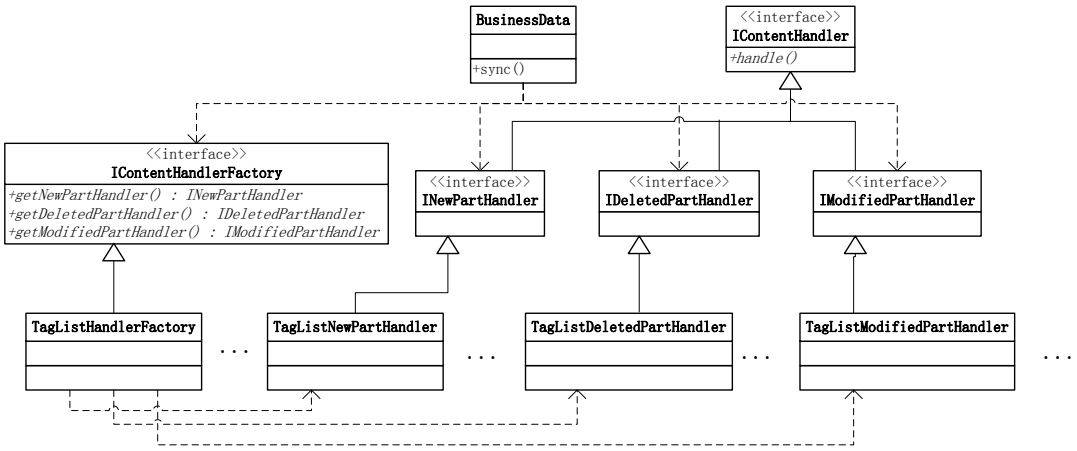

图 3.21 数据同步模块相关类图

数据同步模块主要操作的数据对象为 BusinessdataX.xml 系列文件。它提供数据同步功能，并在同步过程中实现好友关联，事项转发和消息提醒。由 BusinessData 类提供具体的数据同步接口。

同步消息包括 5 个部分：标签列表（tag_list），分组列表（group_list），好友列表（friend_list），重复事件列表（repeat_event_list）和事件列表（event_list）。每个部分存在若干同步项。同步项共 3 种类型：新增的（status 属性为 new），修改的（status 属性为 modified）和删除的（status 属性为 deleted）。因此，共有 15 种同步项。

该模块设计这里采用工厂模式，如图 3.21 所示。BusinessData 通过 sync()方法向控制层提供数据同步的接口。数据同步中根据每个同步项找到对应的工厂（TagListHandlerFactory，GroupListHandlerFactory ， FriendListHandlerFactory ， RepeatEventListHandlerFactory ，EventListHandlerFactory ， NullHandlerFactory ），然后用工厂生成对应的 handler（newPartHandler，modifiedPartHandler，deletedPartHandler）。这里 NullHandlerFactory 用来处理未定义的同步项，处理方法为空，提高程序的健壮性。

此外，为了保证分布式环境下，多客户端操作一个账户的数据一致性，同步消息中还必须包括用于恢复现场的数据。现场恢复主要包括 tag 和 group 两部分。当 event 或 repeat_event 引用了 tag 和 group 时，同步时还需上传相关 tag 和 group 的数据，并且其 status 属性为空。

在数据的同步更新的同时，数据同步模块还要负责好友关联，事项转发和消息提醒。数据同步的顺序图如图 3. 22 所示。

好友关联

好友根据 friend 的 type 属性分为两种：

1. not_reply，新加但没得到对方确认的好友。
2. friend，添加了并且也得到了对方确认的好友。

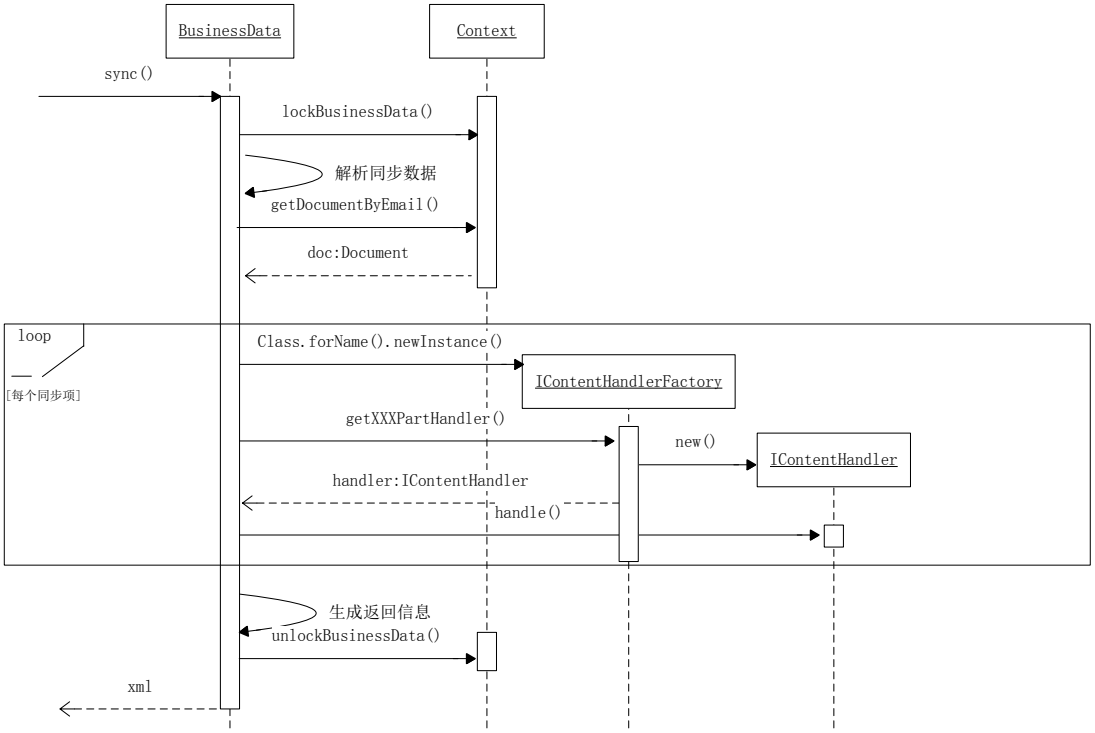

图 3.22 数据同步的顺序图

当收到状态为 new 的 friend 同步项时，系统将在两边都创建新的 type 属性为 not_reply 的 friend 项，并且在被邀请方处生成朋友邀请的 message 项。

当收到状态为 modified 的 friend 同步项时，系统将比较修改后的 friend 的 type 属性和原来的 friend 的 type 属性。如果原来的为 not_reply 而现在的为 friend，则表示更新方同意将更新 friend 项指向的用户加为好友。系统将在把 friend 项指向的用户下对应的 friend 项的 type 属性设为 friend，并生成对应的 message 项。

当收到状态为 deleted 的 friend 同步项，系统会删除用户数据中的 friend 项。并且一并删除其指向的用户相关的 friend 项，生成对应的 message 项。

事项转发

当收到状态为 new 的 event 同步项时，系统将检查其 forward_to 属性。如果非空，则表示该事件需要转发。系统会在 forward_to 指向的用户下生成对应的含 ref 属性的新的 event 项，并生成对应的 message 项。

当收到状态为 modified 的 event 同步项时，系统会比较修改项和原有项的 forward_to 属性。如果二者不同且修改项非空，则系统会在 forward_to 指向的用户下生成对应的含 ref 属性的新的 event 项，并生成对应的 message 项。如果二者不同且原有项非空，则系统会删除原有的 forward_to 属性指向的用户对应的 event 项，并生成对应的 message 项。

除此之外，系统还要根据 event 的完成时间，位置来判断 event 的状态是否更改。如果 event 被完成了，被清除了或被取消了，系统将根据更新后的 forward_to 和 forward_from 属性级联通知各个用户，在每个用户的 message_list 中生成对应的 message 项。

当收到状态为 deleted 的 event 同步项时，如果 event 项 forward_to 不为空，则系统将向下级联删除转发的 event 项，并生成对应的 message。如果 forward_from 不为空，则系统将向上级联删除转发的 event 项，并生成对应的 message。

消息提醒

消息提醒功能通过在 message_list 中添加和删除 message 项完成。当需要给某用户消息提醒时，系统会在其 message_list 中添加对应的 message 项。当该用户更新时，message 项将被作为返回数据的一部分返回给客户端，客户端通过解析返回的 XML 来生成对应的消息提醒。同时，服务器端的 message_list 将被清空。

3.6.4 数据层设计

用户 GTD 的 XML 数据存储在 BusinessDataX.xml（X=1，2…n）中。这些 xml 文件分为两类，一种是包含高级用户数据的文件，另一种是包含除高级用户以外的其他用户数据的文件。

高级用户是这么定义的：当用户数据同步次数达到某一个关键值时，该用户会被升级为高级用户。高级用户的数据将被转移到专门的文件中储存。关键值数据在配置文件中定义，高级用户数据转移在用户帐户管理模块中实现。

对于第一类文件，服务器系统启动时就会将其加载进内存，并让其常驻内存。除此之外，在系统关闭时，系统会对高级用户的文件做负载均衡，平衡每个文件的大小。这里的负载度量单位是文件所包含用户的个数。对于第二类文件，服务器系统启动时会选择固定数量的随机非高级用户文件加载进内存。当需要其他非高级用户文件时，系统才把其动态加载进内存。同时，系统会统计非高级用户文件的访问次数并排序。定时将在内存中访问次数较低的文件写回硬盘，扫出内存。

高级用户文件数量，被加载进内存的非高级文件数量，非高级文件数量都在配置文件中定义。

该模块功能主要由 3 个类实现：UserInfo，Context 和 ContextListener。

UserInfo 的 addSyncTimes()方法提供了当非高级用户同步次数达到要求后，将此用户 GTD 数据转移到随机的高级用户文件中的功能。

Context 具体设计如图 3.23 所示，其中关于 lock 和 unlock 的 4 个方法是为并发访问提供的。getDocumentByName()和 getDocumentByEmail()方法是获得指定 DOM 的接口。它主要有 3 个功能：1 在初始化时将高级用户文件和指定数量的随机非高级用户文件解析为 DOM，加载在内存中，放入 HashTable 中；2 动态加载没有加载到 HashTable 中的文件；3 统计非高级用户文件的使用频率，定时将超出数量的并且使用率低的 DOM 写回硬盘，移出 HashTable。

ContextListener 用来监听处理 tomcat 的 context 创建，结束事件。它的主要功能是在系统关闭时将所有的 DOM 写回硬盘，对高级用户文件进行负载均衡。

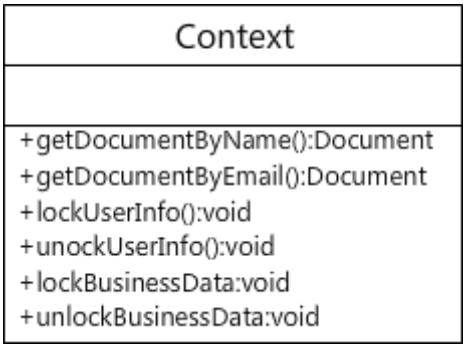

图 3.23Context 类图

配置信息存储在 config.xml 中。ConfigCenter 本身存在默认的配置信息，可以通过从解析 config.xml 更新自己的配置信息，然后在提供接口向外公布。ConfigCenter 的具体设计如图 3.24 所示。

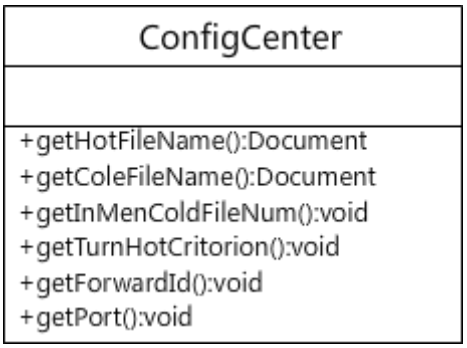

图 3.24ConfigCenter 类图

getHotFileNum()方法用来获得存储高级用户 GTD 数据的文件数量。比如，当文件数量为 2 时，BusinessData1.xml 和 BusinessData2.xml 为存储高级用户 GTD 数据的文件数量。该值默认为 5。

getColdFileNum()方法用来获得存储非高级用户 GTD 数据的文件数量。比如在上一种情况之上，非高级用户的文件数量为 2 时，BusinessData3.xml 和 BusinessData4.xml 就为存储非高级用户 GTD 数据的文件数量。该值默认为 15。

getInMemColdFileNum()方法用来获得常驻内存的非高级用户文件数量。默认值为 5。

getTurnHotCriterion()方法用来获得用户升级为高级用户的所需更新次数。默认值为 100。

getForwardID()方法用来提供新的 forward id（简称 FID），每次提供的 FID 均不同。由于存在转发事件，普通的 UUID（Universally Unique Identifier，通用唯一识别码）还不足以成为事件的主键。如 A 转发给 B 事件 1，而 B 又把事件 1 转发给 A。在这种情况中，简单的 UUID 不足以区别两个事件。因此系统特别定义了 FID。现在事件的 ID 可以表示为，UUID:FID:FID。一个 FID 对应一个转发关系，0 表示无转发关系。第一个 FID 指事件被转发过来的转发关系，第二个 FID 指事件转发过去的转发关系。在上面这种情况中，A 中事件 ID 为 UUID：0：1，B 中时间 ID 为 UUID:1:2，A 中被转发回来的时间 ID 为 UUID:2:0。为了保证 FID 不会重复，客户端都必须通过服务器端才能获得 FID，即调用该方法来获得。

第四章 系统实现

4.1 Sharp MVC 框架的实现

由于篇幅所限，只挑选 Sharp MVC 框架的核心部的实现进行说明。

Notification 类的实现：Notification 类是 Sharp MVC 框架中进行通信时传递的对象，它包含三个属性及属性的 get、set 方法

表 4.1 Notification 主要属性和方法说明

属性		
名称	类型	说明
name	String	通知对象的名称，是通知对象的唯一标识，其他的观察者通过此属性识别所关心的通知对象。
body	Object	通知对象在传递过程中所携带的数据体。
type	String	通知对象的类型

Notifier 类的实现：Notifier 类是所有通知者的父类，它实现 sendNotification 方法负责向注册的观察者对象发送 Notification 对象。

表 4.2 Notification 主要属性和方法说明

属性		
名称	类型	说明
facade	Facade	通知者对象保存对 Facade 对象的引用，通知者对象实际是通过 Facade 对象向观察者发送通知
方法		
名称	返回值	说明
sendNotification	void	向注册过的的观察者对象发送 Notification 对象

Observers 类的实现：Observers 类是针对特定通知的观察者列表类，它包含通知的名称和关心此通知的观察者的列表。

表 4.3 Observers 主要属性和方法说明

属性		
名称	类型	说明
note	String	通知的名称
observers	Vector	观察者列表
方法		
名称	返回值	说明
getObservers	Vector	返回观察者列表
addObserver	void	向列表添加一个观察者
deleteObserver	void	从列表删除指定的观察者
notifyObserver	void	通知所有观察者

Observer 类的实现：Observer 是具体的观察者类，他注册在 Observers 类的观察者列表中，实现 notifyObserver 方法以执行 Observers 发出的通知。

表 4.4 Observers 主要属性和方法说明

属性		
名称	类型	说明
context	Object	观察者的实体
notify	IFunction	观察者对象接收到 Notification 对象后的处理方法
方法		
名称	返回值	说明
notifyObserver	void	接到通知时执行的方法。

Facade 类的实现：Facade 类是一个单例模式类，是 Sharp MVC 框架的通信核心，负责 M，V，C 三部分的通信。

表 4.5 Facade 主要属性和方法说明

属性		
名称	类型	说明
facade	Facade	Facade 对象的引用
model	Model	Model 对象的引用
view	View	View 对象的引用
controller	Controller	Controller 对象的引用
方法		
名称	返回值	说明
registerProxy	void	向 Model 注册一个 Proxy 对象
removeProxy	void	从 Model 删除一个 Proxy 对象
retrieveProxy	IProxy	从 Model 获取一个 Proxy 对象
registerMediator	void	向 View 注册一个 Mediator 对象
removeMediator	void	从 View 删除一个 Mediator 对象
retrieveMediator	IMediator	从 View 获取一个 Mediator 对象
registerCommand	void	向 Controller 注册一个 Command
removeCommand	void	从 Controller 删除一个 Command
sendNotification	void	发送通知方法

Model 类的实现：Model 类是一个单例模式类，负责 Model 部分的操作，维持着所有 Proxy 对象的引用。

表 4.6 Model 主要属性和方法说明

属性		
名称	类型	说明
proxyMap	HashMap	所有 Proxy 对象的引用
view	View	View 对象的引用
方法		
名称	返回值	说明
registerProxy	void	向 Model 注册一个 Proxy 对象
removeProxy	void	从 Model 删除一个 Proxy 对象
retrieveProxy	IProxy	从 Model 获取一个 Proxy 对象

View 类的实现：View 类是一个单例模式类，负责 View 部分的操作，维持着所有 Mediator 对象的引用，同时维持着框架中的所有的观察者和通知的映射。

表 4.7 View 主要属性和方法说明

属性		
名称	类型	说明
mediatorMap	HashMap	所有 Mediator 对象的引用
observerMap	HashMap	所有 Observers 对象的引用
方法		
名称	返回值	说明
registerMediator	void	向 View 注册一个 Mediator 对象
removeMediator	void	从 View 删除一个 Mediator 对象
retrieveMediator	IMediator	从 View 获取一个 Mediator 对象
registerObservers	void	向系统注册一个通知和观察者列表的映射

View 中关于观察者管理的核心代码如下：

表 4.8 观察者管理核心代码

```
//注册观察者方法
public void registerObserver(String notificationName,
                             IObserver observer) {
       if (this.observerMap.get(notificationName) != null) {
          Observers observers =
               (Observers)this.observerMap.get(notificationName);
          observers.addObserver(observer);
       } else {
          this.observerMap.put(notificationName,
                          new Observers(notificationName, observer));
       }
    }
//通知观察者方法
public void notifyObservers(INotification note) {
       Observers observers =
          (Observers) this.observerMap.get(note.getName());
       if (observers != null) {
          observers.notifyObservers(note);
       }
    }
```

Controller 类的实现：Controller 类是一个单例模式类，负责 Model 部分的操作，维持着所有 Proxy 对象的引用。

表 4.9Controller 主要属性和方法说明

属性		
名称	类型	说明
commandMap	HashMap	所有 Command 对象的引用
view	View	View 对象的引用
方法		
名称	返回值	说明
registerCommand	void	向 Controller 注册一个 Command
removeCommand	void	从 Controller 删除一个 Command

4.2 Sharp UI 框架实现

由于篇幅所限，只挑选 Sharp UI 框架的事件机制的实现及 Calendar，Combox 组件的实现进行说明。

SharpEvent 类的实现：

表 4.10 SharpEvent 主要属性和方法说明

属性		
名称	类型	说明
name	String	事件的名称，用来辨别事件的表示
data	HashMap	事件传递时所携带的参数和数据
方法		
名称	返回值	说明
getDataByKey	Object	从数据中根据 Key 来取得相因数据的方法

SharpEventPanel 类的实现：

表 4.11 SharpEvenPanel 主要属性和方法说明

方法		
名称	返回值	说明
addSharpEventListener	void	添加 SharpEvent 事件侦听器方法
removeSharpEventListener	void	删除 SharpEvent 事件侦听器方法
notifySharpEventListener	void	通知所有侦听器方法。

SharpEventPanel 中关于事件机制的核心代码如下：

表 4.12 事件机制核心代码

```
Collection<SharpEventListener> listeners;
public void addSharpEvnetListener(SharpEventListener l){
    if(listeners == null)
        listeners = new CopyOnWriteArraySet<SharpEventListener>();
    listeners.add(l);
}
public void removeSharpEventListener(SharpEventListener l){
    if (listeners == null) {
        listeners = new CopyOnWriteArraySet<SharpEventListener>();
    }else{
        listeners.remove(l);
    }
}
public void notifySharpEventListener(SharpEvent e){
    if(listeners == null){
        listeners =new CopyOnWriteArraySet<SharpEventListener>();
    }else{
        Iterator<SharpEventListener> iterator = listeners.iterator();
        while (iterator.hasNext()) {
            SharpEventListener listener = iterator.next();
```

```
            listener.handleSharpEvent(e);
        }
    }
}
```

其中为了线程安全的问题，Collection 采用 CopyOnWriteArraySet 对象作为实例。

Calendar 组件的实现：

图 4.1Calendar 组件

ScrollPanel 组件的实现

图 4.2ScrollPanel 组件

ComboBox 组件的实现：

图 4.3ComboBox 组件的实现

4.3 客户端实现

4.3.1 事务管理模块实现

事务管理模块的界面实现：事务管理模块使用 Sharp UI 组件进行开发，包含的组件有 SharpEventPanel，SimpleButton，ImagePanel，ImageButton，CheckBox，Calendar ,GhostPanel，BlurPanel 等

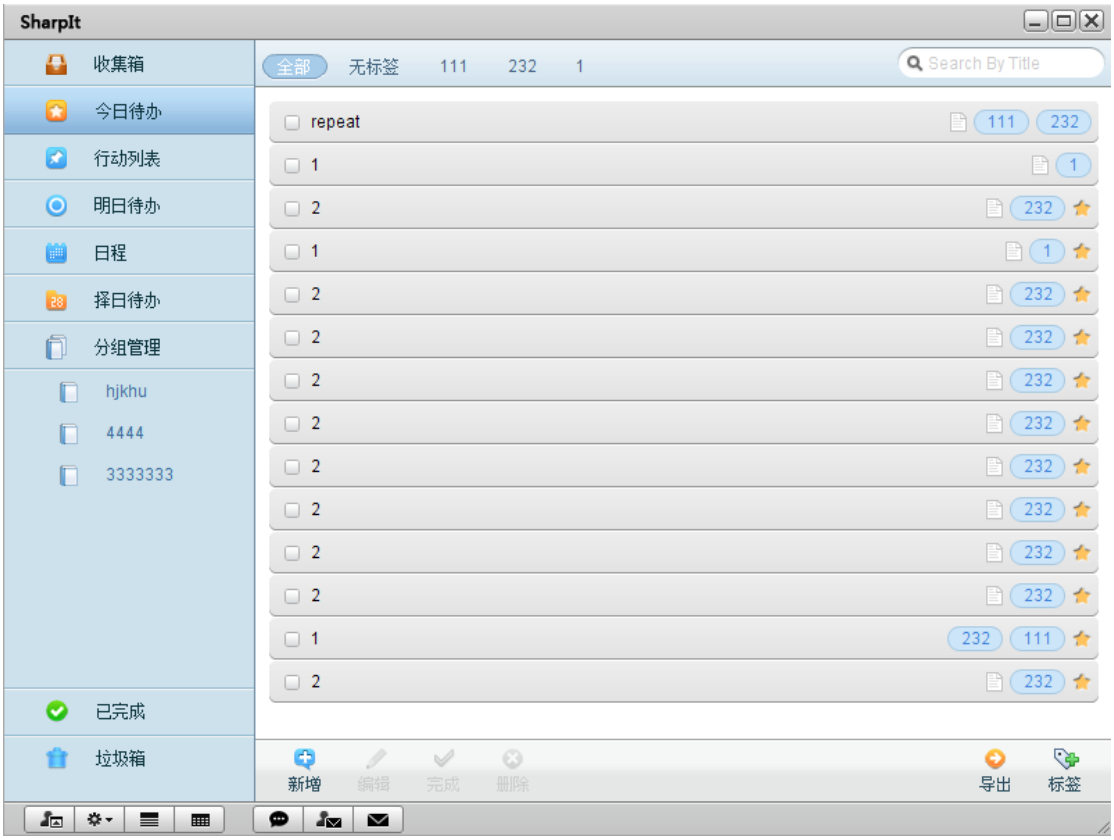

图 4.4 事项列表面板实现图

图 4.5 新建事项面板实现图

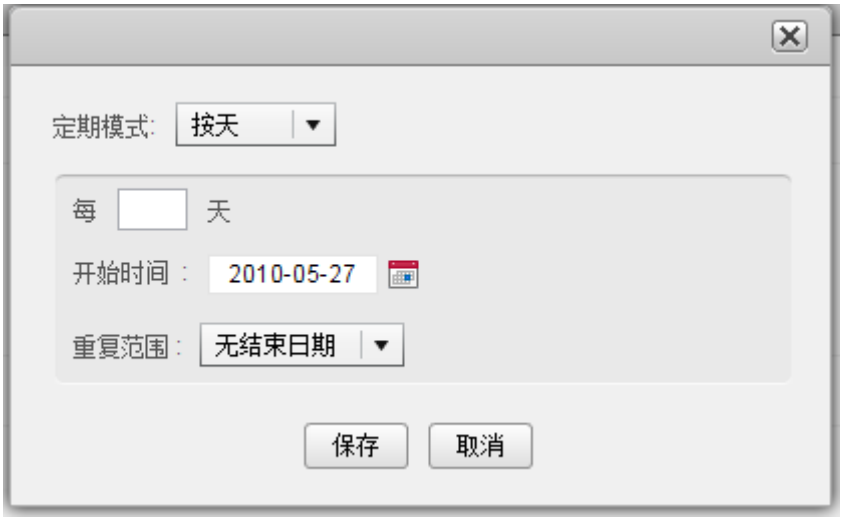

图 4.6 重复设置面板实现图

图 4.7 标签管理面板实现图

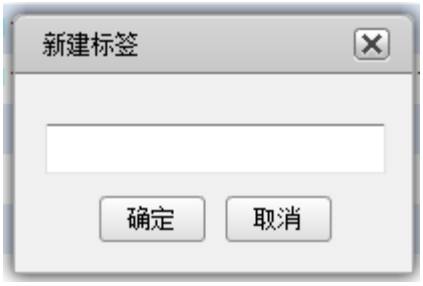

图 4.8 新建标签面板实现图

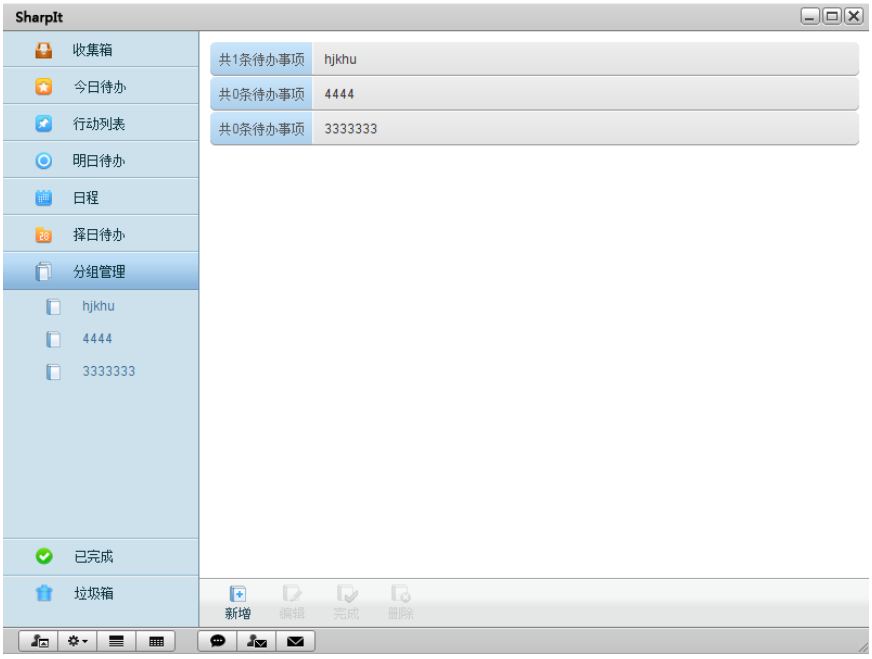

图 4.9 分组管理面板实现图

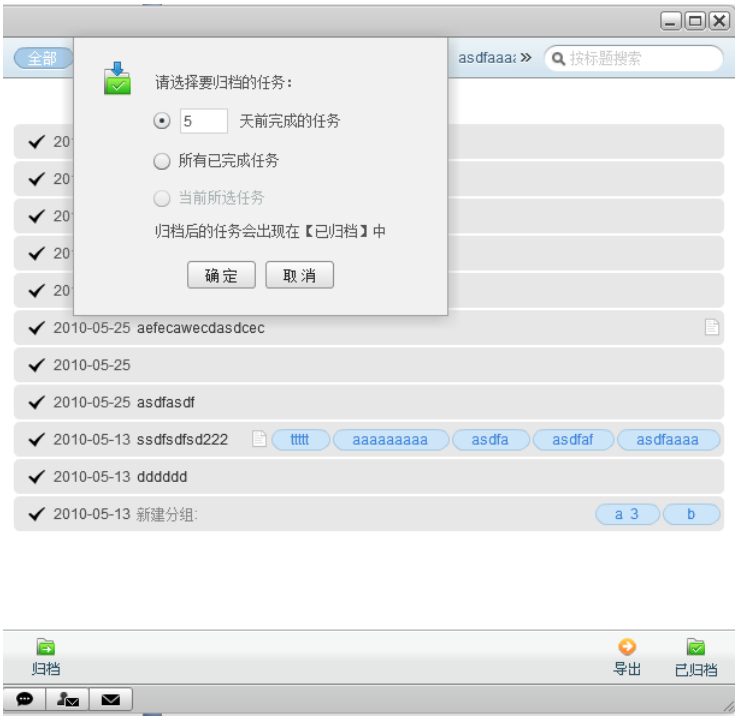

图 4.10 归档管理面板实现图

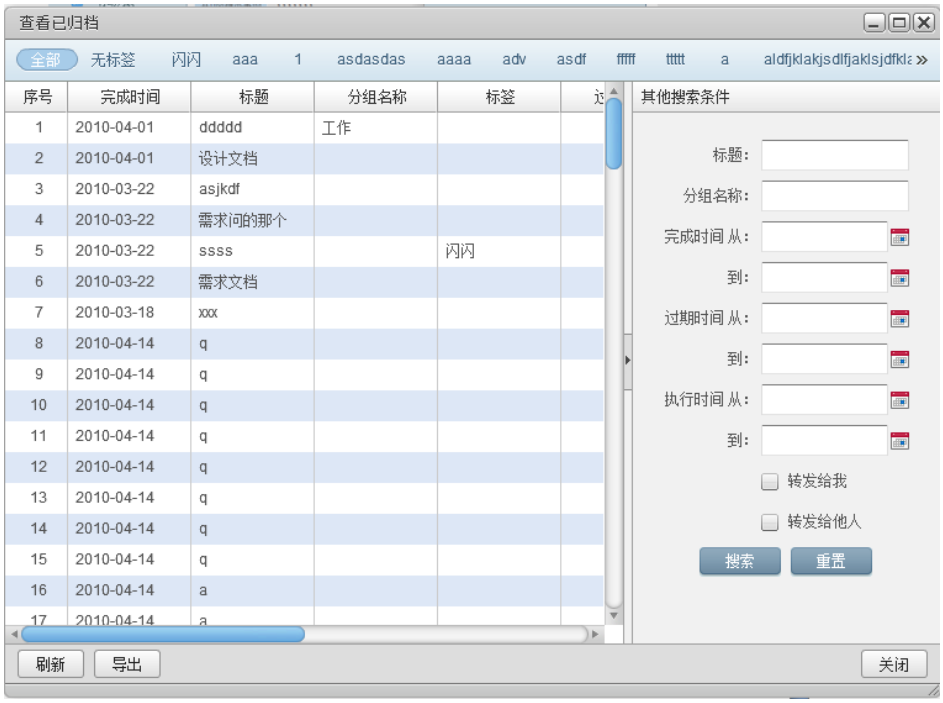

图 4.11 已归档面板实现图

4.3.2 用户管理模块实现

图 4.12 新用户注册面板实现图

图 4.13 登录面板实现图

图 4.14 密码找回面板实现图

4.3.3 联系人管理模块实现

图 4.15 添加联系人面板实现图

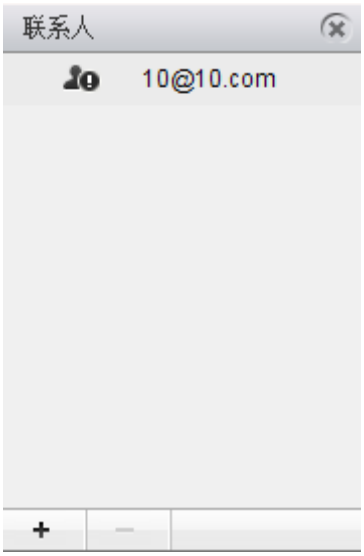

图 4.16 联系人列表面板实现图

4.3.4 本地数据管理模块实现

由于篇幅有限，本文只介绍系统最核心的 Event 类的相关实现，其他的对象，如 Tag、Group 等与 Event 类处理的流程大体相同，故不赘述。

新增事项:

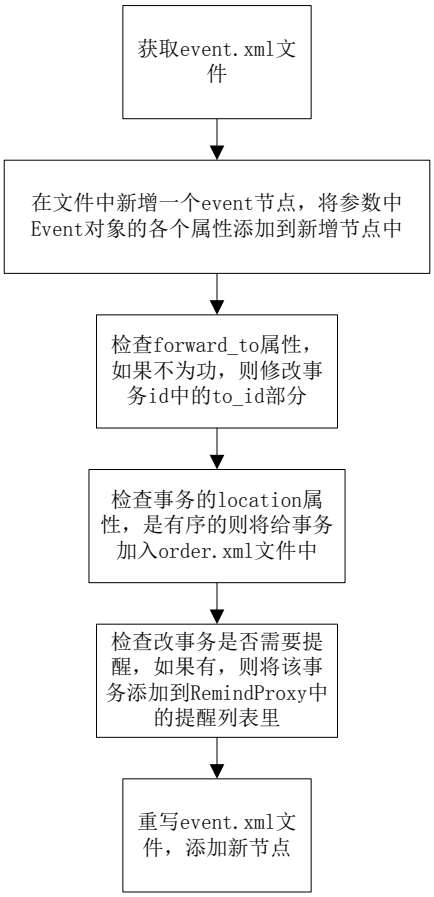

图 4.17 新增事项流程图

获取事项列表：

该方法声明为：public ArrayList<Event> getEventListByLocation（int location）其流程如图 4.18

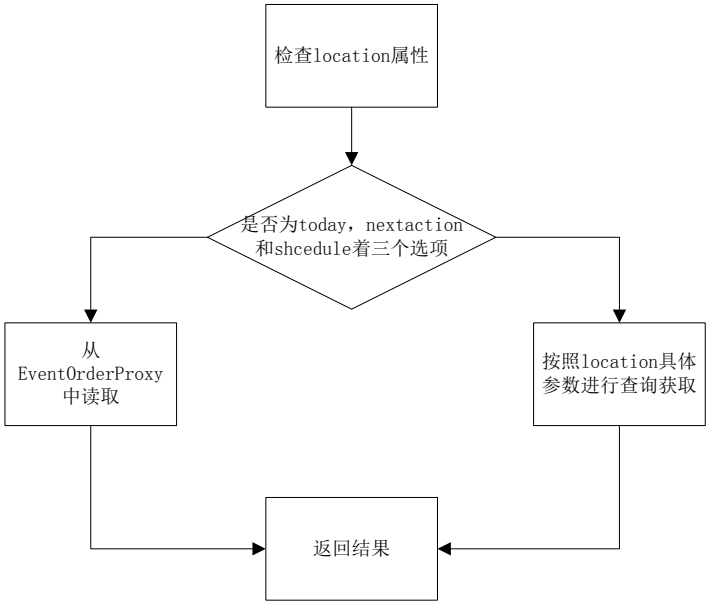

图 4.18：获取 Event 列表流程图

4.4 服务器端实现

选取服务器端较为复杂的同步操作在各层的处理来说明服务器端的实现：

4.4.1 通信层实现

同步操作在通信层中主要通过接受客户端发送来的请求以及封装在 HTTP 头中的更新数据进行操作。在处理完更新后将更新后的数据写在 HTTP 消息体重返回给客户端。

表 4.13 客户端发送更新 XML 文件

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<client email="8@8.com">
<event_list>
</event_list>
<tag_list>
</tag_list>
<group_list>
</group_list>
<friend_list>
</friend_list>
<repeat_event_list>
</repeat_event_list>
</client>
```

表 4.14 服务器返回 XML 文件

```
<?xml version="1.0" encoding="utf-8"?>
<client email="8@8.com">
<event_list>
<event create_time="2010-06-02 15:20" deadline=""
finish_date="2010-06-02" forward_from="" forward_to="" group_id=""
id="313c3988-e4e3-4092-ac1a-40751c742220:0:0" is_profile="true"
location="" name="1" note="" plan_date="" remind_time="" status=""/>
<event create_time="2010-06-02 15:20" deadline=""
finish_date="2010-06-02" forward_from="" forward_to="" group_id=""
id="3fefe4b7-e80e-4988-a515-fea708a4d8a4:0:0" is_profile="true"
location="" name="2" note="" plan_date="2010-06-02" remind_time=""
status=""/>
</repeat_event>
</repeat_event_list>
<friend_list>
<friend email="10@10.com" name="123" status="" tele="" type="friend"/>
</friend_list>
<group_list>
<group id="50f5791e-0374-497d-b5a6-dd6dd08acebc" is_finish="true"
name="g1444" status=""/>
<group id="b0b0c74f-e5c7-405e-ae74-96f3774ad659" is_finish="false"
name="g3" status=""/>
</group_list>
<tag_list>
<tag id="f80c571c-e3d1-4dcf-8d15-dce0e459925e" name="1" status=""/>
</tag_list>
<message_list>
</message_list>
</client>
```

4.4.2 控制层实现

同步操作在控制层的实现为 SyncServlet。SyncServlet 的主要方法是 service()方法来处理和返回 HTTP 请求。在 SyncServlet 中，service 方法首先从请求实体中检出客户端发送过来的 XML 数据。然后调用逻辑层的 businessData.sync()方法来进行服务器和客户端数据的同步。最后将同步的结果写回到响应实体中。

SyncServlet 的流程如图 4.19 所示。

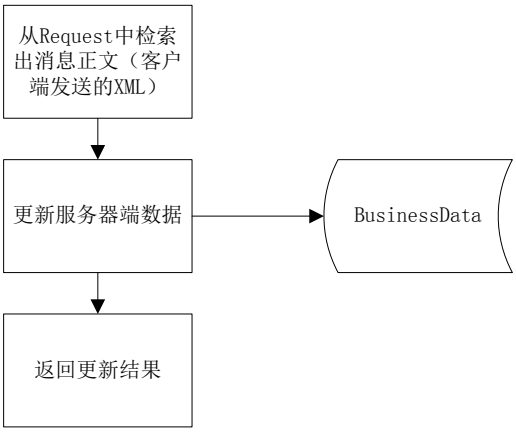

图 4.19 同步流程图

4.4.3 逻辑层实现

在逻辑层，当处理同步时，首先锁定用户数据。然后解析同步数据，找到每个同步项对应的 handler 对同步项进行处理。最后生成返回给用户的数据，解锁并将数据返回给客户端。

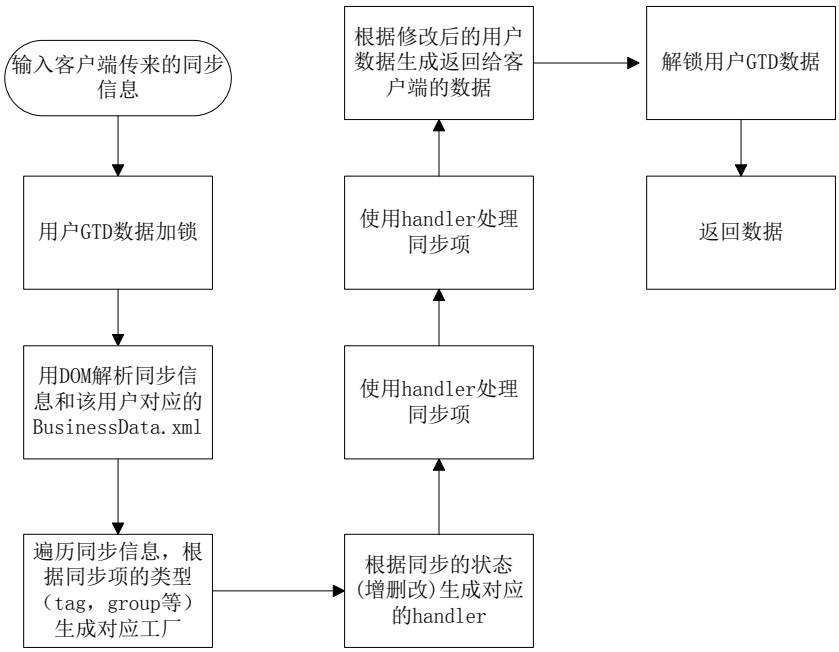

图 4.20 同步流程图

4.4.4 数据层实现

同步的数据层主要包含对用户数据加锁和解锁的操作。Semaphore 类根据互斥信号量原理来实现对资源的上锁。其代码如下：

表 4.15 锁操作实现代码

```
public class Semaphore {
    private volatile LinkedList<Object> waitingList =
                                            new LinkedList<Object>();
    private volatile int s = 1;
    public synchronized void before(){
        if(s > 0){
            s--;
            return;
        }
        else{
            Object o = new Object();
            synchronized(o){
                waitingList.add(o);
                try {
                    o.wait();
                    s--;
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
    public synchronized void after(){
        if(s == 0){
            s++;
            if(waitingList.size() > 0){
                Object o = waitingList.remove(0);
                synchronized(o){
                    o.notify();
                }
            }
        }
    }
}
```

其中 before()方法相当于 P 操作，after()方法相当于 V 操作。私有属性 s 相当于信号量，linked list 用来存放同步对象。

第五章 总结与展望

GTD 行为管理方法作为近几年来逐渐兴起的管理方法，正在被越来越多的人所接受并将其融入到自己的工作，学习和生活中。GTD 工具也逐渐成为人们管理日常事务的不可或缺的工具之一，GTD 工具的用户体验，响应速度，运行稳健性等因素都会影响到人们工作效率。所以一款设计优良，运行稳定，操作简单，随手可及的 GTD 工具必定会受到人们的青睐。

本文通过对 GTD 行为管理方法的主要原则和核心活动的研究，基于 Java 平台设计和实现了一款 GTD 个人事务管理工具。经过模拟测试，系统表现出了良好的管理能力，稳定的运行能力和简单的操作性，相信该系统可以成为 GTD 个人事务管理的利器。

但是由于时间的仓促和开发人员的不足，系统仍有许多值得改进的地方，在后续的开发中可以考虑如下几方面的改进和完善:

1) 增强与其他个人事务管理系统的集成，支持事务在不同系统间的导入和导出。

2) 增强多人协作功能，支持事项在多人间的跟踪处理。

3) 增加智能提示功能，通过分析用户的使用习惯和工作效率智能的安排用户的事务执行顺序。

4） 将系统移至到手机平台，以方便用户随时随地安排和管理事项。

参考文献

[1] Allen David. *getting things done*. Penguin Books.2001

[2]Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates. *Head First Design Pattern.* O'Reilly. 2007

[3]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns*. Addison-Wesley. 1995

[4]Java 简介. http://baike.baidu.com/view/29.htm

[5]Java EE 简介。 http://baike.baidu.com/view/1507.htm

[6]Bruce Eckel. Java *编程思想*. 第四版.机械工业出版.2007

[7] Daniel Liang. *Java 语言程序设计基础篇*. 第六版.机械工业出版. 2008

[8] 希仁编.计算机网络.电子工业出版社.2008

[9]XML 简介. http://baike.baidu.com/view/63.htm

[10]W3C.Document Object Model(DOM) Level 2 Specification Version 1.0.2000-11

[11]Grady Booch, James Rumbaugh, Ivar Jacobson. *UML 用户指南*.第二版.人民邮电出版社.2006

[12]Paul C.Jorgensen. *软件测试*. 第二版. 机械工业出版社. 2008

致谢

感谢贝佳老师给予的指导，正是贝佳老师的帮助才使得我的毕业设计和毕业论文能够顺利完成。

感谢辅导员张蕾老师的关爱，她在我的生活和学业上都给予了无微不至的关怀。

感谢软件学院的各位老师，他们不仅传授我知识，更用认真严谨的治学态度深深影响着我。

感谢我毕业设计小组的成员刘歆，邓垚鸿，邓云和杨永乐同学，他们在整个毕业设计完成阶段给予了我很多帮助和支持。