保密★2 年

申请同济大学工学硕士学位论文

智能手机构件化软件设计模式研究及其在邮件系统中的应用

（国家 863 计划“软件重大专项”项目　课题编号：2001AA113400）

培养单位：电子与信息工程学院

一级学科：计算机科学与技术

二级学科：计算机软件与理论

研 究 生：叶 蓉

指导教师：陈榕　教授

二〇〇八年一月

保密★2 年

A dissertation submitted to
Tongji University in conformity with the requirements for
the degree of Master of Engineering

Patern Research for CAR based Smartphone Software and its Application in the Email System

(Supported by the 863 program of China for
Important Software Project, Grant No. 2001AA113400)

School/Department: College of Electronics and Information Engineering
Discipline: Computer Science and Technology
Major: Computer Software and Theory
Candidate: Rong Ye
Supervisor: Prof. Rong Chen

Jan, 2008

智能手机构件化软件设计模式研究及其在邮件系统中的应用

叶蓉　同济大学

学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年　　月　　日

同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年　　月　　日

摘要

随着通信产业的快速发展，单纯的语音通信功能已经无法满足消费者的需要，集语音通信、数据通信、图像处理等多种功能于一体的个人移动媒体平台-智能手机（Smartphone）得到越来越消费者的青睐。现在的智能手机产品从概念到设计、宣传、上市、推广、热销到最终退出市场，产品的生命周期越来越短。越来越多的智能手机生产企业力求其产品以更低的成本、更多的差异性、更快的响应和更好的服务进入并占领市场。消费者对手机的需求也从过去盲目追求产品功能多样性，发展到现阶段重视产品功能与应用的理性结合，由单纯追求产品全面性转向重视产品的个性化、差异化。要想将智能手机日益丰富和复杂的应用发挥得淋漓尽致，就必须依赖软件技术的大力支持。

论文从软件架构模式(Architectural Pattern)、设计模式(Design Pattern)等不同的模式抽象层次讨论了智能手机软件应用的架构和构件的实现，在实现中，充分考虑了智能手机软件应用的特点，并结合 Elastos 系统中 CAR 构件技术和 XmlGlue 技术，实现了低耦合、灵活性强、可升级、用户界面易于修改的智能手机软件开发模型。

论文在对智能应用软件架构模式的研究中，创新性地在 Elastos 软件平台实现基于 CAR 构件技术、运用 XmlGlue 技术的 MVC 软件架构模式。其核心思想是运用脚本语言来描述智能手机应用 UI（人机交互界面，User Interface)；按照应用功能实现对应用进行抽象，以构件形式形成一系列功能单一的元构件，也就是手机应用引擎构件；再由 CAR 构件封装应用的逻辑，然后由 XML 在脚本语言和 CAR 构件之间搭起通信的桥梁，实现了以脚本方式完成构件组装的功能，最终实现智能手机应用。

论文还对智能手机软件构件化开发意义进行阐述，对智能手机构件开发中遇到的有代表性的问题进行研究分析，结合构件开放、灵活、稳定的特点，合理运用 CAR 构件技术实现面向构件的设计模式。

论文还以智能手机邮件系统实现为例，详细说明其如何运用 MVC 架构模式及其他设计模式实现具体应用的过程。

以上这些研究成果均已运用于上海科泰世纪智能手机软件开发中。

关键词：智能手机，设计模式，CAR 构件，XmlGlue

ABSTRACT

With the fast growth of telecom industry, customer need is unable to be met by simple voice communications, while Smartphone, the personal mobile media platform integrated with voice, data exchange, image process and many other functions are stepwise getting public attention.

Currently the product life cycle of Smartphone is shorter and shorter from concept to design, propagandas, go market, promotion, hot selling to eventually withdraw from the market. More and more Smartphone manufacturers enter and occupy the market at lower cost, more differentiation, quick response and better services. In the mean time, consumer demand on mobile phones also change from the past blind pursuit of product features diversity, to the emphasis at feasible combination of function and application; from simple pursuit of comprehensively to the personality and differentiation. It is a must to reply on the strong support of software technology that Smartphone can best perform with complex application.

In this paper, Smartphone software application framework and CAR component implementation are discussed from the perspective of different abstract level of Pattern such as “Architectural Pattern”,” Design Pattern”, etc. The characteristics of Smartphone software applications, along with combination of CAR component technology and XmlGlue technology in Elastos system are fully taken under consideration, so as to introduce Smartphone software development model in loose coupling, flexible, easy upgradable and easy change of graphic user interface.

During the research of Smartphone software Architectural Pattern, the paper innovatively applies MVC to Elastos system based on CAR component and XmlGlue technology. The essence of the idea is to leverage script language to describe Smartphone software user interface.

According to the application functions to abstract, and to form a series of CAR based meta component, that is, cell phone software engine; Use CAR to encapsulate the application logic, and then build up the communication bridge between script

language and CAR component, so as to realize the way to complete the script component assembly function, and eventually realize the smart mobile applications.

This paper also illustrates the significance of CAR component to Smartphone software application, and analyzes the representative problems usually encountered during the development. It combines the open, flexible, and stable characteristics of CAR component and feasibly applies it on the implementation of CAR based design pattern.

This Paper uses Smartphone email system as an example, to in detail explain the process of leveraging MVC framework and other design patterns to implement specific application.

These foregoing research achievements have already been applied to Smartphone software development of Shanghai Ketide co, ltd and got quite positive results.

Key Words: Smartphone，Design Pattern，CAR，XmlGlue

目录

第 1 章 引言

1.1 概述

在过去的 20 年间，通讯产业是全球增长最为迅速的产业之一，手机作为世界上发展最快的通讯类产品之一，其应用早已突破了基本的语音和数据业务，扩展到包括互联网、广播、影视、数字娱乐及金融服务等不同领域。[1]

随着消费市场的逐步成熟和消费者对于手机功能要求的提高，普通手机单纯的拍摄、传送、MP3 等简单功能已不再满足人们的需要。人们需要更加智能化、人性化的手机来满足日渐提高的商务生活和现代生活的需要，经过几年的培育，智能手机迎来了发展的黄金时期，智能手机市场从 2004-2005 年度开始进入快速增长期，市场上的智能手机层出不穷，市场竞争也更加激烈，而 3G 时代的到来和众多国内厂商的加入，让智能手机在价格、外形、功能、增值服务等方面有了更大的突破，同时也更加受到用户的欢迎，这也使得智能手机在竞争中取胜，从而跻身主流市场。

据瑞典市场分析公司 Berg Insight 的调查显示，2007 年全球智能手机销售量将突破一亿大关，达到 1.13 亿部。Berg Insight 预测，虽然目前智能手机只占手机市场份额的 10%，但未来全球智能手机的年复合增长率(CAGR)将达到 25.6%，到 2012 年智能手机的销售量将达到 3.65 亿部，并在全部手机销售中占到 22% 。

在中国，2007 年上半年全国智能手机的销量达到 1179 万台，同比增长 101.2%。07 年智能手机全年的销量极有可能达到两千万台左右。

巨大的市场也带来了激烈的市场竞争，现在的智能手机产品从概念到设计、宣传、上市、推广、热销到最终退出市场，产品的生命周期越来越短，这样的市场趋势对整个手机产业链提出了优化整合的要求，而今越来越多的智能手机生产企业也日益力求其产品能以更低的成本，更多的差异、更快的响应和更好的服务进入并占领市场。

在消费市场一端，如今的智能手机已经完全脱离了简单通话功能的初衷，

借助丰富的软件应用，现在的手机产品可以完成文字图片传送、音视频录放、数据处理甚至程序运转等多种业务功能，完全演变为一个实实在在的流动多媒体终端。消费者对手机的需求也从过去盲目地追求产品功能的多样化，发展到现阶段重视产品功能与应用的理性结合；由单纯地追求产品的全面性转向重视产品的个性化、差异化，这就对智能手机开发提出了更高的要求。

综上所述，要想将智能手机日益丰富和复杂的应用发挥得淋漓尽致，就必须依赖软件技术的大力支持。应用软件的功能强大，应用间耦合度低，产品易于定制，升级，软件易于修改、可复用性强；按照消费群体的关注点不同尽快推出更加贴近不同消费群体、满足消费者需求的智能手机产品；以上是提高手机生产厂商的核心竞争力的关键所在。而以上这些，是基于良好的软件开发架构和合理的软件设计模式的，国外的知名手机制造公司，如诺基亚，摩托罗拉等，在其手机软件开发方面有着相当成熟的工程规范，而这些，正是国内智能手机软件公司所欠缺的。

1.2 课题来源及选题意义

本课题是基于上海科泰世纪有限公司 863 项目“基于中间件技术的因特网嵌入式操作系统及跨操作系统中间件运行平台”展开，该项目主要完成拥有自主知识产权的“和欣”（Elastos）智能手机嵌入式操作系统及应用程序运行支撑平台研究和开发；本课题属于该项目，主要研究如何实现基于 CAR（Component Assembly Runtime）构件技术的可复用智能手机软件开发模型。

目前，智能手机软件开发的核心技术都在国外，国内企业在这一块的核心竞争力差，核心竞争力差的原因在于没有自己的核心技术：基本都是国外的操作系统，也没有合理的软件载体。而上海科泰世纪有限公司的“和欣”智能手机软件开发平台的出现，极大地弥补了这一不足，本文对课题的研究基于“和欣”智能手机软件开发平台上展开的。

针对目前国内智能手机软件开发过程中暴露出的一系列问题，如：重复劳动过多，开发周期过长，开发成本较高，无法按需订制应用，无法通过升级扩展应用功能等。基于“和欣”智能手机软件开发平台这一载体，本文研究如何结合 CAR 构件技术，XmlGlue 技术，设计出合理的智能手机应用软件架构以及

实现面向构件技术的设计模式，基于合理的软件架构模式和设计模式实现软件构件化开发，提高软件的可复用性，形成合理的智能手机引擎开发库，实现各个应用模块的独立性并保持模块间的松耦合，增强其灵活性，提高软件可复用性，在较短的开发周期内，以较低的成本开发出具有较高质量的智能手机应用软件，加快软件开发力度及开发速度。这对促进智能手机软件行业及企业发展，具有十分重要的意义。

1.3 国内外的发展现状

构件复用已经在学术界和产业界得到了广泛深入地研究和实践，随着硬件平台与软件平台相分离，智能手机的处理系统将通过专业化的分工 ,形成不同的“零部件”，而这种“构件化”的设计方式，将成为智能手机发展的新趋势。从手机硬件平台的“基频处理器和应用处理器”相分离，到手机软件平台的“操作系统和应用软件平台”相分离；可以清晰地看到：智能手机技术越来越向着“构件化”的方向发展。水清木华研究中心认为，这种分离式的“构件化”开发趋势，在未来将使智能手机产品的开发像制造工业产品一样容易。首先，通过专业化分工生产出不同功能的“零部件”；然后，再将这些“零部件”合理地组装起来，形成所需的产品。而且，智能手机开发的“构件化”还可以实现各个层面硬件、软件的复用和构件化生产，极大节约了智能手机产品的开发时间和开发成本。[2]

设计模式在软件设计行业中的起源可以追述到 1987 年，WardCunningham 和 Kent Beck 一起使用 Smalltalk 语言设计用户界面。他们决定运用 Alexander 的理论来发展一个有五个模式的语言，并用来指导 Smalltack 的新手，因此他们完成了一篇“Using Pattern Languages for Object-Oriented Programs（使用模式语言进行面向对象开发）”的论文（发表于 OOPSLA 87 in Orlando）。

在那以后不久，Jim Coplien 开始搜集 C++语言的成例（idioms）。成例是模式的一种，这些成例发表在 1991 年出版的《Advanced C++ Programming Styles and Idioms（高级 C++编程风格和成例）》一书中。[3]

1990 至 1992 年间，Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides 四位作者开始搜集模式的工作。关于模式的讨论和工作会议一再召

开。

1993 年 8 月，Kent Beck 和 Grady Booch 主持了在科罗拉多山区度假村召开的第一届关于模式的会议，模式研究的主要人物都参加了这个会议，在那以后不久，Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides 四位作者的《Design Pattern: Elements of Resuable Object-Oriented Software》一书发表了。此后，参加设计模式研究的人数、被确定为模式结构的数据均呈爆炸性增长。编程模式语言大会（Pattern Languages of Programming，即 PLOP）每年在美国举行一次，大会的论文也汇编成书，公开进行发表 [4]。各种模式也不断被应用到软件工程的各个方面[5]。

1.4 本人的主要工作

本人通过分析智能手机特点及智能手机软件开发中遇到的问题，并结合 Elastos 智能手机开发平台的 CAR 构件技术、XmlGlue 技术，创新性的提出基于 CAR 构件、运用 XmlGlue 技术的 MVC 软件架构模式。其创新性在于运用脚本语言来描述智能手机软件应用 UI（人机交互界面，User Interface）；按照软件应用功能实现对应用进行抽象，以构件形式形成一系列功能单一的元构件，也就是应用引擎构件；再由 CAR 构件封装应用的逻辑，然后由 XML 在脚本语言和 CAR 构件之间搭起通信的桥梁，实现了以脚本方式完成构件组装的功能，最终实现软件应用。

同时，本人分析了构件化设计对智能手机软件开发的意义，并对智能手机应用构件化开发过程中遇到的普遍问题进行分析，将软件设计模式合理运用到开发中，达到了构件间松耦合、易扩展、灵活性高的目的。

1.5 论文的组织结构

论文主要分为七个章节，其结构及要点如下：

第一章引言

本章简述了论文课题的来源以及选题的意义，阐述了目前国内外的发展现状及本人的贡献。并给出论文的结构。

第二章 Elastos 平台关键技术

本章阐述了 Elastos 智能手机整体解决方案及该平台的关键技术。它们是构件化智能手机软件开发、设计模式研究及实现的基础。

第三章设计模式

本章论述了设计模式的定义、基本要素、设计模式的分类、设计模式与 CAR 构件技术的结合以及设计模式在智能手机软件开发中的意义。

第四章 智能手机 MVC 软件架构模式分析与运用

本章阐述了软件架构模式在软件开发中的重要性，通过分析智能手机软件架构需达到的目的给出基于 CAR 构件，运用 XmlGlue 技术的 MVC 软件架构模式并给出邮件系统应用实例。

第五章 构件化智能手机构件开发中设计模式的运用

本章讨论在智能手机构件开发过程中运用的设计模式，通过合理使用设计模式实现构件间的低耦合，并增加了构件的灵活性。

第六章 构件化智能手机邮件应用的实现与模式应用

本章阐述了构件化智能手机邮件应用的具体实现，并论述了软件架构模式、软件设计模式的运用情况。

第七章 总结与展望

本章总结了研究工作，并对今后的工作做了进一步的展望。

第 2 章 Elastos 平台关键技术

科泰世纪的智能手机软件平台有以下三个基本组成部分[6]

1) Elastos 嵌入式操作系统内核 Zener

2) Elastos 网络构件运行平台　Elastos

3) Elastos 智能手机软件集成方案　Elastos Mobile

如图 2.1 所示

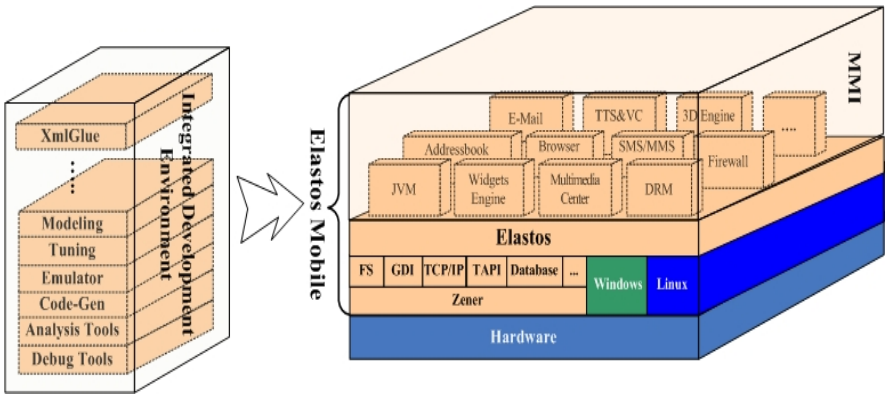

图 2.1 Elastos 平台产品架构图

2.1 Elastos 智能手机软件集成方案

科泰世纪为智能手机专供的软件整体解决方案，此方案涵盖 Elastos 智能手机操作系统、集成的应用参考方案、MMI 参考方案、Elastos 特色应用参考方案等在内的全套应用。Elastos 智能手机软件整体解决方案还提供了一整套完备的集成开发环境及相关开发、调试、自动测试工具，完备的用户手册、技术文档、示范程序等。

Elastos 智能手机软件整体解决方案的设计遵循构件化程序设计思想，按照手机常用的业务模块划分构件，明确定义每个模块的接口。每个业务程序模块都用 URL 编址，可以动态下载，采用工厂化的设计方法，即插即用的设计理念，让系统中的各个模块相对独立，各个子模块之间为松耦合，便于不同 ISV 生产

的软件模块之间的适配和通讯，便于软件模块的版本的下载和升级，便于用户根据需求进行自由组合与动态替换。

2.2 Elastos 智能手机操作系统

Elastos 智能手机操作系统是科泰世纪为智能手机专供的操作系统及引擎的软件代码集合。该操作系统是科泰世纪公司自主创新的、面向智能手机的完全开放的嵌入式操作系统。该系统采用构件、中间件技术，全面支持新兴的网络服务（WEB Service）和面向服务的体系结构（Service-Oriented Architecture，SOA）。按照下一代网络与软件技术的发展方向，和欣的 CAR（Component Assembly Runtime）技术使软件工厂化生产成为可能。“和欣”智能手机操作系统将为中国移动通讯产业链提供一个至关重要的软件开发、运行及服务平台。

作为 Elastos 智能手机整体软件支撑框架的一个核心组成部分，Elastos 嵌入式操作系统体现的技术特点包括：[7]

1、支持无缝计算 (Seamless Computing)。利用 URL 等全球唯一标识技术，将软件模块直接在因特网范围编址，如：//www.elastos.com/car/drivers/tcpip.dll。配合其它创新软件技术，Elastos 可以自动、透明地为用户搜索解码器、驱动等软件模块。

2、遵循绿色软件原则，避免软件安装/反安装，软件模块拷贝过来就可以直接使用，摒弃 Windows/LINUX 注册表之类的设计。根据用户使用习惯不同，每个终端都可以按需配置个性化软件。结合 URL 对软件模块的编址技术，Elastos 为提供了“感觉不到软件”的用户体验。

3、Elastos 发明的 CAR（Component Assembly Runtime）技术，为 C/C++ 语言添加契约导向（面向接口）编程描述，对目标软件模块进行封装，按照规范实现软件工厂化生产，支持不同版本软件构件的互操作，允许不同生产厂家软件的相互替换，实现不同语言（JAVA、C#等）构件相互调用。

4、在软件制造商之间互不公开源代码的前提下，以目标代码为单位，动态拼装不同厂家的软件构件，完成更大的功能模块。这种开放标准，不开放技术的合作方式，为软件工业化、产业化奠定基础。

5、利用 Elastos 的“超净车间”（CleanRoom）进程间防火墙技术，对不可信赖的软件实行隔离。应用程序可以按照安全级别、CPU 指令集等要求，对一个指定的软件构件在本地、跨进程、跨机器或跨网络运行，实现内核安全技术。

6、实现透明的分布式计算模型，回避 Linux、Windows 等基于消息机制的编程模型。在网络间通过消息轮询方式编程，让 N 个应用交换信息的计算复杂性是 N 的阶乘，这是计算理论的禁区。新一代网络编程环境，如 JAVA、.NET 等也都回避了传统的消息机制编程模型。

7、利用绑定回调函数，处理软件构件里面的异步回调事件。在无需修改源代码的情况下，完成对软件模块的双向调用，实现软件按需加载，平滑运行。软件开源基于自愿的前提，无需开源的软件开放运行平台才能真正保障产业发展。

8、用 XML + JavaScript 等脚本语言进行用户界面编程，动态适配 CAR 构件实现的标签（Tags），实现了完整动态的编程环境。利用“换肤”技术提高产品的灵活性；利用 CAR 构件复用提高产品质量和快速开发；利用因特网门户网站生成的动态脚本，可以有效地促进产业商业环境发展。

9、统一的数据模型，利用底层数据库统一管理文件存储、下载网页、小应用（Widgets）缓存等，保障数据一致性，减轻程序员的负担。

10、在操作系统层面率先实现了对面向方面编程（AOP）的支持。允许用户对上下文（Context）编程，实现动态构件聚合，方便用户对不同需求的扩展，包括对软件模块进行动态监听、信息截获、通讯转移、信息同步、离线运行等问题的支持。

2.3 和欣嵌入式操作系统内核 Zener

Zener 是科泰世纪自主设计、实现的嵌入式操作系统内核。Zener 在内核层提供了对网络构件运行环境的支持。Zener 支持多进程、多线程，抢占式、多优先级线程任务调度。Zener 提供 TCP/IP、FAT、CRT 等众多嵌入式系统常见的标准程序库。

2.4 CAR（Component Assembly Runtime）构件技术[8]

Elastos 智能手机操作系统是科泰世纪为智能手机专供的操作系统及引擎的软件代码集

Elastos 平台的 CAR（Component Assembly Runtime）构件技术是科泰世纪按照下一代网络与软件技术的发展方向，自主研发的面向构件的编程模型。它们使软件工厂化生产成为现实。CAR 编程模型规定了一组二进制构件间相互调用的标准，而且在构件中封装自描述信息（又称元数据）。Elastos 网络构件运行平台可以根据构件自描述信息，在运行时把 CAR 构件当作软件零件动态拼装起来，如图 2.2 所示。从而使构件框架可以根据自描述信息向构件提供运行时支持。使得二进制构件能够自描述，CAR 是一种新的中间件技术。

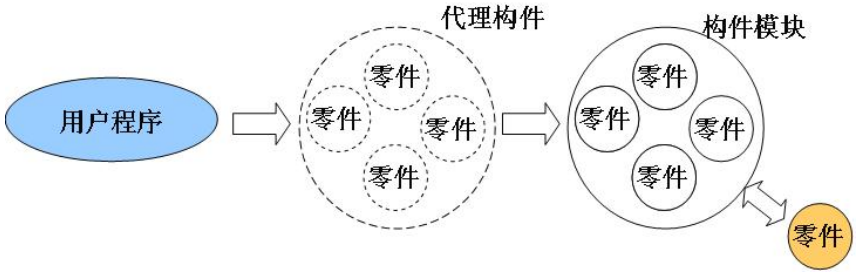

图 2.2 中间件运行环境的模型，动态生成代理构件

CAR 构件技术定义了一套网络编程时代的构件编程模型和编程规范，它规定了一组构件间相互调用的标准，使得二进制构件能够自描述，能够在运行时动态链接。CAR 构件技术继承了 COM 的二进制封装思想，面向接口编程。在逐步融合.Net，Java 技术思想之后，形成独有的二进制构件程序模型。

为了在资源有限的嵌入式系统中实现面向中间件编程技术，同时又能得到 C/C++的运行效率，CAR 构件技术没有使用 JAVA 和.NET 的基于中间代码——虚拟机的机制，而是采用了用 C++编程，用 ElastosSDK 提供的工具直接生成运行于 Elastos 构件运行平台的二进制代码的机制。用 C++编程实现构件技术，使得更多的程序员能够充分运用自己熟悉的编程语言知识和开发经验，很容易掌握面向构件、中间件编程的技术。在不同操作系统上实现的 Elastos 构件运行平台，可以使 CAR 构件的二进制代码实现跨操作系统平台兼容。

对于面向 WEB 服务的应用软件开发，以及开发操作系统这样的大型系统

软件而言，采用 CAR 构件技术具有以下的意义：

(1) 不同软件开发商开发的具有独特功能的构件，可以确保与其他人开发的构件实现互操作，不同的编程语言实现的构件之间可以实现互操作。

(2) 提供一个简单、统一的编程模型，使得构件可以在进程内、跨进程甚至于跨网络运行。同时提供系统运行的安全、保护机制，实现在对某一个构件进行升级时不会影响到系统中的其它构件。

(3) CAR 的开发工具自动实现构件的封装，简化了构件编程的复杂性，有利于构件化编程技术的推广普及。

(4) CAR 构件技术是一个实现软件工厂化生产的先进技术，为建立软件标准提供了参考，可以大大提升企业的软件开发技术水平，提高软件生产效率和软件产品质量。

2.5 XmlGlue 技术

XmlGlue 是基于 CAR 构件技术发展起来的一种编程技术。它对外提供了 XmlGlue 应用的运行环境以及多种脚本语言，方便开发人员快捷地开发应用；对内则在脚本语言和 CAR 构件之间搭起通信的桥梁，实现了以脚本方式完成构件组装的功能。

其技术优势在于：

（1）动态 UI 技术

Elastos 上的“动态换肤”技术是用 XmlGlue 技术实现的。它通过本地化引擎与 XML 动态匹配解释，实现在不需要修改本地二进制引擎代码的情况下，只是简单的替换 UI 的 XML 描述语言来重新封装原有的构件化应用引擎，再结合 JavaScript 实现逻辑控制流操作，从而达到迅速的更换人机交互 UI 的效果;

（2）支持应用跨操作系统

Elastos 提供了一个完整的 XmlGlue Runtime。XmlGlue Runtime 是一个支持 XmlGlue 的网络构件运行平台，可以运行在 Elastos、Linux、WinCE 多个系统之上。对 XmlGlue 的应用而言，Runtime 完全屏蔽了操作系统的差异。XmlGlue

继承了 CAR 技术的优秀特性，所以，只要符合 CAR 标准的组件，都可以在 XmlGlue Runtime 中使用。这样使得 XmlGlue 应用具有很好的扩展和移植能力。

（3）支持 SOA 面向服务的架构

Elastos 是面向 SOA 设计的，CAR 构件技术保证了服务的提供，XmlGlue 编程范式则提供了一种新的使用这种服务的方式。在嵌入式应用千变万化的今天，XmlGlue 将应用的表现层和逻辑层分离开，服务提供商只需提供功能独立的正确的构件，应用开发人员就可以根据需求的变化快速更改构件之间的关联关系，降低了服务提供者和使用者的耦合程度。XmlGlue 强化了 SOA 编程的思想，弱化了服务（构件的生产者）和应用（构件的使用者）之间的关系，使得通过它开发的应用能够方便快捷的使用分布在世界各地的服务，而不需要所有服务都集中到一起，从而降低了应用的运行环境要求，扩大了服务范围。

第 3 章 设计模式

3.1 模式的定义

所谓模式，是一种问题的解决思路，它已经适用于一个实践环境，并且可以适用于其他环境。[9]

当人们在解决问题的时候，最简单的方法不是去发明一种新的解决方案，而是在他们了解的或者以前使用过的解决问题的方法中，找到和目前情况类似的一种方法，然后进行复用，久而久之，以前的这些方法就形成了一些固定的解决问题的方案，通过对这些方案的本质的提炼，就形成了模式。模式的出现，是为了更好地在前人的基础上分析复杂的问题，得到最有效的解决问题的方法。

3.2 设计模式

3.2.1 设计模式的定义

设计模式是对被用来在特定场景下解决一般设计问题的类和相互通信的对象的描述。

设计模式描述了一个在特定设计上下文中重复出现的设计问题，以及一个已经被证实的通用解决方案。解决方案描述了设计的组成部分，它们之间的相互关系及各自的职责和协作方式。设计模式的思想认为在系统设计这一层次上，软件开发可以抽象成一种模式，并可以复用。其核心就在于提供了相关问题的解决方案。设计模式有利于帮助做出有利于系统复用的选择，避免设计破坏系统复用性。总之，设计模式可以帮助设计者更快更好地完成系统设计和开发[10]

3.2.2 设计模式的基本要素

设计模式有四个基本要素：[11]

1 模式名称（pattern name）一个助记名，它用一两个词来描述模式的问题，解决方案和效果。命名一个新的模式增加了设计词汇。模式名可以帮助程序员思考，便于程序员之间交流设计思想及设计结果。找到恰当的模式名也是设计模式编目工作的难点之一。

2 问题（problem）描述了应该在何时使用模式。它解释了设计问题和问题存在的前因后果，它可能描述了特定的设计问题，也可能描述了导致不灵活设计的类或对象结构。

3 解决方案（solution）描述了设计的组成成分，它们之间的相互关系及各自的职责和协作方式。因为模式就像一个模板，可应用于多种不同场合，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的抽象描述和怎样用一个具有一般意义的元素组合（类或对象组合）来解决这个问题。

4 效果（consequences）描述了模式应用的效果及使用模式应权衡的问题。尽管在描述设计决策时，并不总提到模式效果，但它们对于评价设计选择和理解使用模式的代价及好处具有重要意义。

3.2.3 设计模式的分类

模式分类应具有下列特征：[12]

（1）分类图式应该包括少而精的分类标准，且简单易学；

（2）每个分类标准应该反映模式的自然属性；

（3）分类图式应该给用户提供一个“路标”；

（4）分类图式队新模式的集成应该是开放的。

软件设计模式的分类有许多不同的方法：

（1）根据应用领域的不同，设计模式可以分为通用领域设计模式和特殊领域设计模式，特殊领域包括并发领域，分布式领域，持久化机制领域和时间领域等；

（2）从面向对象角度，设计模式有两种分类方式，一种是按照设计模式的目的分类，另一种是从其范围进行分类。

从其目的来看，可以分成下列三种模式类型：

创建型（Creational）模式：与对象的创建有关，即描述怎样创建一个对象，它隐藏对象创建的具体细节，使程序代码不依赖具体的对象；

结构型（Structural）模式：负责处理类或对象的组合，即描述类和对象之间怎样组织起来形成大的结构，从而实现新的功能；

行为型（Behavioral）模式：描述算法以及对象之间的任务（职责）分配，它所描述的不仅仅是类或对象的设计模式，还有它们之间的通讯模式。

根据设计模式是用于类还是用于对象，可将其分为类模式和对象模式。

类模式：该模式处理类和子类之间的关系，这些关系通过继承建立，在编译时便确定下来，是静态的；

对象模式：该模式处理对象之间的关系，这些关系在编译时无法确定，在运行时按照实际情况变化调用，具有动态性。

（3）将软件的架构、设计、程序实现的任何种类模式划分为三种不同层次的模式：架构模式（Architectural Pattern）、设计模式（Design Pattern）、成例（Idiom）也称为代码模式（Coding Pattern）。[13]

这三者之间的区别在于三种不同的模式存在于它们各自的抽象层次和具体层次上。架构模式是一个系统的高层次策略，涉及到大尺度的组件以及整体性质和力学。架构模式的好坏可以影响到总体布局和框架性结构。设计模式是中等尺度的结构策略。这些中等尺度的结构实现了一些大尺度组件的行为和它们之间的关系。模式的好坏不会影响到系统的总体布局和总体框架。设计模式定义出子系统或组件的微观结构。代码模式（或成例）是特定的范例和与特定语言有关的编程技巧。代码模式的好坏会影响到一个中等尺度组件的内部、外部的结构或行为的底层细节，但不会影响到一个部件或子系统的中等尺度的结构，更不会影响到系统的总体布局和大尺度框架。

代码模式或成例(Coding Pattern 或 Idiom)

代码模式（或成例）是较低层次的模式，并与编程语言密切相关。代码模式描述怎样利用一个特定的编程语言的特点来实现一个组件的某些特定的方面或关系。较为著名的代码模式的例子包括双检锁（Double-Check Locking）模式

等。

设计模式(Design Pattern)

一个设计模式提供一种提炼子系统或软件系统中的组件的，或者它们之间的关系的纲要设计。设计模式描述普遍存在的在相互通讯的组件中重复出现的结构，这种结构解决在一定的背景中的具有一般性的设计问题。设计模式在特定的编程语言中实现的时候，常常会用到代码模式。比如单例（Singleton）模式的实现常常涉及到双检锁（Double-Check Locking）模式等。

架构模式(Architectural Pattern)

一个架构模式描述软件系统里的基本的结构组织或纲要。架构模式提供一些事先定义好的子系统，指定它们的责任，并给出把它们组织在一起的法则和指南。有些作者把这种架构模式叫做系统模式。

本文使用的是第三种分类方式，即按照不同层次进行划分。

3.3 设计模式与 CAR 构件技术

设计模式和构件本身是有差别的，主要表现在以下两个方面：[14]

第一，设计模式比构件更加抽象，构件可以在代码中体现，但在代码中只能体现设计模式的实例。可以用程序语言把构件写下来，并且能直接学习、执行、复用（对可复用构件而言）。与之相比，设计模式在每次被复用时都需要被实现，但设计模式还解释了设计的意图，权衡和效果。

第二，设计模式比构件具有更为广泛的意义：构件往往针对特定的应用，其实现受特定的构件模型的约束。而设计模式由于抽象度高，几乎可用于所有种类的应用中，即使是专门的设计模式（如分布式系统或并行程序设计的设计模式），也不会像构件模型那样对具体实现做出限定，从而更加灵活。

但并不是说二者就是毫无关系的，就设计模式而言，是对被用来在特定场景下解决一般设计问题的类和相互通信的对象的描述。解决问题的载体，可以是面向对象编程模型的类和对象，也可以是构件。从提倡软件复用的角度来看，使用构件技术实现设计模式，会使得设计模式的实现更具有可复用性、也更优

雅。本文讨论了基于 CAR 构件技术的设计模式在智能手机软件开发中的实现。

3.4 设计模式在智能手机软件开发中的意义

设计模式是一个很宽泛的概念，按照不同的抽象层次，可以分为：软件架构模式、软件设计模式和代码模式。

在智能手机软件开发中引入设计模式的意义在于：[15]

第一，智能手机产品是一种更新换代非常迅速的产品，设计模式为开发者提供了一个很好的设计经验，模式中所描述的解决方案是人们从不同角度对一个问题进行研究，然后得出的最通用，最灵活的解决方案，其有效性是经过大量的实践检验的。开发者采用合理的设计模式，可以尽可能迅速的实现合理的解决方案。

第二，不同型号智能手机的应用具有重复性。设计模式为软件重用提供了一条途径，每个设计模式都可以是软件设计中的可重用元素或单元。多个模式可以组合起来构成完整的系统。这种基于模式的设计运用于智能手机应用开发，使得软件开发具有更大的灵活性，可扩展性和可重用性。

第三，设计模式的基本思想是将程序中的可变部分和不可变部分进行分离，尽量减少对象间的耦合度，某个对象的修改不会导致其他对象的变动，在智能手机软件开发中使用设计模式，使得修改带来的影响最小，从而缩短手机终端产品开发周期。

第 4 章 智能手机 MVC 软件架构模式分析与运用

4.1 设计模式与软件架构

设计模式（Design pattern）和软件架构（Architecture）是开发系统时经常使用的设计方式[16]。软件架构规定了应用的体系结构，阐述了整个设计、协作构件之间的依赖关系、责任分配和控制流程，软件架构是构件技术、软件体系结构研究和应用软件开发三者反复结合的产物。软件架构的关键还在于架构内对象间交互模式和控制流模式。而设计模式是对在某种环境中反复出现的问题以及解决问题的方案的描述。

从软件架构的层次进行模式抽象，软件架构模式（Architectural Pattern）描述软件系统里的基本的结构组织或纲要。架构模式提供一些事先定义好的子系统，指定它们的责任，并给出把它们组织在一起的法则和指南。这种架构模式也叫做系统模式。

4.2 软件架构模式在智能手机开发中的意义

意义一、运用合理的软件架构设计将功能（应用要做什么）、表现（应用的表现形式）与外观（应用的用户界面呈现）完美的区分开来，例如，对于邮件应用来说，要能够将功能（邮件收发、管理）与表现（邮件应用的表现形式）和外观（邮件应用的用户界面呈现）三者区分开，从而降低应用开发、维护、定制的成本，针对不同用户开发功能相同但表现不同的应用，可以以最小的代价、最短的时间完成相应的功能；

意义二、运用合理的软件架构设计可以将应用软件的关注度分离，提高应用软件的开发效率。让程序开发人员只关注应用业务逻辑的实现，由 UI 开发人员实现更完美的页面展示；

意义三、运用合理的软件架构设计让手机应用成为一款 DIY 的产品。DIY，Do It YourSelf，让普通消费者具备有选择地 DIY 自己的手机应用软件的能力，

最大程度地发挥已有硬件功能，保证手机应用的先进性，能够与时俱进；让手机技术发烧友，通过 DIY 拥有个性化的手机应用 UI 风格。

4.3 MVC 软件架构模式分析与实现

针对智能手机的特点并结合 Elastos 智能手机开发平台独有的技术，独创性地提出了基于 CAR 构件，运用 XmlGlue 技术的智能手机软件 MVC 架构模型。

4.3.1 运用 XmlGlue 技术的 MVC 模式分析

在基于 CAR 构件，运用 XmlGlue 技术的 MVC 软件架构模式中将应用的实现分为三层，其结构如图 4.1 所示。

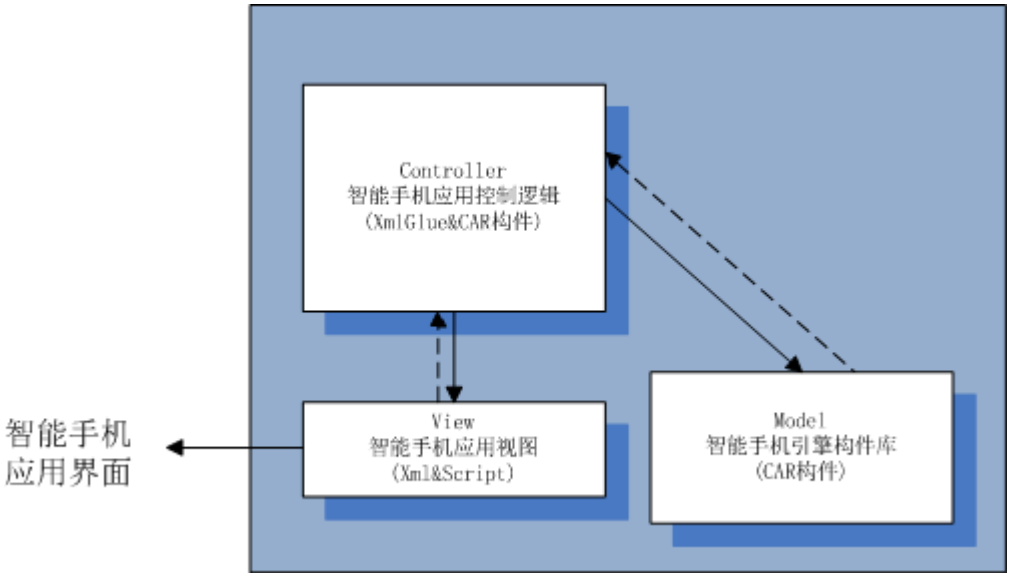

图 4.1 构件化智能手机应用软件架构模型

首先分析图 4.1 中的 Model，Controller 和 View 三个部分。

Model（模型）部分，这是智能手机引擎构件库部分，其实现完全基于 CAR 构件系统。这是将手机应用的功能进行抽象，完全按照 CAR 构件系统的规范进行开发实现而形成大量功能单一的元构件集合，智能手机应用开发基于这些基础构件，只要构件库中的手机各种应用基础构件足够丰富，则开发新的应用则变得高度简化：开发人员可利用现有构件快速组合出应用程序的主体，只有应

用的特殊需求需要重新实现，这将极大提升软件开发效率和软件质量。

View（视图）部分，这一部分采用符合 W3C 文法的 XML 标记和多种脚本语言来描述具体应用。View 采用这样的实现形式有两个优势：第一、脚本语言擅长客户端逻辑的控制，由它来执行客户端程序逻辑的控制；XML 语言适合于数据的存储和描述，由它来描述应用的架构，界面的布局以及构件及构件间的关系，两者相结合，脚本语言弥补了 XML 在描述界面元素逻辑上的缺陷，XML 弥补了脚本语言不适合逻辑的实现的缺陷；第二、采用脚本语言开发 UI 界面，降低了开发难度，提高了开发效率，使得用户 DIY 手机应用成为可能。

Controller（控制器）部分，是整个应用系统的核心部分。这一部分是对引擎库的合理封装，将构件元素组合成符合手机应用的 CAR 构件；同时这一部分还运用 XmlGlue 技术实现 XML 语言、脚本语言与 CAR 构件系统之间的交互机制（包括 XML 和脚本对 CAR 构件的正向的调用以及 CAR 构件系统的反向事件回调）以及错误的处理。

4.3.2 层间通讯的实现

下面来阐述层次间的通讯过程，首先规定：“用户交互层”为上层，而“手机引擎层”为下层，三层结构的层间通信通过上层对下层的正常接口方法调用及下层注册相应事件方法给上层，由下层触发事件而引起上层做出响应。如图 4.2 所示。其中，手机引擎层是 CAR 构件实现的，逻辑控制层调用引擎层的部分也是 CAR 构件实现的，因此，二者之间的调用使用的是 CAR 构件的调用和回调机制；用户交互层是脚本+XML 的实现方式，通过 XmlGlue 的脚本回调构件的方式，即将脚本语言编写的方法作为回调接口的事件方法注册给接收器对象，并通过接收器调用到所注册的脚本方法来实现回调。

这里需要注意的一点是三层结构中尽量让相邻的两层进行直接通讯，也就是说，UI 界面层不直接调用手机引擎层。这不是说 UI 界面不可调用引擎层，而是不推荐这种方式，这样可以使应用软件的结构更加清晰。

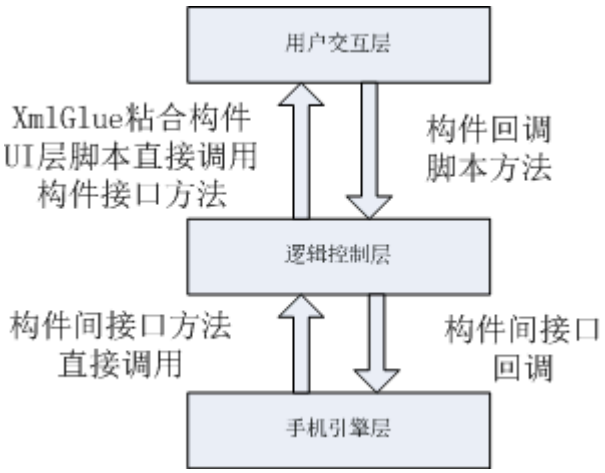

图 4.2 层间通信示意图

4.4 智能手机 MVC 软件架构模式的创新性

基于 CAR 构件，运用 XmlGlue 技术的 MVC 智能手机软件架构模型的技术创新点主要有以下四个方面：

第一，智能手机的整体应用架构设计基于 CAR（Component Assembly Runtime）构件技术思想，建筑在 Elastos 网络构件运行平台之上。在系统中的各个模块相对独立，每个模块都可以抽象成为一个独立的构件。由于传统的动态链接库 DLL 动态调用是基于地址(Address)的调用，DLL 尺寸的改变必然导致函数地址的改变。所以难以做到应用的二进制动态升级与动态拼装。而对构件模块 DLL 的功能调用则是基于接口（Interface）的，各个子模块之间没有直接的调用耦合。系统通过 URL 来动态加载相关 DLL，创建构件对象实例。应用对功能服务的调用与接口顺序有关，与具体的函数地址无关。子模块的修改或升级只是在构件内部的局部的修改，更不会导致整个系统或产生应用的重新链接要求。应用可以方便地根据用户的产品需求进行二进制构件复用拼装。一次编译，重复可用。Elastos 构件化技术为应用的二进制动态升级提供了软件技术的基础支撑。

第二，在 CAR 构件技术的基础上，对智能手机的应用编程模型采用了 MVC (Model-View-Controller)的模型。MVC 结构是为那些需要为同样的数据提供多个视图的应用程序而设计的，它能够很好地实现数据层与表示层的分离。MVC

作为一种开发模型，很适合用于移动分布式网络应用系统的设计，以及用于确定系统各部分间的组织关系。针对手机产品的界面设计可变性的需求，MVC 把 Elastos 手机的人机交互系统的组成分解成模型、视图、控制器三层部件。

视图部件把表示模型数据及逻辑关系和状态的信息以特定形式展示给用户。它从模型获得显示信息，对于相同的信息可以有多个不同的显示形式或视图。

控制器部件是很薄的一层，处理用户与软件的交互操作的，其职责是控制提供模型中任何变化的传播，确保用户界面与模型间的对应联系；它接受用户的输入，将输入反馈给模型，进而实现对模型的计算控制，是使模型和视图协调工作的部件。

模型部件保存由视图显示，由控制器控制的数据；它封装了问题的核心数据、逻辑和功能的计算关系，它独立于具体的界面表达和 I/O 操作。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其它依赖于这些数据的视图都应反映到这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。这实际上是一种模型的变化-传播机制。

这种基于构件中间件技术的 MVC 编程模型能够很好屏蔽不同产家的底层软件技术的差异，用户可以很方便地在同一个 UI 视图下采用不同产家或不同版本号的软件引擎，也可以在不同的 UI 视图下使用重复使用同一个产家提供的应用引擎。视图的变化不会影响到模型与控制器的变化。能够快速的在已有成熟稳定引擎基础上，很快的形成新的应用。以计算器为例，基本上任何一款手机对于计算器的功能及复杂度要求的需求都是相同的，不同只是 UI 的整体效果与控件风格。如图 4.3 所示：

图 4.3　不同的 UI 效果和控件风格

简单来说，在传统的应用中，要体现出上述在布局上，包括控件风格、计算功能上都明显不同的 UI，必须重新开发出两套完全不同应用。如果再有第三种 UI 体现，那么这个开发工作需要不断重复。而在 Elastos 平台上，上述的 UI 仅仅是 View 层面的不同，底层的计算引擎可以不做任何的修改就简单地在不同的终端上进行平移。这将大大简化手机设计商的现有的开发复杂度，为相关企业的有效软件积累提供可靠支持。

第三，采用 XmlGlue 技术“粘和”用户界面与实际逻辑处理，界面采用 XML 和多种脚本结合的方式编写，逻辑部分则采用构件技术实现。简化应用的实现，应用设计人员可以像设计 HTML 一样方便的设计应用程序，几行 XML 可完成原来很多行 C/C++代码的工作。而业务开发人员则只需关注于应用业务逻辑的实现，不用再用 C/C++来编写繁琐的 UI（User Interface）了。同时，由于 XML 和脚本语言的动态性，它们皆可由服务器程序在运行时动态生成，形成智能手机软件应用程序无限多种变化的可能。

第四，采用脚本实现 UI，给予手机消费者 DIY 自己个性的手机邮件应用界面的可能性，消费者只需具备一定的脚本语言编程能力，通过 Elastos 集成开发环境，遵循 Elastos UI 规范，重新组织基于 XML 的 UI 脚本，描绘出相关功能模块的 UI 布局与效果，定义各 UI 控件的控制逻辑，并形成最终的可执行 UI。就可形成个性十足的手机邮件应用界面。

4.5 构件化智能手机邮件应用 MVC 软件架构模式的实现

根据上面分析得到的软件架构模型，本节将讨论构件化智能手机电子邮件应用的具体实现，将邮件应用分为“用户交互层”——“逻辑控制层”——“手机引擎层”三个层次，如图 4.4 所示

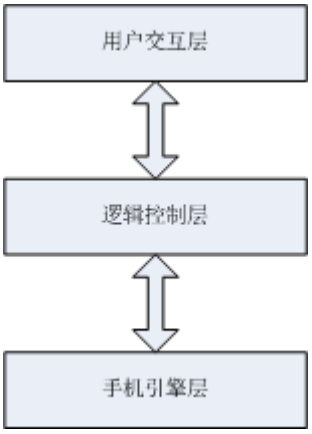

图 4.4 构件化智能手机邮件应用层次

在具体实现时，也按照三个部分分别进行实现。

4.5.1 模型(model)与邮件应用引擎层

手机引擎层是手机应用软件的最底层。它负责把底层的功能进行抽象封装，以接口的形式向外提供。

在和欣智能手机软件开发的过程中，手机引擎通过 CAR 构件形式具体实现。通过为每个 CAR 构件设置一个 128 位 GUID 来标识一个手机引擎 CAR 构件，称为 CLSID(class identifier，类标识符或类 ID)。在 CAR 构件的实现中，这个 CLSID 已经对用户不可见，将在 CAR 的编译与运行过程中，由编译工具与运行环境自动生成与管理。用 CLSID 标识对象可以保证在全球范围内的唯一性。

和欣智能手机引擎接口描述了手机引擎构件对外提供的服务。在构件和构件之间、构件和客户之间都通过接口进行交互。一个手机引擎构件可以实现多个引擎接口，而一个特定的引擎接口也可以被多个手机引擎构件来实现。

手机引擎构件一旦发布，智能手机应用开发人员可以使用预先定义的引擎接口提供的合理的、一致的服务来进行应用开发。这种接口定义之间的稳定性

使应用开发者能够构造出坚固的应用。

引擎设计为应用基本功能实现，可满足不同智能手机相同功能开发的需求，可实现代码复用。手机引擎层是对手机底层功能的抽象，通过手机引擎层的设计，可以形成许多手机引擎构件，例如：通信管理引擎、系统设置引擎、多媒体引擎、地址簿引擎、数据共享引擎等。当开发不同的智能手机应用软件时，如果采用的开发平台不变，大多数的手机引擎可以进行复用，因此，通过积累，可以形成完善的智能手机引擎开发库。

对于邮件应用来说，引擎需实现以下几类应用基本功能：

（1） 电子邮件帐户管理；

（2） 电子邮件编/解码、读/写、删除等；

（3） 电子邮件发送/接收；

（4） 邮件信箱的管理，自定义信箱的建立、修改及删除等；

（5） 电子邮件设置和签名；

对于引擎层的构件，要保持其原子性，尽量减少构件的重复开发。下面给出基于应用分类的邮件应用的构件及构件接口设计。

表 4.1 邮件应用构件接口说明

构件	接口	接口描述
MsgBoxManager.car	IMsgBoxManager	信箱管理引擎接口
	IMsgBoxEntity	信箱信息引擎接口
	IObjectEnumerator	信箱信息枚举接口（已重载）
Email.car	IPop3	发送协议引擎接口
	ISmtp	接收协议引擎接口
	IMessage	邮件编/解码引擎接口
	IAccountManager	邮件帐号管理引擎接口
	IAccountEntity	邮件帐号引擎接口
	IStorageManager	邮件存储管理引擎接口
	IStorageEntity	单个存储邮件管理引擎接口
	IEmailSetting	邮件设置引擎接口
	ISignature	邮件签名引擎接口
ConnManager.car	IConnectManager	网络连接引擎接口

对于信箱管理，在手机应用中，有电子邮件、短信、彩信等都有信箱管理功能，在设计引擎实现时将信箱管理抽象出来，形成一个构件 MsgBoxManager.car，供多个应用继承，重载使用。该构件对外提供信箱内容管理 IMsgBoxEntity 和信箱管理 IMsgBoxManager 两个接口。

对于邮件的管理和帐号管理，因为是邮件应用所独有的，所以单独实现，在实现的过程中，将其功能进行分类，接口按不同功能来设计，包括：发送/接收协议引擎接口：IPop3，ISmtp；邮件编/解码引擎接口：IMessage；邮件设置引擎接口：IEmailSetting；邮件签名引擎接口：ISignature；邮件账号管理相关引擎接口：IAccountManager、IAccountEntity；邮件存储管理相关引擎接口：IStorageManager、IStorageEntity；

因为邮件的发送和接收都必须通过 PPP 协议连接至 GPRS 网络，而需要这个连接的应用除了邮件应用外，还包括浏览器、彩信等，因此将网络连接功能抽象为一个单独 ConnManager.car 构件，该构件的 IConnectManager 接口实现了网络的连接/断开的管理。

引擎构件在实现中，遇到的具体问题和一些基于设计模式的研究在论文的第六章中作了具体的研究和阐述。接口方法的实现细节则在论文的第七章进行阐述。

4.5.2 控制(control)与邮件应用逻辑控制层

上面的描述给出了邮件应用引擎功能，引擎层为单个应用需求提供单一的功能，构件的粒度是比较小的，很多构件都是元构件（最基本的不含有其它构件的单元）。而逻辑控制层则通过将引擎层的构件进行组合，形成一个大的复合构件，为上层的 UI 界面层服务。

在构件化智能手机的邮件应用服务中，引擎层提供：邮件数据包的接收，发送，打包，解包，邮件信箱管理，邮件的存储等最基本的邮件功能实现，而如何将邮件引擎接收并解包出的数据进行逻辑整合并按需呈现给用户，就是在逻辑控制层需要完成的工作了，因此可以把逻辑控制层的构件看成是外层构件，引擎层的构件可以看成是内层构件，外层构件和内层构件是包含关系。外层构件通过调用内层构件的接口来完成功能的拼装。

邮件应用的逻辑控制层功能划分如表 4.2 所示：

表 4.2 邮件应用逻辑控制层接口表

功能名称	接口	相关引擎接口调用
邮件收/发管理	ISendRecv	IConnMgr； IPop3；ISmtp； IStorageManager；IStorageEntity； IAccountManager；IAccountEntity； IMessage；
邮件帐户管理	IAccountCtrl	IAccountManager；IAccountEntity； IEmailSetting；
信箱管理	IEmailBoxCtrl	IEmailBoxManager；IEmailBoxEntity； IStorageManager；IStorageEntity；
邮件管理	IEmailCtrl	IStorageManager；IStorageEntity； IEmailBoxManager；IEmailBoxEntity； IEmailSetting； ISignature；

从表中可以看出，逻辑层构件 IEmailCtrl 对引擎资源进行了整合，以已经开发完成的邮件构件为基础，通过组合方式快速构造出邮件应用，一切都变得简单起来。逻辑控制层的具体接口实现请参见论文第六章。

4.5.3 视图(view)与邮件应用用户交互层

在架构分析中，在用户交互层，界面逻辑采用 JavaScript 脚本语言开发页面交互逻辑，采用 XML 语言描述界面布局，采用 Elastos 平台独有的 XmlGlue 技术实现脚本语言对构件的操作。

具体实现如表 4.3 所示：

表 4.3 用户交互层实现表

文件名	功能
emailui.xml	emailui.xml 主要实现三大功能 ● 页面布局：定义出主页面大小及位置，主页面标题栏位置及大小，页面左右键位置及大小；布局的实现在 emailui.XML 的 <w:form>……</w:form>标签间实现； ● “粘合”构件：将需要的控制层构件“粘和”进来，例如 emailctrl = Elastos.Using("emailctrl.dll")；就引入了邮件逻辑控制的构件； ● “粘和”脚本语言：在 script 标签中说明使用的脚本语言并引入

	UI 层实现逻辑功能的脚本 emailui.js：<script language="JavaScript" src="emailui.js">……</script>；并调用脚本函数 InitEmailForm()加载邮件应用。
Emailui.js	Emailui.js 主要实现邮件应用的具体逻辑，其功能函数包括： InitEmailForm 主邮件应用加载函数 CEmailClientForm 电子邮件用户界面逻辑实现 CEmailMainForm 邮件应用主界面逻辑实现 CEmailBoxClientForm 电子邮件邮箱界面逻辑实现 CWriteClientForm 写邮件界面逻辑实现 CReadClientForm 读邮件界面逻辑实现 CExtractMainForm 邮件提取主界面逻辑实现 CExtractListForm 列表提取界面逻辑实现 CAttachmentListForm 附件列表界面逻辑实现 CAccountMainForm 电子邮件帐户主界面逻辑实现 CAccountManageForm 帐户管理界面逻辑实现 CEmailSettingForm 电子邮件设置界面逻辑实现 CSignatureForm 电子邮件签名设置界面逻辑实现 CMailboxManageForm 邮件信箱管理界面逻辑实现 CConnectMgrForm 邮件收/发管理界面逻辑实现 CMemoryStatusForm 邮件内存占用情况界面逻辑实现 CMoveToListForm 列表移动界面逻辑实现

第 5 章 智能手机构件开发中设计模式的应用

论文第四章从软件系统的基本结构组织角度讨论了智能手机 MVC 软件架构模式，本章将从构件设计角度运用设计模式到智能手机软件应用子系统中的构件设计及构件之间关系的纲要设计中。

构件化设计的需求在 PC 软件中并不突出，但是到了智能手机应用中就变得迫切起来，其主要原因有以下两个：

原因一、智能手机的运算能力和内存空间有限，如果在启动时加载大量构件，就会造成启动速度缓慢，占用内存空间多的问题，采用构件技术，主程序在启动时只加载少量构件，大部分构件在使用时才加载起来，做到“按需加载、点击运行”；

原因二、智能手机是一种移动设备，如何维护、更新设备上的应用程序是软件开发中要解决的问题，如果一次程序的更新要更新全部程序，则会造成带宽资源的浪费，采用构件化的设计使程序便于远程的维护更新，每次更新时只需要更新一部分构件就可以了。

在本章中，本人通过合理运用设计模式解决在智能手机软件开发中普遍存在的、在相互通讯的组件中重复出现的结构，形成低耦合、高内聚的引擎构件；通过运用设计模式分离应用中变化和不变点，加强引擎构件的灵活度。

5.1 单实例引擎接口与 Singleton 设计模式

5.1.1 问题的提出

在智能手机引擎中，有两类引擎接口有单实例的特性，一类是针对数据库操作的，在这一类接口对象中，有一类是描述数据对象的，如：IAccountEntity、IStorageEntity 等，用它们描述数据，还有一类接口对象，是专职数据库操作的，例如：IAccountManager、IStorageManager；“和欣”智能手机软件整体解决方案中，使用的嵌入式数据库 ElaDB，ElaDB 的线程模型限定了一个数据库的连

接只能一个线程使用，不能在线程间传递使用。对于数据对象，一般不能定义为单实例，因为它们会在同一时刻存在不同的实体，而数据库管理的接口，一般将其定义为单实例，这样做，可以保证数据库操作对象的唯一性，方便使用者使用并可以减少无谓的内存损耗，同时方便对接口的加锁操作。

另一类，是针对手机应用的特殊需求的，例如应用存在网络 Socket 通信以及与数据库引擎存在存取交互的情况，此时为了提高 Socket 通信的效率，该通信过程是实行单线程控制的，也就是对于同一个应用线程，即使用户发出多个连接请求，这多个连接请求也只可映射到单一应用组件中去。例如：在邮件应用中，接收是通过 IPop3 构件接口对象完成的，也就是说，此时的 IPop3 是一个特殊的构件接口对象，必须保证其在系统中只存在一个实例，这样才能确保它的逻辑正确性以及良好的效率。

以上两种情况是手机引擎中很常见的单实例情况。

5.1.2 通常的解决方法

通常对单实例问题的解决方法是可以用一个全局变量使得一个对象可以被访问，但它不能防止你实例化多个对象。

5.1.3 Singleton 设计模式解决方案

Singleton 模式具有可以实现对唯一实例的受控访问；使用这种模式，可以缩小名空间，避免那些存储唯一实例的全局变量污染名空间的情况发生；允许对操作和表示的精化；允许可变数目的实例并具有比类操作更灵活的特点。

如何保证构件接口的实例对象只有一个且这个实例易于被访问？一个更好的方法是让构件对象自身负责保存它的唯一实例，由此保证没有其他实例可以被创建，并且这个构件对象提供一个访问该实例的方法，这就是 Singleton 模式。图 5.1 为构件化的 Singleton 模式图

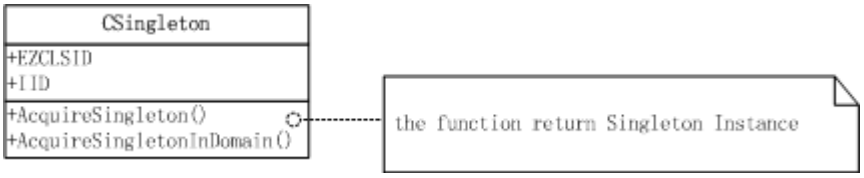

图 5.1 构件化 Singleton 模式图

从图中可以看出，在采用 CAR 构件实现的 Singleton 模式中，生成 Singleton 对象有两种方式，分别是：

```
static ECODE AcquireSingleton(
                    /*[out]*/ ISingleton **pISingleton)
static ECODE AcquireSingletonInDomain(
                    /*[in]*/    PDOMAININFO pDomainInfo,
                    /*[out]*/ ISingleton **pISingleton)
```

两种实现的区别其实就是接口对象域是否相同，本文在 6.1.4 中会详细说明它们的具体实现。

5.1.4 模式应用实现

前面的叙述表明“和欣”智能手机应用是面向 CAR 构件的软件开发，这里的 Singleton 模式实现和传统的面向对象的 Singleton 模式有所区别，下面给出基于 CAR 构件的 Singleton 模式实现细节：

```
static ECODE AcquireSingleton(
                    /*[out]*/ ISingleton **pISingleton) {
    return EzCreateObject(CLSID_CSingleton,
                        CTX_SAME_DOMAIN,   //同一 DOMAIN
                        IID_ISingleton,
                        (POBJECT*)pISingleton);
}
```

```
static ECODE AcquireSingletonInDomain(
                    /*[in]*/    PDOMAININFO pDomainInfo,
                    /*[out]*/ ISingleton **pISingleton) {
    return EzCreateObject(CLSID_CSingleton,
                        pDomainInfo, //可以自己选定的 DOMAIN
                        IID_ISingleton,
                        (POBJECT*)pISingleton);
}
```

EzCreateInstance 是创建对象时使用的。用 EzCreateObject 以指定的 CLSID 创建一个未初始化的类对象；这个类对象可是是单件模式的，当然也可以是非单件模式的。

下面以 IPop 接口（单件）为例，说明如何在 CAR 构件中实现 Singleton 模式。

```
CARAPI _CPop3CreateObject(IObject **ppObj) {
    _s_CPop3_spinlockObj_.Lock();
BEGIN：
    switch (_s_CPop3_ObjState_) {
    case _SingletonObjState_Uninitialize:
        pNewObj = (CPop3 *)new(pLocation) _CSandwichCPop3;
        pNewObj->m_lRef++;
        pNewObj->_Initialize_();
        _s_CPop3_ObjState_ = _SingletonObjState_Initialized;
        break;

    case _SingletonObjState_Initializing:
```

```
        do {
            EzSleep(1, NULL);
            if (_SingletonObjState_Uninitialize == _s_CPop3_ObjState_) {
                _s_CPop3_spinlockObj_.Unlock();
                goto BEGIN;
            }
        } while(_SingletonObjState_Initialized != _s_CPop3_ObjState_);
        /* Don't break here */

        case _SingletonObjState_Initialized:
            _s_pCPop3_ObjPtr_->m_lRef++;
            pObj = (_IObject *)_s_pCPop3_ObjPtr_;
            break;
    }

    *ppObj = (_IObject*)pObj;
    }
```

5.1.5 效果

使用 Singleton 模式，可以很好的管理构件接口对象的单实例及多实例情况，构件服务器端清楚构件的情况，可以针对不同的情况及状态提供单实例或多实例构件对象，而构件客户端在调用时，无须关注起改变，仍然可以正常调用。这样的做法，将改变隔离在构件服务器端，提高了软件的灵活性，满足了不同的构件接口需求。

5.2 信箱管理实现与 Factory Method 设计模式

5.2.1 问题的提出

智能手机的数据信息类应用都涉及到数据的保存，在“和欣”智能手机应用开发设计中，目前针对数据类业务的应用主要是：邮件应用，短信、小区广播应用，彩信应用，还将开发快讯，WAP 信息应用等，这些应用的数据存储都是基于 ElaDB 数据库的，存储都是通过引擎提供的信箱管理接口实现的。为了保持 MMI 设计的一致性，针对这些应用的信箱都包括：收件箱、发件箱、草稿箱、已发信息/邮件箱、已收信息/邮件箱、已删信息/邮件箱、垃圾箱、保密箱、用户自定义信箱等。这样保证了它们的用户界面呈现是非常相似的，但是在实现中，由于业务数据的实际差异，相应的数据库表结构也会有所差异，数据细节不同又产生了具体实现的差异。同时，针对不同的业务需求，又会对这些数据类业务进行删减，重组，分离和结合，综合以上几点因素，为信箱管理引擎设计合理的架构，让其高效，灵活的工作是很有必要的。

5.2.2 通常的解决方法

通常的解决方案，可以为每个应用重复开发信箱管理应用来达到实现灵活的要求，这会使得代码过度冗余；也可以采用抽象集成来减少代码的重复，这会使得代码过度耦合，这两种方式都无法达到高效，灵活的设计要求。

5.2.3 Factory Method 设计模式解决方案

Factory Method（工厂方法）定义一个用于创建对象的接口，让子类决定实例化哪一个类。面对一个经常变化的具体类型，紧耦合关系会导致软件的脆弱。Factory Method 使得一个类的实例化延迟到子类。这种设计模式主要用于隔离类对象的使用者和具体类型之间的耦合关系，由此解决了“单个对象”的变化。

其实 Factory Method 设计模式是基于不断的代码重构而提出的。

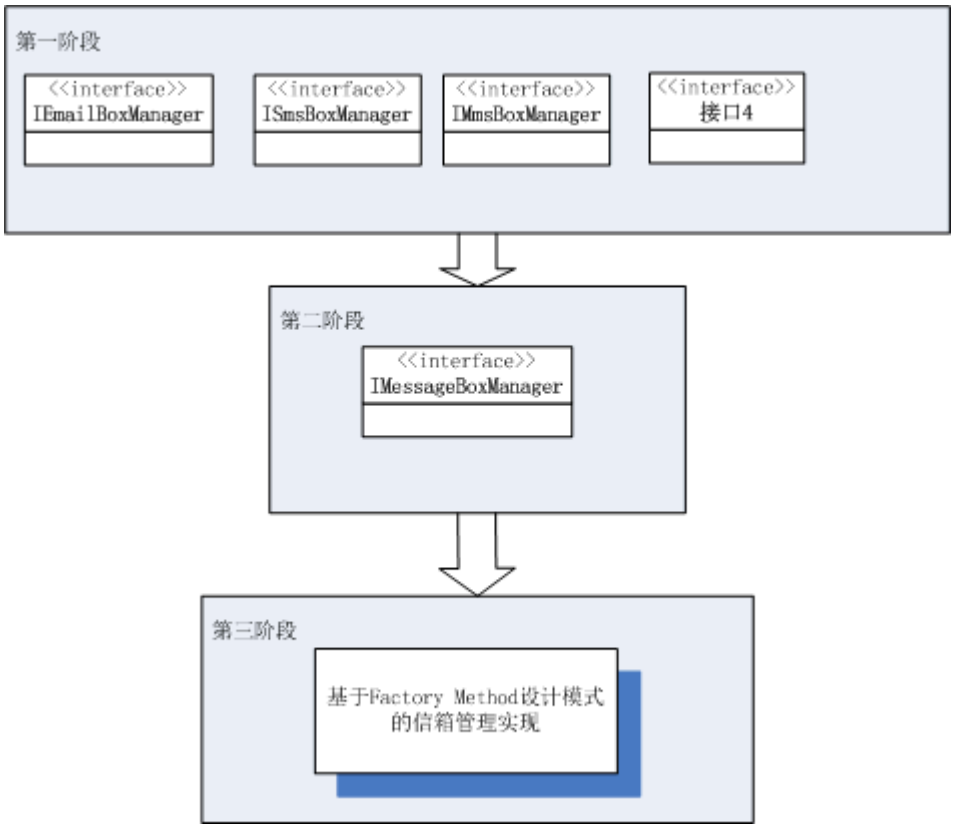

图 5.2 信箱管理模块开发重构阶段图

从上图可以看出，对于信箱管理模块软件架构的设计，是有一个演化过程的。

首先开发者应当明确，信箱管理是手机数据业务类软件一个不可或缺的组成部分，运用合理的模式实现这个模块会为应用开发、维护以及代码重用奠定良好的基础。

本人之前参与过短信/彩信应用开发，在开发中就有信箱管理这个模块应用，在开发邮件应用的过程中，又一次遇到了类似的开发需求，且由于要保证这些应用 MMI 的一致性，在同一款手机中，这些应用的信箱管理非常类似。如果在每一个应用开发中都开发这一模块，将造成代码重复实现。就好像图中的阶段 1，采用为每一个需求应用单独开发一个信箱管理引擎的方式，这带来了代码的重复冗余。

尝试通过分析这些应用信箱管理的相似性，将类似的业务进行合并和抽象到 IMessageBoxManager 接口来统一实现，即图中的阶段 2。在此，专门抽象了

Interface IMessageBoxManager，这也是后面再次进行抽象的依据。

表 5.1 IMessageBoxManager 接口描述

方法	方法描述
EnumByType([in]BoxType eType, [out]IObjectEnumerator**ppIEntityEnum);	通过枚举信箱类型，获得不同的信箱实体
Create([out]IMsgBoxEntity **ppIBoxEntity);	在信箱中创建新信箱对象
Add([in]IMsgBoxEntity *pIBoxEntity);	在信箱中为新信箱添加具体信息
Update([in]IMsgBoxEntity *pIBoxEntity);	更新信箱信息
DelById([in]INT nBoxID);	删除信箱信息
GetById([in]INT nBoxID, [out]IMsgBoxEntity **ppIBoxEntity);	按信箱 ID 号获取信箱实体
GetMsgCountByStatus([in] BoxType eType, [in] MessageStatus uMsgStatus, [in] EzArray<INT> eaExceptBoxId, [out] INT * pnMsgCount);	查询数据库中某种类型信息，某种信箱的信息数
IsNameDuplicate([in]BoxType eType, [in]WString wsBoxName, [out]BOOL *pbDuplicate);	查询是否重名
SetViewMode([in]BoxType eType, [in]BoxViewmode eViewMode);	设置显式方式

此时，在程序中是通过结构化的分离来实现各个模块的需求的，

首先要给出信箱应用的所有类型：

```
enum BoxType {
        BoxType_Sms = 0x0,
        BoxType_Mms,
        BoxType_CB,
        BoxType_Wap,
        BoxType_Email,
        BoxType_Message          //include sms,mms and wap
}
```

然后通过每一个具体的方法进行区分，实现不同的代码，这样的确降低了代码的冗余度，但是却大大提高了代码的耦合度。在这个阶段中，不同信息的信箱管理是通过在 IMessageBoxManager 的每一个具体方法的实现过程中，以结构化的方式来实现区分的，显然，这个架构也不够合理。

但是开发者应当意识到，仅仅从一个代码耦合的高度来看待信箱管理引擎设计的问题是远远不够的，目前，手机业务定制的灵活性已经成为手机终端软件开发商非常关注的问题之一，本人在软件开发过程中，曾遇到过要应用定制的问题，如果信箱管理针对不同应用的代码耦合度太高，将导致需要删减应用时无法完全删除对应应用的实现；当需要增加一个新的信息应用时，需要改动的代码太多而带来软件性能稳定性下降；当需要将某几个应用合并在一起时，更加无法达到快速、高效的整合。

依据上面的分析，引入 Factory Method 模式，为了达到信箱管理功能多样性及配置灵活性，通过抽象来实现多样性，通过松耦合来实现灵活性。也就到达了图中的阶段 3。

5.2.4 模式应用实现

下面给出基于 Factory Method 模式的信箱管理应用的具体实现。

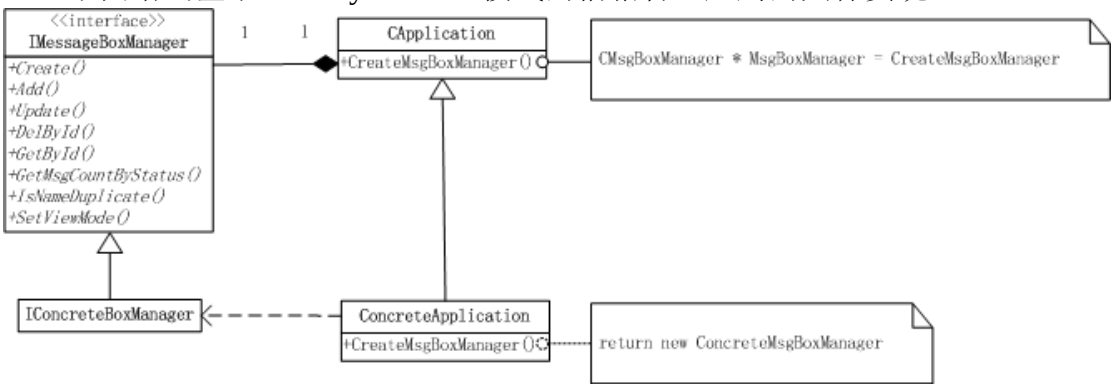

图 5.3 采用 Factory Method 的解决方案

如图 5.3 所示，具体做法是将 IMessageBoxManager 定义为一个抽象接口，在这个抽象接口中定义出对于信箱管理所必需的接口方法。对于需要信箱管理的应用可自行定义出一个实际应用的信箱管理接口，例如：IEmailBoxManager；由该接口实现具体的方法。

采用 Factory Method 模式对手机应用实际调用的信箱管理接口进行封装，如图 5.3 所示，在 IMessageBoxManager 中给出 CreateMsgBoxManager()虚方法，该方法会被实际应用信箱管理接口继承，并实现。此时，具体应用调用时，可以通过不同的实现类调用到不同的实现对象。其具体实现如下：

```
ECODE CEmailBoxCtrl::CreateMsgBoxManager(
    /* [out] */ IObject **ppIEntity) {
      ……
      CEmailBoxManager *pEntity = NULL;
    CEmailBoxManager::New(&pEntity);
    ……
    ::IObject::QueryDefault((_IObject*)pEntity, ppIEntity);
    pEntity->Release();
}
```

5.2.5 效果

使用 Factory Method 模式，使实现具有良好的框架，采用这种方式的信箱管理，其优点在于：

第一、可以很方便的扩充新的信箱管理业务，例如 WAP 信息管理，CB 信息管理等；

第二、应用的耦合具有很高的灵活性，可以随意合并和拆分信息应用；

第三、这种方式也便于应用定制。

5.3 数据库数据操作与 Interator 设计模式

5.3.1 问题的提出

在智能手机软件应用开发中，存在大量和数据库的交互操作，例如，对于

智能手机邮件应用就包括：邮件信息，手机用户的邮件帐户信息，用户邮件签名设置信息，用户邮件设置信息等等。

在面向对象的软件设计过程中，数据库表中承载的也是经过抽象的对象。如果是一个基本数据类型，例如整形 INT，若要从数据库中查找出一系列的数据，并要对其进行遍历操作时，很容易想到使用一个整形数组或是一个整形链表来实现；在面对的是抽象的数据对象时，这样的方式显得不合时宜，此时，设计一个容器来承载数据对象族，实现对对象的遍历，访问，在顺序访问一个聚合对象中各个元素时，也不会暴露该对象内部的表示。

5.3.2 通常的解决方法

通常的解决方法是将数据模型抽象成一个类，然后采用对类的基本操作方法来操作单个对象。对于一组对象则将对象用数组或者链表的方式进行存放，来实现查找，遍历。采用这样的解决方案，其缺点在于，首先对于单个类的操作，没有合理的封装数据，其次，对于采用数组或链表的方式，其枚举过程比较繁琐。

5.3.3 Iterator 设计模式解决方案

Iterator（迭代器）模式提供了一种顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。通过使用 Iterator 模式，可以支持以不同的算法遍历一个聚合，迭代器可以简化聚合的接口并且支持在同一个聚合上实现多个遍历。

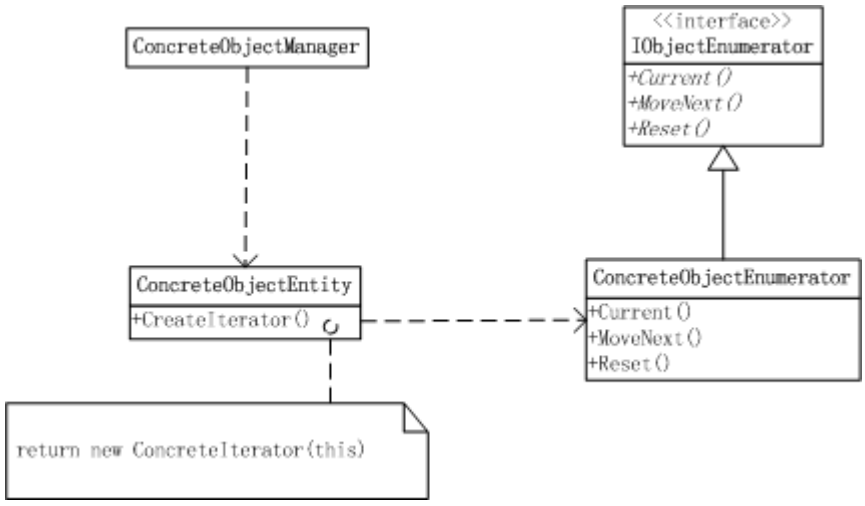

图 5.4 采用 Iterator 设计模式的解决方案

5.3.4 模式应用实现

提供一个对象枚举器接口 IObjectEnumerator，

IObjectEnumerator::Current() 获取枚举器中的当前对象

IObjectEnumerator::MoveNext() 设置下一个对象为当前对象

IObjectEnumerator::Reset() 重置枚举器为初始状态

设计一个抽象的对象枚举器接口 IObjectEnumerator，并为这个抽象接口设计了三个虚方法，分别是：用于获取枚举器中的当前对象的 Current()方法；用于设置下一个对象为当前对象的 MoveNext()方法和用于重置枚举器为初始状态的 Reset()方法。然后再在涉及到各类数据库对象操作的地方实现这些方法。

这里以智能手机邮件应用中的实现为例进行说明，在该应用中涉及到的实际对象包括 CMail 邮件对象，CSignature 邮件用户签名对象，CAccount 邮件用户帐户对象等，它们的对象操作就是不相同的。需要分别实现对应的 Current()、MoveNext()和 Reset()方法。下面表格给出三个迭代器对象的 Current 方法实现细节中的差异：

表 5.2 不同迭代器对象的 Current 方法

Current 方法	实际不同对象的 SetupEntity 方法
ECODE CEmailEnumerator::Current(/* [out] */ IObject ** ppIObject)	ECODE CEmailEnumerator::SetupEntity(/* [out] */ IObject **ppIEntity)

{
……
ec = SetupEntity(ppIObject);
……
}
	{
	……
	CStorageEntity * pEntity = NULL;
	CStorageEntity::NewByFriend(&pEntity);
	pEntity->SetID(nMailID);
	pEntity->SetSubject((WString)wsbSubject);
	pEntity->SetDate(nDateTime);
	pEntity->SetFromTo((WString)wsbFromTo);
	pEntity->SetSize(nSize);
	pEntity->SetStatus(uStatus);
	pEntity->SetPriority(uPriority);
	
	::IObject::QueryDefault((_IObject*)pEntity,
	ppIEntity);
	……
	}

ECODE CSignatureEnumerator::Current(
/* [out] */ IObject ** ppIObject)
{
……
ec = SetupEntity(ppIObject);
……
}
	ECODE CSignatureEnumerator::SetupEntity(
	/* [out] */ IObject **ppIEntity)
	{
	……
	CSignature * pSignature = NULL;
	CSignature::NewByFriend(&pSignature);
	pSignature->SetID(nSignID);
	pSignature->SetContent(wsbContent);
	ISignature::Query(pSignature,
	(ISignature**)ppIEntity);
	……
	}

ECODE CAccountEnumerator::Current(
/* [out] */ IObject ** ppIObject)
{
……
ec = SetupEntity(ppIObject);
……
}
	ECODE CAccountEnumerator::SetupEntity(
	/* [out] */ IObject **ppIEntity)
	{
	……
	CAccountEntity *pEntity = NULL;
	CAccountEntity::NewByFriend(&pEntity);
	pEntity->SetID(nAccountID);
	pEntity->SetNickName((WString)wsbNickName);
	pEntity->SetUserName((WString)wsbUserName);
	pEntity->SetPassword((WString)wsbPassword);
	pEntity->SetSmtpServer((WString)wsbSmtpServer);
	pEntity->SetPopServer((WString)wsbPopServer);

	pEntity->SetSmtpPort(nSmtpPort);
	pEntity->SetPopPort(nPopPort);
	pEntity->SetIsDefault(bDefault);
	pEntity->SetReplyTo((WString)wsbReplyTo);
	pEntity->SetAutoDel(bAutoDel);
	pEntity->SetUseSSL(bUseSSL);
	pEntity->SetUseAPOP(bUseAPOP);
	pEntity->SetAddress((WString)wsbAddress);
	pEntity->SetUseSmtpAuth(bUseSmtpAuth);
	
	::IObject::QueryDefault((_IObject*)pEntity,
	ppIEntity);
	……
	}

对于邮件帐户管理和邮件管理由于比较复杂，这里各抽象出两个接口，Interator 接口主要针对 IEntity 接口，实现单一对象的访问，修改等操作，而 IManager 接口主要为管理多个对象提供的。

5.3.5 效果

使用 Iterator 模式，将对数据库表的访问和遍历用单独的迭代器对象实现，从而简化了对数据库数据操作的复杂度，同时，每一个迭代器对象可以保持其自身的遍历状态，这样可以进行多个迭代，提高了效率。

5.4 逻辑控制层与用户交互层通信与 Observer 设计模式

5.4.1 问题的提出

本文讨论智能手机软件架构采用的是基于 CAR 构件的 MVC 模型。上文也详细描述了手机引擎层，逻辑控制层和用户交互层的结构，其中层间通信采用构件间调用和回调实现构件间的通信。现在的问题在于，如何保证消息在层间传递的合理性，即一个构件接收的信息如何合理传递给需要响应的构件而不是

所有的构件。

5.4.2 通常的解决方法

一般来说，若手机引擎层中对象事件的发生将导致逻辑控制层中某些对象事件产生响应，则需要在逻辑控制层对应对象中申明手机引擎层的对象，并调用对象方法完成对应功能。与此类似，若逻辑控制层中对象事件的发生将导致用户交互层中某些对象事件产生响应，则对应的在用户交互层对应对象中申明逻辑控制层的对象，并调用对象方法完成对应功能，这导致二层代码之间的强依赖关系，由于代码的紧耦合而导致代码不稳定。

5.4.3 Observer 设计模式解决方案

Observer（观察者）模式是一种对象行为型模式，它定义对象间一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

Observer 模式描述了如何建立这种关系，这一模式中的关键对象是目标（subject）和观察者（observer）。一个目标可以有任意数目的依赖它的观察者。一旦目标的状态发生改变，所有的观察者都得到通知。作为对这个通知的响应，每个观察者都将查询目标以使其状态与目标的状态同步。

这种交互也称为发布-订阅（publish-subscribe）。目标是通知发布者。它发出通知时并不需要知道谁是它的观察者，可以有任意数目的观察者订阅并接收通知。

那么，手机引擎层在收到数据模型变化的信息之后，将会通过回调的形式通知逻辑控制层关注这条信息的应用模块，然后逻辑控制层在通知到用户交互层，由于并非所有的应用都关注相同的到来的信息，要采用怎样的方法，让逻辑控制层可以将消息正确、高效的分发出去呢？这个需求和 Observer（观察者）模式的功能不谋而合了。此时这一模式中的关键对象目标（Subject）就是手机逻辑控制层要分发消息的对象，而观察者（observer）就是用户交互层要得到逻辑控制层的消息的那些对象。所以选用 Observer（观察者）模式来实现手机用户交互层和逻辑控制层之间的通信。

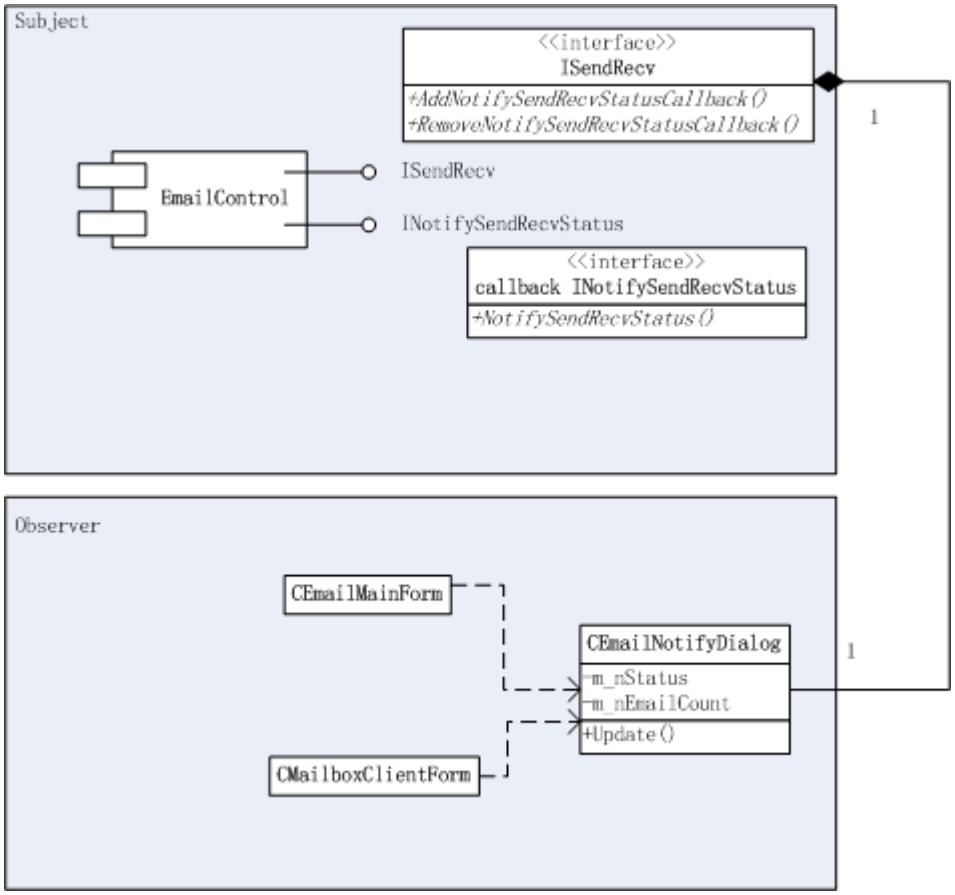

图 5.6　采用 Observer 设计模式的解决方案

5.4.4 模式应用实现

邮件应用的 Observer（观察者）设计模式采用基于 CAR 构件的 CALLBACK（回调）机制来实现的。

（1）什么是回调机制？

软件模块总是存在着一定的接口，以提供相互之间进行调用。从调用方式上，可以把它们分为三类：同步调用、回调和异步调用。同步调用是一种单向调用，调用方要等待对方执行完毕才返回；回调是一种双向调用模式，也就是说，被调用方在接口被调用时也会调用对方的接口；异步调用是一种类似消息或事件的机制，不过它的调用方向刚好相反，接口的服务在收到某种信息或发生某种事件时，会主动通知客户方（即调用客户方的接口）。回调又常常是异步调用的基础。

回调，讲的更具体一点，就是模块 A 调用模块 B，而模块 B 中又存在调用模块 A 中的一个函数 c 的代码，下图所示的形式就叫回调，其中的函数 c 就是回调函数：

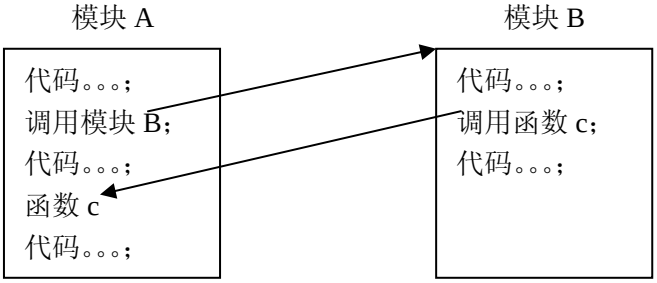

图 5.7 回调示意图

（2）回调的具体的作用？

如以上模型，A 要调用 B，可是又希望 B 在处理过程中用到 A 中的某些信息，具体是什么信息，怎么处理这些信息，都由 A 决定。而 B 只要知道调用 A 中的函数 c 能得到这方面的信息就行，函数 c 怎么实现就不用管了。

其实 B 不一定由 A 来调用，当 B 所在的系统发生某件事件时，系统会自动调用 B 并通过 B 模块来调用由 A 定义的函数 c，起到了 B 对 A 的通知作用。

（3）CAR 构件的回调机制

在 CAR 构件技术里，支持一个或多个回调接口的 CAR 构件对象叫做可连接对象。可连接对象支持两种接口，一种是普通接口，它里面的成员函数由服务器端自己实现；另一种就是回调接口，其中每个成员函数代表一个事件（event），处理每个事件的函数由客户端实现。当特定事情发生时，如定时消息或用户鼠标操作发生时，构件对象产生一个事件，客户程序可以处理这些事件。

Microsoft 提供的可连接对象技术需要用户去实现客户程序与构件对象的连接、事件的激发、接收器的编写等，而且只能以接口为单位注册，即不能为接口中每个成员方法分别注册。在 CAR 的回调事件机制中，客户程序可以单独注册事件的某一个事件处理函数，大大简化了编程。

在 CAR 的回调机制中，也有一种接收器(sink)对象，它的方式和作用都与

前者有区别。CAR 里的接收器对象相当于一个客户端回调函数的容器，在客户端的地址空间里，负责与可连接对象进行通信。只要有可连接对象存在，那么在客户端肯定要有接收器的存在。多个回调接口可对应一个接收器对象，这样可以减少通信花费的开销。当接收器与可连接对象建立连接后，客户程序可将自己实现的事件处理函数（回调函数）向接收器进行注册，而不是向可连接对象进行注册，又一次减少了通信开销。接收器会自动把在它里面注册的函数的回调接口指针告诉构件对象，构件对象在条件成熟时激发事件，如果客户注册了自己的回调接口方法，那么就会被调用，否则就调用接收器默认实现的回调接口方法。事实上，接收器是向客户程序屏蔽的，客户只需通过 AddXxxCallback 和 RemoveXxxCallback（Xxx 代表回调事件名）函数来注册和注销事件处理函数，完全不用关心接收器的存在。

在编写构件程序时，用户只需关心何时激发事件，而在编写客户端程序时，用户只需在适当的时候注册事件处理函数。其它的工作，如接收器对象的创建与实现、接收器与可连接对象建立通信的具体过程、事件的分发回调过程等都由 CAR 实现。这样，用户在编写具有回调接口的构件和编写使用该构件的客户端程序都会变得相当简单。下图就是 CAR 构件的回调机制模型：

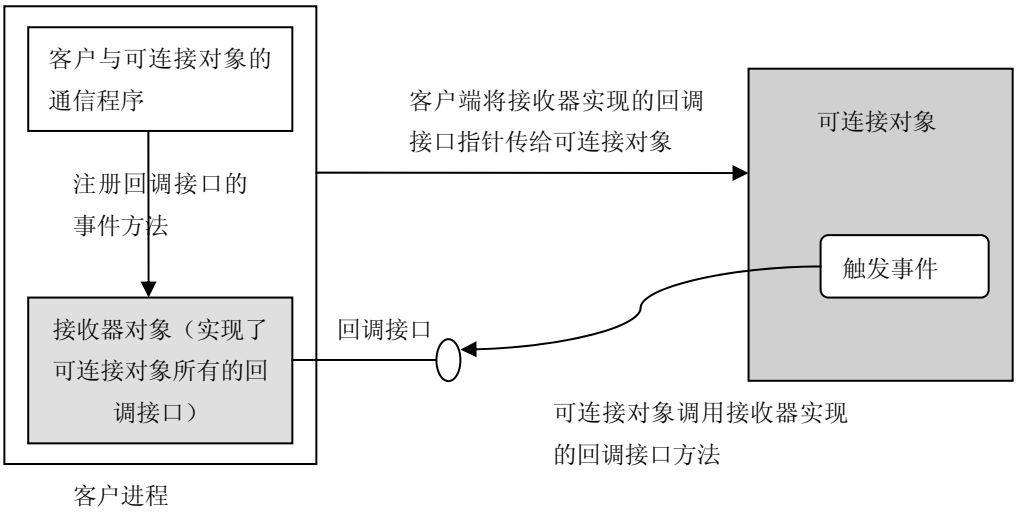

图 5.8 CAR 构件的回调机制模型

在上图的回调机制模型里，接收器对象是客户进程里一个构件对象，它默

认实现了可连接对象所支持的所有的回调接口，客户程序可向接收器对象注册客户自己的回调接口的事件方法。可连接对象是接收器的客户，在触发回调接口事件的时候，如果客户端注册了自己的回调接口方法，那么就会被调用，否则就调用接收器默认实现的回调接口方法。客户端在与可连接对象建立通信后，客户端将接收器实现的所有回调接口指针传给了可连接对象，可连接对象触发事件就是通过回调接口指针调用回调接口方法。

（5）运用 CAR 的 CALLBACK 机制的 Observer（观察者）模式的实现

下面给出邮件在发送和接收过程中的存在多种变化：

```
enum NotifySendRecvStatus {
        NotifySendRecvStatus_Connect            = 0,
        NotifySendRecvStatus_Connect_Failed,

        NotifySendRecvStatus_Send_Status = 0x10,
        NotifySendRecvStatus_Send_Login,
        NotifySendRecvStatus_Send_LoginFailed,
        NotifySendRecvStatus_Sending,
        NotifySendRecvStatus_Sent_Succeed,
        NotifySendRecvStatus_Sent_Failed,

        NotifySendRecvStatus_Recv_Status = 0x100,
        NotifySendRecvStatus_Recv_Login,
        NotifySendRecvStatus_Recv_LoginFailed,
        NotifySendRecvStatus_Recv_LoginSucceed,
        NotifySendRecvStatus_Receiving,
        NotifySendRecvStatus_Recv_MaxLimited,
```

```
        NotifySendRecvStatus_Received_Succeed,
        NotifySendRecvStatus_Received_Failed,

        NotifySendRecvStatus_Storage_Full = 0x200
}
```

那么上面的这些变化需要及时地通知到用户交互层，并让用户可以做出及时地响应，例如，成功收到新邮件，希望用户读取；信箱存储空间已满，新邮件无法正常存取，那么希望用户可以及时清理邮箱等等，这些信息都要由控制层及时分发到相应的用户交互层上去。

```
控制层：class CSendRecv
{
    constructor();
    interface ISendRecv;
        callback interface INotifySendRecvStatus; //这里就是回调接口
}
UI 层：CEmailNotifyDialog.cpp 中
ECODE CEmailNotifyDialog::Init()
{
// Event handler
CSendRecv::AddNotifySendRecvStatusCallback(m_pISendRecv,
            this, &CEmailNotifyDialog::OnNotifyMessage);//注册这个回调
}
```

注册了这个回调之后，消息就可以通过 OnNotifyMessage 这个方法的参数 nStatus 传递过来了。但此时会存在发送接收线程和主 UI 线程不是同一个线程的情况，在 Elastos 的图形系统中，目前无法支持在非图形线程中调用图形方法

的情况，因此需要通过调用 InvokeUserEvent()这个方法来转到主 UI 线程，即图形主线程中来。

```
ECODE CEmailNotifyDialog::OnNotifyMessage(
    /* [in] */ ISendRecv * pSender,
    /* [in] */ INT nStatus,
    /* [in] */ INT nNumber)
{
    return InvokeUserEvent(nStatus, nNumber);
}
```

接收 InvokeUserEvent 方法传递来的 nStatus 状态并对状态进行分析，处理的 OnUserEvent()方法：

```
ECODE CEmailNotifyDialog::OnUserEvent(
    /* [in] */ IForm * pSender,
    /* [in] */ INT nStatus,
    /* [in] */ INT nNumber) {
    ……
    switch (nStatus) {
        case NotifySendRecvStatus_Receiving:
            ……
            //邮件正在接收中
            ……
            break;
        case NotifySendRecvStatus_Received_Succeed：
            ……
```

```
                //邮件成功接收
                ……
                break;
            //多种状态分析
}
}
```

CEmailMainForm.cpp 中：

```
CEmailMainForm::OnReceiveMail {
CEmailNotifyDialog emailNotifyDialog; //接收
}
```

CMailboxClientForm.cpp 中：

```
CMailboxClientForm::OnSend(
/* [in] */ IObject * pSender) {
CEmailNotifyDialog emailNotifyDialog; //发送
}
```

CWriteMessageClientForm.cpp 中：

```
CWriteMessageClientForm::SendEmail() {
    CEmailNotifyDialog emailNotifyDialog; //重发，转发
}
```

使用完毕之后，要解除这个回调，

CSendRecv.cpp 中：

CSendRecv::RemoveNotifySendRecvStatusCallback(

m_pISendRecv,this, &CEmailNotifyDialog::OnNotifyMessage);

5.4.5 效果

使用 Observer 模式，用户应用交互层的对象通过注册逻辑控制层的变化回调，达到目标和观察者之间的抽象耦合，它们处于邮件应用服务的不同抽象层次，保证了目标（逻辑控制层的具体消息）和观察者（用户应用交互层对消息关注的对象）之间的耦合是抽象的、并且是最小的。

5.5 应用间交互与 Mediator 设计模式

5.5.1 问题的提出

智能手机一般都有大量的应用，图 5.9 和图 5.10 分别给出手机邮件应用和其他应用交互图和手机地址簿应用和其他应用交互图。

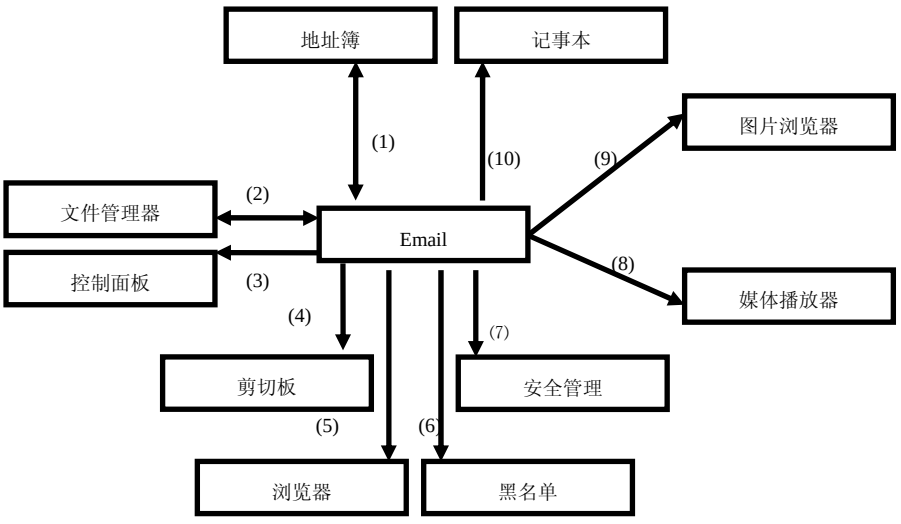

图 5.9 Email 与其它应用交互图

下面分析应用间的交互过程

（1） 从地址簿选择联系人发送 Email，保存 Email 发件人地址到地址簿。

（2） 从文件管理器选择文件，插入附件到 Email，Email 来信中包含对象保存到文件管理器。

（3） 收发 Email 前需要调用网络设置进行网络的连接操作

（4） 调用剪贴板模块进行邮件内容的剪贴、复制、粘贴操作。

（5） 通过浏览器连接 URL

（6） 将 Email 发件人设为 email 黑名单

（7） 调用安全模块的密码输入及验证界面

（8） 调用媒体播放器应用预览 Email 的附件

（9） 调用图片浏览器应用预览 Email 的附件

（10） 调用记事本应用预览 Email 的附件

如果单纯从邮件应用服务本身来看，可以利用邮件应用服务这个“中介者”来实现多应用之间交互的工作，但事实是否如此呢？

下面简单分析一下地址簿和其他应用的交互过程。如下图所示

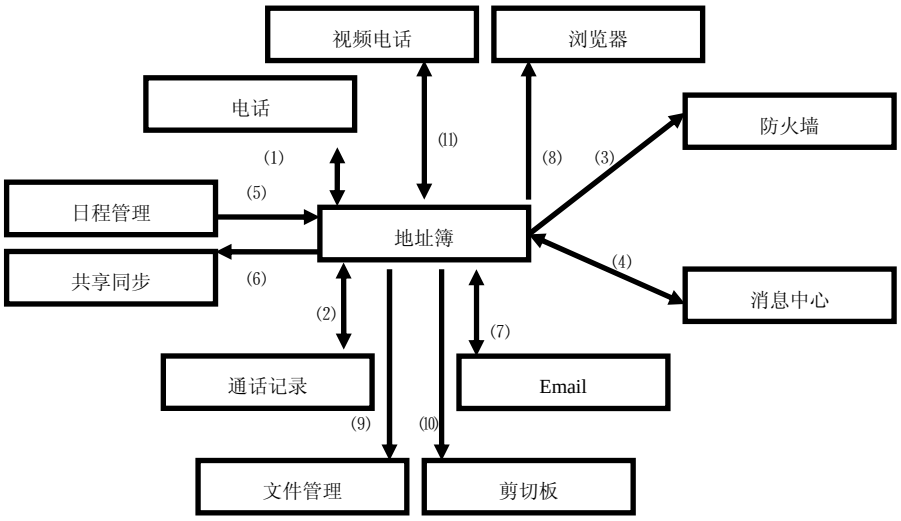

图 5.10 地址簿和其它应用的交互图

（1）从地址簿给联系人号码拨打电话；从电话中将通话号码保存到地址簿；

来电时从地址簿取出号码相对应的姓名显示出来。

（2）从地址簿取出号码相对应的姓名显示出来；将通话记录中的号码添加到地址簿

（3）将地址簿中的号码添加到防火墙

（4）从地址簿给联系人号码发送消息；消息中心从地址簿添加收件人；从消息中心将号码保存到地址簿；将发件人号码从地址簿取出号码相对应的姓名显示出来

（5）从地址簿添加联系人

（6）从地址簿发送联系人名片

（7）从地址簿给联系人号码发送 email；email 从地址簿添加收件人；从 email 将地址保存到地址簿；将发件人地址从地址簿取出地址相对应的姓名显示出来

（8）连接联系人主页

（9）从文件管理选取铃声及图片

（10）在文本输入时使用剪切板。

（11）从地址簿给联系人号码拨打视频电话；从视频电话中将通话号码保存到地址簿；视频电话来电时从地址簿取出号码相对应的姓名显示出来。

通过上面的分析可以看出，在智能手机应用中，每个应用都是其组成部分，如果在应用之间直接进行交互，并非不可以，只是这种交互是无序的，会导致应用之间的紧耦合，一个应用的需求变更会影响到所有和它交互的应用，按照隔离变化，构建松耦合软件架构的目的，这样的方式是不好的，因此，需要提供一个专门的“中介者”来实现应用与应用之间的交互。

5.5.2 通常的解决方法

通常来说，如果一个应用要使用到其它应用，那么将其它应用作为该应用中的一个成员来操作，这在应用间关系比较简单的时候，是比较有效的实现。但是，对于智能手机应用来说，它们之间存在着错综复杂的交互管理，此时采

用通常的解决方法则会使实现过于耦合，不利于代码的结构和管理。

5.5.3 Mediator 设计模式解决方案

在软件构建过程中，经常会出现多个对象相互关联交互的情况，对象之间常常会维护一种复杂的引用关系，如果遇到一些需求变更，这种直接的引用关系将面临不断的变化。在这种情况下，可以使用一个“中介对象”来管理对象间的关联关系，避免相互交互的对象之间的紧耦合引用关系，从而更好地抵御变化。这就是 Mediator（中介者）模式，它是一种行为型模式，它用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式的相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。下图给出了使用 Mediator 设计模式的智能手机应用间通信的解决方法。

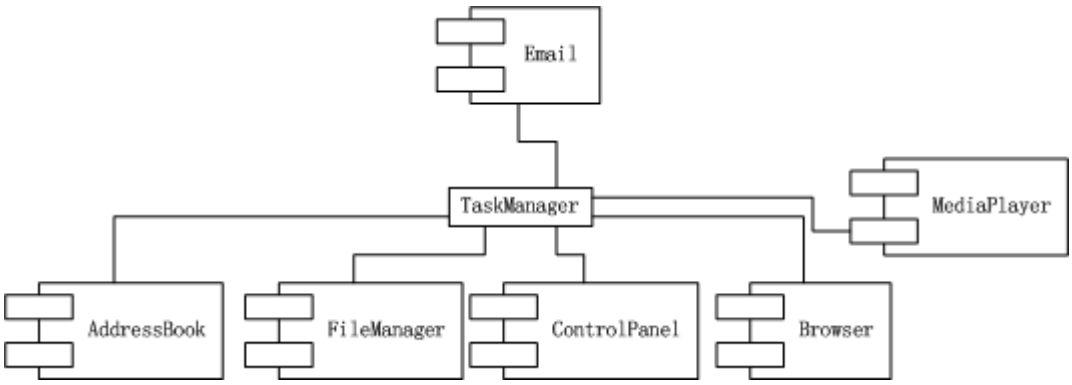

图 5.11　采用 Meidator 设计模式的解决方案

5.5.4 模式应用实现

在“和欣”智能手机应用开发中，抽象出一个 TaskManager 充当“中介”的角色，将应用的载体抽象成一个 Task 构件，其结构如下：

```
class CTask {
        constructor(
            WString wsAppName,
            IMainApplet* pIMainApplet,
```

```
        IEvent *pIReadyEvent);
    interface ITask;
    callback interface ITaskCallback;
}
```

将ITask构件与应用的状态，是否可以关闭等信息结合在一起，组成TaskInfo信息供 Taskmanger 使用，其结构如下：

```
typedef struct _TaskInfo {
    ITask * pITask;
    TaskState emState;
    BOOL bCanEnd;
} TaskInfo;
```

在 Taskmanager 中维护一个 TaskInfoArray 的队列，并由 TaskManager 来实现消息的分发。队列提供的方法主要包括：

ITaskMgr 接口中：GetTasks(ITask);

ActivateTask(ITask);

EndTask(ITask);

GetTaskCount(INT);

消息分发给各个应用，一般来说，这些消息可能就是其他应用的一个信号，而 TaskManager 这个中介实现了应用间消息的分发和多应用交互的管理。

例如，在邮件应用中，CEmailApp.cpp 是邮件应用 UI 层的切入点，在 CEmailApp.cpp 中：

```
ECODE CEmailApp::AtEntry( //切入方法
/* [in] */ EzArray<WString> argcv) {
    ……
```

```
        SpAcquireMyTask((_IObject*)this, &m_pITask);
                        //将邮件应用加入 TaskManager 队列当中去
        CTask::AddRequestServiceCallback(m_pITask,
                                        this, &CEmailApp::RequestService);
                        //并注册 TaskManager 的回调事件，允许其接收来自其他应用的信息
        ……
    }
```

和之前讨论的一样，由于“和欣”图形系统的要求，这里要有一个转线程的操作，这个操作是通过“和欣”编程模型中自定义的具有自描述性质的数组 argcv 的判断来实现的，若传入数组中只有一个数据，则表示 TaskManager 启动应用，若数组中有多个参数，则表明是由其他应用调入邮件应用，在邮件应用内部，由 CEmailMainForm 再做进一步处理，具体代码分析如下：

```
ECODE CEmailApp::RequestService(
        /* [in] */ ITask* pSender,
/* [in] */ EzArray<WString> argcv) {
        ……
        if (argcv.Used() == 1) {
    ec = m_pEmailMainForm->InvokeUserEvent(EmailRequestService_Show, 0);
        }
        else {
                ec = m_pEmailMainForm->ParameterPass(argcv);
        }
        ……
}
```

我们简单看看 CEmailMainForm.cpp 中 ParameterPass 方法的一些实现：

```
ECODE CEmailMainForm::ParameterPass(
    /* [in] */ EzArray<WString> argcv)
{
    ……
    INT used = argcv.Used();

    m_eaService = EzArray<WString>::CreateObject(used);
            //利于自描述数组建立一组应用对象
    if (used == 1) {
        return m_pForm->Show();
    }

for (INT i = 0; i < used; i++) {
        ……
        m_eaService[i] = _wcsdup((WCHAR*)(argcv[i])); //传入应用名
        ……
}
ECODE CEmailMainForm::Show()
{
    InvokeUserEvent(); //转到主 UI 线程做操作
}

ECODE CEmailMainForm::OnUserEvent(    //在主 UI 线程做操作
```

```
        /* [in] */ IForm* pIObject,
        /* [in] */ INT nIndex,
        /* [in] */ INT nParam){
……
        nServiceType = _wtoi(m_eaService[1]);
        //完成对应的操作
        ……
}
```

最终在构件使用完，析构时要解除注册的回调事件。

```
ECODE CEmailApp::AtExit()
{
        ……
CTask::RemoveRequestServiceCallback(m_pITask, this,
                                                    &CEmailApp::RequestService);
        ……
}
```

5.5.5 效果

使用 Mediator 模式，形成应用间交互的解耦。各个应用的松散耦合，对于应用本身需求的改变以及中介者对象实现形式的变化都有很好的实现效果。

第 6 章 构件化智能手机邮件应用的实现与模式应用

6.1 构件化智能手机邮件应用概述

6.1.1 邮件系统的原理图

图 6.1 邮件系统原理图

上图显示的是 Email 从发送到接收的一个双向的典型过程，其中 SMTP 和 POP3 服务器是服务器软件，它们运行在邮件服务器上，在发送 Email 的时候，Sender Client 通过 SMTP 协议与 SMTP Server 通信，经 SMTP Server 将用户邮件写入用户邮箱。而 Receiver Client 则通过 POP3 协议与 POP3 Server 通信，经 POP3 Server 将用户邮箱中的邮件取回。

6.1.2 邮件系统总体结构图

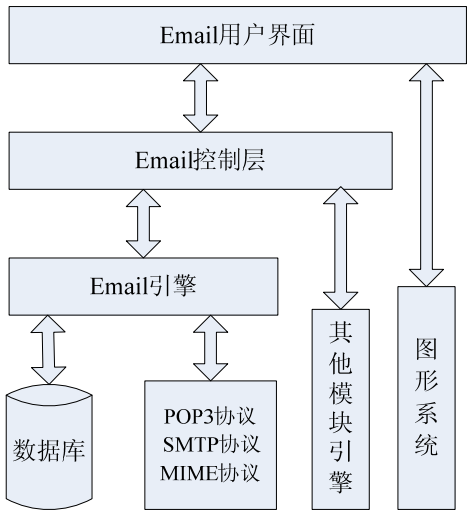

图 6.2 邮件系统总体结构图

从上图可以看到，整个系统大体可以分为四个部分： UI 层、控制层、引擎、协议。

上层 UI：主要负责与用户的交互功能，如邮件的收发、读取及相关设置等。

Email 控制层：主要实现邮件夹、邮件、账号、邮件过滤等信息的管理，并为 UI 提供功能接口。

Email 引擎：主要实现邮件的收发和编解码操作，以及对邮件、邮箱、账号等信息的数据操作功能。

协议：主要实现 POP3、SMTP、MIME 协议，及与邮件服务器的通信。

6.1.3 邮件系统功能图

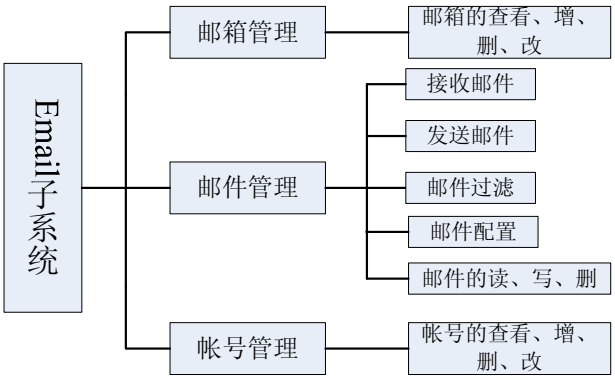

图 6.3 邮件系统功能图

邮件系统的功能主要有三大类，邮件管理，邮箱管理和帐号管理。其中，邮件管理是整个邮件系统的核心模块，包含了邮件的发送和接收，邮件的编辑、读写、删除及配置等；

6.1.4 邮件数据存储

Mail 部分的数据从整体上主要分为两类：邮件原始信息、解析后的邮件信息、附件信息和其他信息，其他信息包括邮箱信息、账号信息、设置信息等。

因为数据上的这两种划分，所以对数据的操作与保存方式也有所不同。对于邮件信息，在接收时，对原始信息进行部分解码，提取出部分信息供用户浏

览时用，然后将邮件的原始信息保存到数据库中，只是在用户打开该邮件时再进行解析；发送邮件时，保存部分供用户浏览的信息和打包后的邮件信息到数据库中。这样只有用户使用该邮件时才进行解析，同时也可以很好的保持数据的一致性。对于其他信息，也保存到数据库中。

6.1.5 接收邮件的业务流程

对于 Email 子系统，最主要的功能就是收发邮件，其他的功能都是围绕着这点展开的，接收邮件的流程如图 6.4 所示：

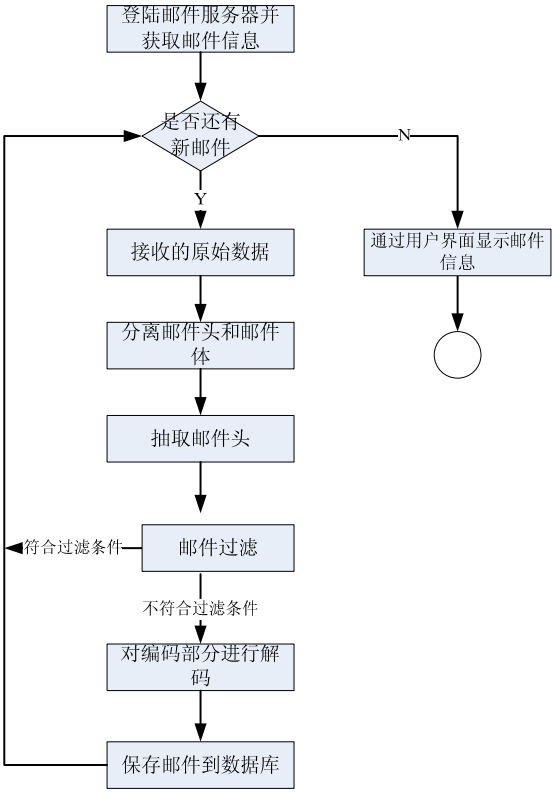

图 6.4 接收邮件的业务流程

6.1.6 发送邮件的业务流程

Mail 部分的数据从整体上主要分为两类：邮件原始信息、解析后的邮件信息、附件信息和其他信息，其他信息包括邮箱信息、账号信息、设置信息等。

因为数据上的这两种划分，所以对数据的操作与保存方式也有所不同。对

于邮件信息，在接收时，对原始信息进行部分解码，提取出部分信息供用户浏览时用，然后将邮件的原始信息保存到数据库中，只是在用户打开该邮件时再进行解析；发送邮件时，保存部分供用户浏览的信息和打包后的邮件信息到数据库中。这样只有用户使用该邮件时才进行解析，同时也可以很好的保持数据的一致性。对于其他信息，也保存到数据库中。业务流程如图 6.5 所示。

6.1.7 阅读邮件的业务流程

在智能手机邮件应用的邮件阅读过程中，包含几个主要的步骤：从数据库中读取邮件信息；解析邮件系统并显示邮件信息；如果用户要阅读附件，则需要解析邮件的附件，将解析出的文件保存到临时目录中去并调用对应的应用查看附件内容。如图 6.6 所示。

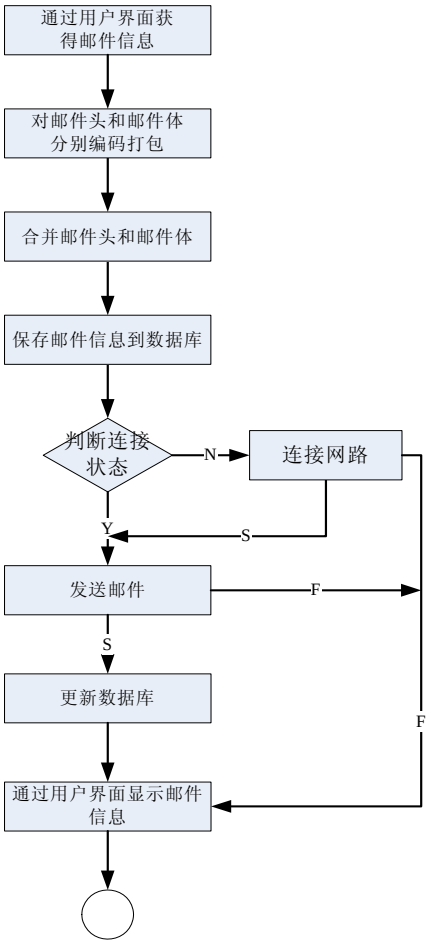

图 6.5 发送邮件业务流程

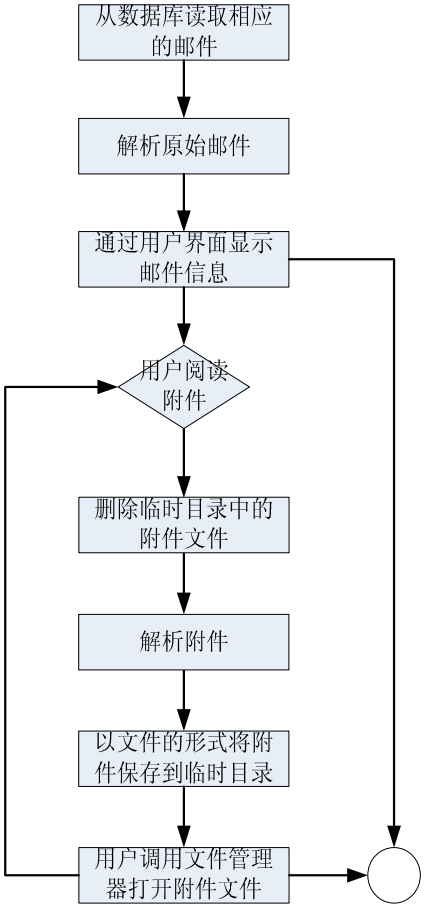

图 6.6 阅读邮件业务流程

6.2 构件化智能手机邮件应用层次结构分析

图 6.7 给出了邮件应用的实现结构图。

其软件架构采用“引擎层”如前所述，手机引擎层为单个应用提供单一的功能，构件的粒度要求达到最小。

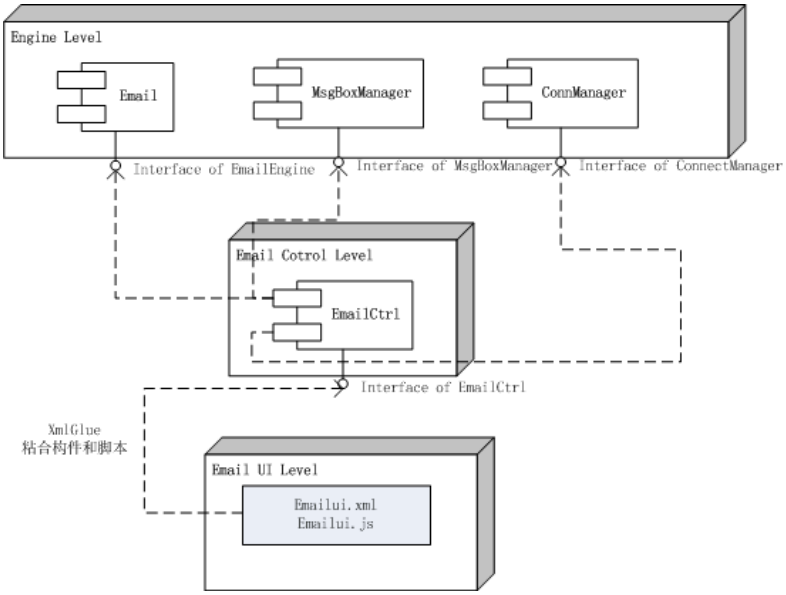

图 6.7 构件化智能手机邮件应用结构图

6.3 构件化智能手机邮件应用的引擎层实现

如前所述，手机引擎层为单个应用提供单一的功能，构件的粒度要求达到最小，这一层的构件都是元构件（最基本的不含其它构件的单元）。这里所说的应用，不能完全对应高层的一个应用，也有可能是一个应用中的一个需求。

通过对构件化智能手机邮件应用需求的分析以及对引擎层特点的理解，将“和欣”智能手机邮件应用引擎层分成三个模块：

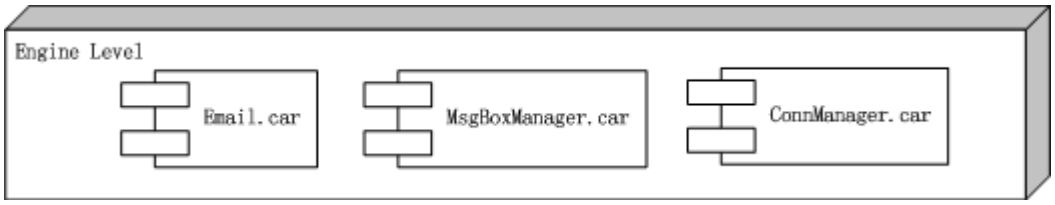

图 6.8 构件化智能手机邮件应用引擎模块图

分别是：

邮件引擎：针对邮件应用独有的功能，例如：邮件的编解码，接收发送，数据库的存储等的抽象；

信箱管理引擎：由于信息管理在智能手机应用中是一个使用很广泛的接口，有包括邮件应用，短信应用，多媒体短信（彩信）应用，WAP 信息应用，小区广播应用等都会使用到，因此在这里将这个引擎单独作为一个引擎进行提供。

网络连接引擎：这个引擎单独拿出的原因也是因为在智能手机应用中有许多应用会涉及到与移动网关的连接，例如：邮件的收发，彩信和 WAP 信息的收发，以及浏览器应用。因此这个引擎也单独形成一个模块。

6.3.1 邮件引擎

按照需求分析，将邮件应用需要的底层功能进行归纳分解为与协议相关的引擎，与编解码相关的引擎，邮件接收下来后与数据库相关的操作引擎。

一、协议与编解码引擎

IETF 为在 Internet 上传递电子邮件制定了一系列的协议，这主要包括：发送邮件的 SMTP[17]协议，接收邮件的 POP3[18]协议，规定邮件格式的 RFC822[19]和 MIME[20][21]协议，以及其它与电子邮件相关的协议。

（1） 邮件接收引擎接口 IPop3

POP3 邮局协议它采用客户/服务器模式，实现将邮件从服务器收取到客户端本地主机的功能。客户端收到邮件后将其从服务器上删除，断开与服务器的连接，而对邮件的处理是在客户端本地进行。使用 POP3 收取邮件减少了客户端与服务器的连接时间，减轻了网络负担。

POP3 服务期通过 TCP 的 110 端口监听连接请求。客户端与服务器连接建立后，通过相互交换的 POP3 命令与响应完成邮件的收取。

完整的 POP3 会话周期有三种状态：“认证”状态，“操作”状态和“更新”状态。一旦客户端开始 TCP 连接，并得到 POP3 服务期返回的问候信息后，会话就进入了“认证”状态，此状态下用户必须向服务其确认自己的身份。如果客户端成功地确认了自己的身份，会话则进入“操作”状态，此时客户端可以向服务器要求提供各种服务，并可以无序的，重复的发送 POP3 命令直到客户端发送 QUIT 命令。服务器接收到 QUIT 命令后，会话进入“更新”状态，POP3 服务器删除已标记的邮件，释放本次会话所占用的资源并终止连接。[22]

表 6.1 IPop3 接口详述

方法	方法描述
Connect([in]IAccountEntity *pIAccountEntity);	连接网络，传入邮件帐号实体接口，若传入为空则返回 E_INVALIDARG；若网络连接失败，返回 E_CONNECT_FAILED；成功返回 NOERROR
Login([in]WString wsUserName, [in]WString wsPwd);	登陆邮件服务器，传入邮件帐号用户名 wsUserName 和邮件帐号密码 wsPwd；若网络没有连接返回 E_EMAIL_NET_UNCONNECT；若用户名无效，返回 E_EMAIL_USER_INVALID；若密码不正确，返回 E_EMAIL_PWD_INVALID；成功返回 NOERROR
GetEmailCount([out]INT *pnCount);	获取服务器上需要下载的邮件总数(如果设置自动删除邮件为 False，在此实现 Message ID 的匹配)，传出需要下载的邮件的总数，若下载过程中出现网络中断情况，则返回 E_EMAIL_NET_HALT；成功返回 NOERROR
GetSizeByIndex([in]INT nIndex, [out]INT *pnSize);	获取某个指定邮件的大小，输入指定邮件的 Index，返回指定邮件的大小，在获取过程中如果网络中断则返回 E_EMAIL_NET_HALT；成功返回 NOERROR
GetFromByIndex([in]INT nIndex, [out]EzWStrBuf wsbFrom);	获取指定邮件的发件人，输入指定邮件的 Index，返回发件人，若获取过程中网络终端，则返回 E_EMAIL_NET_HALT；成功返回 NOERROR
ReceiveBuf([in]INT nIndex, [out]EzByteBuf * ebbEmailBuf);	接收指定的邮件(如果设置自动删除邮件为 False，在此实现 Message ID 的保存)，输入指定邮件的 Index，返回邮件的原始数据。在接收过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果 ebbEmailBuf 为空，返回 E_INVALIDARG；如果邮件尺寸太大，返回 E_MAIL_OVERSIZE；成功返回 NOERROR

Receive([in]INT nIndex, [out]IMessage **ppIMessage);	接收指定的邮件，输入指定邮件的 Index，返回接收的 message 接口对象；设置过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果 ppIMessage 为空，返回 E_INVALIDARG；如果邮件尺寸太大，返回 E_MAIL_OVERSIZE；成功返回 NOERROR
ReceiveHeader([in]INT nIndex, [out]EzByteBuf *ebbEmailHeader);	接收指定的邮件头，输入指定邮件的 Index，返回对应的邮件头；接收过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果 ebbEmailHeader 为空，返回 E_INVALIDARG；成功返回 NOERROR
Disconnect();	断开网络连接，断开过程中，如果网络中断，返回 E_EMAIL_NET_HALT；成功返回 NOERROR
DelCurEmail([in] ReceiveType uType);	标记删除标志或将 MsgID 保存到数据库中，标记删除标志过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果操作数据库储出错，返回相应的错误；成功返回 NOERROR
AbortReceiving();	中断正在进行的接收，成功返回 NOERROR
ReceiveByHeader([in] INT nEmailIndex, [out] EzByteBuf * ebbEmailBuf);	根据邮件头的 id 接收完整的邮件，输入指定邮件的 Index，返回邮件的原始数据；下载过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果操作数据库储出错，返回相应的错误；如果该封邮件已经被下载，返回 E_EMAIL_DOWNLOADED；成功返回 NOERROR

（2） 邮件发送引擎接口 ISmtp

SMTP 为简单邮件传输协议，主要保证电子邮件能够可靠和高效的传输。SMTP 主要应用在两种情况：一是电子邮件从客户端传输到服务器；二是邮件从一个服务器转发到另一个服务器。当收件人和发件人在同一个网络上时，SMTP 服务器把邮件直接转发给收信人信箱；当双方不在同一个网络上时，发信方的 SMTP 服务器做客户端，将邮件发送给收信人所在网络的 SMTP 邮件服务器。

SMTP 协议的基本命令有 7 个，分别是：HELO，MAIL，RCPT，DATA，REST，NOOP 和 QUIT 等。客户端向服务器发送邮件时，首先向 SMTP 服务器提出连接请求，服务器接受此客户端连接后，双方可以开始通信。[23]

ISmtp 接口主要适用 STMP 协议中电子邮件从客户端传送到服务器端的协

议规范进行开发。

表 6.2 ISmtp 接口详述

方法	方法描述
Connect([in]IAccountEntity *pIAccountEntity);	连接网络，输入邮件帐号实体接口，如果 pIAccountEntit 为空，返回 E_INVALIDARG；如果网络连接失败，返回 E_CONNECT_FAILED；成功返回 NOERROR
VerifyUser([in]WString wsUserName, [in]WString wsPassword);	验证发送者用户名，输入发送者用户名和密码，验证过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果 wsUserName 或 wsPassword 为 NULL，返回 E_INVALIDARG；如果验证用户名失败，返回 E_USERNAME_INVALID；如果验证密码失败，返回 E_PASSWORD_INVALID；成功返回 NOERROR
Send([in]IMessage *pIMessage);	发送邮件方法，输入要发送的邮件接口对象，发送过程中，如果网络中断，返回 E_EMAIL_NET_HALT；如果 pIMessage 为空，返回 E_INVALIDARG；成功返回 NOERROR
Disconnect();	断开网络连接方法，断开过程中，如果网络中断，返回 E_EMAIL_NET_HALT；成功返回 NOERROR
AbortSending();	中断正在进行的发送，成功返回 NOERROR

对于邮件接收发送的引擎接口设计要主要到一点，协议会发生变化，例如针对 SMTP 扩展的 ESMTP 协议，这里的设计将两个协议单独形成两个接口，便于进行协议升级。

（1） 邮件信息引擎接口 IMessage

邮件编解码:邮件的编码与解码

邮件的编解码、获取和设置邮件的基本信息、附件的编解码

表 6.3 IMessage 接口详述

方法	方法描述
InitFromBuf([in]EzByteBuf ebbEmailData);	根据邮件的原始数据初始化 IMessage，输入邮件的原始数据，如果 ebbEmailData 为空，返回 E_INVALIDARG；在进行空间分配时，如果系统资源不足，返回 E_OUTOFMEMORY；成功返回 NOERROR

ToBuf([out]EzByteBuf *ebbEmailData);	将 IMessage 信息编码成可以发送的邮件信息，输出邮件的原始数据,内部分配由用户释放，如果 ebbEmailData 为空，返回 E_INVALIDARG；在进行空间分配时，如果系统资源不足，返回 E_OUTOFMEMORY；成功返回 NOERROR
GetAttachmentInfo([out]EzArray<WString> * peaAttachNames, [out]EzArray<INT> * peaAttachSize);	获取附件相关的信息，其中 peaAttachNames 包含名字和索引号的数组，peaAttachSize 每个附件的大小；在进行数组空间分配时，如果系统资源不足，返回 E_OUTOFMEMORY；附件的个数为 0 时，返回 E_EMAIL_PROPERTY_NULL；成功返回 NOERROR
GetAttachmentCount([out]INT *pnCount);	获取该邮件中附件的数量，返回附件的个数，成功返回 NOERROR
GetAttachment([in]INT nIndex, [out]EzWStrBuf wsbAttachPath);	获取指定的附件的方法，输入附件的索引号，输出附件保存成临时文件后的全路径；索引号对应的附件不存在，返回 E_ATTACH_NOT_EXIST；附件进行解码保存成文件时出错，返回 E_EMAIL_SAVE_ERROR；成功返回 NOERROR
AddAttachment([in]WString wsFileName);	添加附件方法，输入附件全路径文件名，如果 wsFileName 非法，返回 E_INVALID_ARGUMENT；文件操作出错，返回 E_EMAIL_FILE_OPERATION_FAIL；成功返回 NOERROR
RemoveAttachment([in]INT nIndex);	删除附件，输入需删除附件的索引号，如果索引号对应的附件不存在，返回 E_ATTACH_NOT_EXIST；成功返回 NOERROR
GetSubject([out]EzWStrBuf * wsbSubject);	获取当前邮件的主题，输出邮件主题，考虑到 Subject 大小不确定，所以由内部分配，外部释放；成功返回 NOERROR
GetTo([out]EzWStrBuf * wsbTo);	获取收件人，输出邮件收件人，考虑到 To 大小不确定，所以由内部分配，外部释放；成功返回 NOERROR
GetFrom([out]EzWStrBuf wsbFrom);	获取邮件发件人，输出邮件发件人，成功返回 NOERROR

GetCc([out]EzWStrBuf * wsbCc);	获取邮件抄送人，输出邮件抄送人，考虑到 Cc 大小不确定，所以由内部分配，外部释放；成功返回 NOERROR
GetBcc([out]EzWStrBuf * wsbBcc);	获取邮件密送人，输出邮件密送人，考虑到 Bcc 大小不确定，所以由内部分配，外部释放；成功返回 NOERROR
GetDate([out]LONG *plDate);	获取邮件发送时间，输出邮件发送时间；成功返回 NOERROR
GetPriorityType([out]PriorityType *puPriorityType);	获取邮件的优先级，输出邮件的优先级，成功返回 NOERROR
GetReplyTo([out]EzWStrBuf wsbReplyTo);	获取邮件的回复地址，输出邮件的回复地址，成功返回 NOERROR
GetSensitivity([out]Sensitivity *puSensitivity);	获取邮件的敏感度，输出邮件的敏感度，成功返回 NOERROR
GetKeywords([out]EzWStrBuf * wsbKeywords);	获取邮件的关键字，考虑到 Keywords 大小不确定，所以由内部分配，外部释放；成功返回 NOERROR
GetCharSet([out]CharSet *puCharSet);	获取邮件头编码字符集，输出邮件头编码字符集，成功返回 NOERROR
GetBodyEncoding([out]EncodingType *puEncodingType);	获取邮件体编码方式，输出邮件体编码方式，成功返回 NOERROR
GetContentSize([out]INT * pnSize);	获取邮件内容大小，参数输出邮件内容大小，成功返回 NOERROR
GetContent([out]EzWStrBuf * wsbContent);	获取邮件内容，参数输出邮件内容，考虑到 Content 大小不确定，所以由内部分配，外部释放，成功返回 NOERROR
GetMessageID([out]EzWStrBuf wsbMsgID);	获取消息 ID，参数输出邮件消息 ID，成功返回 NOERROR
SetSubject([in]WString wsSubject);	设置邮件的主题，参数输出邮件主题，成功返回 NOERROR
SetTo([in]WString wsTo);	设置邮件收件人，参数输入邮件收件人，成功返回 NOERROR
SetFrom([in]WString wsFrom);	设置邮件发件人，参数输入邮件发件人，成功返回 NOERROR
SetCc([in]WString wsCc);	设置邮件抄送人，参数输入邮件抄送人，成功返回 NOERROR
SetBcc([in]WString wsBcc);	设置邮件密送人，参数输入邮件密送人，成功返回 NOERROR
SetPriorityType([in]PriorityType	设置邮件的优先级，参数输入邮件的优先级，

uPriorityType);	如果 PriorityType 为非法值，返回 E_INVALIDARG；成功返回 NOERROR
SetReplyTo([in]WString wsReplyTo);	设置邮件的回复地址，参数输入邮件的回复地址，如果 wsReplyTo 非法，返回 E_INVALIDARG；成功返回 NOERROR
SetSensitivity([in]Sensitivity uSensitivity);	设置邮件的敏感度，参数输入邮件的敏感度，如果 uSensitivity 非法，返回 E_INVALIDARG；成功返回 NOERROR
SetKeywords([in]WString wsKeywords);	设置邮件的关键字，参数输入邮件的关键字，如果 wsKeywords 非法，返回 E_INVALIDARG；成功返回 NOERROR
SetCharSet([in]CharSet uCharSet);	设置邮件头编码字符集，参数输入邮件头编码字符集，如果 uCharSet 为非法值，返回 E_INVALIDARG；成功返回 NOERROR
SetBodyEncoding([in]EncodingType uEncodingType);	设置邮件体编码格式，参数输入邮件体编码格式，如果 uEncodingType 为非法值，返回 E_INVALIDARG；成功返回 NOERROR
SetContent([in]WString wsContent);	设置邮件内容，参数输入邮件内容，成功返回 NOERROR
SetAttachNames([in]INT nArrayLength, [in]EzArray<WString> eaAttachNames);	设置邮件附件，参数输入附件数组长度 nArrayLength，附件全路径名组成的数组 eaAttachNames，成功返回 NOERROR

二、数据库引擎

“和欣”智能手机邮件应用的数据都是数据库存储的。存储的内容包括：邮件信息，手机用户的邮件账户信息，用户邮件签名设置信息，用户邮件设置信息等等。

（1） 账号管理引擎接口

Email 帐号（EmailAccount）数据库表结构如表 6.4 所示：

表 6.4Email 帐号数据库表

字段名	中文名称	字符类型	字节数	说明
AccountID	帐号 ID	integer		Autoincrement primary key
NickName	昵称	varchar	60	
UserName	用户名	varchar	64	登陆邮件服务器的用户名
Password	密码	varchar	20	登陆邮件服务器的密码
SmtpServer	Smtp 邮件服务器	varchar	64	Smtp 服务器的域名或 IP
PopServer	Pop 邮件服务器	varchar	64	Pop 服务器的域名或 IP
SmtpPort	Smtp 端口号	varchar	4	Smtp 服务器的端口号
PopPort	Pop 服务器端口号	varchar	4	Pop 服务器的端口号

Address	邮件地址	varchar	128	Email 邮件地址
IsDefault	默认账户	Boolean		是否是默认帐号
ReplyTo	回复地址	varchar	128	邮件的回复地址
AutoDel	自动删除	Boolean		是否从服务器上自动删除
UseSSL	SSL 连接	Boolean		是否使用 SSL 连接
UseAPOP	启用 APOP	Boolean		是否启用 APOP

针对 email 的属性，一个智能手机的持有者可以拥有多个邮件账户，因此，对邮件账户的管理就要包括对若干账户的管理以及对每一个账户的详细信息的管理。可以用一个对象接口来实现其功能，但在进行对象接口设计时，遵循一个接口提供一类方法的原则，将实现分成 IAccountManager 多账号管理和 IAccountEntity 单帐号管理区分开。

其中，IAccountManagerEmail 帐号管理:实现 Email 帐号的各种操作，Email 信息帐号的查看、增加、删除、修改和设置

表 6.5 IAccountManager 接口详述

方法	方法描述
EnumEntity([out]IObjectEnumerator **ppAccountEnum);	枚举邮件帐号，参数返回枚举器接口对象，如果枚举失败，返回相应的数据库错误码；如果 ppAccountEnum 为空，返回 E_INVALIDARG；成功返回 NOERROR
Create([out]IAccountEntity **ppIAccountEntity);	创建一个新的帐号，以创建 IAccountEntity 对象的方式输出新的帐号对象，如果 ppIAccountEntity 不为空，返回 E_INVALIDARG；如果 ppIAccountEntity 创建不成功，返回 E_OUTOFMEMORY；成功返回 NOERROR
Add([in]IAccountEntity *pIAccountEntity);	新增帐号，参数输入新增的帐号信息，如果 pIAccountEntity 为空，返回 E_INVALIDARG；如果数据库保存时出错，返回相应的数据库出错代码；成功返回 NOERROR
Update([in]IAccountEntity *pIAccountEntity);	更新帐号信息，参数输入更新的帐号信息，如果 pIAccountEntity 为空，返回 E_INVALIDARG；如果数据库更新时出错，返回相应的数据库出错代码；成功返回 NOERROR
DelByID([in]INT nAccountID);	删除指定的帐号，参数输入帐号 ID，删除时数据库出错，返回相应的数据库出错代码；成功返回 NOERROR
DelAll();	删除所有的帐号，如果删除时数据库出错，

	返回相应的数据库出错代码；成功返回 NOERROR
GetByID([in]INT nAccountID, [out]IAccountEntity **ppIAccountEntity);	根据指定的 ID 获得相应的帐号，输入帐号 ID，以 IAccountEntity 方式输出帐号信息，如果 ppIAccountEntity 不为空，返回 E_INVALIDARG；如果数据库中不存在该 ID 对应的记录，返回 E_RECORD_NOT_FOUND ；如果 ppIAccountEntity 创建不成功，返回 E_OUTOFMEMORY；成功返回 NOERROR
GetDefault([out]IAccountEntity ** ppIAccountEntity);	获得默认的帐号，以 IAccountEntity 形式参数输出帐号信息，如果 ppIAccountEntity 不为空，返回 E_INVALIDARG；如果数据库中不存在该 ID 对应的记录，返回 E_RECORD_NOT_FOUND ；如果 ppIAccountEntity 创建不成功，返回 E_OUTOFMEMORY；成功返回 NOERROR
GetCount([out]INT *pnCount);	获取帐号的总数，参数输出帐号总数，操作中如果数据库出错，返回相应的数据库出错代码；成功返回 NOERROR
SetToDefault([in]INT nAccountID);	将指定的帐号设置成当前帐号，参数输入配置项，如果数据库更新出错时，返回相应的数据库错误码；成功返回 NOERROR
StartTransaction();	开始事务处理（设置数据库的自动更新属性为否），操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
Commit();	提交事务处理（将事务中进行操作更新到数据库），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
Rollback();	事务回滚（如果事务中某个操作失败调用事务回滚操作，以保持操作的原子性），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR

IAccountEntity 是 Email 帐号实体，用于存储，管理 Email 帐号的基本信息，并提供关于这些基本信息的操作包括获取和设置 Email 帐号信息的操作

表 6.6 IAcoountEntity 接口详述

方法	方法描述
GetID([out]INT *pnAccountID);	获取帐号 ID，参数输出帐号 ID，成功返回 NOERROR

GetNickName([out]EzWStrBuf wsbNickName);	获取帐号昵称，参数输出帐号昵称，成功返回 NOERROR
GetUserName([out]EzWStrBuf wsbUserName);	获取帐号用户名，参数输出帐号用户名，成功返回 NOERROR
GetPassword([out]EzWStrBuf wsbPassword);	获取帐号密码，参数输出帐号密码，成功返回 NOERROR
GetSmtpServer([out]EzWStrBuf wesbSmtpServer);	获取 Smtp 服务器 IP 或域名，参数输出 Smtp 服务器 IP 或域名，成功返回 NOERROR
GetPopServer([out]EzWStrBuf wsbPopServer);	获取 Pop 服务器 IP 或域名，参数输出 Pop 服务器 IP 或域名，成功返回 NOERROR
GetSmtpPort([out]int *pnSmtpPort);	获取 Smtp 服务器端口号，参数输出 Smtp 服务器端口号，成功返回 NOERROR
GetPopPort([out]int *pnPopPort);	获取 Pop 服务器端口号，参数输出 Pop 服务器端口号，成功返回 NOERROR
GetIsDefault([out]BOOL *pbIsDefault);	获取帐号是否是默认帐号的属性，参数输出布尔值，成功返回 NOERROR
GetReplyTo([out]EzWStrBuf wsbReplyTo);	获取回复的电子邮件地址，参数输出回复电子邮件地址，成功返回 NOERROR
GetAutoDel([out]BOOL *pbDel);	获取下载信件后是否从服务器上删除该信件，参数输出是否自动删除，成功返回 NOERROR
GetUseSSL([out]BOOL *pbUseSSL);	获取是否使用 SSL 连接，参数返回是否使用 SSL 连接，成功返回 NOERROR
GetUseAPOP([out]BOOL *pbUseAPOP);	获取是否使用 APOP 验证，参数输出是否使用 APOP 验证，成功返回 NOERROR
GetUseSmtpAuth([out]BOOL *pbSmtpAuth);	获取是否使用 smtp 验证，参数输出是否使用 APOP 验证，成功返回 NOERROR
GetAddress([out]EzWStrBuf wsbAddress);	获取帐号的邮件地址，参数输出邮件地址，成功返回 NOERROR
SetID([in]INT nAccountID);	设置帐号 ID，参数输入要设置的帐号 ID，成功返回 NOERROR
SetNickName([in]WString wsNickName);	设置帐号的昵称，输入要设置的帐号昵称，成功返回 NOERROR
SetUserName([in]WString wsUserName);	设置帐号用户名，参数输入要设置的帐号用户名，成功返回 NOERROR
SetPassword([in]WString wsPassword);	设置帐号密码，参数输入要设置的帐号密码，成功返回 NOERROR
SetSmtpServer([in]WString wsSmtpServer);	设置 Smtp 服务器的 IP 或域名，参数输入 Smtp 服务器的 IP 或域名，成功返回 NOERROR
SetPopServer([in]WString	设置 Pop 服务器的 IP 或域名，参数输入 Pop 服务器

wsPopServer);	的 IP 或域名，成功返回 NOERROR
SetSmtpPort([in]int nSmtpPort);	设置 Smtp 服务器端口号，参数输入 Smtp 服务器端口号，成功返回 NOERROR
SetPopPort([in]int nPopPort);	设置 Pop 服务器端口号，参数输入 Pop 服务器端口号，成功返回 NOERROR
SetIsDefault([in]BOOL bIsDefault);	设置帐号的是否是默认帐号的属性，参数输入布尔类型的默认值，成功返回 NOERROR
SetReplyTo([in]WString wsReplyTo);	设置回复的电子邮件地址，参数输入回复电子邮件地址，成功返回 NOERROR
SetAutoDel([in]BOOL bDel);	设置下载信件后是否从服务器上删除该信件，参数输入自动删除标志值布尔值，成功返回 NOERROR
SetUseSSL([in]BOOL bUseSSL);	设置是否使用 SSL 连接，参数传入是否使用 SSL 连接布尔值，成功返回 NOERROR
SetUseAPOP([in]BOOL bUseAPOP);	设置是否使用 APOP 验证，参数传入是否使用 APOP 验证布尔值，成功返回 NOERROR
SetUseSmtpAuth([in]BOOL bUseSmtpAuth);	设置是否使用 Smtp 验证，参数传入是否使用 smtp 验证布尔值，成功返回 NOERROR
SetAddress([in]WString wsAddress);	设置帐号的邮件地址，参数输入邮件地址，成功返回 NOERROR

邮件存储管理引擎接口

邮件（Email）数据库表结构如下：

表 6.7 邮件数据库表结构

字段名	中文名称	字符类型	字节数	说明
MailID	邮件 ID	integer		Autoincrement primary key
BoxID	所属邮箱 ID	integer		记录该邮件属于哪个邮箱
Subject	邮件主题	varchar	100	
DateTime	时间	integer		以整型存储邮件发送的日期时间
MailFromTo	发件人/收件人	varchar	80	发件人/收件人的邮件地址
Size	邮件大小	integer		该邮件的大小
OriginalData	邮件体	Blob		以 Blob 的形式存储邮件
Status	邮件状态	integer		是否已经阅读

按照对邮件信息的理解和操作的需要，将邮件的存储分为：IEmailManager 和 IEmailEntity 两个接口

IEmailEntity，描述 Email 的基本信息，并提供关于这些基本信息的操作，包括 Email 基本信息的设置和获取操作

表 6.8 IEmailEntity 接口详述

方法	方法描述
GetID([out]INT *pnMailID);	获取当前邮件的 ID，参数输出邮件的 ID；成功返回 NOERROR
GetBoxID ([out]INT *pnBoxID);	获取当前邮件的邮箱目录 ID，参数返回邮件的邮箱目录 ID，成功返回 NOERROR
GetSubject([out]EzWStrBuf wsbSubject);	获取当前邮件的主题，参数返回邮件主题，成功返回 NOERROR
GetStatus([out]Status *puStatus);	获取当前邮件的状态，参数返回当前的邮件状态，成功返回 NOERROR
GetPriority([out]PriorityType *puPriority);	获取当前邮件的重要性，参数返回邮件重要性，成功返回 NOERROR
GetSize([out]INT *pnSize);	获取当前邮件的大小，参数返回邮件大小，成功返回 NOERROR
GetFromTo([out]EzWStrBuf wsbFromTo);	获取当前邮件的发件人或收件人，参数输出邮件的发件人或收件人，成功返回 NOERROR
GetName([out]EzWStrBuf wsbName);	获取当前邮件发件人的姓名，参数返回邮件发件人姓名，成功返回 NOERROR
GetDate([out]LONG *plDate);	获取当前邮件的发送时间，参数输出邮件发送时间，成功返回 NOERROR
GetOriginalData([out]EzByteBuf ebbData);	获取邮件的原始数据，参数输出邮件的原始数据，成功返回 NOERROR
SetID([in]INT nMailID);	设置当前邮件的 ID，参数输入需要设置的邮件的 ID，成功返回 NOERROR
SetBoxID([in]INT nBoxID);	设置当前邮件的邮箱目录 ID，参数输入需要设置的邮箱目录的 ID，成功返回 NOERROR
SetSubject([in]WString wsSubject);	设置当前邮件的主题，参数输入邮件主题，成功返回 NOERROR
SetStatus([in]Status uStatus);	设置当前邮件的状态，参数输入要设置的邮件状态，成功返回 NOERROR
SetPriority([in]PriorityType uPriority);	设置当前邮件的重要性，参数输入邮件重要性，成功返回 NOERROR
SetSize([in]INT nSize);	设置当前邮件的大小，参数输入要设置的邮件大小，成功返回 NOERROR
SetFromTo([in]WString wsFromTo);	设置当前邮件的发件人或收件人，参数输入要设置的邮件发件人或发件人，设置成功返回 NOERROR
SetName([in] WString wsName);	设置当前邮件发件人的姓名，参数输入要设置的邮件收件人或发件人姓名，设置成功返

	回 NOERROR
SetDate([in]LONG lDate);	设置当前邮件的发送时间，参数输入邮件发送时间，成功返回 NOERROR
SetOriginalData([in]EzByteBuf ebbData);	设置当前邮件的原始数据，参数输入需要设置的邮件的原始数据，设置成功返回 NOERROR

IEmailManager 主要是对保存在数据库中 Email 记录进行操作，包括 Email 记录的查看、新增、更新、删除、删除某个目录所有邮件

表 6.9 IEmailManager 接口详述

方法	方法描述
GetTotalSize([out]INT *pnMailSize);	获得邮箱中邮件总的大小，参数输出邮件总的大小，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
GetInfoByBoxId([in] INT nBoxId, [out] INT *pnTotalNum, [out] INT *pnUnreadNum);	获得邮箱某个邮件夹中邮件的信息，参数输入要获取的信箱目录 ID，参数输出邮件总数 nTotalNum 和未读邮件总数 nUnreadNum，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
EnumByBoxID([in]INT nBoxID, [out]IObjectEnumerator **ppEmailEnum);	根据信箱目录 ID 枚举邮件，参数输入要枚举的信箱目录 ID，参数输出重载的枚举器接口对象，如果枚举失败，返回相应的数据库错误码；如果 ppEmailEnum 为空，返回 E_INVALIDARG；成功返回 NOERROR
EnumUnread([in]EzArray<INT> eaExceptBoxId, [out] IObjectEnumerator ** ppIEmailEnum);	根据信箱目录 ID 枚举未读邮件，枚举出除数组中 BoxID 外的所有未读邮件，参数输入需要枚举的信箱目录 ID 数组，参数输出重载后的枚举器接口对象，如果枚举失败，返回相应的数据库错误码；如果 ppEmailEnum 为空，返回 E_INVALIDARG；成功返回 NOERROR
GetByID([in]INT nMailID, [out]IStorageEntity **ppIStorageEntity);	阅读指定的邮件，参数输入指定邮件的 ID，参数输出邮件实体的接口对象，如果操作数据库出错，返回相应的数据库错误码；如果解码出错返回相应的错误；ppIMStorageEntity 不为空，返回 E_INVALIDARG；当指定的 ID 不存在时，返回 E_RECORD_NOT_FOUND；成功返回 NOERROR
Create([out]IStorageEntity **ppIStorageEntity);	创建一个新的邮件实体，参数输出邮件实体的接口对象，如果 ppIStorageEntity 不为空，

	返回 E_INVALIDARG；如果 ppIStorageEntity 创建失败，返回 E_OUTOFMEMORY；成功返回 NOERROR
Add([in]IStorageEntity *pIStorageEntity);	添加一个邮件到数据库，参数输入要添加的邮件实体的接口对象，如果操作数据库出错，返回相应的数据库错误码；如果 pIStorageEntity 为空时，返回 E_INVALIDARG；成功返回 NOERROR
Update([in]IStorageEntity *pIStorageEntity);	更新数据库中指定的邮件，参数输入要更新的邮件实体的接口对象，如果操作数据库出错，返回相应的数据库错误码；如果 pIStorageEntity 为空时，返回 E_INVALIDARG；成功返回 NOERROR
DelByID([in]INT nMailID);	删除指定的邮件，参数输入要删除的邮件的 ID，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
DelAllByBoxID([in]INT nBoxID);	删除指定邮箱目录中的邮件，参数输入要删除邮箱目录的 ID，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetStatus([in]INT nMailID, [in]Status uStatus);	设置邮件的已读或未读的状态，参数输入要设置的邮件的 ID，参数输入要设置邮件的状态，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetBoxID([in]INT nMailID, [in]INT nBoxID);	将指定的邮件移到指定的邮箱目录，参数输入指定的邮件的 ID，指定的邮箱目录的 ID，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
StartTransaction();	开始事务处理（设置数据库的自动更新属性为否），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
Commit();	提交事务处理（将事务中进行操作更新到数据库），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
Rollback();	事务回滚（如果事务中某个操作失败调用事务回滚操作，以保持操作的原子性），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR

邮件设置引擎接口

邮件设置（EmailSetting）数据库表结构如下：

表 6.10 邮件设置数据库表结构

字段名	中文名称	字符类型	字节数	说明
BoxMaxSize	邮箱最大容量	integer		设置邮箱的最大的容量是多少，以邮件的个数计算
MailMaxSize	允许的邮件大小	integer		发送和接收的邮件的大小不能超过这个尺寸
AutoSign	自动签名	Boolean		是否自动签名
AutoDel	自动删除	Boolean		是否自动删除

IEmailSetting 接口完成 Email 设置，管理 Email 设置属性的工作， Email 设置属性的查看和修改主要包括邮箱的最大容量、邮件的最大尺寸、是否自动签名、是否自动从服务器删除已下载的邮件等属性。

表 6.11 IEmailSetting 接口详述

方法	方法描述
GetBoxMaxSize([out]INT *pnSize);	获取邮箱的最大容量，参数输出邮箱的最大容量，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
GetMailMaxSize ([out]INT *pnSize);	获取发送邮件限制的最大尺寸，参数返回邮件的最大尺寸，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
GetReceiveMailMaxSize([out]INT* pnSize);	获取接收邮件限制的最大尺寸，参数输出邮件的最大尺寸，若参数不合法返回错误码，成功返回 NOERROR
GetAutoSign ([out]BOOL *pbSign);	获取是否自动签名，参数返回是否获取自动签名的标志，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
GetReplyWithOriginalText([out] BOOL * pbReplyWithOriginalText);	获取是否附加原文回复，参数返回是否自动签名的标志，参数不合法返回错误码；成功返回 NOERROR
GetReceiveAll([out] BOOL * pbReceiveAll);	获取是否接收所有帐号邮件，参数返回是否接收所有帐号邮件标志，参数不合法返回错误码；成功返回 NOERROR
GetConId([out] INT * pnConId);	获取本模块需要的网络配置 Id，参数输入邮箱的最大容量，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetConId([in] INT nConId);	设置本模块需要的网络配置 Id，参数输入需配置的邮箱的最大容量，如果操作数据库出

	错，返回相应的数据库错误码；成功返回 NOERROR
SetBoxMaxSize([in] INT nSize);	设置邮箱的最大容量，参数输入邮箱的最大容量，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetMailMaxSize([in]INT nSize);	设置发送邮件的最大尺寸，参数输入邮件的最大尺寸，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetAutoSign([in]BOOL bSign);	设置是否自动签名，参数输入是否自动签名标志，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetReceiveMailMaxSize([in]INT nSize);	设置接收邮件的最大尺寸，参数输入邮件的最大尺寸，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetReplyWithOriginalText([in] BOOL bReplyWithOriginalText);	设置是否附加原文回复，参数输入是否附加原文回复标志，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
SetReceiveAll([in] BOOL bReceiveAll);	设置是否接收所有帐号，参数输入设置是否接收所有帐号标志，如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR

邮件签名引擎接口

邮件签名（EmailSignature）数据库表结构如下：

表 6.12 邮件签名数据库表结构

字段名	中文名称	字符类型	字节数	说明
SignatureID	签名 ID	integer		Autoincrement primary key
Content	签名内容	varchar	40	
IsDefault	默认签名	Boolean		是否是默认签名

接口 ISignature 负责签名管理，管理 Email 的签名进行 Email 签名信息的查看、新增、更新和删除操作。

表 6.13 ISignature 接口方法详述

方法	方法描述
EnumEntity([out]IObjectEnumerator **ppSignEnum);	枚举邮件签名，参数输出重载的枚举器接口对象，如果枚举失败，返回相应的数据库错误码；如果 ppSignEnum 为空，返回 E_INVALIDARG；成功返回 NOERROR
Add ([in]WString wsContent);	添加签名，参数输入签名的内容，如果

	wsContent 为空，返回 E_INVALIDARG；如果保存到数据库时出错，返回相应数据库出错代码；成功返回 NOERROR
Update([in]INT nSignatureID, [in]WString wsContent);	更新指定的签名，参数输入要更新的签名的 ID，以及要更新的签名内容，如果 wsContent 为空，返回 E_INVALIDARG；如果保存到数据库时出错，返回相应数据库出错代码；成功返回 NOERROR
DelByID([in]INT nSignatureID);	删除指定的签名，参数输入要删除的签名 ID，如果操作数据库时出错，返回相应数据库出错代码；成功返回 NOERROR
DelAll();	删除所有的签名，成功返回 NOERROR
GetByID([in]INT nSignatureID, [out]EzWStrBuf wsbContent);	获取指定签名 ID 的内容，参数输入要获取签名内容的 ID，参数输出 wsbContent 签名内容；如果指定的 ID 不存在，返回 E_RECORD_NOT_FOUND；如果操作数据库出错时，返回数据库相应的错误码；成功返回 NOERROR
GetCount([out]INT *pnCount);	获取签名的总数，参数返回签名总数，操作数据库出错时，返回数据库相应的错误码；成功返回 NOERROR
GetID([out]INT *pnSignatureID);	获取签名的 ID，参数返回要获取的签名 ID；成功返回 NOERROR
GetContent([out]EzWStrBuf wsbContent);	获取签名的内容，参数返回要获取的签名内容 wsbContent ，成功返回 NOERROR
SetID([in]INT nSignatureID);	设置签名的 ID，参数输入签名的 ID 值，成功返回 NOERROR
SetContent([in]WString wsContent);	设置签名的内容，参数输入签名的内容，成功返回 NOERROR
SetToDefault([in]INT nSignatureID);	将指定的签名设为默认，参数输入要指定的默认签名的 ID 值，成功返回 NOERROR
GetDefault([out]INT * pnSignatureID, [out]EzWStrBuf wsbContent);	获取默认的签名，参数输出默认的签名的 ID 及签名的内容；如果 pnSignatureID 为空,返回 E_INVALID_ARGUMENT；如果设置时数据库出错，返回相应的数据库错误码；成功返回 NOERROR
StartTransaction();	开始事务处理（设置数据库的自动更新属性为否），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR

Commit();	提交事务处理（将事务中进行操作更新到数据库），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR
Rollback();	事务回滚（如果事务中某个操作失败调用事务回滚操作，以保持操作的原子性），如果操作数据库出错，返回相应的数据库错误码；成功返回 NOERROR

6.3.2 信箱管理引擎

如前所述，信箱管理在智能手机应用开发中是一个不可或缺的组成部分，按照“和欣”智能手机邮件，短信，彩信，WAP 信息和小区广播的 MMI 设计，其信箱都包括有：收件箱，发件箱，草稿箱，已发信息/邮件箱，已删信息/邮件箱，垃圾箱，保密箱和用户自定义箱。信箱管理的主要任务，就是对这些信箱信息进行查看和更新。

表 6. 14 邮件信箱管理数据库表结构

字段名	中文名称	字符类型	字节数	说明
BoxID	信箱 ID	integer		Autoincrement primary key
BoxName	信箱名称	varchar	20	
ViewMode	查看方式	integer		邮件查看时的排序方式包括：时间排序、大小排序、发件人排序
Icon	邮箱图标	varchar	512	存储图标的全路径
Type	邮件夹类型	Boolean		有两种类型，系统提供的和用户自定义的两种，系统提供的邮件夹不能更名和删除。0 表示系统提供，1 表示用户自定义的

6.3.3 网络连接管理引擎

因为网络连接管理是智能手机应用的一个常用模块，例如：电子邮件应用，浏览器应用，彩信应用，快讯应用，一些聊天和炒股软件都要用到网络连接管理这个接口。因此将其抽取出来。

6.4 构件化智能手机邮件系统的逻辑控制层实现

构件化智能手机邮件系统的逻辑控制层的实现主要由四个接口完成，这四个接口分别是：IEmailBoxCtrl、IEmailAccountCtrl、IEmailCtrl、ISendRecv；其是通过对引擎层原子接口的组合调用来实现具体的功能的。

图 6.9 给出了逻辑控制层的静态结构图。

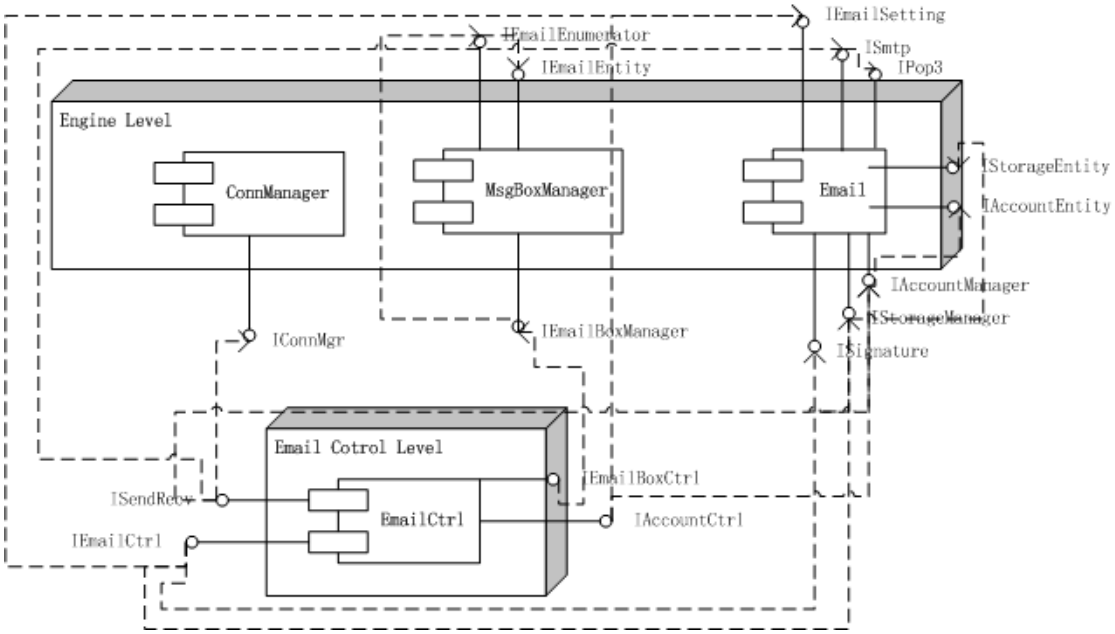

图 6. 9 逻辑控制层静态结构图

6.5 构件化智能手机邮件系统应用用户交互层实现

用户交互层的实现，采用了 XML 语言+JavaScirpt 脚本语言的实现形式。其中 emailui.xml 实现界面的布局以及构件及构件间的关系。

6.5.1emailui.xml 的实现

Emilui.xml 的总体结构如下：

```
<?xml version="1.0" encoding="gb2312" ?>
<x:xglue xmlns:x=http://www.koretide.com/xml-glue xmlns:w="elactrl.dll" trace="1">
    <w:form>
```

```
        页面布局实现
    </w:form>
<script language="JavaScript" src="emailui.js"> //申明脚本语言和脚本实现文件
    <![CDATA[
        粘合构件
        调用脚本方法
    ]]>
</script>
</x:xglue>
```

（1）页面布局的实现

在页面布局的实现中，有对主 Form 的设置：

```
<w:form x:id="_form" controlStyle="FormStyle_Client, ControlStyle_Visible"left="0" top="28"
    width="240" height="292">
```

其中定义了主 Form 的 ID，属性，以及 Form 宽、高、左边距及上边距。

对标题栏的设置：

```
<w:label x:id="_titleLabel" controlStyle="LabelStyle_Transparent"left="7" top="0"
    width="240" height="26" foreColor="16777215"backColor="4294967295" />
```

其中定义了标题栏的 ID，属性，标题栏的宽、高、左边距及上边距，标题栏的前景色和后景色。

对主界面左右键的设置：

```
<w:label   x:id="_leftLabel" controlStyle="ControlStyle_Visible, ControlStyle_NoBackground,
    LabelStyle_Left" left="4" top="267" width="80" height="26" foreColor="16777215"
    backColor="4294967295"/>
```

```
<w:label x:id="_rightLabel" controlStyle="ControlStyle_Visible, ControlStyle_NoBackground,
    LabelStyle_Right"   left="154" top="267" width="80" height="26" foreColor="16777215"
    backColor="4294967295" />
```

其中定义了左右键的 ID，属性，左右键的宽、高、左边距及上边距，左右键的前景色和后景色。

（2）粘合构件

```
elagdi = Elastos.Using("elagdi.dll");
elactrl = Elastos.Using("elactrl.dll");
emailctrl = Elastos.Using("emailctrl.dll");
```

在这里粘合了 UI 层要使用的构件。

（3）调用脚本方法

```
InitEmailForm()   //Show an email form
```

该方法是 emailui.js 中实现的一个方法，具体功能是打开邮件主应用。

6.5.2emailui.js 的实现

emailui.js 文件，其主要实现了用户交互层的逻辑，其中包括了和逻辑控制层的交互，js 文件中各个函数之间的调用。表 6.15 给出界面类函数的说明。

表 6. 15 界面类函数表

CEmailMainForm	邮件应用主界面
CUnreadForm CInboxForm COutboxForm CSentForm CDeletedForm CTrashForm CSecrecyForm	邮件信箱界面，包括：未读箱、收件箱、发件箱、已发邮件箱、已删邮件箱、草稿箱和保密箱
CReadClientForm CWriteClientForm	阅读邮件界面和编辑邮件界面
CExtractMainForm	有关提取的应用界面，包括：提取的主界面

CExtractListForm CAttachmentListForm CMoveToListForm	CExtractMainForm，对邮件列表的操作的 CExtractListForm 界面，附件列表的操作 CAttachmentListForm 界面，列表移动操作 CMoveToListForm 界面
CAccountMainForm CAccountManageForm	邮件帐户主界面和帐户管理界面
CEmailSettingForm CSignatureForm CMemoryStatusForm	邮件一些相关信息界面，如邮件设置界面 CEmailSettingForm，邮件签名编辑界面 CSignatureForm，已存邮件内存状态界面 CMemoryStatusForm 界面
CConnectMgrForm	邮件收发中，网络连接界面

操作类函数如表 6.16 所示。

表 6.16 操作类函数表

InitEmailForm	启动电子邮件应用的主函数，在 emailui.xml 文件中调用，可以区分用户主动启动邮件应用还是由其他应用启动
InitMenuTables InitMailboxForm	两个辅助的初始化函数，分别初始化菜单和邮件信箱
IsReadOnlyMailbox IsWritableMailbox	两个辅助函数，区分邮箱性质
MessageBox RegexMatch	两个辅助函数，MessageBox 是信息提示框，封装了 ICommonDialog RegexMatch 用于检查信箱地址是否有效

下面以邮件信箱主界面的一部分来说明脚本函数之间的调用以及和逻辑控制层的交互是如何实现的。

```
//---------------------------------------------------------------------------
// CMailboxClientForm
//---------------------------------------------------------------------------
var CMailboxClientForm = function()
{
    this.boxid;
    this.Init = function(bid)   //首先是界面初始化
   {with (this) {
```

```
        boxid = bid;
    // Label 将左右键内容写入
        leftLabel.SetText("选项");
        leftLabel.MouseDown = OnLabelClick;
        rightLabel.SetText("返回");
        rightLabel.MouseDown = OnLabelClick;

    // ListView 初始化界面列表
        mainListView = elactrl.ListView.CreateObject();
        mainListView.Init(elactrl.ControlStyle.ControlStyle_UserRender
                          | elactrl.ControlStyle.ControlStyle_Focused
                          | elactrl.ControlStyle.ControlStyle_Visible
                          | elactrl.ListViewStyle.ListViewStyle_MultiLine
                          | elactrl.ListViewStyle.ListViewStyle_NoHead
                          | elactrl.ListViewStyle.ListViewStyle_Gradient
                          |elactrl.ListViewStyle.ListViewStyle_Loop,
                          "mainListView",
                          0, 26, SCREEN_WIDTH, SCREEN_WIDTH, form);
        mainListView.SetBackColor(elagdi.Color.Color_Transparent);
        mainListView.InsertColumn(0, 218, "one", 0);
        mainListView.InsertColumn(1, 218, "two", 0);
        mainListView.ReSelect = OnListViewClick;
        mainListView.Select = OnListViewSelect;
var emailCtrl = emailctrl.EmailCtrl.CreateObject();
```

```
        //这里新建一个逻辑控制层构件对象，让下面可以调用到构件接口方法
emailCtrl.LoadEmail(mainListView, boxid);
                                //这里实际调用了 IEmailCtrl 构件的 LoadEmail()方法
if (mainListView.GetItemCount() > 0) {mainListView.SetFocusIndex(0);}
      }
}

this.OnListViewClick = function(sender, index)
                                                //该方法响应界面列表条目点击事件
{ with (s_thisForm) {
     var emailCtrl = emailctrl.EmailCtrl.CreateObject(); //新建构件对象
    var msgid = mainListView.GetItemData(index);
    var sp = emailCtrl.GetStatusAndPriority(msgid);
                                          //按照 ID 值查询数据库中邮件的详细信息
     var msgstatus = sp.puMessageStatus;
    var msgpriority = sp.puPriorityType;

if (IsReadOnlyMailbox(boxid, msgstatus)) {
     view.showDialog(GetFile("emailui.xml"),
     "CReadClientForm", msgid, boxid, msgstatus);
//调用 emailui.js 的 CReadClientForm 方法，用 GetFile 方法导入 emailui.xml 文件，
//用于配置 CReadClientForm 的界面
}
else if (IsWritableMailbox(boxid, msgstatus)) {
```

```
        view.showDialog(GetFile("emailui.xml"),
                                "CWriteClientForm",msgid, boxid, EmailOperation_Read);
//调用 emailui.js 的 CWriteClientForm 方法，打开 CWriteClientForm 的界面
}

with (emailengine.Status) {
       if (msgstatus == UNREAD) {
           emailCtrl.SetStatus(msgid, READ);
      }
      else if (msgstatus == TRASH_UNREAD) {
           emailCtrl.SetStatus(msgid, TRASH_READ);
      }
}
emailCtrl.LoadEmail(mainListView, boxid);
index = Math.min(index, mainListView.GetItemCount() - 1);
if (index >= 0) {mainListView.SetFocusIndex(index);}
                    }
               }
           …….

}
```

第 7 章 总结与展望

本课题是围绕上海科泰世纪有限公司 863 项目“基于中间件技术的因特网嵌入式操作系统及跨操作系统中间件运行平台”展开，本课题将 CAR 构件和 XmlGlue 技术运用于智能手机软件开发平台的研究，实现了基于 CAR 构件，运用 XmlGule 技术的 MVC 架构模式及设计模式构件化的研究，并将研究成果运用于和欣智能手机邮件应用的软件架构设计及实现。

本文首先介绍了 Elastos 智能手机软件开发平台的相关技术，包括 CAR 构件技术及 XmlGlue 技术，并介绍了基于构件技术的 Elastos 智能手机软件开发平台，它们是本文智能手机应用开发的基础之一。

论文充分分析了智能手机的发展现状并结合 Elastos 智能手机开发平台特点，创造性提出基于 CAR 构件、运用 XmlGlue 技术的智能手机 MVC 软件架构模式，并对智能手机构件开发中的普遍问题进行分析，并将设计模式如何运用到构件开发中，进行了深入、详尽的探讨。希望可以从中总结出一些切实有用并具备通用性的构件化智能手机应用开发的方法。缩短智能手机软件开发的周期，提高软件开发的效率。

并将智能手机 MVC 软件架构模式及各种设计模式运用到智能手机邮件应用中。

随着通信技术的发展以及人们对移动通信业务中信息服务功能需求的增长，手机使用者对新型手机和应用的需求逐步扩大，而手机制造商则面临如何快速占领市场的问题，基于此目标，降低手机应用软件的开发难度、提高应用软件的复用度和缩短应用软件开发周期就成了手机制造商必须要面临的问题，基于智能手机的发展趋势和手机制造商面临的一系列问题，本文探讨了智能手机应用的软件构架模式和应用中设计模式的一些研究及应用，希望可以起到抛砖引玉的作用。

致 谢

转眼间，两年半的硕士研究生生活就要结束了，两年半，在人生的长河中只是沧海一粟，但却是我人生中很有意义的一段日子。因为它领我走入计算机软件研究的领域，开始了解科研的意义；经历很多事，得到很多帮助，让我学着用一颗感恩的心来生活。

首先我衷心地感谢我的导师陈榕，陈老师学识渊博、思维敏捷、平易近人，他始终站在学科的前沿，以自己的执著和热情致力于祖国的嵌入式操作系统事业！陈老师的言传身教，对我们的严格要求和悉心指导，都深深地影响着我，让我对软件编程、项目管理及学术研究都有了全新的认识。

其次，我要深深感谢顾伟楠老师和裴喜龙老师，正是他们细心、无私的指导让我能顺利完成研究生阶段课题的研究，他们的指导会让我终生受益。

我要感谢上海科泰世纪有限公司的许多同事，张利俊、闻英波、余宏、张永斗、沈金等等，正是他们在我实习阶段给予我的无私帮助让我能够顺利完成手机应用开发工作。

我还要感谢上海第二工业大学计算机与信息学院蒋川群院长、网络工程系主任张世明老师和网络工程系的各位老师，是他们的理解和支持让我能够顺利地完成研究生阶段繁重的学习和科研任务。

再次，我要感谢05计算机系2班的所有给予我帮助的同学们。

最后，我要真诚的感谢我的家人对我不变地支持和关爱，是他们对我无微不至的关怀才使得我能够专心于课题的研究。感谢他们在我烦躁与困惑时给予我关怀和宽慰。

参考文献

[1] 中国 3G 行业与手机游戏的发展趋势，J2ME 开发网，http://www.j2medev.com

[2] 智能手机发展趋势报告：趋向“构件化”发展，通信产业网，http://www.ccidcom.com/Index.html

[3] James W.Cooper. The Design patterns java Companion[M]:1998

[4] Buschmann,f 等.面向模式的软件体系结构 卷 1：模式系统（可荣等译）[M].北京：机械工业出版社，2003

[5] 涂文婕．武汉工程大学：用 JAVA 泛型实现设计模式：[硕士学位论文] ．武汉：武汉工程大学

[6] Elastos 构件运行平台与手机解决方案产业化推广策划 2006-12. http://www.elastos.com

[7] Elastos 技术白皮书 2006-12. http://www.elastos.com

[8] Elastos CAR 构件与编程模型技术文档 20061202. http://www.elastos.com

[9] 莫勇腾著，深入浅出设计模式（C#/Java 版）.北京：清华大学出版社，2006

[10] James. Objected-Oriented Modeling and Design. USA:Prentice-Hall Inc. 1991

[11] Erich Gamma.Richard Helm.Ralph Johnson.John Vlissides.译者：李英军等译.设计模式：可复用面向对象软件的基础.北京：机械工业出版社，2000

[12] 设计模式在企业管理系统中的研究与应用：[硕士学位论文]．辽宁：大连理工大学

[13] 软件的架构与设计模式之模式的种类，网易学院.教程，http://tech.163.com/05/0608/10/1LNH3BQD00091589.html

[14] 基于设计模式的软件开发方法研究与实践：[硕士学位论文]．成都：电子科技大学

[15] John Zukowshi. Learn Java with Jbuilder6; Apress. March 26, 2002

[16] Business processes-attempts to find a definition. Ann Lindsay, Denise Downs, Ken Lunn. Information and Software Technology, 2003, (45):1015-1019

[17] Jonathan B. Postel. Simple Mail Transfer Protocol RFC0821 1982

[18] J. Myers，Rose M. Post Office Protocol - Version 3 RFC1939 1996

[19] David H. Crocker. Standard for ARPA Internet Text Messages RFC0822 1982

[20] N. Freed. Multipurpose Internet Mail Extensions(MIME) Part One:Format of Internet Message Bodies RFC2045 1996

[21] N. Freed. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types RFC2046 1996

[22] 钱诚慎．SMTP 电子邮件客户端与服务器端设计与实现：[硕士学位论文]．辽宁：大连理工大学

[23] 韩金侠．POP3 电子邮件客户端与服务器端设计与实现：[硕士学位论文]．辽宁：大连理工大学

个人简历　在读期间发表的学术论文与研究成果

个人简历：

叶蓉，女，1980 年 9 月生。

2002 年 7 月毕业于南京师范大学 计算机科学与技术专业 获学士学位。

2005 年 9 月考入同济大学电子与信息工程学院计算机专业攻读硕士研究生。

已发表论文：

叶蓉，陈榕. 基于 CAR 构件的用户自定义事件机制的研究.电子技术应用，2007.9 月

叶蓉，陈榕. 基于 CAR 构件的和欣智能手机开发模型的研究.电子技术应用，2007.10 月

叶蓉，陈榕. 一种实现控件与多语言文本动态绑定的方法.计算机技术与发展，2008.1 月

叶蓉，陈榕. 运用 CAR 智能指针实现 CALLBACK 机制.计算机技术与发展，2008.2 月

研究成果：

参与国家 863 项目“基于中间件技术的因特网嵌入式操作系统及跨操作系统中间件运行平台”的研发并取得一定成绩。