同济大学
TONGJI UNIVERSITY

硕士学位论文

基于云计算的移动设备软件分享模型的研究与实现

（国家核高基资助项目　编号：2009ZX01039-002-002）

姓　　名：王保卫

学　　号：0820080261

所在院系：电子与信息工程学院

学科门类：工学

学科专业：计算机应用技术

指导教师：顾景文 陈榕 教授

副指导教师：顾伟楠 教授

二〇一一年三月

同济大学
TONGJI UNIVERSITY

A dissertation submitted to

Tongji University in conformity with the requirements for

the degree of Master of Science

The Research of Mobile's Software Share Model Based-on Cloud Computing And The Realization On Mobile

Candidate: Baowei Wang

Student Number: 0820080261

School/Department: School of Electronics and Information Engineering

Discipline: Engineering

Major: Computer Application Technology

Supervisor: Jingwen Gu, Rong Chen

Associate Supervisor: Weinan Gu

March, 2011

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3G无线网络以及wifi网络的建设，人们一直努力把基于有线网络的应用延伸到移动互联网中来，但是由于移动终端设备的弱计算能力、弱供电能力等特点，简单的靠移动终端完成网络与本地计算的协调依然存在很多问题，而更常见的做法是将网络计算能力服务化，通过云计算与移动终端的协调工作。在移动互联网中，越来越多的用户终端需要信息或计算资源的沟通，这个沟通的过程就是分享。

本文是在对分享的社会需求、计算需求等综合分析的基础上，对其进行抽象，提出一种软件的分享模型SCS（Super Component Share）。实现该模型的移动终端设备之间可以分享其软件列表中的软件应用程序，并基于Elastos操作系统，采用CAR构件技术实现了这个分享模型。

Elastos操作系统对于该分享模型的需求有着良好的技术支持。它为3G时代创新的应用模型及运营模式提供全方位的基础软件支持。另外在软件开发上，Elastos开发环境提供关于CAR（Component Assembly Runtime）编程模型和CAR构件运行时的最优支持。CAR不仅可以携带自描述信息（Class-Info），其编程模型还支持软件零件化生产。

论文首先提出了一个分享模型SCS，它以移动互联网为基础，利用云计算提供的服务，实现了在各个终端用户分享节点之间分享软件应用程序的功能。然后对模型中各个部分进行深入研究，完成了分享模型SCS的功能设计。根据模型设计，本文给出了分享模型SCS移动终端部分的基本设计和实现，并通过应用实例的实现和运行说明了这个模型的可用性，虽然离完整的商用还有一定距离，但是它对现有的一些电信应用如乐媒、彩信等相比，扩充了电信行业在移动互联网领域的新业务，它具有统一软件架构，方便业务扩展等，已经在一些工程中得到良好地应用。

关键字：云计算，CAR构件技术，构件化软件，软件分享、移动设备

ABSTRACT

With 3G wireless networks and wifi network construction, people always works hard to apply the application that based on the wired network to extend to wireless Internet. But because of the mobile device have the characteristic of weak compute ability、weak power ability and etc. Simply with the mobile to complete the network and local compute always have many problems, but through changing the network compute to services can resolve these problems. Through the coordination of cloud computing and mobile devices many weak points of mobile cannot be a problem. In the mobile internet, there are many application is realized to make information or compute resources communication between people and people or people and organization or organization and organization. Theses communications processes is called share.

This paper presents a software sharing model called SCS(Super Component Share) which is based on the analysis and abstract of the sharing of social needs, the computational needs. The realization of the SCS can make the two mobile devices share one or more software application in their devices. Based on the Elastos operating system, with the Car component technology, the paper realized this model.

Elastos operating system have a good technology support for the sharing model SCS and it provide a full range of basic software support to new innovative application model and operating mode in 3G times. Also in the software development, the development environment of Elastos provides the best support for CAR programming model and CAR component Runtime. CAR not only carries self-describing information but its programming model also support part of software production.

The paper first promote a sharing model SCS, which is based on mobile net, with the services that cloud computing support, realized the features of many mobile devices can share their software application with each other. And with the research of every part of the model SCS, completed the design of the features of this model. With the design, this paper realized the share engine in the mobile device and through the realizing of application declares the possible of this model. Although it have distances to be commercial, but compares to some of Telecom Applications like LuckMedia、

SMS and etc. It extends the new business in mobile internet. It has unified software architecture and facilitates business expansion, etc. It has been applied in some projects and has a good performance.

Key Words: Cloud Computing, CAR，Component-based Software, Sharing Software, Mobile Device

目录

第1章 前言

1.1 背景介绍

随着移动通信技术和 Web 应用技术的不断发展与创新，移动互联网业务将成为继宽带技术后互联网发展的又一个推动力，为互联网的发展提供一个新的平台，使得互联网更加普及，并以移动应用特有的随身性、可鉴权、可身份识别等优势。为传统的互联网类业务提供了新的发展空间和可持续发展的新商业模式。同时，移动互联网业务的发展为移动通信带来了无尽的应用空间。促进了移动网络宽带化的深入发展[1]。

3G 技术的发展，使网络带宽大大提高，为移动互联网发展奠定了基础。网络传输速率的提升，为软件下载和高带宽应用提供了良好的网络环境；网络时延进一步降低，提高了业务应用的交互能力；数据传输成本的降低，促进了数据业务的广泛普及[2]。从单一的文字短信到图文彩信，再到视频传送，让通讯双方的距离越来越近，信息量越来越大，随时随地的分享成为移动终端之间数据交流的需求。

由于移动终端的弱供电能力、若计算能力，在移动终端上存储大量的信息以及进行大量的本地计算已经不能满足人们对于大数据量的数据沟通和分享的需求。在这种情况下，本地计算网络服务化、大量的丰富的软件及其它资源网络存储化已经变成了一种发展趋势。因此云计算与移动互联网以及移动终端相结合，将能够不断的促进移动终端的从大量的计算中解放出来，使得移动终端之间进行大数据量的分享变的快速和简单。通过将传统互联网分享产品和移动终端相结合，推出移动分享平台，为广大移动终端用户提供更加时尚、更加便利的交流方式[3]。目前出现的最为流行的一种软件分享方式是软件商店，例如 App store——iPhone 软件商店以及多家 Android 平台软件商店，为第三方软件的软件开发者提供了一个方便而又高效的软件分享平台，通过这种方式提高了应用软件的丰富性，适应了移动终端用户对个性化应用软件的需求，从而使得手机软件业开始进入了一个高速的、良性发展的轨道。然而，从严格意义上说，App store 存在一定的封闭性，用户只能主动到应用商店上从数以万计的应用软件中查找自己需要的软件，给终端用户造成了极大的不便。

综上所述，目前的分享技术一般都停留在服务器存储内容、客户端主动获取内容的方式上，这种方式可以让终端用户去找到自己想要的内容，并且获取自己

想要的信息，但是它不能满足移动互联网下终端用户之间的信息沟通和交流。目前以终端为中心，实现终端用户之间数据分享平台尚在起步阶段，并没有被广泛的应用。因此，本论文将深入分析和解决终端用户之间进行软件的分享问题，研究并设计了分享模型 SCS。

1.2 课题的内容和意义

本文以核高基重大专项为背景，探讨了什么是软件分享以及如何实现软件在移动终端之间的传输分享，并在此基础上实现了移动设备的软件分享引擎以及应用开发。在 Elastos 上，CAR 构件技术已经非常成熟，其编程模型已经具备了坚实的理论和实践基础，并已经运用到了实际开发中，所以本文采用 CAR 构件技术作为软件分享模型的实现基础。

论文首先提出了一个分享模型 SCS，它以移动互联网为基础，通过服务器辅助传输，实现了在各个终端用户分享节点之间分享软件应用程序的功能。然后对模型中各个部分进行深入研究，完成了分享模型 SCS 的功能设计。根据模型设计，本文给出了分享模型 SCS 移动终端部分的基本设计和实现，并基于 XmlGlue 技术开发了应用程序实例。

分享模型 SCS 要求作为分享节点的某个终端设备具有可以分享的软件列表。分享节点可以决定分享的方法、需要分享的应用，终端分享引擎将根据用户的选择将其打包成 Capsule[4]并请求服务器与接收端建立此分享的网络连接。SCS 将会根据请求内容对其进行过滤以决定是否答应此请求并记录分享信息，然后根据结果建立分享收发节点之间的网络传输连接。

该分享模型通过软件构件以及应用软件的分享，可以达到升级软件功能，开展某项业务的能力，通过某个终端用户发起分享，可以使得同一地域或不同地域中的某个组中的每个终端都可以通过应用软件或者软件构件得到某项业务的使用权，从而极大的方便了业务的开展。以用户为中心的分享模型 SCS 对于软件以及软件构件的分享将成为未来移动互联网中应用分享技术的一个趋势。

分享模型 SCS 改变了原有的手机软件传播和营销模式，减低了手机厂商的软件成本，降低软件提供商的门坎和成本，使得软件提供商可以有机会面向最终用户，可以极大的方便软件的传播以及满足终端用户对于某种软件的需求。

1.3 论文的组织结构

论文首先介绍了移动互联网环境下移动终端设备间构件化软件分享背景以

及研究的内容和意义。然后，介绍了云计算、Elastos、CAR 构件和移动设备数据分享技术背景。其次，通过分析以上问题，设计了在互联网基础上的智能终端之间的构件化软件分享模型 SCS 以及分享策略。并根据分享模型以及分享策略实现了智能手机客户端的构件化软件分享引擎 ShareEngine 以及应用层 UI。最后，对构件分享模型的进行了阶段性总结，同时对后续工作进行展望。

论文共有六章：

第一章是引言部分，介绍了移动智能终端之间的软件以及数据分享的背景以及国内外研究现状，并介绍了课题研究的主要内容和意义。

第二章是相关技术与概念，主要介绍了云计算以及移动互联网下移动智能设备 Elastos 平台的基本特点。然后，介绍了构件技术相关理论和 Capsue 技术。

第三章是软件分享模型的分析和设计，首先研究和设计了 SCS 分享模型的总体架构，并深入分析了该架构模型的各个重要组成部分。

第四章是终端分享引擎 ShareEngine 的设计与实现，首先根据分享模型总体架构，详细研究终端引擎 ShareEngine 的各个功能模型，并进行设计与实现。

第五章是应用实例，从分享模型的终端应用功能需求设计、工作流设计以及界面实现三个部分分别介绍了如何实现终端应用程序。

第六章是工作总结与展望，总结本课题已取得的研究工作，并对以后的研究工作提出了展望。

第2章 相关技术与概念

2.1 云计算

云计算（Cloud computing）[5]是一种 IT 资源的交付模式，指通过网络（包括互联网 Interne 和企业内部网 Intranet）以按需、易扩展的方式获得所需的硬件、平台、软件及服务等资源、提供资源的网络被称为“云”。其计算能力通常是由分布式的大规模的集群和服务器虚拟化软件搭建。

云端的虚拟计算资源，通常是一些大型服务器集群，包括计算服务器、存储服务器、宽带资源等等。云计算将所有的计算资源集中起来，并由软件实现自动管理，无需人为参与。这使得应用提供者无需为繁琐的细节而烦恼，能够更加专注于自己的业务，降低成本。

云计算是分布式计算技术的一种，属于网络应用模式，其最基本的功能是通过网络将庞大的计算处理程序自动分拆成无数个较小的子程序，再交给由多个服务器所组成的庞大系统，服务器集群经搜寻、计算分析之后将处理结果回传给用户[6]。通过这项技术，云计算服务提供者可以在很短的时间内处理数以千万计的信息。

云计算具有以下特点：

（1）大规模。“云”具有相当的规模，Google 云计算已经拥有百万台服务器， Amazon、IBM、微软、Yahoo 等的“云”均拥有几十万台服务器。企业私有云一般拥有数百上千台服务器。庞大的计算机群使得“云”具有了强大的计算能力[5]。

（2）虚拟化。云计算支持用户在任意位置、使用各种终端获取应用服务。这些终端所请求的资源来自云端，而不是来自固定的有形的实体。应用在云端某处运行，但实际上用户无需了解、也不用担心应用运行的具体位置。只需要一台终端，就可以通过网络服务来获取我们需要的一切。

（3）通用性。云计算并不是针对特定的应用开发的，它可以支持不同的业务应用需要，同一个云端可以同时支撑不同的应用运行。

（4）可扩展性。云端的规模为满足应用和用户规模增长的需要可以根据实际情况动态伸缩。

（5）按需服务。云端是一个庞大的资源池，你可以按需购买相应的服务。

（6）廉价。云端采用极其廉价的服务器节点来构成云，云端的自动化集中

式管理使大量企业无需负担日益高昂的数据中心管理成本，其通用性使资源的利用率较之传统系统大幅提升，因此用户可以充分享受“云”的低成本优势。

由于云计算的众多优点，目前已经有很多公司纷纷推出自己的云计算服务平台，下面我们将分别介绍当前主流的云计算服务实现方案：

（1）Google 云计算[5]基础架构由四个部分组成，分别是建立在集群基础上的文件系统（Google file system）、Map/Reduce 分布式计算系统、分布式锁服务系统（Chubby）和分布式数据库系统（BigTable）。Google 提供的云计算服务使得用户可以通过浏览器就可以使用。

（2）Amazon 云计算又称为 AWS（Amazon Web）[7]，是一组服务的总称，包含了存储服务（Amazon Simple Storage Service）、计算服务（Amazon Elastic Computer Cloud）、消息传递服务（Amazon Simple Queue Service)和数据集服务（Amazon Simple DB)。AWS 提供了一整套的云的基础架构，同时提供了 Web Service 接口，用户可以通过浏览器来使用这些服务。

（3）开源云计算系统（Hadoop）[8]是由 Apache 软件基金会研发的，类似于 Google 的文件系统和 MapReduce，提供高效、稳定的分布式存储和运算。用户可以在不了解底层细节的情况下开发分布式程序。

（4）微软云计算 Windows Azure 是一个应用程序平台[9]，是构建在微软的数据中心的私有云服务平台，包含云操作系统、关系数据库（SQL Azure）以及基于.NET 的开发环境。它通过云存储和 Web Service 接口为用户提供在线服务。微软倡导的云计算是云和端的计算。

（5）基于应用虚拟化的云计算是通过在后端服务与终端显示之间增加了一个虚拟层，应用实际运行在虚拟层中，而将应用运行的显示界面推送到终端上显示。这种技术使得用户可以远程访问程序，就好像他们在本地终端上使用一样。

综上所述可以知道，这些云计算方案都是给终端来提供服务的，只是对终端的要求的高低有所不同，微软倡导的云计算是云+端的计算，软件的计算仍旧需要依赖终端来进行处理，而其他云计算技术中的运行、计算、存储都在云端，对于终端设备的要求很低，只需要能上网的终端就可以通过互联网来进行文档处理、远程计算等操作。云计算的核心思想是“分享软件服务”。让用户可以通过各类移动或固定终端，随时随地使用云计算提供的软件服务[46]。终端用户之间可以通过社交网络，通过云计算分享平台，相互“推送”软件服务。而用户无需关注技术实现和运行的细节，诸如安全、传输、升级等。

2.2 Elastos操作系统

Elastos操作系统是构件化的网络嵌入式操作系统，具有多进程、多线程、抢占式、多优先级任务调度等特性[10]。目前，Elastos已经可以在包括PC、ARM、MIPS等多种体系架构上运行。Elastos提供的功能模块全部基于CAR（Component Assembly Runtime）构件技术，这是Elastos操作系统的精髓。CAR构件技术规定了构件间相互调用的标准，每个CAR构件都包含自描述信息，可以在运行时动态裁剪组装，它贯穿于整个Elastos操作系统技术体系中。

从传统的操作系统体系结构的角度来看，Elastos操作系统可以看成是由微内核、构件支持模块、系统服务器组成的[11]。

微内核：主要可分为四大部分：硬件抽象层，对硬件的抽象描述，为该层之上的软件模块提供统一的接口；内存管理，规范化的内存管理接口，虚拟内存管理；任务管理，进程管理的基本支持，支持多进程，多线程；进程间通信，实现进程间通信的机制，是构件技术的基础设施。

构件支持模块：提供了对CAR构件的支持，实现了构件运行环境。构件支持模块并不是独立于微内核单独存在的，微内核中的进程间通讯部分为其提供了必要的支持功能。

系统服务器：在微内核体系结构的操作系统中，文件系统、设备驱动、网络支持等系统服务是由系统服务器提供的。在Elastos操作系统中，系统服务器都是以动态链接库的形式存在。

同时，Elastos操作系统提供的功能模块全部基于CAR构件技术，是可拆卸的构件，应用系统可以按照需要剪裁组装，或在运行时动态加载必要的构件，还可以用自己开发的构件替换已有模块。

Elastos操作系统的最大特点也是它最大的优势，即全面面向构件技术，在操作系统层提供了对构件运行环境的支持；用构件技术实现了“灵活”的操作系统。

在新一代互联网应用中，越来越多的嵌入式产品需要支持Web服务，而Web服务的提供一定是基于构件的。在这种应用中，用户通过网络获得服务程序，这个程序一定是带有自描述信息的构件，本地系统能够为这个程序建立运行环境，自动加载运行。这是新一代互联网应用的需要，是必然的发展方向。Elastos操作系统就是应这种需要而开发，率先在面向嵌入式系统应用的操作系统中实现了面向构件服务的技术。

实现Web服务的关键技术之一是面向构件、中间件的编程技术。Web服务提供的软件服务就是可执行的功能模块，就是构件。构件是包含了对其功能的自描述信息的程序模块。Elastos操作系统支持在网络环境下动态查找、动态链接构件，为Web服务提供了支撑平台[11]。

Elastos操作系统可广泛应用于信息家电、工业控制、传统工业改造、国防、

商业电子等领域，已经开发了PDA/掌上电脑、数控机床、工业远程监控设备、医疗仪器等应用。

2.3 构件技术

2.3.1 构件技术概述

构件技术是指通过对一系列可复用的软件构件的组装来开发新的软件系统的软件技术[12]。软件复用是指充分利用过去软件开发中积累的成果、知识和经验，来开发新的软件系统，使人们在新系统的开发中着重于解决出现的新问题、满足新需求，从而避免或减少软件开发中的重复劳动[13]。

近几年来，以面向对象为基础发展起来的软件构件技术，从某种层面上说，克服了以往的面向对象技术的某些缺陷，提高了软件复用程度[14]。

面向对象是将系统划分为多个对象（数据和相关方法的结合体），通过对象之间的通信和互操作形成耦合系统，它提供了客观世界与软件系统进行对应的编程思维方式，其具体技术包括封装性、多态性和继承性。

构件技术是在面向对象技术的基础上发展起来的[14]。面向对象技术通过类的封装和继承成功实现代码级的复用。类的封装性，实现数据抽象和信息隐蔽，继承性提高了代码复用性。但是面向对象的复用脱离不了代码复用的本质，对象之间的关系在编译时被固定，模块之间的关系是静态的，无法解决软件动态升级和软件模块动态替换等问题。

构件技术通过二进制的封装以及动态链接技术解决软件的动态升级和软件的动态替换问题[15]。构件技术对一组类的组合进行封装，并完成一个或多个功能的特定服务，同时为用户提供多个接口。整个构件隐藏了具体的实现，只通过接口提供服务。这样，在不同层次上，构件均可以将底层多个逻辑组合成高层次上的粒度更大的新构件。构件之间通过约定的接口进行数据交换和信息传递，构件的位置是相互透明的，可以在同一个用户进程空间，也可以在不同的用户进程空间，甚至在不同的机器上，而且不同的构件可以用不同的语言编写，只要它们符合事先约定的构件规范。

构件是具有强制性、封装性、透明性、互操作性和通用性的软件单元[12]。构件的粒度可大可小，可以是一个简单的按钮实现模型，也可以是潮流计算、状态估计等应用。构件使用与实现语言无关的接口定义语言（IDL）来定义接口。IDL文件描述了数据类型、操作和对象，客户通过它来构造一个请求，服务器则为一个指定对象的实现提供这些数据类型、操作和对象。

目前有三种比较成熟的构件技术，它们是微软公司提出的组件对象模型COM（Component Object Model）[47]及其分布式扩展DCOM（Distributed COM）[15]、对象管理组织OMG的通用对象请求代理体系结构CORBA（Common Object Request Broker Architecture)[16]]以及SUN的JavaBeans组件技术。

2.3.2 CAR 构件技术

CAR （Component Assembly Runtime）是上海科泰世纪科技有限公司开发的一种构件技术，它是一个面向构件的编程模型，表现为一组编程规范[17]。它规定了构件间相互调用的标准[11]。该标准包括了对构件、类、对象、接口等的定义，并且规定了访问构件对象的方法。这种二进制构件能够自描述，能够在运行时动态链接。其中Component Assembly主要有两层含义：一是软件零件，特指“目标代码单元”，在CAR编程规范中是DLL；另一个是软件组件，是软件零件的集合。软件组件不但包含一组DLL（也可以是单个DLL），而且还包含了零件清单、数字签名、零件依赖关系等信息。

构件自描述指的是构件能够描述自己的数据信息，这是通过在构件里面加入元数据的方式来实现[18]。元数据是描述数据的数据（data about data），它是对数据的抽象，主要描述了数据的类型信息。在Elastos中，构件中的元数据主要包括模块信息（ModuleInfo）、接口信息（InterfaceInfo）、类信息（ClassInfo）等。这些信息是通过编译CAR文件得到的，是CAR文件的二进制表述。

元数据描述了接口的函数布局和函数参数属性,也描述了构件之间的关系。有了元数据，不同构件之间的调用才成为可能，构件的远程化、进程间通讯、自动生成Proxy和Stub及自动Marshalling、Unmarshalling才能正确地进行。在运行时，Elastos运行环境会在加载构件时加载元数据，然后根据元数据实现构件的动态组装。

CAR构件具有的特性包含以下几个方面：

（1）构件自描述性，在CAR构件中，构件的元数据与构件的实现代码一起被打包进构件模块文件中。

（2）可重用性，CAR构件的重用建立在二进制代码重用的基础上，使得构件可以被多次使用。

（3）支持面向方面的编程，CAR的AOP机制可以使得用户方便的通过动态聚合两个CAR构件类来生成一个具有两个CAR构件类所有接口实现的新的构件类。

（4）具备远程过程调用的能力，CAR构件技术支持远程接口调用，构件服务和构件服务的调用者可以处于操作系统的不同空间，调用者如同在同一地址空

间里面使用构件服务。

（5）具有命名服务机制，命名服务机制是指将一个组件和指定的字符串绑定的过程，组件使用者可以远程通过组件名来调用该组件来获得服务。

（6）回调事件机制，客户程序可以将自己的事件处理函数进行注册，把函数指针告诉构件对象，构件对象在条件成熟时激发事件，回调事件处理函数。

（7）构件缓存机制，CAR构件的缓存机制是建立在IE缓存机制的思想基础上的，是一套构件化的缓存管理机制，能够支持网络操作系统的构件自动下载和运行。

（8）构件版本控制，可以利用CAR构件技术将版本管理器做成一个构件，用来专门管理其它CAR构件的版本。

（9）点击运行机制，在网络环境下软件无需事先在本地计算机上安装，在需要时通过网络中的相关对象来实现动态加载构件。

（10）CAR网络服务，CAR网络服务构建于CAR构件运行平台上，能够将运行的构件实例发布为一种网络服务，可以通过WWW标准网络服务协议来远程访问这种服务。

CAR技术简化了构件的开发过程，编写CAR文件后用CAR编译器编译便可生成基本的代码框架，开发人员在此基础上开发出自己的构件为客户端提供服务，提高了构件开发的速度及质量。

Elastos构件运行平台为CAR构件提供了编程运行环境，提供了一套符合CAR规范的系统服务构件及支持构件相关编程的API函数，实现和支持了系统构件及用户构件相互调用的机制。二进制构件直接运行在Elastos构件运行平台上，满足了嵌入式系统对应用软件的运行效率和实时性的要求。

对于使用CAR构件技术的程序员来说，他们可以受益于以下方面：（1）易学易用性（2）软件复用性（3）系统易升级性（4）可以动态加载构件（5）可以利用第三方软件丰富系统功能（6）可以提高系统的可靠性和容错性（7）可靠的系统安全性。

CAR构件技术是一个实现软件工厂化生产的先进技术，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；软件工厂化生产需要有零件的标准，CAR构件技术为建立软件标准提供了参考，有利于建立企业和行业的软件标准。

2.3.3 构件化软件

软件构件是指软件系统中具有相对独立功能、可以明确辨识、接口由契约指定、和语境有明显依赖关系、可独立部署、可组装的软件实体[13]。它由接口、实

现和部署三大要素构成[19]。其中接口主要解释构件所能完成的功能，实现是让此构件运作的代码，部署是构件的存在形式，即为二进制代码或可执行文件。

构件化软件开发是软件产业发展的重要途径，它将改变软件的生产方式，引起软件行业的变革[14]。但构件的复用能力、复用程度、构件标准化、智能化、构件的检索和应用等，都存在不足，是今后的研究方向。基于构件的软件开发具有适用范围广、重用效率高的特点，其效率的发挥需要大量标准的构件群来支持，这就需要标准统一的构件封装技术规格和应用于具体领域的大量构件。这两个方面现在是构件化发展的阻碍。构件封装技术规格的统一需要资深软件公司和国际性的软件机构深化合作、统筹兼顾来制定实用、有效、科学的标准。领域构件群的创建需要学术组织和有责任有远见的软件公司加大投入，深化构件化软件开发的应用，在业界形成模范作用。而中小型软件公司应该建立自己的构件库，在开发中尽量应用构件化软件开发方法，总结经验，为构件化时代的到来和繁盛做准备。有了统一的规格标准和各个领域丰富的构件群，那么构件化的开发将极大提高软件开发效率，解决软件危机[20]。

目前应用领域中的基本移动终端应用包括各种类型的终端应用软件，如娱乐类，办公类，业务类等软件。根据当前主流智能手机应用软件的特点，终端应用主要应用在信息服务、电子商务、数字个人助理、系统功能增强工具、游戏和多媒体娱乐和企业级应用软件等应用领域。

2.4 软件的分享

移动设备的快速发展，尤其是智能手机，而以3G智能手机为代表的移动设备的一个很大的特点就是与移动互联网相连接，使得智能手机成为移动互联网世界中的一个网络终端，这就大大方便了手机用户，使得他们可以随时随地上网。但是移动设备用户之间的通信方式以及数据分享方式都仍然是依靠运营商的短信和通话服务，而且产生的大量费用使得很多用户无法承受。

一直以来，人们探索着什么才是手机市场中最好的分享软件的方法。在智能手机发展的最初阶段，由于没有合适的传播途径供用户选择、网络通讯各方面技术还不够成熟等原因，让一款手机软件流行起来的最佳途径只能是通过说服运营商在手机中预装这种软件这种单一的方式，是一种难度非常大的商务过程。另外，由于软件成本、手机厂商的喜好等原因，手机厂商不可能预装众多的软件。在这一阶段，由于各种技术原因和手机厂商对第三方软件的垄断性，极大地限制了整个手机软件行业的发展。

分享是一种手机软件的传播和获取方式，即发送方通过某种途径，将自己手

机终端中的应用软件或者软件的构件发送给接收方，接收方可以在自己的手机终端中直接运行接收到的软件应用程序或者软件构件，它突破了传统的用户获取手机应用软件的方式。

在分享过程中，用户之间的大数据量的数据传递仍然有很大问题，一方面是网络带宽的限制，另一方面是移动运营商高额的收费使得用户无法承受。但是移动互联网的深入发展，使网络带宽和传输速度得到了大幅度的提升，使智能手机用户间通信以及数据分享成为可能，因此以移动互联网为基础，将数据分享机制纳入智能设备中来作为手机的一个功能已经成为必要的需求。

苹果公司目前也在研究分享这方面的技术，主要是提出了一种分享的方式，利用端到端（P2P）技术实现移动终端间之间的软件分享。它基于移动设备上的物理存储设备来存储软件应用列表，并通过用户选择将某些软件及其配置信息打包分享给其他终端用户。

创智数码公司将传统互联网分享产品和移动互联网相结合，推出移动分享平台PowerShare移动分享系统，该系统包含多媒体内容分享服务器、流媒体平台以及内容管理平台和移动终端几个部分组成。主要是用来方便移动终端之间分享多媒体信息，包括视频、图片、音乐、文件等。

2.5 Capsule技术

Capsule技术是上海科泰世纪提出的一种技术，是一种将商业服务和各种轻量级的网络小应用以及数字化内容同时封装在同一个可分发、可部署、可管理的应用程序包装容器之中的移动网络应用开发技术[4]。采用Capsule技术开发的移动网络应用称之为数字胶囊，它包括两个部分，一是应用程序包装容器（Capsule），一是被封装的内部应用及数字资源（Payload）。

采用Capsule技术将需要分享的构件化软件打包成一个Capsule，这样，有利于终端用户之间对于构件化软件的分享，通过Capsule来对其进行管理，方便用户对构件化软件的使用和管理。

在开发模型上，Capsule本身又是一种特殊的轻量级widget应用程序，采用和普通widget应用程序类似的开发技术和开发方法，可以像普通widget应用程序一样运行在Elastos、Android、WinMobile、Windows、Linux等平台之上。Capsule使用基于Elastos移动应用中间件的XmlGlue Widget应用开发技术，用XML编写UI界面、用Lua、Javascript等脚本语言编写业务逻辑、用CAR构件技术开发功能组件，通过XmlGlue引擎，实现脚本可以透明地加载和调用本地CAR构件，实现本地功能扩展。

Capsule是对数字时代的数字化内容（包括数字软件及服务，数字内容，数据业务等等）的封装和描述，它可以实现并提供以下功能：可封装多种形式的应用，将应用以多种分享形式分发出去，有利于数字化内容和应用的传播和商业运营。能够以不同的策略完成应用所需要的相关软件模块的下载、更新和加载。具有安全验证和用户权限管理功能，更强的安全管理与认证机制。面向Web服务的架构，以更灵活的方式自动实现相关软件服务的发现和绑定。能够以多种协议、甚至是用户自定义的协议与服务提供方或资源提供方建立连接。支持泛存储，软件模块、服务和数字资源都以GID的方式标识，资源的获取与存储位置无关，摆脱URL局限性约束。

Capsule技术是以Elastos技术体系作为其支撑和研发基础，实现移动网络数字资源应用零维护、点击运行、即插即用、自适应异构的计算环境和多变的终端设备，让软件对用户透明，让服务内容直接呈现给终端用户。它的结构图如图2.1所示。

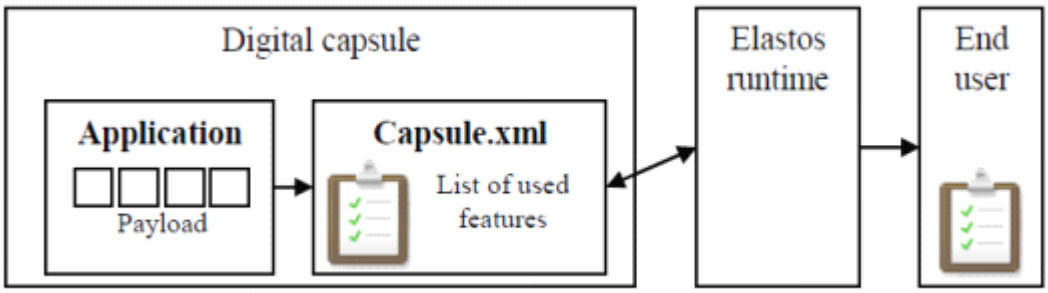


图 2.1 Capsule 结构图

采用Capsule技术封装的构件化软件具有新的功能，它具有自己的脚本，并可以通过对引擎功能的调用以及根据描述信息来管理构件化软件，以及从描述信息地址获取该构件化软件所依赖的其它构件化软件。Capsule应用可以被拥有使用权的用户自由地分享给其他用户，而不需要经过Capsule应用的运营方或厂商的同意，相反Capsule应用的运营方还应该为用户之间的这种分享行为提供相应的服务。

用户通过购买软件的副本和授权使用的产品序列号来使用软件及其提供的服务。而免费软件和共享软件是软件厂商主动放弃了全部或者部分版权，而允许用户或第三方免费使用和传播软件。

Capsule应用与一般软件相比，存在的一个最大的区别就是Capsule应用可以被拥有使用权的用户自由地分享给其他用户，而不需要经过Capsule应用的运营方或厂商的同意，相反Capsule应用的运营方还应该为用户之间的这种分享行为提供相应的服务。

用户自由地分享Capsule应用还体现在分享程度的可制定上，一般软件都有使用期限和使用次数的限制，也可以无限制期限和次数。但是对于Capsule应用用户来说，他可以自由决定将其所拥有的使用期限和使用次数的部分或全部分享给其他用户，而一般软件的使用者是不允许转让或分享使用期限和使用次数的。

第3章 SCS分享模型研究

3.1 SCS分享模型研究

3.1.1 SCS 分享模型概述

分享模型（SCS，Super Component Share)，是一个 Elastos 上实现移动终端之间进行软件构件或者应用程序分享的模型，为当今 3G 时代的移动终端之间构件或者软件分享的方式的单一性不能满足终端用户的需求而提出的，最终目的是为了通过传输服务商实现 Elastos 上的软件的构件或者应用软件在终端之间的分享，从而实现以用户为中心的终端分享节点间构件或者应用软件的分享。

SCS 模型的原理类似于通常在手机上用到的彩信系统，彩信是通过 GPRS 来发送或接收的。彩信在技术上并不是一种短信，而是在 GPRS 网络的支持下，以 WAP 无线应用协议为载体传送图片、声音和文字等信息。彩信业务可实现即时的手机端到端、手机终端到互联网或互联网到手机终端的多媒体信息传送。彩信的 WAP 协议传输方式受到带宽的限制，传输的内容大小受到限制，而 SCS 分享模型采用移动互联网，利用 3G 技术，打破了这种限制。SCS 分享模型通过移动互联网，利用 HTTP 协议进行在网络间传输，有效的拓宽了并加速和方便了终端用户之间的分享内容，能够实现即时的移动终端到终端、移动终端到互联网或互联网到移动终端的构件化软件的分享。

SCS分享模型从应用列表中选择某个应用并将其分享给其它接收终端用户，分享节点可以决定分享的方法、需要分享的应用，终端分享引擎将根据用户选择对其打包，成为一个成Capsule后，请求服务器与接收端建立分享的网络连接。SCS分享模型中的云端服务器将会根据请求内容对其进行过滤以决定是否答应此请求并记录分享信息，然后根据结果建立分享收发节点之间的网络传输连接，并完成分享的整个过程。

3.1.2 SCS 分享模型总体架构

SCS分享模型总体架构分为三个部分，分别是云端服务器、构件化软件的容器Capsule和客户端。图3.1为SCS分享模型架构图。

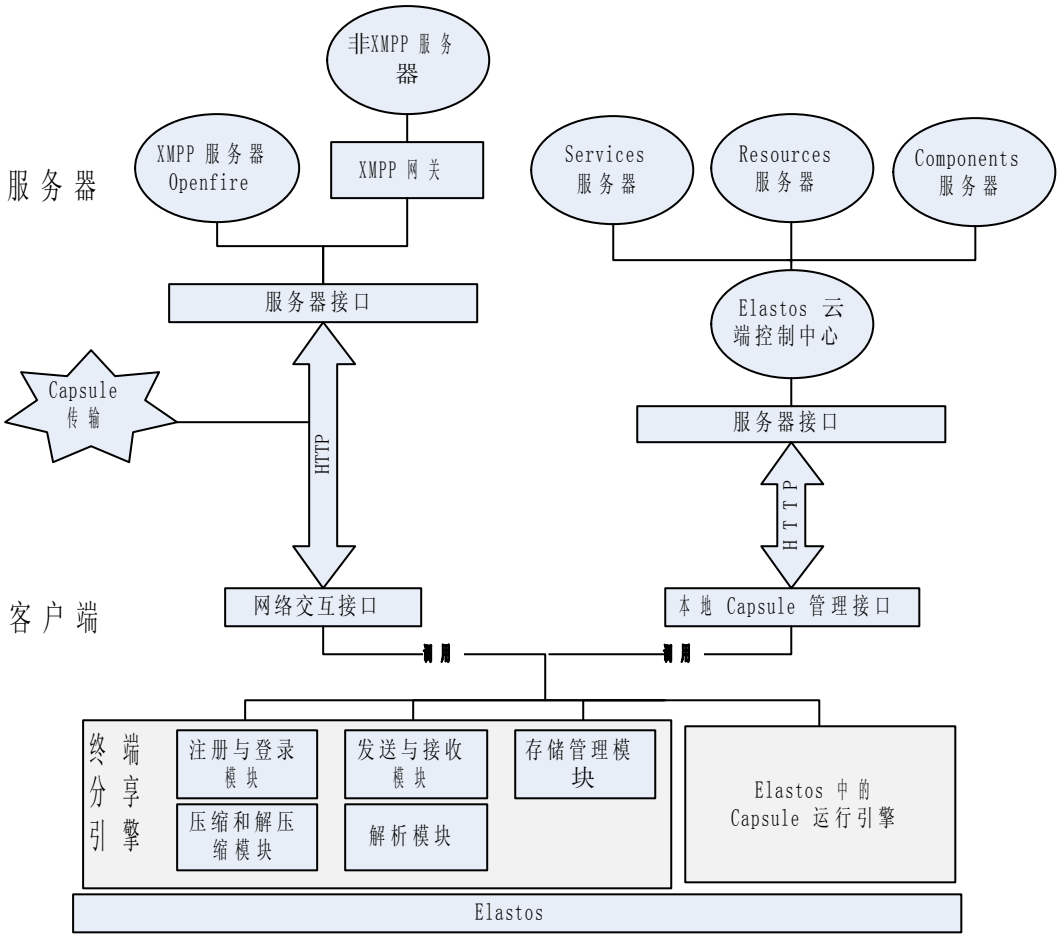


图 3.1 SCS 分享模型的总体架构图

其中，云端服务器包括两部分，一部分是负责Capsule的传输所需要的传输服务器包括Openfire（XMPP服务器）、XMPP网关，还包括非XMPP服务器。一部分是获取服务的服务器，包括从服务器上获取一些依赖性软件或者构件、服务或者某种资源等，包括Elastos云端控制中心，负责分发Capsule的一些请求到具体的服务器上（这些服务器包括Components服务器、Resources服务器、Services服务器等），如：当分享的应用或者构件需要依赖其它的构件时，就会请求Elastos云端控制中心，控制中心根据请求描述信息，将该请求转发到Components服务器，服务器查找到该构件后将相关信息返回终端，终端根据Capsule将该依赖构件下载到本地。

本文的重点在于通过利用传输服务器来进行终端之间的Capsule应用的分享，下面是SCS分享模型中终端之间Capsule分享的基本抽象模型，如图3.2所示：

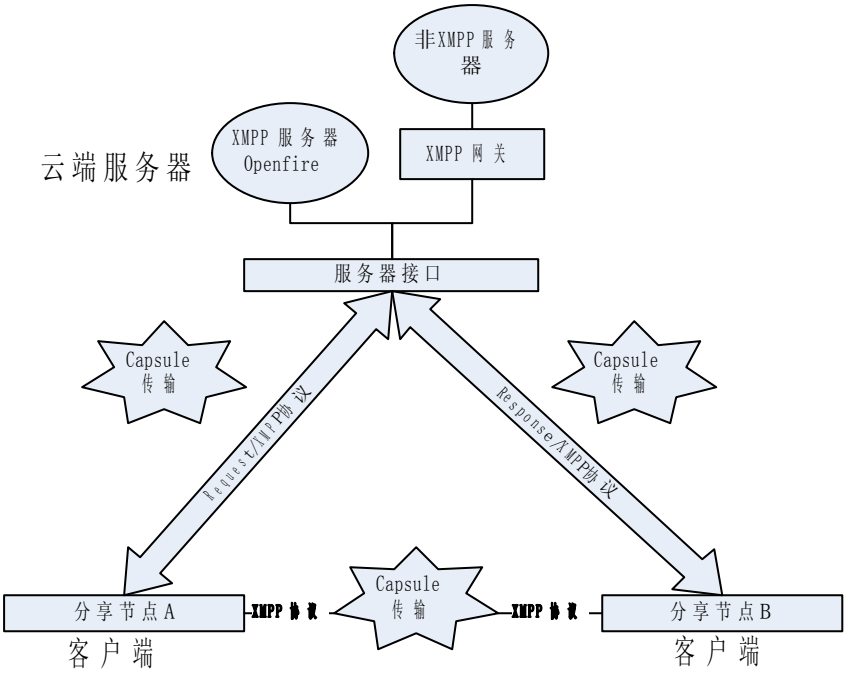


图 3.2 Capsule 应用的分享模型

Capsule 应用的分享模型包括 Openfire（XMPP 服务器）[21]、XMPP 网关和终端软件分享系统引擎及其应用实现。Openfire 服务器可以是单个服务器，也可以是多个服务器构成的集群，能够快速有效的扩展该分享模型的计算能力，也可以通过扩展服务器来增加新的业务功能。为了快速的构建并实现该分享模型，服务器采用开源的 openfire，而通信协议采用 XMPP 协议来实现终端与云端的通信。

终端之间 Capsule 的分享状态分为离线和在线两种，在线状态是指要分享数据的终端之间可以接收到网络信号，并且可以连接到网络的状态。离线状态则相反，是指终端中一方失去网络信号或者处于网络关闭或者关机而不能连接网络的状态。

当终端的接收方处于离线状态时，发送方将请求云端建立连接，云端将会记录一系列发送信息，包括软件的大小、软件的版本、软件是否允许再次分享、软件的拥有者以及软件的发送者和接收者等信息。同时云端检测接收端状态，若处于离线状态，将暂时缓存该分享数据，当接收方处于在线状态时，将会通知接收方是否接收数据，若接收方拒绝接收数据，那么该消息将会反馈到软件发送方，同时将缓存数据删除；若接收方接受发送数据，那么数据将发送给接收方。当接收方接收完毕，云端将反馈消息给发送方该软件已经发送到接收方。

当接收方处于在线状态时，云端将会记录一系列软件的发送信息，和处于离线状态时记录的信息一致，并通知接收方是否接收信息。

为了保证通信会话的安全，将采用请求授权的方式来进行安全连接，数据发

送方在请求连接分享数据时，云端将根据该用户来进行生成唯一授权码，该授权码每隔一段时间就会自动更新一次，云端会将该更新的授权码发送给数据发送方，发送方通过终端的分享服务来更新授权码。请求连接中必须包含这个授权码，当云端同意连接后，将可以接收数据，并将数据缓存起来。当接收者接收时，也以同样的方式来建立安全连接，从而保证数据的安全传输。

通过服务器建立安全连接，让客户端之间进行数据传输，通过服务器的数据缓存可以实现客户端之间的大数据分享，如业务构件的分享、应用软件的分享、用户大量文件的批量分享等。

客户端通过终端的分享收发功能模块接收到一个构件化软件分享请求时，将会通过终端的一系列操作来实现对该分享内容的管理，包括解压缩该分享包、解析描述信息、存储数据、显示分享内容、构件化软件的启用和停用，对构件化软件的失效时间的控制等。

3.1.3 SCS 分享模型的特点

基于云计算的 SCS 分享模型具有很多的特点，体现在以下几个方面：

（1）该模型具有很强的扩展性

可以在该模型的基础上进行扩展，实现新的功能以满足新的需求。终端之间的软件分享是通过 Capsule 容器来对其进行封装的，终端可以通过对 Capsule 进行配置以实现不同的传输策略，也可以通过扩展服务器为该 Capsule 的分享提供更多的技术和服务支持。

（2）该模型可以实现有效的终端间数据的分享

该模型利用云计算的强大的计算能力，通过采用云端服务器控制传输的策略实现了移动终端之间数据的有效分享。服务器根据不同的应用可以有两个作用，一是在网络不好或者接收者处于离线状态的情况下实现终端之间的离线分享，一是可以记录并控制数据在该模型下的运行轨迹，并在网络情况好或者接收者处于在线状态的情况下建立端到端的数据分享。

（3）构件化软件的分享具有很高的安全性

终端之间需要分享构件化软件时，分享终端将会通过引擎对其进行打包压缩，利用 Capsule 技术，将构件化软件填充进一个 Capsule 中，分享节点在传输前可以对该 Capsule 进行数字签名和加密，从而可以保证需要分享的构件化软件的安全性。

3.2 构件化软件的有效生命周期

在该分享模型 SCS 中，分享的构件化软件都需要有一个生命周期。基本上，一个分享的构件化软件有 4 种状态，分别是启动状态、前台状态、后台状态、销毁状态。当接收者从服务器或者分享节点上接收到一个分享的构件化软件以后，就会将该构件化软件存储起来，并在需要时启动它。所有程序的状态只有 ShareEngine 可以设置，一般被分享的构件化软件无权更改自己的或者其他软件的状态。图 3.3 描述了一个被分享的构件化软件的生命周期，当被分享的构件化软件要从一个状态转变为另一个状态时，应用会主动调用对应的回调函数（AtEntry、OnRunAtBackground、OnRunAtForeground、AppExit）。

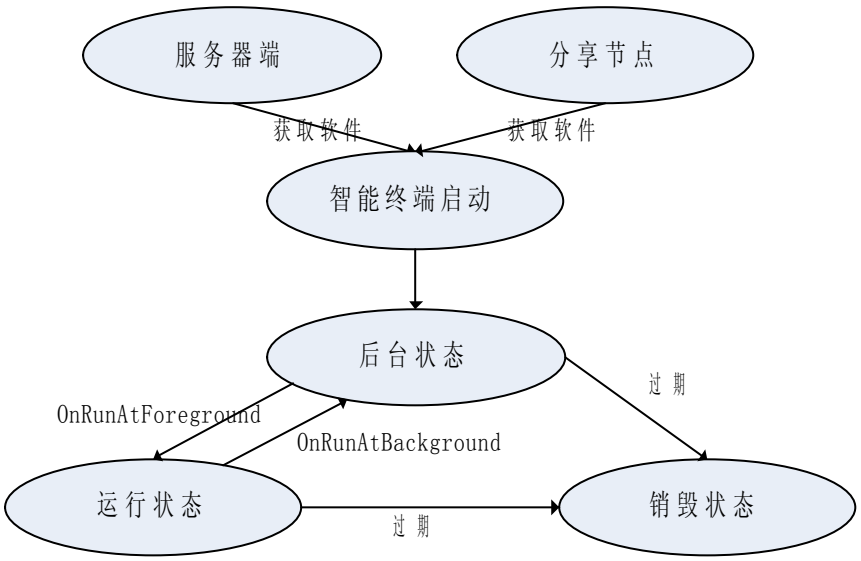


图 3.3 被分享的构件化软件生命周期

3.3 Capsule装载构件化软件的原理

由 Capsule 技术可知一个 Capsule 应用实际上可以看成两个部分，一是 Capsule 应用程序包装容器，一是被封装的内部应用及数字资源。

在 Capsule 中可以封装多种形式的应用程序和资源，被封装的应用程序可以是*.eco 可执行程序，也可以是 widgets 应用，也可以是 html 文件与 JavaScript 脚本，甚至可以是另一个 Capsule 应用。

可以封装在 Capsule 中的资源包括：程序文件、配置文件、数据文件、文本图片音视频等多媒体文件，甚至还包括若干设备驱动和软件构件以及 Capsule.xml 和 Payload.xml 文件组成的，它们以规定的路径保存在规定的目录结构里。当要打包一个构件化软件到 Capsule 中并将其分享时，可以按不同的策略选择需要的文件和资源进行打包。

在用户选择一个构件化软件来分享时，终端分享引擎 ShareEngine 将会压缩

该软件并收集该软件的配置信息，然后形成一个 Capsule.xml 配置文件，并将该配置文件与软件的压缩包一起打包产生一个 Capsule 应用，并拥有唯一 ID 来标识该软件。该 Capsule 应用将会变成分享的实体，通过的分享模型在终端用户之间相互分享。

3.4 SCS分享模型的云计算

云计算的核心思想是“分享软件服务”。让用户可以通过各类移动或固定终端，随时随地访问云计算机提供的软件服务[46]。移动终端用户之间可以通过移动互联网，互相“推送”软件服务。软件服务从云（Cloud）上来，在端（Client）上跑，用户无需关心软件服务的安装及运行细节。

本文讨论的分享模型 SCS 中的云计算负责分享的软件的检测、记录、资源查找以及传输等控制，它由传输服务、存储服务、安全访问控制服务、资源查找服务等组成。通过云计算，移动终端之间分享软件可以通过分享该软件的一些配置信息，这样移动终端接收到以后可以通过将该配置信息发送到 Elastos 云控制中心，由该中心请求资源查找服务，将该配置信息中涉及到的依赖性软件或者构件收集起来，将结果返回给接收端，实现终端之间的软件分享。当终端分享本地软件到另一个终端时，软件分享终端将会通过 Capsule 技术将其打包，然后发送到云端传输服务中心，该中心将会检测该软件的详细配置信息，然后根据该软件的完整性来进行判断是否要发送依赖性软件或者构件的查找请求。

云端服务器架构采用开源服务器软件 Openfire，它支持 XMPP 协议，是一套根据 XMPP 协议开发的服务器端软件，具有着良好的性能和很高的安全性，而且部署管理也非常简便[22]。

客户端之间的消息传递必须通过云端服务器的过滤和转发，也可以通过服务器建立客户端之间的一个直接连接，所以在经过服务器建立这个连接后，就可以进行软件的分享了，在这一过程中，云端服务器还能够对传输进行有效的控制。

云端服务器架构图如图 3.4 所示。

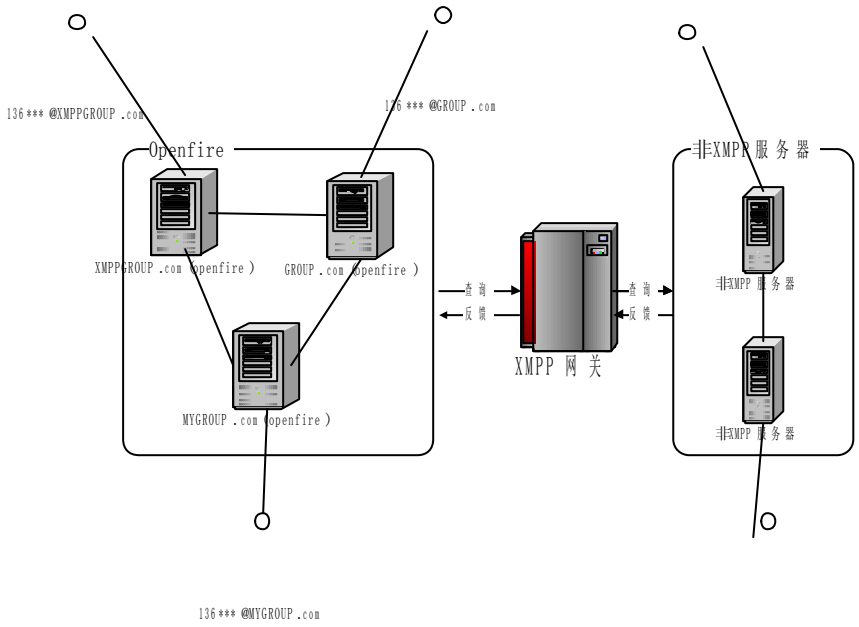
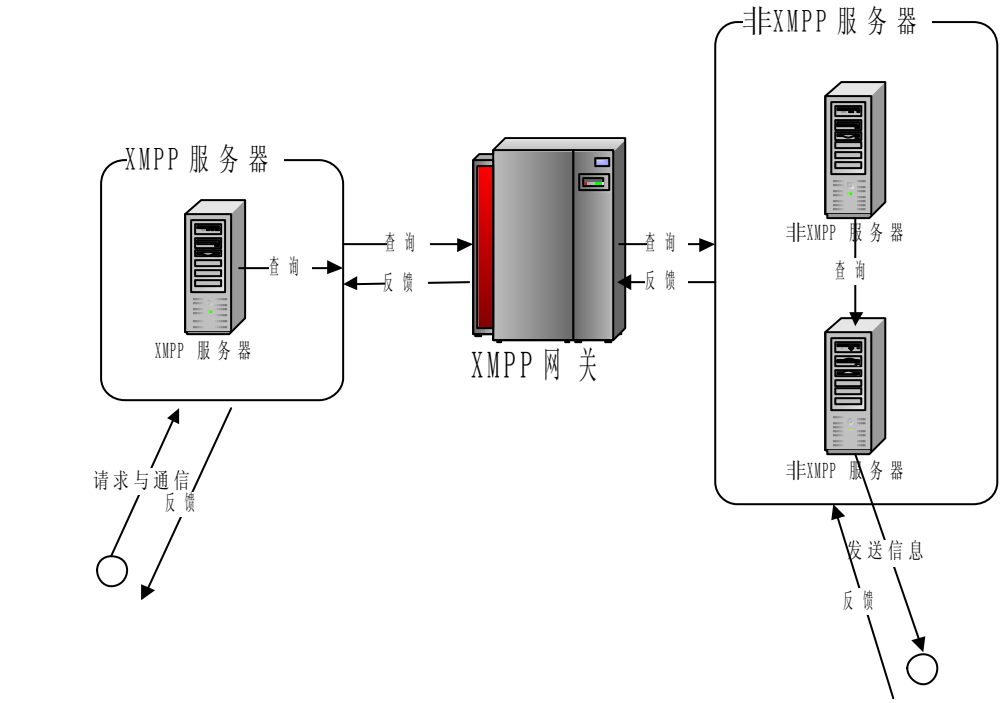


图 3.4 云端服务器架构图

3.4.1 服务器的分布式扩展

XMPP网络体系是模仿的E-Mail系统构造起来的，每一个用户都有自己的本地服务器，可以从该服务器上接收消息。消息在这些服务器之间进行传输后传递到终端，因此，可以在服务器端添加任意多个XMPP服务器[23]，这些服务器能够响应终端的请求，并通过服务器之间的协作，完成终端要求的功能，并在终端之间建立的连接进行传输。每一个XMPP服务器都独立于其它服务器，并且拥有其自己的一系列用户列表，通过网络上的任一个XMPP服务器都可以与其它服务器进行通话[24]，XMPP地址与E-Mail地址的形式是一致的，如：test@jabber.org。在这里使用的是电话号码来作为用户名，如：136*****@jabber.org。通过电话号码作为用户名可以实现终端的联系人列表的选择更加方便。

XMPP服务器的扩展可以扩展到非XMPP服务器，这种扩展需要通过在服务器端架设协议转换网关来实现，该转换网关需要遵守RFC3922协议。这个协议规定了XMPP与公共显示和即时消息（CPM）的映射。例如要实现XMPP服务器与非XMPP服务器的连接，其示意图如图3.5所示。



XMPP 终 端 用 户 A

非XMPP 终 端 用 户 B

图 3.5 XMPP 分布式服务器架构图

3.4.2 服务器之间通信机制

一个典型的多网络通信的模型图如图3.1所示。图示左边为XMPP服务网络，右边为非XMPP服务网络。在XMPP服务网络中，有三台XMPP服务器，三台XMPP服务终端，和一台XMPP网关。在非XMPP服务网络中有两台其它的非XMPP服务器，两台非XMPP服务的客户端。

在 XMPP 服 务 网 络 中， 三 台 XMPP 服 务 终 端， 分 别 属 于 三 个 域 XMPPGROUP.com、GROUP.com与MyGROUP.com，图中实线连接的两个终点表示的是真实的物理通信链路。

（1）XMPP网络内的终端通信

假 设 终 端 客 户 A （ 136***@XMPPGROUP.com ） 希 望 与 终 端 客 户 B （136***@MYGROUP.com/home）进行通信，首先终端客户A发送的消息必须传递到XMPPGROUP.com这台服务器，又由这台服务器转发给终端客户B所在MYGROUP.com服务器，最后经过MYGROUP.com服务器转发给终端客户B。在物理通信链路上，此次通信过程经过了两个端点服务器的中转，而不是终端客户A和B之间的直接通信，这使得服务器在安全性上有一定的提升，确保了通信双

方都必须通过验证才可以进行通信。

（2）XMPP网络与非XMPP网络的终端通信

当终端客户B希望与其它非XMPP服务器系统中的终端进行通信时，首先是终端客户B将信息发送给其所在域的XMPP服务器XMPPGroup.com，服务器XMPPGROUP.com查询到发送信息的接收方不在XMPP网络中，即在其服务器的服务中查找是否有相应的网关可以将XMPP协议翻译成外部非XMPP服务器所能识别的信息，最终服务器XMPPGROUP.com将终端客户B发送的消息递交给XMPP网关服务器处理，网关将翻译后的信息传递到相应的目的非XMPP网络中的服务器，通过该服务器将消息发送给接收终端，达到双方通信的目的[25]。

（3）实体通过不同资源的通信

假设136***@XMPPGROUP.com/phone是XMPP中终端用户A的另一个实体，通过资源名称“/phone”标识了与“/home”的不同。XMPP系统正是通过这种利用资源名称的不同，标识了同一帐户可以在同一时间与多个终端实体进行通信而互不影响。在此模型图中终端客户136***@XMPPGROUP.com/phone与终端用户A相互通信的过程中，并不影响同一用户的另外一个实体B与其它终端客户之间的通信。

3.4.3 数据安全性传输机制

服务器之间的安全机制包含了两个方面：一是数据的机密性和完整性，一是认证。数据的保密性指的是保证网络的各个环节都能得到数据完整性和机密性，也就是用来保证数据在传输过程中数据不被篡改和窃听。认证就是要保证通信双方的身份的真实性，它不同于信息保护和加密处理，服务器间引入了回拨机制，通过认证可以有效避免通信过程中利用服务器欺骗方式进行信息伪造的情况[26]。回拨认证通过DNS技术来实现，利用回拨认证技术，一个服务器可以确认与自己建立连接的服务器是否经过合法授权[28]。这个过程一般会涉及到三个方面：源服务器、目的服务器和授权服务器。其中授权服务器可以提供对源服务器的DNS验证查询[27]，它可以是源服务器本身，也可以是源服务器网络中的一个独立服务器。一般回拨认证过程如下：

（1）源服务器和目的服务器之间的认证。由源服务器告诉目的服务器需要认证，然后目的服务器回应源服务器采用认证并分配认证会话ID。

（2）源服务器发送密钥到目的服务器。目的服务器接收到密钥后并不对密钥进行验证处理，因为目的服务器没有保存任何源服务器的信息，所以它需要通过把密钥传递给授权服务器来进行验证。

（3）目的服务器与授权服务器连接。连接的过程与源服务器和目的服务器

之间建立认证连接的过程相同，授权服务器同样会为处理认证过程分配一个验证会话ID。

（4）目的服务器提交密钥到授权服务器来验证。这个过程目的服务器会把源服务器提交来的密钥以及双方的会话ID提交给授权服务器。授权服务器验证密钥，然后返回认证结果给目的服务器。

（5）目的服务器受到认证结果并通知源服务器认证的结果。

为了保证服务器间发送的数据不被篡改或偷听，服务器之间的通信引入了TLS机制，通过使用TLS的数据加密功能，从而可以保证数据在传输过程中的安全。TLS协议由两部分组成：TLS记录协议和TLS握手协议[29]。TLS记录协议有很多优点：（1）使用对称加密机制，对称加密所产生的密钥对每个连接都是唯一的，且此密钥基于另一个协议（如握手协议）协商。进行信息传输包括使用密钥进行信息完整性检查。（2）用于MAC计算的安全哈希功能（SHA、MD5等）。在没有MAC的情况下记录协议也能操作，但一般只能用于这种模式，即有另一个协议正在使用记录协议传输协商安全参数。（3）TLS记录协议用于封装各种高层协议。握手协议通过这种封装协议封装后，允许服务器与客户机在传输和接收数据前相互认证、协商加密算法和加密密钥。加密密钥是通过双方协商确定的，对第三方来说协商加密是难以获得的，即使是进入连接中间的攻击者也不能。以下例子说明了服务器S1和服务器S2之间使用TLS保护数据流的过程。

步骤1：S1给S2发送初始化流：

```
<stream:stream
  xmlns='jabber:server'
  xmlns:stream='http://etherx.jabber.org/streams'
  to='test.com'
  version='1.0'>
```

步骤2：S2给S1发送应答流：

```
<stream:stream
  xmlns='jabber:server'
  xmlns:stream='http://etherx.jabber.org/streams'
  from='test.com'
  id='sts1'
  version='1.0'>
```

步骤3：S2发送TLS给S1，这中间包含验证机制和其他流属性：

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
```

```
    <required/>
    </starttls>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
        <mechanism>DIGEST-MD5</mechanism>
        <mechanism>KERBEROS_V4</mechanism>
    </mechanisms>
</stream:features>
```

步骤4：S1发送TLS给S2：

```
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

步骤5：S2通知S1继续处理：

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

步骤6：S1和S2尝试通过TCP完成TLS握手。

步骤7: 如果TLS 握手成功，S1 初始化一个新的流给S2:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='test.com'
    version='1.0'>
```

步骤8: S2 发送给S1一个流头信息,其中包含任何可用流特性:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='test.com'
    id='sts2'
    version='1.0'>
    <stream:features>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
        <mechanism>DIGEST-MD5</mechanism>
        <mechanism>KERBEROS_V4</mechanism>
        <mechanism>EXTERNAL</mechanism>
    </mechanisms>
    </stream:features>
```

步骤9：S1继续进行SASL握手。

3.5 SCS的通信协议

分享模型中的通信协议采用XMPP协议，这种协议具有很强的扩展性、开放性和安全性，能够保证终端之间的构件化软件的安全和快速传输。

3.5.1 XMPP 协议

XMPP协议是基于可扩展标记语言（XML)的协议，是用来实现任意两个终端、终端与服务器、服务器与服务器之间准实时的结构化信息交换的通信协议[30]。XMPP提供了一个标准规范，可以通过XMPP提供的可扩展的框架用来交换数据，是一种开放式的传输XML流的协议[31]，具有如下的优点：

（1）开放性

XMPP的前身是Jabber，由开源形式组织开发的网络即时通信协议。XMPP目前被IETF国际标准组织完成了标准化工作。它的初衷就是构建一套免费的、开源的即时通信协议。任何企业和个人都可以免费的使用XMPP协议。

（2）安全性

XMPP协议采用了多种安全措施用来保证传输的信息的安全，充分考虑到了在复杂的网络传输条件下，传输的信息流的安全性问题。任何信息的传递都要经过TLS安全传输层协议、SASL和安全层协议[32]，这种多层安全验证机制可以保证信息的安全，其中涉及到多种强度算法如BIGEST-MD5[33]、KERBEROS_V4、PLAIN等，利用这些算法的证书可以保证安全的信息传输。可以说XMPP在目前的即时通信协议中具有比较高的安全性[28]。一个XML流的传输安全认证层如图3.6所示：

图 3.6XML 流的传输安全认证层

（3）可扩展性

XMPP协议的最大特点就是基于结构化的XML数据传输，因为XML语言具有易扩展的特性，所以在满足XMPP协议传输的XML节元素结构标准之下，可以方便的添加新的属性或添加新的子节点来扩展现有的协议功能。也就是，在通过通信双方确认的情况下，可以通过自定义XML结构，可以在XML中加入任何的新的元素或属性来表示出新的功能。XSF利用XMPP良好的扩展性正致力于制定各种XMPP协议的扩展协议，来不断的满足即时通信中涉及到的新的需求。

（4）跨平台

XMPP协议已经形成一个标准，可以根据这个标准在各种平台下实现，在不同的平台下，都是根据该平台来实现相同的XMPP协议，例如服务器、移动终端设备、嵌入式设备等。

通过分析XMPP协议的特点，可以看出XMPP具有很高的安全性、可扩展性，可以很好的满足分享系统的需求，并且由于其开放性，能够实现与其他即时通信系统的互联互通，因而选择XMPP协议进行分享系统的开发。

与 XMPP 协议并存的还有一些其它开放式的协议，例如 SIMPLE（SIP for Instant Messaging and Presence Leveraging Extensions）协议[35]。SIMPLE 协议也是 IETF 的标准协议，是一个目前为止制定的较为完善的一个即时通信协议，它和 XMPP 协议一样，该协议也符合 RFC2778 和 RFC2779 的要求，与 XMPP 协议不同的是，SIMPLE 协议是基于 SIP 协议传输通信数据的。SIP 最初是由 IETF 制定来为终端服务的协议，一般情况下应用在建立语音通话领域，一旦连接以后，依靠即时协议(RTP)来进行实际上的语音发送。SIP 不仅仅被用在语音中，也可以用于视频传输领域中。SIMPLE 是基于 SIP 协议发展起来的即时通信技术，对 SIP 进行了一定的扩展。SIMPLE 协议出现的时间比 XMPP 协议出现的时间早，出现了大量的商用化的产品和方案，目前微软、IBM 等企业的即时通信系统很多都是基于 SIMPLE 协议开发的，应用范围较广泛，技术也较成熟。而 XMPP 技术属于新生技术，但是由于 XMPP 极具扩展性，在 XML 基础上可以很容易扩展到不同的应用和系统，并且 XMPP 以及扩展协议形成标准化协议的数量与范围应用更为广泛。但 XMPP 协议与 SIMPLE 协议相比，不足之处在于语音和视频传输方面的支持不足，SIMPLE 基于 SIP 协议使其在这方面具有先天优势，并且技术也比较成熟，但是 XMPP 协议也没有停止这方面的发展，语音和视频传输的 XMPP 扩展协议也在制定与完善中。XMPP 在扩展性和标准化等方面的优势，使得越来越多国际公司企业的纷纷对其支持，例如 Google 等，可以说，未来 XMPP 比 SIMPLE 更具有市场优势。SIMPLE 与 XMPP 协议的详细比较如表 3.1 所示。

表3.1 SIMPLE与XMPP对比

协议	SIMPLE	XMPP

基础	SIP	XML 协议
功能	支持各种即时消息通信	支持各种即时消息通信
扩展能力	一般	很强
主流厂商支持	微软、IBM 等	Google、HP 等
前景	已经率先广泛应用	后来居上

3.5.2 XMPP 架构

XMPP 架构图如图 3.7 所示。其中包含四个部分：XMPP 服务器、XMPP 网关、XMPP 终端以及 XMPP 协议。其终端之间的通信包含 XMPP 终端与 XMPP 终端之间的通过 XMPP 服务器进行的通信以及 XMPP 终端通过网关与非 XMPP 终端之间的通信如与 SMTP 服务器终端的通信等。

XMPP 协议与其他的应用层协议有着相似之处。在这些协议中，具有惟一名称的终端可以通过相关的服务器与另外一个具有惟一名称的终端进行通信。每个终端执行自己的协议表单，而服务器在表单中提供路由功能。服务器还可以针对不同域之间的路由进行通信，例如在 test1.com 和 test2.com 之间。此外，网关可用于在外部消息传递域和协议之间进行转换。图 3.7 中的示例，网关通往一个短信服务（SMS）域和一个 SMTP 域。在这种情况下，网关大多数都是被用来在 IM 协议之间进行转换。作为一个可扩展的协议，XMPP 对于在不同的端点协议间提供统一连接性来说是一个理想的中枢协议。XMPP 网关允许终止一个给定的客户端到服务器的会话，并且向目标端点协议发起一个新的会话，同时进行必要的协议转换。

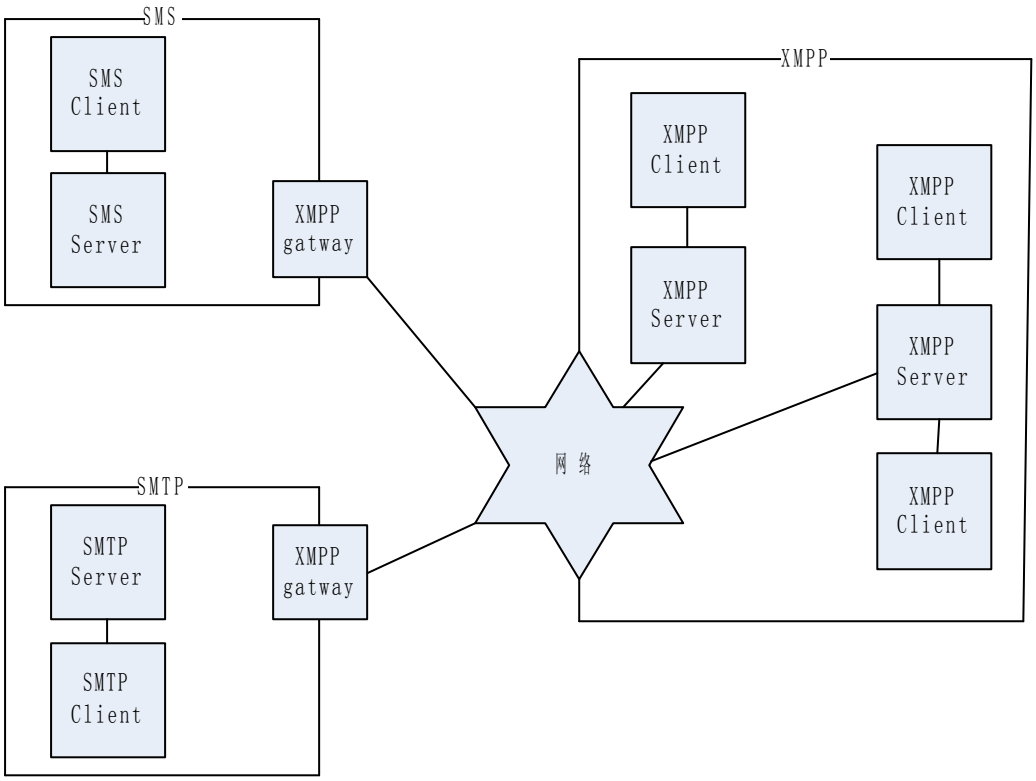


图3.7 SCS中的XMPP架构

3.5.3 XMPP 数据传输结构

XML，即可扩展标记语言。它是一种简单的数据存储语言，和 HTML 一样，通过使用一系列简单的标记来描述数据。XMPP 协议就是利用 XML 的结构化数据存储的形式进行信息传递的。在 XMPP 协议中定义了两种 XML 实体：

（1）XML 流

XML 流类似于一个装载器的功能，用于装载一些信息，这些信息在两个实体之间通过网络相互传输。XML 流是由<stream>标签开始的，以标签</stream>。在流的整个生命周期，初始化它的实体可以通过流发送大量的 XML 元素，用于流的控制信息和 XML 节信息[31]。

（2）XML 节

XML 节是一个离散的语义单位，它存在于通过 XML 流向另一个实体发送的结构化信息中，一个 XML 节在根元素<stream/>的下一级，它是一个完整均衡的元素，是成对出现的，也就是说它拥有打开和关闭标签。XMPP 协议中， TLS 握手，SASL 握手和服务器回拨所需要发送的 XML 元素不属于 XML 节，其它所有的信息传递都属于 XML 节[36]。

初始化方和接收方在相互发送的 XML 节中一般都含有以下三种最常用的属

性值：to 属性：接收标识此 XML 信息节的实体地址；from 属性：发送此 XML 信息节的实际地址；xmlns 属性：标识此 XML 节具体使用的子元素或属性的命名空间。每一个 XML 节元素的命名空间在 XMPP 标准协议中都是标准化定义的，它有很重要的用途，XMPP 的扩展协议有很多，支持不同扩展协议的通信双方在相互通信时只能通过 xmlns(xml name space)属性值来确认双方是否能识别对方所发送的 XML 节信息。在 XMPP 通信中主要的节类型是<presence/>节、<message/>节和<iq/>节[37]。下面来分别解释：

（1）<presence/>

<presence/>节用于表示出席信息，终端客户可以通过该节来发送自己的出席信息情况，发送实体将自身的出席情况通过该节广播出去，服务器将会转发该出席信息到订阅他的各个实体。一般分两种情况，一种情况是客户端发送实体到具体的订阅者，这时信息中应该带有“to”属性，这样系统就会通知指定的接收实体该客户端的出席情况，如果不带“to”属性，那么服务器将会广播到所有订阅者[38]。

（2）<message/>

<message/>节是实体之间传递消息的主要部分。主要是用来包含通信双方所要传送的消息，同时该属性中会包含通信双方的地址信息以及消息类型。

如下面的代码：

```
<message to=“remote@xmpp.com” from=“test@xmpp.corn”
    type=“chat” xml:lang=“cn” >
    <body>Hi </body>
</message>
```

上面的例子清晰地告诉了一个基本的<message/>节的构成，其中有属性值 to、from、xml：lang 以及 type。这些属性的意思分别是目的地址、源地址，所使用的语言以及消息节的类型。其中<body/>元素节的值就是此次发送信息的内容。

（3）<iq/>

<iq/>节表示请求和回复，是一种实体间的请求应答机制，其中的数据定义在该节的子元素中，例如要获取一个联系人名册，那么就要发出请求，那么就可以通过以下示例代码来获得：

```
<iq from=“test@xmpp.com/home” type=“get” id=“roster-1” >
    <query xmlns=“jabber:iq:roster” />
<iq/>
```

3.5.4 XMPP 建立通信链路

根据 XMPP 协议标准，任何符合 XMPP 规范的实体之间通信包括客户端与服务器、服务器与服务器之间都是通过 TCP 协议来建立连接的[40]，在规范中服务器端得访问端口建议使用 XMPP 在 IANA 注册过的 5222 端口，但并不是强制的。

为了保证信息的安全传输需要有安全的保障，在实体间进行通信时都要经过实体间的协商并验证通信实体之间的合法性。为了增强安全性，实体间通信一般都是使用了 TLS 握手机制，服务器端之间的通信协商时需要使用 TLS 的，而客户端与服务器的协商并没有强制使用 TLS。一般在建立 TLS 连接后，就开始了验证的流程。XMPP 协议中是使用的 SASL 来验证的[39]，之后实体间通信就可以发送需要发送的 XML 数据信息了

让来看一下 XMPP 中，两个实体之间通过服务器传递 XML 流的过程。其中 Sx 表示服务器，Cx 表示客户端。

```
C 1：
<?xml version=“1.0”?>
<stream:stream to=“test.com”xmlns=“jabber:client”
  xmlns:stream=“http://etherx.jabber.org/streams”version=“1.0”>
S1：
<?xml version=“1.0”?>
<stream:stream from=“xmpp.com”id=“id1”
  xmlns=“jabber:client”xmlns:stream=“http://etherx.Jabber.org/streams”
  version=“1.0”>
```

过程 C1 和 S1：首先 C 端发送<stream/>，表示打开流标记，同样 S 端会返回一个<stream/>流标记，表明了 C 与 S 开始建立连接。然后 C 端与 S 端分别进行了协商建立了 TLS 连接与 SASL 验证身份，这和服务器端建立连接的过程类似。

```
C2：
<presence xml:lan92“cn”>
  <show>chat</show>
</presence>
C3：
<iq from=“test@xmpp.com/home” type=“get”id=“roster-1”>
```

```
    <query xmlns2“jabber:iq:roster”/>
  </iq>
  S2：
  <iq to=“test@xmpp.corn/home”type=“result”id=“roster-1”>
    <query xmlns=“jabber:iq:roster”>
    <item j id=“remote@test.com”name=“Remote”subscription=“both”>
    <gvroup>F</group>
    </item>
    </query>
  </iq>
  C4：
  <message from=“test@xmpp.com/home” to=“remote@test.corn”xml:lang=“ca”
>
    <body>Hi</body>
  </message>
  S3：
  <message from=“remote@test.com” to=“test@xmpp.corn/home”
       xml:lang=“cn”>
    <body>Hi，test</body>
  </message>
  C5：
  </stream:stream>
  S4：
  </stream:stream>
```

过程 C2：在 C 的身份经确认之后，C 会发送了一个自身的出席信息给服务端 S，服务端将会发送 C 的出席信息给 C 订阅者。过程 C3： C 向服务端 S 请求 C 的名册列表，在过程 S2 中 S 端返回 C 的名册列表。过程 C4：C 开始给其它终端“remote@test.corn" 发送信息，S 端返回由“remote@test.corn”返回的消息。最后通信结束 C 端与 S 端分别发送一个<stream/>关闭标签，关闭会话，整个交互过程结束。

总结上面的例子，在一个会话中，会话的开始时通信双方首先必须发送一个流打开标记，然后是双方协商是否使用 TLS，协商后，开始利用 SASL 验证身份，之后，实体就可以给服务器发送自身的出席信息，这时，终端就可以和其它终端

进行通信了，然后终端会请求终端客户的联系人名册当然这不是必须的，在获得了联系人列表后，终端客户可以从用户列表中选择一个用户向其发送消息。在成功的发送一个消息后，那么表示一次会话成功。这时终端客户可以选择关闭会话，也可以选择继续发送消息。

3.5.5 Capsule 的传输机制

XMPP 具有良好的可扩展性和开放性，已经被广泛应用在即时通信领域中。通过 XMPP 协议及其扩展协议，可以实现 Capsule 应用压缩文件传输等功能，其中的文件传输需要使用额外的 SOCKS5 代理协议实现。

SOCKS 是一个代理协议[41]，是 1990 年开发的，此后成为 RFC 中的开放标准， 其协议描述在 RFC1928 文档中。SOCKS 协议分为 V4 和 V5 两个标准。V5 在 V4 基础上增加了 UDP 代理、对客户端的授权和认证[42][48]。V4 协议只支持 TCP 代理，允许客户端通过 SOCKS 代理完全地连接到外部的服务器，相比较 V5 更加灵活[43]。SOCKS5 在使用 TCP/IP 协议通讯的前端机器和服务器机器之间扮演一个中介角色，使得内部网中的前端机器变得能够访问 Internet 网中的服务器，或者使通讯更加安全。SOCKS5 服务器通过将前端发来的请求转发给真正的目标服务器， 模拟了一个前端的行为。在这里，前端和 SOCKS5 之间也是通过 TCP/IP 协议进行通讯，前端将原本要发送给真正服务器的请求发送给 SOCKS5 服务器，然后 SOCKS5 服务器将请求转发给真正的服务器。

SOCKS5 工作的流程就是首先客户端向服务器发出连接请求及相关的信息，服务器接收到消息后，检查安全配置，然后返回给服务器，客户端再对服务器所作选择进行验证，之后客户端向服务器发送协商请求信息以及相关的接口信息，完成后服务器将会处理客户端的请求并设置代理，从而建立与应用服务器的连接，同时向客户端返回协商请求结果以及一些必要信息[44]。至此，客户端已经可以将数据发送到服务器，服务器代理将进行数据传递操作。

当实体之间需要传输文件时，发送方首先会找到接收方的地址，然后发送请求到 XMPP 服务器，服务器会查找接收方的网络地址并告知接收方有文件需要接收，这时接收方消息尝试与 SOCKS5 服务器建立连接，建立成功后就会通过 XMPP 服务器发送反馈消息给发送方。发送方接收到反馈的消息后也会与 SOCKS5 服务器建立连接，这样，当 SOCKS5 服务器与发送方和接收方都建立连接后就会建立一个数据流传输通道，然后实体间的文件传输才真正开始。如图 3.8 所示：

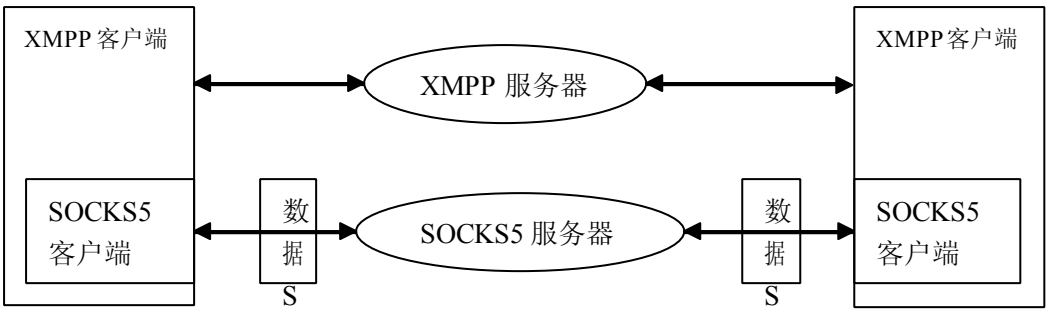


图 3.8 Socks5 文件传输结构图

SOCKS 传输协议是 XMPP 协议的扩展协议，这个协议的标准定义在 XEP-0096、XEP-0065 等协议中。SOCKS5 服务器建立连接的过程也就是文件的传输过程，根据 XEP-0096 协议定义的两个实体间文件传输的标准，可以获得要传送的文件的基本消息。XMPP 用户间也可以建立 P2P 的字节流的传输，也可以通过第三方来转发该字节流，这个标准定义在 XEP-0065 中。

实体间文件传输的过程大致经历下面三个阶段：

- 请求与响应
- 协商传输方法
- 建立 SOCKS5 字节流

在实体间需要协商传输方法时，发送方会提供几个可选的传输方法给接收方，接收方选择后悔通知发送方，如果发送方接受该方法，那么 SOCKS5 服务器将会建立文件传输的数据连接。如果协商不成功，会继续以上过程，直到成功或者出错。

建立 SOCKS5 字节流的阶段是真正的文件传输的核心部分，收发双方根据协商的传输方式，服务器给双方建立传输文件的字节流。这时，服务器会创建一个 SOCKS5 的连接实例，收发双方都会连接到给实例上，发送方发送文件数据到服务器的该实例上，服务器将该数据根据与接收方的连接将数据流传递给接收方。

3.6 终端分享引擎ShareEngine模型

3.6.1 ShareEngine 概述及功能简介

该分享引擎中所分享的构件化软件具有一定的时效性和周期性，可以实现短时期内的应用的发布和使用，每个构件化软件都会附有一个 expire-time 即失效时间，当失效时间到达后该构件化软件将会自动失效，无法运行。如果想要继续使

用，可以进行再次分享的请求。

ShareEngine 分享系统实现了以下功能模块，如图 3.9 所示

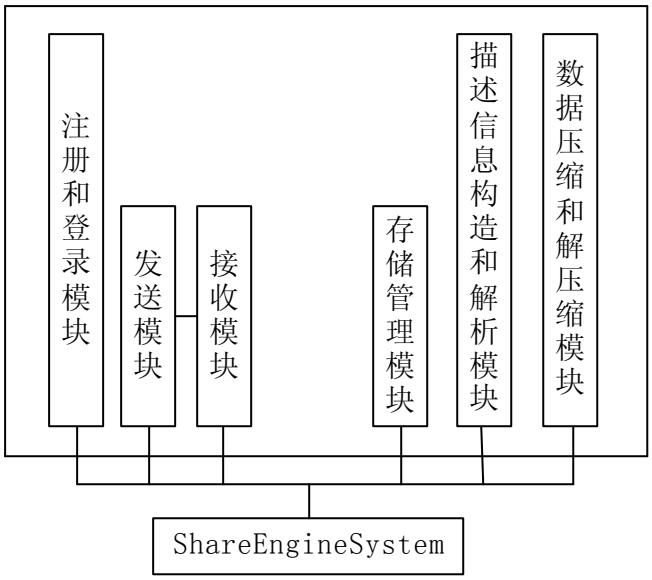


图 3.9 ShareEngine 功能模块

终端的分享功能是指客户之间使用 ShareEngine 来通过服务器过滤、记录和连接功能来进行传输分享构件化软件，客户可在客户端实现对分享内容的接收、存储、查看、管理、转发或者发送。

整体功能介绍 ：

（1）可在手机进行构件化软件的接收、存储、查看、管理、转发或者发送；

（2）服务器将实时推送分享信息到手机，具有流量小、速度快的特点，接收方可以选择是否接收该分享；

（3）与手机电话簿关联，地址存取方便；

（4）支持特定时间段的静音功能；

（5）支持与情景模式同步以及调用系统音乐；

（6）可自定义签名，密码保护；

（7）可对构件化软件设置密码保护，设置密码后，接收端接收到该分享后需要输入密码才能使用；

（8）暂停构件化软件接收；

（9）在 ShareEngine 中可设置对暂停/恢复构件化软件的接收，暂停接收后仍可通过 ShareEngine 发送分享构件化软件。

（10）支持构件化软件的有效期管理，当设置的构件化软件的有效期过后，该构件化软件自动失效，将无法使用。

3.6.2 ShareEngine 的原理

智能终端的 ShareEngine 的设计主要基于以下 3 点：

（1）服务器，用来负责记录和控制智能移动终端之间分享的构件化软件，并且记录该构件化软件的所有信息以及在不同终端之间的移动轨迹。

（2）移动通信服务商，如中国移动、中国电信、中国联通，掌握了很多移动终端的联系人的信息，并能够随时的给该号码发送一些数据：短信，彩信，电话接入请求等等。

（3）移动智能终端，具有 ShareEngine 以及该引擎的客户端应用，并且根据三层架构原则，可以利用本地其它引擎功能了解当前手机的状态，比如是否有信号、是否已联网等。

基于以上三点，在 Elastos 中间件上智能手机实现 ShareEngine 完全可行。移动通信服务商成为中介是最好的选择。

3.6.3 Capsule 的分享过程

终端 A 需要分享 Capsule 给 B 的抽象时序图如图 3.10 所示。从 A 和 B 的状态来看，主要有两种情况：

- B 在线，服务器检测要发送的基本信息并记录下来，之后通知 B 是否同意接收，若同意接收，那么服务器和 B 建立连接，进行传送该分享，接收完毕后，服务器返回给 A 一个消息：B 已经接收完毕；若不同意，则服务器返回给 A 一个消息：B 不同意接收该分享。
- B 不在线，服务器检测要发送的基本信息并记录下来，之后通知 A：B 不在线，是否仍要发送，若 A 仍要发送，那么该分享将会被服务器缓存起来，放进一个发送队列中，当 B 在线时，通知 B 是否接收，若同意，则建立与 B 的连接，接收完毕后，服务器返回给 A 一个消息，告诉 A：B 已经接收完毕；若不同意，则返回给 A 一个消息，告诉 A：B 已经拒绝了该分享的接收。

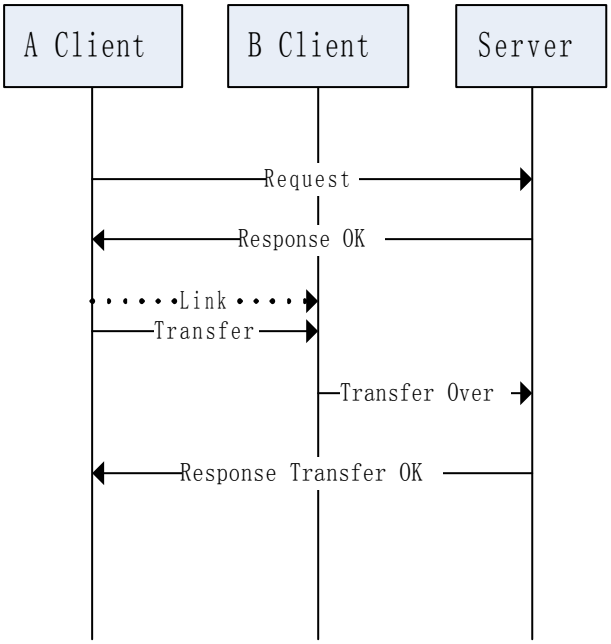


图 3.10 终端之间 capsule 传输抽象过程

分享 Capsule，其执行时序为:

（1）A 向服务器发送请求；

（2）服务器响应，查看终端 B 是否处于在线状态；

（3）服务器通知终端 B 处于可以分享状态；

（4）A 向服务器发送需要向 B 发送的内容以及该内容的详细信息（这些详细信息由引擎获取）；

（5）服务器检查内容的合法性；

（6）服务器告诉 B：A 想分享给他一个 Capsule，询问其是否想要接收；

（7）若 B 同意接收，那么 B 将通过 ShareEngine 的接收模块将该 Capsule 接收到自己的终端上，然后通过解析模块来对 Capsule 的信息进行解析，根据不同的类型，通过数据管理器将其存储起来；

（8）接收完毕后 B 将向服务器发送一个 Response，这样，服务器就知道了 B 已经接收到 Capsule；

（9）服务器返回给 A 一个 Response，告诉 A：B 已经接收到了 A 给 B 分享的 Capsule。

这样一个完整的分享过程完成。

第4章 终端ShareEngine的引擎接口设计和实现

4.1 ShareEngine各个功能模块设计

终端分享引擎ShareEngine包含很多功能，有注册与登录模块、发送模块、接收模块、存储管理模块、解析模块、数据压缩和解压缩模块，下面来分别详细介绍这些功能模块。

■ 注册与登录模块：

终端应具备一次登陆就能与网络上的服务器连接并处于使用就绪状态，各功能模块使用的UI界面风格统一，且功能之间的切换不需要重新登陆。

同样，终端应具备一次退出就可与网络上的服务器连接断开，但不影响某些离线功能的使用。

启动终端应用时应该判断用户标识（比如手机号码）是否已经注册，若已经注册，则不做动作。若没有注册则在网络连接的状态下进行注册。在用户需要发送分享或者接收分享时，该分享引擎都会判断是有处于网络状态，是否已经注册，发送方发送分享后，云端服务器将会给接收方发送一个push消息，判断接收方是否愿意接收该分享，如果同意，接收方分享引擎中的接收模块就会判断是否已经登录，若登录则不做动作等待接收该分享，若还未登录则启动登录程序进行登录，并建立接收该分享的与云端服务器的连接。

终端开机时应该具备终端自动登陆功能，也允许用户登陆后退出，然后再次手动登陆。自动登陆时，如果登陆失败，自动重试若干次，若仍然失败则放弃登陆，等待用户手动登陆。退出终端应用时，立即断开与网络服务器的连接。

客户端与服务器建立连接的顺序:

客户端发送注册请求；服务器接到请求将客户端的信息注册完成后得到一个随机授权码，然后将该授权码返回给客户端，该授权码是每隔一段时间就会更新的，每次更新后，服务器都会将新的授权码发送给客户端引擎。这样客户端在与其它客户端进行分享时只需附上该授权码就可以得到服务器的身份认证。客户端接收到授权码保存起来。

■ 发送模块：

负责选择所要发送的构件或者软件和需要分享的接收者，并调用发送协议发送出去。

可以从数据管理模块中获取自己已经有的所有构件信息和自己手机上的所

有软件信息。可以通过选择联系人来确定接收者。该模块所具有的功能如下：

（1）发送之前需要调用数据压缩模块来将需要的分享数据进行打包压缩。

（2）从数据管理模块选择需要发送的构件或者应用

（3）从终端本地获得需要分享的接收联系人

（4）发送分享请求

（5）获取分享请求的反馈

（6）根据反馈发送分享

■ 接收模块：

接收到服务器的消息并确认是否同意接收该分享，如果不同意接收，将会给服务器一个拒绝的response；如果同意接收，那么服务器将会给客户端建立连接，从而将分享接收下来。接收下来后交由解析模块来进一步操作。

■ 存储管理模块：

数据管理模块负责管理发送和接收到得构件的信息，以及负责该构件的禁用或者删除，主要是涉及到分享引擎的所涉及到得数据信息的数据库管理操作，当然也包括与终端本地的构件管理器互操作，以得到该终端上的构件或者软件数据。

■ 解析模块：

解析模块对接收到的分享进行解析，通过调用数据压缩模块，获取其中的xml信息列表，并将这些基本信息通过调用数据管理模块将这些数据添加到数据库中。然后根据xml提供的关于该分享的一些信息包括依赖关系以及安装策略来对该分享的构件进行安装。

（1）对接收到的构件或者软件进行解压并对其中附带的xml表信息进行解析。

（2）解析完成后通过数据管理模块将这些数据添加进数据库。

（3）将解压出来的文件按照xml中的描述信息将这些构件放入到不同的目录中。

■ 数据压缩和解压缩模块：

包括以下几部分：调用系统API获取将要发送的构件或者应用的详细信息，

并将这些信息记录在一张xml表中，同时检查根据构件或者应用对其他构件或应用的依赖性，并将那些构件的位置或者信息也写入xml表中。最后将分享的构件或者应用以及xml信息表和其它依赖性的构件或者软件同时打包压缩。打包时，需要判断依赖性的构件或者应用程序是否包含私有数据，若有私有数据将通知用户是否一起发送。

（1）构件的压缩和相关软件依赖性解析及其自身信息的解析

（2）应用软件的压缩和相关信息的解析

4.2 终端ShareEngine的接口设计与实现

引擎层ShareEngine实现了对各个功能模块的封装，通过暴露接口给应用层来间接为应用开发提供服务。与上节ShareEngine引擎的功能模块相对应，ShareEngine引擎所包含的功能模块可以大致分为五类。分别是发送和接收模块、数据压缩和解压缩模块，描述信息构造和解析模块以及存储管理模块。

4.2.1 注册和登录模块

（1）注册部分：下面着重分析的注册接口RegistrationHandler。

RegistrationHandler 接口：接收来自Registration实体的事件信息，继承该接口的类可以被Registration实体注册一个RegistrationHandlers子类，Registration实体触发的行为所返回的结果可以转发到这个字类。

这个类里面定义了一些消息类型：

```
enum RegistrationResult{
        RegistrationSuccess = 0,
        RegistrationNotAcceptable,
        RegistrationConflict,
        RegistrationNotAuthorized,
        RegistrationBadRequest,
        RegistrationForbidden,
        RegistrationRequired,
        RegistrationUnexpectedRequest,
        RegistrationNotAllowed,
        RegistrationUnknownError
};
```

RegistrationHandler定义的接口如下：

```
interface RegistrationHandler {
        handleRegistrationFields (
            [in] String from,
            [in] Int32 fields
            [in] String instructions);
```

```
            handleAlreadyRegistered ([in]    String from);
        handleRegistrationResult (
            [in]    String from
            [out] RegistrationResult regResult);
        handleDataForm (
            [in]    String from
            [out] DataForm form);
        handleOOB (
            [in]    String from
            [out] OOB oob);
  }
```

其中handleRegistrationFields用于处理注册字段

handleAlreadyRegistered是判断该终端是否已经注册。

handleRegistrationResult是注册完成后返回的结果

handleDataForm是终端到服务器注册时所附带的注册数据信息

handleOOB如果服务器不提供注册但是向用户指向一个外部的url

为了实现注册，实现了一个Registration类，该类实现如下：

```
class Registration : public IqHandler
{
    public:
        class Query : public StanzaExtension
        {
        public:
        Query( DataForm* form );
         …..
        const DataForm* form() const { return m_form; }
        const std::string& instructions() const { return m_instructions; }
        int fields() const { return m_fields; }
        const RegistrationFields& values() const { return m_values; }
        bool registered() const { return m_reg; }
        bool remove() const { return m_del; }
        const OOB* oob() const { return m_oob; }
        // reimplemented from StanzaExtension
        virtual const std::string& filterString() const;
```

```
        // reimplemented from StanzaExtension
        virtual StanzaExtension* newInstance( const Tag* tag ) const
        {
          return new Query( tag );
        }
      };
      Registration( ClientBase* parent, const JID& to );
      Registration( ClientBase* parent );
      void fetchRegistrationFields();
      bool createAccount( int fields, const RegistrationFields& values );
      void createAccount( DataForm* form );
      void removeAccount();
      void changePassword(
                        const std::string& username,
                        const std::string& password );
      void registerRegistrationHandler( RegistrationHandler* rh );
      void removeRegistrationHandler();
      virtual bool handleIq( const IQ& iq ) { (void)iq; return false; }
      virtual void handleIqID( const IQ& iq, int context );
      private:
#ifdef REGISTRATION_TEST
    public:
#endif
      enum IdType
      {
          FetchRegistrationFields,
          CreateAccount,
          RemoveAccount,
          ChangePassword
      };

       ClientBase* m_parent;
        RegistrationHandler* m_registrationHandler;
  };
}
```

（2）登录部分：每次终端启动时登录模块都会根据配置来判断该终端是否已经登录，若没有登录将启动登录程序，当然，用户可以在配置管理中设置是否允许开机登录。其登录过程如图4.1所示：

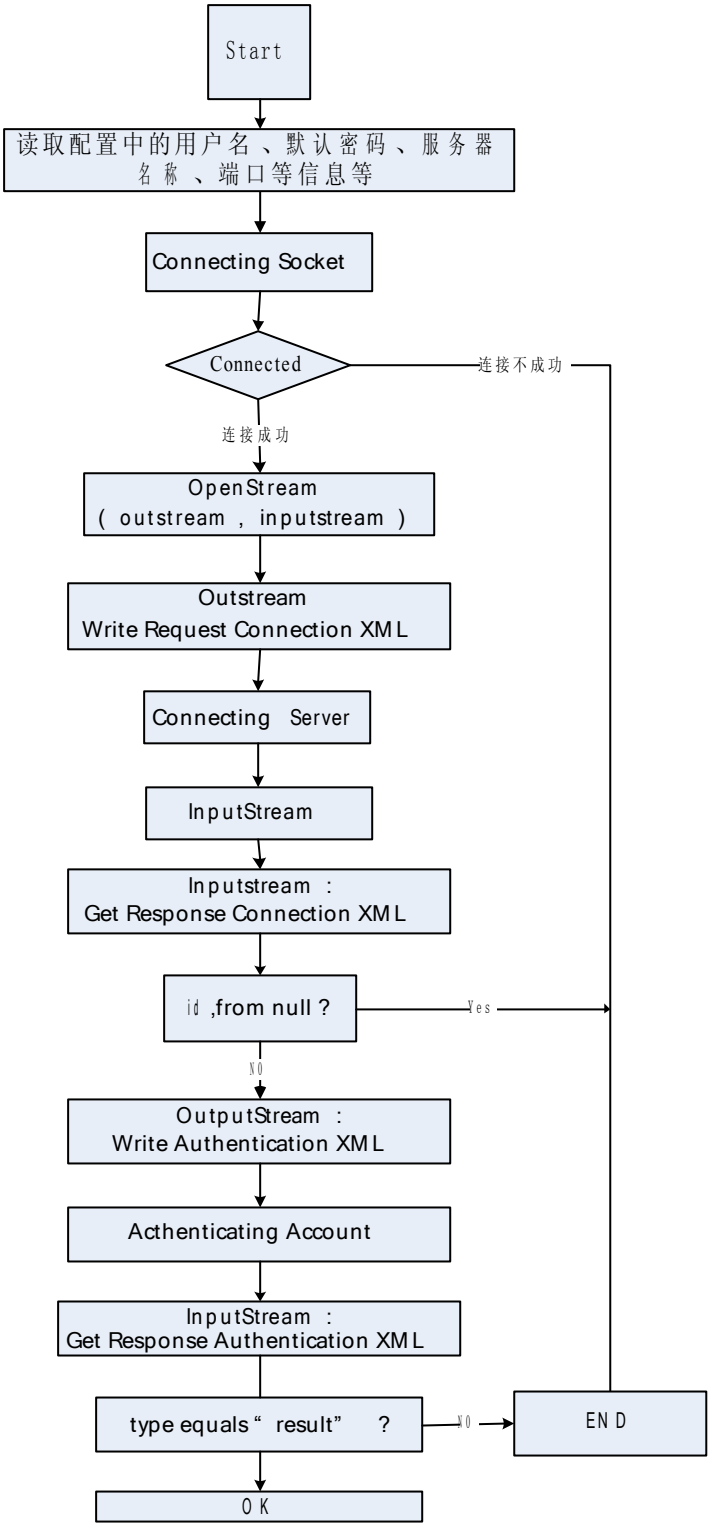


图 4.1 分享终端登录过程

4.2.2 发送和接收模块

发送和接收模块负责编辑需要发送的分享信息以及接收分享内容以及服务器广播的消息。

（1）发送部分负责对不同类型的构件化软件进行附带基本的信息并执行发送操作，这个过程是Capsule装载的过程，该软件被装载进一个Capsule中，然后附带的描述信息将会被加入到Capsule.xml中，其各接口之间的关系如图4.2所示。

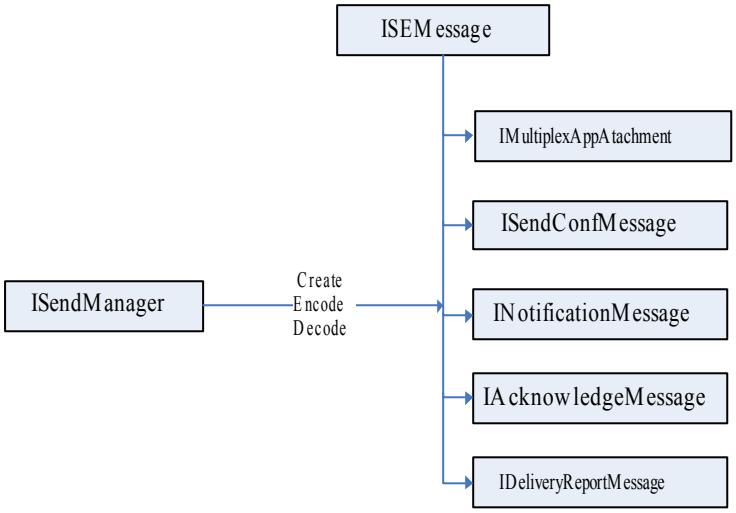


图 4.2 ShareEngine 构件化软件编辑管理模块引擎接口

上图中的不同接口对应着不同的功能，其中接口IMultiplexAppAtachment是用来选择需要分享的内容，当做一个主体附件，同时调用数据压缩的功能将该分享内容的描述信息附在被分享的软件上进行打包；接口ISendConfMessage对应的功能是发送方发出分享请求并附带分享描述信息，发送服务器在收到发送请求，并解析分享描述信息，同时将这些消息存储到服务器的分享记录中，当确认发送分享信息合法后，发给发送方一个确认消息；接口INotificationMessage对应的功能是服务器发送给接收方的分享请求信息，询问接收方是否接收该分享请求；接口IAcknowledgeMessage对应的功能是接收方在接收到分享请求时发送给服务器的确认消息；接口IDeliveryReportMessage对应的功能是当接收方接收分享完毕时返回给发送方一个分享成功的消息。

所有这些接口均继承自接口ISEMessage，该接口提供一些通有的方法，例如获取该构件化软件的版本号、类型等。另外，接口ISendManager是被分享的软件的编辑引擎的管理接口，它负责创建针对不同功能的接口，并负责对将要被分享的软件进行编码和解码。

这里着重分析一下发送分享内容的管理接口IMultiplexAppAtachment，因为将使用此接口来实现对分享内容的发送。。IMultiplexAppAtachment接口的定义如下：

```
interface IMultiplexAppAtachment{
        GetMultiplexAppCount( [out] Int32* count);

         GetMultiplexAppInformation(
              [in]    Int32 index,
              [out] StringBuf_<MAX_MMS_PATH> name,
              [out] Int32 *dataLength);

        RetrieveMultiplexApp (
              [in]    Int32 index,
              [out] StringBuf_<MAX_MMS_CONTENT_TYPE> contentType,
              [out] BufferOf<Byte>** data);

        AddMultiplexApp (
              [in] String path,
              [in] String contentType);

        AddMultiplexAppContent(
              [in] String contentType,
              [in] BufferOf<Byte> content);

        DeleteMultiplexApp ( [in] Int32 index);

        ClearAllMultiplexApp ();
 }
```

IMultiplexAppAtachment是编辑分享内容的内容附件接口，提供了添加多个构件化软件附件的方法：AddMultiplexApp ()、AddMultiplexAppContent ()，删除附件的方法：DeleteMultiplexApp ()、ClearAllMultiplexApp ()，得到附件个数的方法：GetMultiplexAppCount ()，获取某一附件信息的方法：GetMultiplexAppInformation ()，以及检索某一附件、获取其Content Type和数据的方法：RetrieveMultiplexApp ()。

（2）其中接收部分是负责接收分享的，包括目的服务器转发的分享消息以及分享内容。根据接收到的消息类型分两种操作方式来处理：一种是分享的通知消息，通知消息在分享内容传送前发给接收终端用户。一种是分享的传输连接消息，将由服务器发起，通知收发双方建立端到端的连接。

接收模块的核心实现代码如下。

```
virtual void handleFTBytestream( Bytestream* bs )
{
        m_bs.push_back( bs );
        bs->registerBytestreamDataHandler( this );
        if( bs->connect() )
        {
        ……
        }
}

virtual const std::string handleOOBRequestResult( const JID& /*from*/, const JID& /*to*/, const std::string& /*sid*/ )
{
        return std::string();
};

virtual void handleBytestreamData( Bytestream* /*s5b*/, const std::string& data )
{
        ……
}

virtual void handleBytestreamError( Bytestream* /*s5b*/, const IQ& /*stanza*/ )
{
      ……
}

virtual void handleBytestreamOpen( Bytestream* /*s5b*/ )
{
      ……
}
```

```
virtual void handleBytestreamClose( Bytestream* /*s5b*/ )
{
        printf( "stream closed\n" );
        m_quit= true;
}
```

4.2.3 存储管理模块

Elastos平台使用的是SQLite数据库。SQLite是一款轻型的数据库，它能够支持Windows/Linux/Unix等等主流的操作系统，同时能够跟很多程序语言相结合。处理速度也比较快。

存储管理模块中的信息存储都使用数据库，由数据库来保证数据库一致性。该模块负责信息数据库的管理以及配置管理，包括分享信息内容的保存和提取等但对于每个分享消息，在数据库中的键值如图4.3：

分享ID	分享Subject	分享Date Time	分享人	分享内容Size	分享源数据	分享内容有效期

图 4.3 数据库字段

为了实现该存储模块对于分享内容的存取和管理，定义了存储模块的各个接口，其各接口之间的关系如图4.4所示。

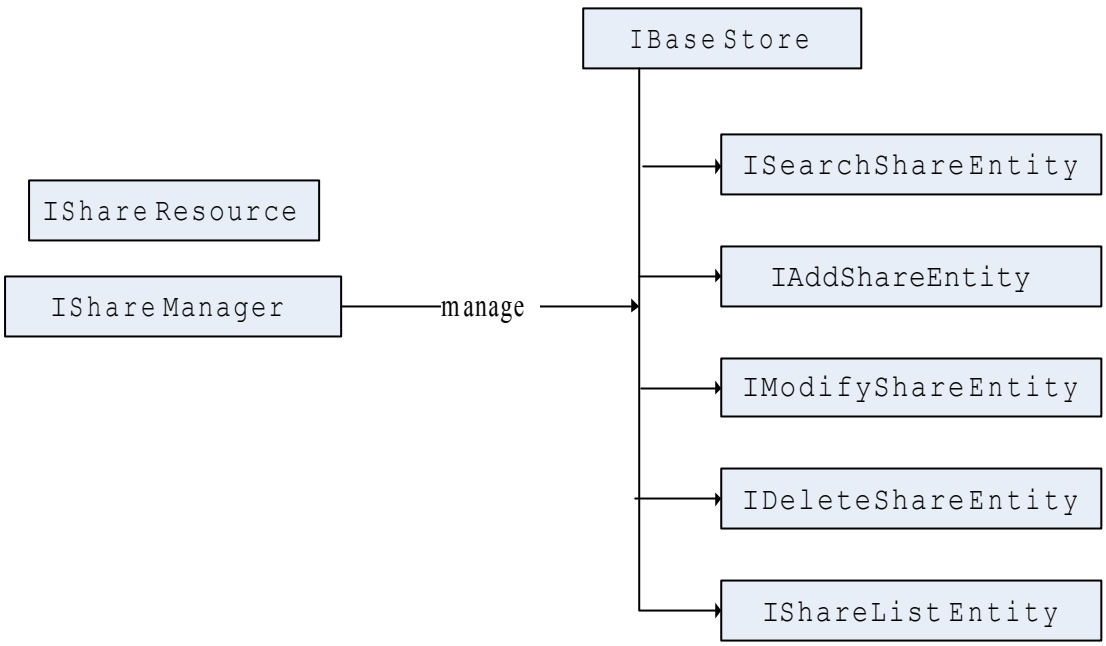


图 4.4 存储管理模块引擎接口

在存储管理方面，这些接口的方法实际对应了对信息数据库中对应分享信息的操作，这些接口均继承自IBaseStore接口，IBaseStore提供了一些通用的存储操作。IShareResource是资源管理接口，负责管理常用构件化软件等资源。IShareManager是信息存储管理接口，包括了对数据库的增、删、改、查等基本操作，并创建针对不同类型被分享的构件化软件信息的IXxxEntity接口。

以下代码列出了不同类型软件信息的枚举定义，以及IBaseStore的一些实现。可以看到IBaseStore将会定义SetType()和GetType()来进行信息类型邦定，这样就可以通过IBaseStore接口实现对对应类型软件分享信息的操作。

```
enum MultiplexAppShareType {
        AppShareType_Component,
        AppShareType_App,
        AppShareType _All,
        AppShareType _Text,
        AppShareType_Share_Report,
        AppShareType_ Share_ Notify,
        AppShareType_Invalid = 0xff
}

interface IBaseStore{
        ……
        SetType( [in] AppShareType msgType);
        GetType( [out] AppShareType * msgType);
        ……
}
```

4.2.4 描述信息构造和解析模块

描述信息构造和解析模块在终端用户在发送分享前和接收到分享时将会被调用。该描述信息具有很重要的意义，因为对于将要分享的构件化软件，系统需要根据每一个软件所对应的描述信息来对其进行管理，包括对其的存储，删除，转发和应用等。

其中描述信息生成功能是在发送分享之前用来对所要分享的软件信息进行收集和整理，变成一个XML信息描述文件，通过的Capsule技术，会将收集的XML描述信息文件与需要被分享的软件一起打包进的Capsule分享实体中去，然后通过分享发送模块将该Capsule分享出去。该描述文件的格式如下：

```
<author >Author</author >
<name>TestApp</name>
<size>1024KB<size>
<path>/system/path</path>
<expiretime>3</ expiretime >
 <requirement >….</requirement >
…..
```

该描述信息的获取是从系统构件库或者应用程序管理器的数据库中获得，在发送分享之前，需要将这些信息存储到本地已发送分享列表中。在接收模块中，当接收到分享后分享引擎部分将调用解压缩模块对其进行解压并调用解析程序对该分享内容的XML描述文件进行解析具体的针对该文件的解析将在控制逻辑中给应用程序提供，这里要在ShareEngine中实现对其解析的基本函数，其引擎中的解析程序如下：

```
Parser::DecodeState Parser::decode( std::string::size_type& pos, const std::string& data )
{
 ……
    if( diff < 3 || diff > 9 )
       return DecodeInvalid;
    switch( data[pos + 1] )
    {
       case '#':
       {
            char* end;
            const long int val = std::strtol( data.data() + pos + idx, &end, base );
            if( *end != ';' || val < 0 )
               return DecodeInvalid;
            …….
        }
        break;
        ……..
        default:
          return DecodeInvalid;
    }
    …..
```

```
        return DecodeValid;
    }

    Parser::ForwardScanState Parser::forwardScan( std::string::size_type& pos,
const std::string& data, const std::string& needle )
    {
        if( pos + needle.length() <= data.length() )
        {
            if( !data.compare( pos, needle.length(), needle ) )
            {
                pos += needle.length() - 1;
                return ForwardFound;
            }
            …….
        }
      …..
    }

    int Parser::feed( std::string& data )
    {
        if( !m_backBuffer.empty() )
        {
            data.insert( 0, m_backBuffer );
            m_backBuffer = EmptyString;
        }
        std::string::size_type count = data.length();
        for( std::string::size_type i = 0; i < count; ++i )
        {
            const unsigned char c = data[i];
            if( !isValid( c ) )
            {
                cleanup();
                return static_cast<int>( i );
            }
        …..
```

```
        return -1;
    }
```

4.2.5 数据压缩和解压缩模块

数据压缩和解压缩模块功能包含两个方面：一方面在移动终端发送分享时，负责对将要被分享的构件化软件进行压缩，并附带对该软件的详细信息的描述XML文件以及需要通知对方的一些提示信息等，该软件是否适用于该终端可以通过终端系统中的应用管理服务来判断。另一方面是移动终端在接收分享时，负责对发送的压缩包进行解压，将其解压到一个临时目录中，并调用数据解析模块进行解析，解析完毕后，将调用存储管理模块将这些分享的描述信息存储到已接收数分享的据库中，从而可以在该功能下通过应用程序来对其进行管理。

其接口定义如图4.5所示：

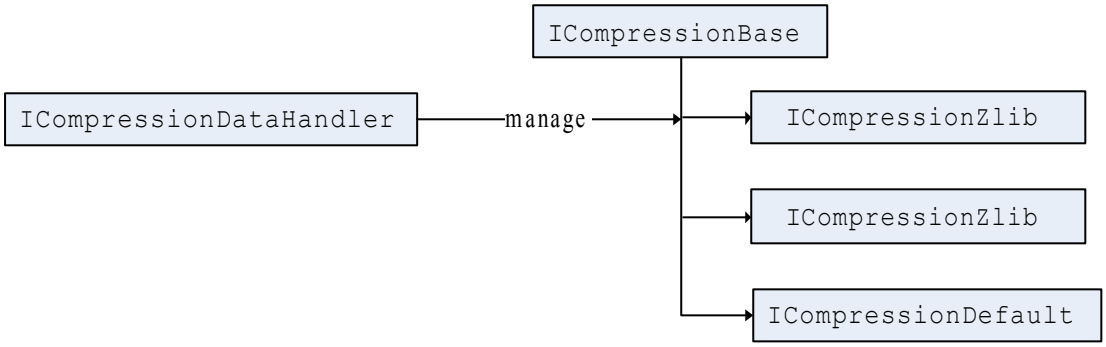


图 4.5　数据压缩和解压缩功能接口关系图

接口ICompressionDataHandler是用来接收数据压缩和解压缩的一个Handler，通过它可以调用数据压缩和解压缩的功能实现数据的压缩和解压缩。该接口不会被应用层直接调用，通过引擎控制层来对其封装后再交由应用层调用。ICompressionBase接口是数据压缩和解压缩的基类接口，该接口定义了数据压缩和解压缩的初始化并提供数据压缩和解压缩两个函数来供上层调用。ICompressionZlib接口是用来实现对数据进行压缩和解压缩成Zlib格式。ICompression LZW接口是用来实现对数据进行压缩和解压缩成LZW格式。ICompressionDefault接口是提供给上层一个默认的压缩和解压缩功能，有两种格式的数据压缩和解压缩，一种是Zlib格式，另一种是LZW格式。

下面介绍ICompressionDefault接口实现的数据默认的压缩和解压缩方法，CompressionDefault的默认构造函数需要对两种数据的处理方式进行判断。

```
CompressionDefault::CompressionDefault( CompressionDataHandler* cdh,
Method method )
```

```
  : CompressionBase( cdh ), m_impl( 0 ){
      switch( method )
      {
        case MethodZlib:
#ifdef HAVE_ZLIB
          m_impl = new CompressionZlib( cdh );
#endif
          break;
        case MethodLZW:
#ifdef HAVE_LZW
          m_impl = new CompressionLZW( cdh );
#endif
          break;
        default:
          break;
      }
    }
```

其具体压缩和解压缩方法调用了两种压缩和解压缩格式的具体实现函数，如下：

```
void CompressionDefault::compress( const std::string& data )
{
    if( m_impl )
      m_impl->compress( data );
}

void CompressionDefault::decompress( const std::string& data )
{
    if( m_impl )
      m_impl->decompress( data );
}

void CompressionDefault::cleanup()
{
    if( m_impl )
```

```
        m_impl->cleanup();
}
```

这里只列出Zlib格式的数据压缩和解压缩具体实现方法，LZW格式的数据压缩和解压缩方法的具体实现类似。

```
void CompressionZlib::compress( const std::string& data )
{
    if( !m_valid )
        init();
    if( !m_valid || !m_handler || data.empty() )
        return;
    long unsigned int CHUNK = data.length() + ( data.length() / 100 ) + 13;
    Bytef* out = new Bytef[CHUNK];
    char* in = const_cast<char*>( data.c_str() );
    m_compressMutex.lock();
    m_zdeflate.avail_in = static_cast<uInt>( data.length() );
    m_zdeflate.next_in = (Bytef*)in;
    int ret;
    std::string result;
    do {
        m_zdeflate.avail_out = static_cast<uInt>( CHUNK );
        m_zdeflate.next_out = (Bytef*)out;
        ret = deflate( &m_zdeflate, Z_SYNC_FLUSH );
        result.append( (char*)out, CHUNK - m_zdeflate.avail_out );
    } while( m_zdeflate.avail_out == 0 );
    m_compressMutex.unlock();
    delete[] out;
    m_handler->handleCompressedData( result );
}
void CompressionZlib::decompress( const std::string& data )
{
    if( !m_valid )
        init();
    if( !m_valid || !m_handler || data.empty() )
        return;
```

```
    int CHUNK = 50;
    char* out = new char[CHUNK];
    char* in = const_cast<char*>( data.c_str() );
    m_zinflate.avail_in = static_cast<uInt>( data.length() );
    m_zinflate.next_in = (Bytef*)in;
    int ret = Z_OK;
    std::string result;
    do
    {
       m_zinflate.avail_out = CHUNK;
       m_zinflate.next_out = (Bytef*)out;
       ret = inflate( &m_zinflate, Z_SYNC_FLUSH );
       result.append( out, CHUNK - m_zinflate.avail_out );
    } while( m_zinflate.avail_out == 0 );
    …..
}
```

第5章 终端的应用实例实现

5.1 终端应用的功能设计

SCS 终端分享模型的常用功能如下：

（1）编辑功能。包括将要被分享的构件化软件的添加和删除、显示接收方号码、抄送号码进行分享等功能。

（2）消息的显示功能。包括分享软件的文件的标题及描述的显示，以及对接收方号码，抄送号码等的显示功能。

分享软件的展示是直接显示这些分享软件的文件名。对于需要显示分享软件的情况，可以进一步通过存储管理模块查看。

（3）发送和接收功能。该功能包括能够将编辑好的需要分享的软件发送出去，并能够接收和显示收到的分享软件消息及其内容。

（4）启用和停用分享软件的功能。该功能包括能够提取分享中的分享软件，并将其保存到接收到或者已发送的分享软件文件夹中，通过存储管理模块来决定是否启用该分享软件，若选择启用，那么 ShareEngine 将根据该分享软件的描述信息将其放入指定的目录下，从而可以使用，若选择停用，那么 ShareEngine 将从指定目录下删除该分享软件。

（5）存储管理功能。该功能包括能够对已发送分享文件列表、已接收分享文件列表进行管理，能够删除分享软件和新建软件分享。

（6）参数设置功能。该功能主要是用来对发送出去的分享软件设置有效时间，就是说发送分享者决定该分享的内容的有效期，该分享内容一旦过期，接收者将无法再使用该分享内容，另外还可以设置是否接收离线传输的文件以及关闭或者打开分享引擎等。

5.2 终端应用的工作流设计

本节将设计分享系统的几个主要功能的工作流，这些工作流在最根本上决定了模块的划分以及主要的控制逻辑。

（1）首先介绍分享系统编辑和发送构件化软件的主要工作流，如图 5.1 所示。

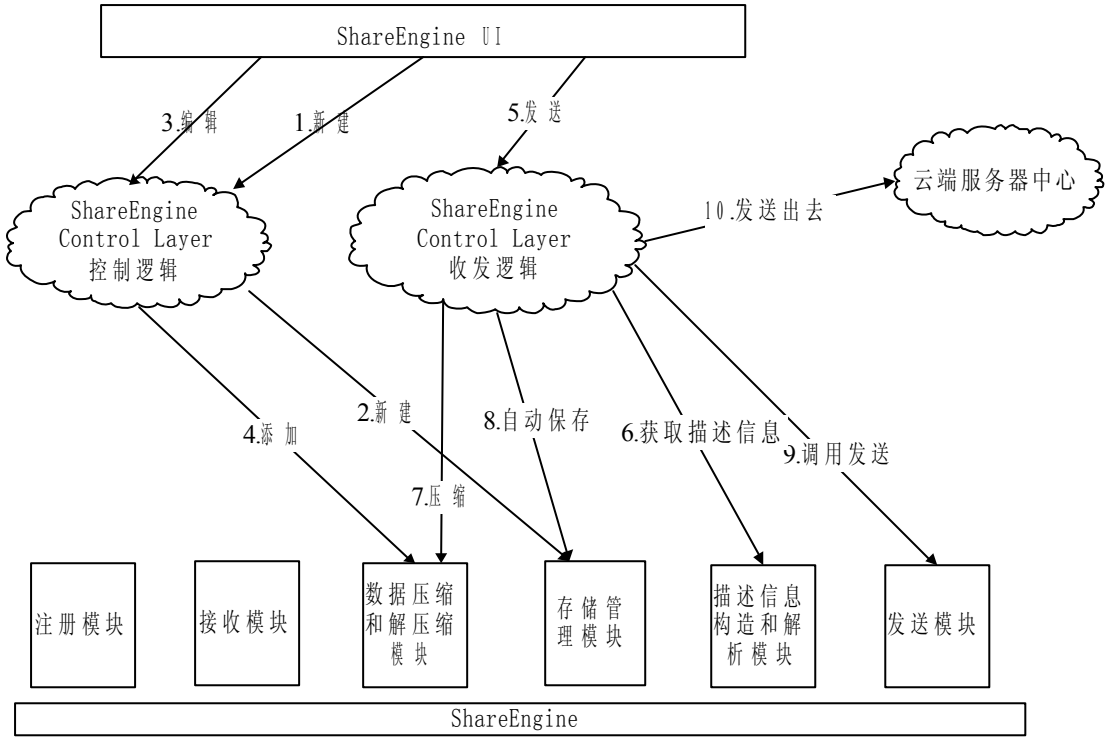


图 5.1 分享系统编辑发送软件的工作流

新建一个构件化软件分享信息时，首先将调用存储管理模块在临时性草稿存储模块新建一条分享信息，之后发起分享的用户对这条分享信息的编辑都将在这个临时性的草稿上进行。

编辑构件化软件分享信息时，用户在添加一个构件化软件时，会调用构件化软件 type 识别器来判断构件化软件的类型是什么，并将该类型记录在该构件化软件的描述信息里。发送模块会根据不同的格式来进行不同的处理操作。

发送构件化软件分享信息时，将会调用数据压缩功能对该分享信息进行压缩，并自动存储到已发送分享列表中，然后发送模块会被控制逻辑调用向服务器发送请求，并将该分享发送出去。

（2）分享模型接收构件化软件分享信息的工作流。在这个工作流中省去了对通知消息和确认消息的处理，重点描述了获取到构件化软件分享信息后的处理流程。

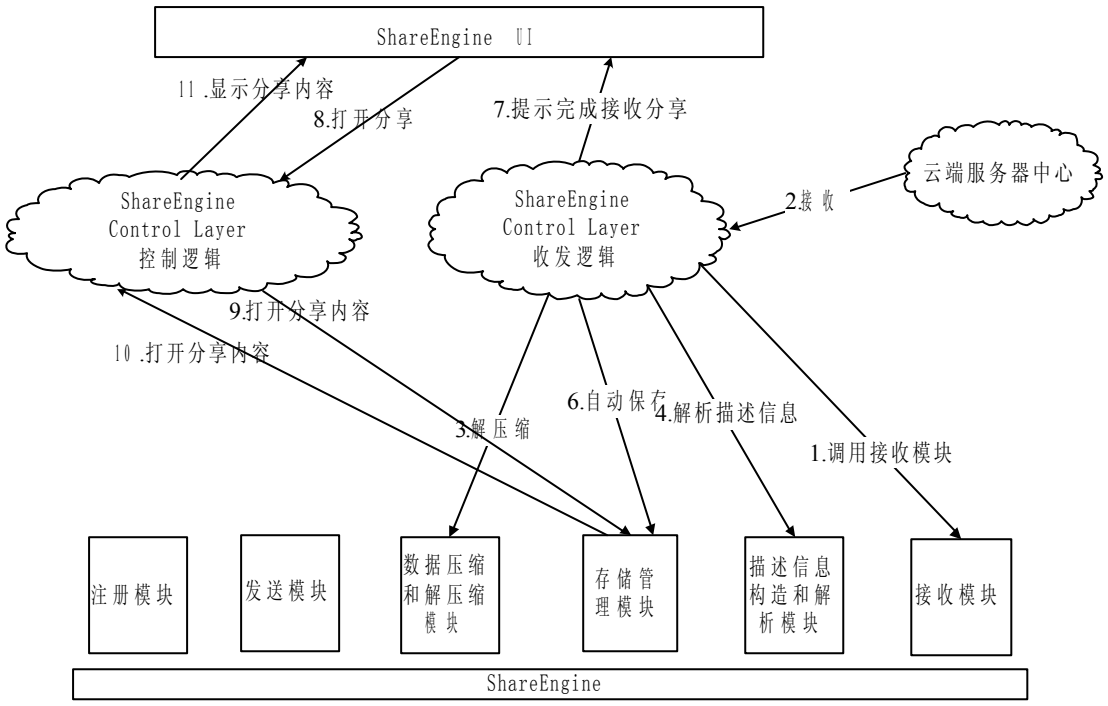


图 5.2 分享系统接收软件分享信息的工作流

在 ShareEngine 收发逻辑功能接收到一条分享信息时，将会自动对其进行解压缩，然后通过解析该分享信息所附带的描述信息来获取该分享信息的具体描述，然后，将其保存到已收到的分享信息列表中，随即将该分享内容告知用户。用户可以打开该分享内容进行查看，也可以对其进行处理，如对该构件化软件的启用和停用、编辑和删除等。

5.3 终端ShareEngine应用层UI实现

应用层 UI 界面直接面对用户，利用下层模块提供的功能支持，提供给终端用户一个简单友好的操作界面。UI 界面部分能够响应用户的控制操作行为，向下层模块传递请求，并从下层模块获取所请求的数据，同时，将这些数据以某种形式显示出来，以满足用户的需求。

终端应用的界面显示采用 XML 和脚本结合的方式编写，通过 Elastos 平台上的 XmlGlue 技术实现 XML 语言对逻辑控制层构件的调用。XmlGlue 能够将 XML 解释为相应的图形元素，并通过图形系统构件来将界面显示出来[45]。XML 文件主要实现三大功能：（1）通过 XML 布局来编写界面，即通过布局来描述各个标签控件，并设置它们的属性参数。例如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。页面布局在 XML 文件中的一对< form>和</ form>标签之间实现。（2）在 XmlGlue 中嵌入脚本语言用来处理用户与界面

视图之间的交互。在 script 标签中说明使用的具体的脚本语言，并导入功能逻辑的代码脚本：<script language="Lua" >……</script>。脚本主要实现应用的具体逻辑，负责与分享引擎 ShareEngine 的交互。（3）可以加载构件功能模块，即将需要的引擎构件加载进来，例如 var file ctrl = Elastos.Using("supershare.dll")，引入了 ShareEngine 构件。通过 ShareEngine.CreateObject()即可实例化相应构件类，调用相关接口函数即可获取服务。

应用程序由多个视图部分组成。在 Elastos 平台上，XmlGlue 在内置的全局对象 application 中添加了 forward(URI ，…) 方法，在 view 中添加了 showDialog(URI， …)方法。其主要视图文件如表 5.1 所示：

表5.1 ShareEngine UI的主要视图文件

supermain.xgl	ShareEngineUI 主界面
unreadList.xgl	ShareEngineUI 信息处理界面
inList.xgl	已收分享信息列表，包含对已收信息的处理
outList.xgl	已发分享信息列表，包含对已发送或正等待发送的分享信息的处理
drafts.xgl	临时分享信息列表，对正在编辑的分享信息的处理
fileEdit.xgl	分享信息编辑界面
fileView.xgl	分享信息查看界面
supersetting.xgl	分享信息设置界面
ListManage.xgl	文件夹管理，对自定义文件夹进行管理，如新建、重命名等
superreportList.xgl	报告列表页面，对发送报告的管理

例如，一个具体的构件化软件分享的实际效果如图 5.3 所示。

图 5.3 Elastos 平台上的分享编辑的效果图

第6章 工作总结与展望

6.1 工作总结

在本篇论文中，作者主要进行了以下几方面的研究和设计工作：

（1）提出了分享模型 SCS

该分享模型通过在 Elastos 平台上实现，以移动互联网为基础，以云计算辅助传输为条件，能够在移动终端之间对构件化软件进行分享。

（2）研究了分享模型 SCS 中的架构设计以及关键技术

研究了模型 SCS 的架构设计，并研究了 XMPP 协议的特点、协议格式、通信链路的建立以及终端之间的传输机制等。并根据分享模型以及业务需要设计了终端之间构件化软件的分享功能。

（3）使用了 Capsule 技术

为了方便以终端为中心的构件化软件的分享的简便传输和管理，采用了科泰公司的 Capsule 技术，该技术可以在 Elastos 平台上很容易将的构件化软件也就是基本应用或者应用中的构件装载起来并获取该构件化软件的基本配置信息一起装载进一个 Capsule 中。这样，在该分享模型中实际的传输实体为 Capsule，通过终端对 Capsule 的解压缩和解析可以很容易的对分享的构件化软件进行管理。

（3）实现了终端构件化软件分享系统

作者在 Elastos 平台上，通过使用 CAR 构件技术实现了终端的构件化软件分享系统，实现的主要工作有设计并实现分享系统引擎 ShareEngine 设计及实现并且利用 XmlGlue 技术设计该分享系统的用户界面。

6.2 工作展望

3G 时代的到来标志着互联网从电脑网络时代向手机网络时代过渡，它意味着使用掌上智能终端将能够实现高速无线上网，它把人们带进了一个日益互动的世界，它意味着从封闭性升华到了开放性，意味着通信向网络的转变。

随着手机硬件水平的不断发展，目前的一款高端智能手机的运算能力与十年之前 PC 机的运算能力相比已基本相当；在软件方面，手机软件的智能化已成为未来的发展趋势。特别是和欣手机平台的出现，首次将 CAR 构件技术和 XmlGlue 技术应用在了智能手机的开发上，使手机软件的开发更加方便，进一步降低了对开发人员的专业要求。

在这种趋势下，如何为手机用户提供一个更加自由和快捷的传播和获取个性化手机应用软件的方式，如何为第三方软件开发者提供一个方便而又高效的软件销售和传播平台，成为了一个迫切需要解决的问题。3G 时代的加速到来和手机软件智能化水平的不断提高，决定了对第三方软件的传播方式的研究将成为未来 3G 通信领域的重要研究课题。

这个模型通过使用 XMPP 实时通信协议实现了移动终端之间的数据分享的问题，建立了以终端用户为中心的数据分享和传播的新途径，它类似于移动终端的彩信功能，但是它打破了彩信的限制，以移动互联网为基础，可以在终端用户之间分享各种数据，还可以在服务器的控制下开展各种业务，如为了实现某个部门各个成员间的协作，而需要一些终端临时具有一些特殊功能，这样就可以在某个范围内，由领导者发出广播，由服务器将该特殊功能的构件或者应用分发到该范围内的各个终端；为了实现两个人同时可以玩一个游戏，实现游戏中的对战或者合作，那么其中一个人就可以发送该分享请求给另一个人，将该应用软件或者具有该功能的构件发送过去，实现多用户协作完成某一项任务。该业务具有广泛的现实意义，可以应用到保安系统中，可以应用到公安部门的人员协作作战中等等，在业务开展时，可以对构件或者应用的有效期以及是否允许二次分发等设置在分享时作一个设定，在任务结束后，使得该构件或者应用自动失效，有效地控制了该业务构件或者业务应用的非法传播。

本篇论文提出了一个分享模型，这个模型通过使用 XMPP 实时通信协议实现了移动终端之间的软件和构件分享的问题，建立了以终端用户为中心的软件和构件分享和传播的新途径，它类似于移动终端的彩信功能，但是它打破了彩信的限制，以移动互联网为基础，可以在终端用户之间分享各种应用和构件，还可以在服务器的控制下开展各种业务，如为了实现某个部门各个成员间的协作，而需要一些终端临时具有一些特殊功能，这样就可以在某个范围内，由领导者发出广播，由服务器将该特殊功能的构件或者应用分发到该范围内的各个终端。

该模型具有很强的扩展功能和业务扩展能力，但是由于个人精力有限，很多工作还未完善，如：增加服务器的控制功能，通过对服务器进行扩展，可以实现用户群之间的软件或者构件分享的传输轨迹记录，实现不同组之间的业务软件或者构件的控制性传播；还可以与运营商以及内容提供商进行合作，开发更加实用的业务等等。

致谢

硕士的学习和生活已近尾声，时光飞逝，蓦然回首，受益良多，感慨颇多。从踌躇满志地踏入校园到带着学有所成的些许自豪地回首，硕士生涯里经历了一件件终身难忘的事情，我的学习和生活在磨练中不断地螺旋上升，借以下的文字寄托我深深地感激之情。

衷心感谢导师陈榕教授对我的精心指导和在硕士期间给予的帮助和照顾。他一丝不苟的治学态度、深厚的知识底蕴、前瞻性的眼光以及执着地追求真理的精神与激情对我产生了深刻的影响，他的言传身教使我终身受益。

衷心感谢顾伟楠教授对我在学习和论文上的精心指导，使我逐渐掌握了如何进行科学研究的方法，感谢他的无私帮助和教诲。

衷心感谢裴喜龙老师一直以来对我用心亲密地言传身教，传授学术研究方法和做人处事的道理，及时调整和纠正我的不足之处，他的无私精神令人敬佩，他给予我极大的帮助和教诲将铭记予心，对我的人生将产生深远的影响。

衷心感谢我的同事徐凡、李红刚等在工作上对我的支持和帮助。

衷心感谢申波以及其他和我一同工作生活的同学对我一直支持和帮助。

最后，要感谢我的家人，我有一个非常幸福的家庭，在我的身边始终有着父母默默的关怀和支持，您们给予我力争上游的无穷动力，作为您们的孩子，我感到无上的骄傲和自豪。

参考文献

[1] 朱旭.移动互联网及其热点技术分析. 中国高新技术企业，2010

[2] 闵栋.3G时代移动互联网应用生态系统发展浅析. 通信管理与技术，2010

[3] 中国移动2010年移动互联网业务发展策略. 2010

[4] 上海科泰世纪科技有限公司．Elastos Digital Capsule 开发文档. 2010

[5] 钟伟彬,周梁月，潘军彪等.云计算终端的现状和发展趋势,移动互联网终端技术，2010

[6] 董晓霞,吕廷杰.云计算研究综述及未来发展. 北京邮电大学学报，2010

[7] Amazon Elastic Compute Cloud .http://aws.amazon.com/ec2/.

[8] Hadoop.http://zh.wikipedia.org/wiki/Hadoop.

[9] 罗达强.探析Windows Azure Platform微软云计算平台.硅谷, 2010(16)

[10] 陈榕, 刘艺平. 技术报告: 基于构件、中间件的因特网操作系统及跨操作系统的构件、中间件运行平台(863课题技术鉴定文件), 2003.

[11] Elastos资料大全.上海科泰世纪科技有限公司，2008年10月

[12] 王志坚,费玉奎,娄渊清.软件构件技术及其应用.科学出版社,2005

[13] 杨芙清,梅宏.构件化软件设计与实现[M].北京:清华大学出版社,2008.

[14] 陈鑫,李鑫,郭超等.构件化软件开发方法的研究.科技创新导报,2010 研 究 报 告

[15] 潘爱名. COM原理与应用. 清华大学出版社, 1999. ISBN 7-302-02268-2

[16] CORBA .http://zh.wikipedia.org/wiki/CORBA,2011

[17] 杨晓倩，陈榕. 基于CAR构件技术的普适计算平台研究. 计算机与数字工程. 2007.10

[18] 郑炜，陈榕，苏翼鹏，殷人昆. CAR构件编程技术中的自描述特性. 计算机工程与应用. 2005.09

[19] 许帧.基于构件的软件开发方法及实现[J].软件导刊,2009(11):17-19.

[20] 程依,秦斌.构件化软件开发探究. 软件导刊,2009

[21] Ghossoon M. W. Al-saadoon.A Platform to Develop a Secure Instant Messaging Using Jabber Protocol. Journal of Computer Science,2009

[22] 潘凤,王华军,苗放等.基于XMPP协议和Openfire的即时通信系统的开发.计算机时代,2008

[23] 陈武,杨世达.基于XMPP 协议的服务器集群研究. 软件导刊,2009

[24] 赵宸.基于Jabber的通信分布式中心.吉林大学硕士论文，2005

[25]PeterSA.Draft-ietf-xmpp-core224XMPP:core[EB/OL]
http://www.xmpp.org/specs/draft-ietf-xmpp-core-24.html, 2004205201.

[26] Peter Saint-André,Kevin Smith,Remko Tronçon.building real-time applications with Jabber technologies,2009

[27] Ottawa, Ontario. Secure Public Instant Messaging: A Survey.2004

[28] Dierks ,Allen,The TLS Protocol Version1.0, RFC2246,1999.

[29] 苗凯.XMPP的安全机制分析.通信技术.通信技术，2003

[30] XMPP RFCS. （Request For Comments, Internet标准草案）[S].2004

[31] RFC3920 ,Extensible Messaging and Presence Protocol(XMPP)：Core/Introduction[]

[32] Myers,J. Simple Authentication and Security Layer(SASL),RFC2222, 1997

[33] RFC 1321, The MD5 message-digest algorithm, 1992

[34] 张云川.标准化的即时通信协议.武汉科技大学学报,2005

[35] 何刚,田森平,田辉平.基于SIP的移动即时通讯系统. 计算机与现代化,2010

[36] RFC3920-2000,Instant messaging/presence protocol requirements[S]

[37] Peter SA.XMPP instant messaging and presence[EB].RFC3921,2004.

[38] RFC2779-2000,Instant messaging/presence protocol requirements[S].

[39] E.Rescorla. SSL and TLS: Designing and Building Secure Systems. Addison Wesley, 2001, http://www.iang.org/ssl/rescorla 1.html [Accessed: June28, 2004].

[40] 黄剑.基于XMPP的端到端连接建立机制的研究与实现.国防科技大学硕士论文,2009

[41] XIA Yong-quan1, ZHANG Bing-quan1, XU Jie-ping2 The Analysis and Application of Socks5 Proxy Technology,2003

[42] Internet standard protocol RFC1928 SOCKS Protocol Version 5[Z].

[43] 夏永泉，张秉权，许洁萍. SOCKS5代理技术分析及应用.2003

[44] XEP-0065,SOCKS5 Bytestreams,2010

[45] 上海科泰世纪科技有限公司．Message UI Specification 1.0.0．2007

[46] 陈榕.另类云计算，另类物联网. 中国计算机学会通信,2010,7(5):56~60

[47] Box, Don. Essential COM. Menlo Park, CA: Addison-Wesley Publishing Company, 1998. ISBN 0-201-63446-5.

[48] J.Rosenberg,J.Weinberger,C.Huitema,R.Mahy,STUN-Simple Traversal of User Datagram Protocol(UDP)Through Network Address Translators(NATs).http://www.faqs.org/rfcs/

个人简历、在读期间发表的学术论文与研究成果

个人简历：

王保卫，男，1985年10月生。

2004年09月至2008年07月，淮北师范大学，计算机科学与技术系，获学士学位

2008年09月至今，同济大学，计算机应用专业，硕士研究生

发表论文：

[1] 王保卫，申波，陈榕. 基于XML的移动设备人机交互引擎研究.计算机应用（已发表，2010年09月发表）