工程硕士

浙江工业大学

硕士学位论文

论文题目：基于 FFMPEG 的视频转换系统

作者姓名 马洪堂

指导教师 吕丽民

学科专业 计算机技术

所在学院 信息学院

提交日期 2009-4-8

浙江工业大学硕士学位论文

基于 FFMPEG 的视频转换系统

作者姓名：马洪堂

指导教师：吕丽民、徐晔

浙江工业大学信息工程学院

2009 年 4 月

Dissertation Submitted to Zhejiang University of Technology
for the Degree of Master

BASED ON FFMPEG VIDEO CONVERSION SYSTEM

Candidate: MaHongtang

Advisor: LvLimin Associate Professor、XuYe

College of Information Engineering
Zhejiang University of Technology
Apr 2009

浙江工业大学

学位论文原创性声明

本人郑重声明：所提交的学位论文是本人在导师的指导下，独立进行研究工作所取得的研究成果。除文中已经加以标注引用的内容外，本论文不包含其他个人或集体已经发表或撰写过的研究成果，也不含为获得浙江工业大学或其它教育机构的学位证书而使用过的材料。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。

作者签名：马世堂　　日期：2009年5月24日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权浙江工业大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密□。

（请在以上相应方框内打“√”）

作者签名：马世堂　　日期：2009年5月24日

导师签名：[illegible]　　日期：2009年5月24日

基于 FFMPEG 的视频转换系统

摘　要

本课题属企业应用项目，来源于互联网视频传播的应用。近年来数字视频工业得到了快速的发展，视频的传播和视频分享也形成了一个行业，逐渐形成了很大的一个市场蛋糕。随之而来的技术问题和视频品质就成了困扰发展的主要障碍。建立起一套降低视频技术门槛，无人值守的视频转换平台也就成了视频 Web 传播的迫切需要。在这样的背景下，研究视频转换就具有非常实际的应用意义。

该项目在协同工作理论的基础上建立起来的，采用计算机支持的协同模型来提供本系统的服务环境。计算机支持下的协同工作（CSCW）作为计算机的一个明确的研究方向已有多年的历史，随着网络技术，尤其是 Internet 的发展，在网络计算机环境下如何支持人们的协同，提高群体工作效率，已成为一个新的研究和应用领域。本文主要研究将适合群体协同、交互信息的 CSCW 应用于科学研究，构造网上科研协同平台，提高工作效率。

本文首先 CSCW 的概念进行了阐述，对活动理论、协调理论、任务管理器常见的 CSCW 协作理论模型进行了研究分析，运用 UML 建立它们相应的模型，然后对 CSCW 的关键技术进行了详细的论述，最后研究 CSCW 系统的对层次结构模型，为视频转换系统的服务环境研究提供了理论基础和依据。

接着论文介绍基于 FFMPEG 的转换系统的服务器端架构原理，结合目前视频转换的应用现状，采用 CSCW 协同工作理论建立的视频转换系统，并对各个层次所采用的技术进行了深入研究，提出了用于服务器的一种集成框架的结构。该结构可以增加系统的兼容性，大幅提高视频转换的效率。为了完成处理多用户请求的任务，本文深入探讨了服务器如何处理多个客户机的请求的处理方法，提出了单进程、多线程等方法，分析了各种方法的优缺点和适用的场合，给出了一个基于 FFMPEG 技术的业务基础架构平台的具体实现。

本文对国内外协同工作方面的进展进行了研究，通过对协同工作工具现状的分析和对需求的探讨，提出了采用 B/S 结构的工作模式的协同工作工具体系结构，使用基于 Microsoft .net、C#等技术设计并实现了 B/S 模式下的视频采集模块、数据处理模块、异常处理模块。最后，对本文工作做出总结和进一步工作的展望。

关键词：FFMPEG，视频转换，CSCW，B/S

BASED ON FFMPEG VIDEO CONVERSION SYSTEM

ABSTRACT

The subject project is an enterprise application, from the applications of Internet video distribution. In recent years, digital video industry has been rapid development, the spread of video and video-sharing also formed an industry, gradually formed a huge cake in a market. But followed by technical problems and video quality problems became a major obstacle to development. Establish a set of video technology to lower the threshold, automatically converting videos web platform has become an urgent need to spread. Against such a background, the practical meaning of study on Video Converter Application is very important.

The project set up on teamwork theory, the use of computer supported cooperative model to provide the service environment of the system, the ones that stood in computer had already to work (CSCW) in coordination had a history of more than ten years, with the network technology, especially the development of internet, how is it to support people to work in coordination with under the environment of network computer, improve colony's working efficiency, has already become a new research and application. It suit colony of research it is main in this text in coordination with, mutual CSCW of information apply scientific research of, the online scientific research of the structure improves working efficiency in coordination with the platform.

The concept of CSCW at first of this text has been explained, to activity theory, coordinate theory, task management device and face target activity support model, etc. Common CSCW cooperate theory model analyses, use UML to set up their corresponding model, and has classified it according to the characteristic of CSCW, then introduced the application of CSCW, has carried on the detailed argumentation to the key technology of CSCW, the ones that studied CSCW system offered theoretical foundation and basis for serving environment research in coordination in scientific research to the Video Conversion System model finally.

Then thesis introduced the server-side architecture principle of based on FFMPEG video conversion system, combine the current status of the application of video conversion, we using cooperative work CSCW theory set up the video conversion system, and get a full range of research to the system, Put forward for an integrated framework of server structure. This structure can increase the system compatibility; substantially improve the efficiency of videos conversion. Achieve in order to handle many user requests the task, in this paper, in-depth exploration into how to deal with a number of server requests client approach, methods such as single process\multi-thread, analyzed the various methods of advantages and disadvantages and application of occasions, then give an implementation program of FFMPEG technology-based infrastructure platform.

Abroad progress of respect carry on research, through right to work analysis and discussion of the demand, tool of current situation in coordination in coordination both at home and abroad, and analyses and compare to some existing in coordination with the tool, the system structure of the tool that put forward three stories of the work pattern of adopting B/S structure and work in

coordination. Use the module on the basis of Microsoft .net，C# technical design and realize video capture module, data processing module，exception handling module that have realized under B/S mode. Finally, make the prospect that is summarized and working to the work of this text further.

Key Words: FFMPEG，Video Conversion，CSCW，B/S

目　录

第1章 绪　　论

1.1 引言

近年来数字视频工业得到了快速的发展,数字视频摄像机也成为了人们日常生活的一部分,但是同时也增加了视频的格式和编码方式[1]。因此,如何能够从一种格式转换成另一种更适合的格式也成了一种迫切需求,而 FFMPEG 为音视频的转换提供了强大的支持。本文主要讨论了如何在互联网视频传播应用中,借助在 Linux 系统下开源的 FFMPEG，实现一个无人值守的第三方平台对视频的格式进行转换以及视频的上传与发布。从而使各种流媒体和 Web 结合起来，解决目前互联网视频传播中的 Web 对视频格式转换、压缩以及网络传播中的技术障碍。

网络视频经过两年的历练，已经成为中国互联网新兴业务中的佼佼者，虽然面临很多政策性的制约，但其发展态势良好，已成为重要的互联网应用之一。不仅诞生了多达百家以上的专业视频网站，所有的门户网站也都有自己的视频内容；不仅有大量的新闻类、影视类内容，也有了数量更加庞大的用户自拍内容，也许正是这种用户提供内容的方式，才使得网络视频市场繁荣如此。

在去年年底，DCCI 互联网数据中心发布了《Netguide2008 中国互联网调查报告》。在报告中，DCCI 分别从网络视频分享服务、P2P 视频播放平台服务、视频点播/直播服务、视频搜索服务四个方面对中国网络视频市场进行了分析。报告认为，在 2009 年，网络视频监管政策、视频内容审核等将成为影响视频服务发展的关键因素，同时网络视频也将迎来快速发展的一个时期。网络视频已经得到广泛的认可，市场蛋糕也已经慢慢形成，中小企业的视频展示业蓬勃发展起来，所以，互联网已经从原来的图文发展到网络视频的时代，这也标志着新一轮的互联网革新已经开始[2]。

在互联网的视频传播应用中，困扰技术开发人员已久的就是对视频格式的转换和压缩。互联网视频传播过程中，受到机房硬件和带宽的限制，压缩视频文件的大小成为了关键的研究课题；另外为了提高互联网用户的操作体验，在播放器的选择上，也都倾向于用户认可的 flash 播放器。而作为这一系列的基本条件，视频传播就需要分享视频者将视频手动转换成统一的 FLV 文件，这就给用户造成了很大的瓶颈，而转换的需要人工手动处理，也成了互联网视频传播的一大障碍。而本项目的完成，将解决上述的种种问题。

本系统实现了无人值守的视频上传、视频转换和视频发布，降低了人力资源的成本，极大地提高了视频处理的效率，也将推动互联网视频传播应用的发展；视频分享网站的技术开发人员无需了解视频处理的整个流程，只需要和以前的图文企业网站开发流程一样，实现基本的数据收发功能，结合本系统，就可以实现视频的互联网传播，这就降低了视频处理的技术门槛；视频处理集成在了可安装的 Windows 服务中，整个网站系统的稳定性和及时性得到了极大的保证，整个视频处理也变得十分流畅。

因此本项目的完成，可以极大地推动网络视频的发展，特别是能极大的推动中小企业视频展示的进程。

1.2 课题研究的背景和意义

1.2.1 背景

随着我国信息化建设的不断发展,人们的生活方式也发生了巨大的变化。目前,视频分享已经成为人们日常生活的一部分,随之而来的是以土豆、56、六间房为主流的 web2.0 的视频分享网站，视频娱乐平民娱乐化,随时随地看电影已经不再是梦想，同时企业的电子商务中的展示也迫切需要多元化的视频展示，但是由于流媒体播放受目前带宽等硬件环境的影响,并不是所有格式的视频都能够达到高品质用户体验的播放,而且 Web 服务器的存储成本偏高,不能够存储大容量的视频文件,因此需要对视频的格式进行转换和压缩。为了方便用户预览视频的内容,要把视频转换成了适合于在网络上在线播放的格式 FLV 格式。而目前处理视频格式功能最强大的就莫过于 Linux 下的开源项目 FFMPEG[3]，但是在视频的 Web 传播中，格式转换、视频压缩、视频发布都需要加入人工处理，造成整个分享视频的流程脱节不流畅，形成了一些发展的瓶颈，主要有以下几个问题：

FFMPGE 是 Linux 的开源项目，在 Windows 下兼容性不好，因此操作系统兼容方面需要进一步加强。

互联网视频传播中，技术人员对视频的整个处理流程没有办法整合，处理过程中通常要加入人工手动处理，极大地影响了整个系统运行的效率；

处理流程严重脱节，容错性能低下；

本文主要讨论的视频转换系统就是采用 C#语言在.net 框架下,利用 FFMPEG 平台开发而成的。本系统着力解决了以上问题，把视频转换压缩等整个视频处理流程集成到了一个可安装操作系统的服务中，同时兼顾了系统的容错性、稳定性和安全性，并且在处理过程中完全实现了无人值守的智能化操作。同时技术开发人员不必了解视频的整个处理过程，更无需了解 FFMPEG 的应用，这就极大地减轻了技术人员的压力。

1.2.2 意义

随着 Web2.0 观念的提出，视频共享的理念奠定了基础，P2P 技术的发展，去年的奥运会，加热了网络视频产业的发展。在线观看视频成为网民浏览互联网主流形式之一，伴随着技术上的突破，在线上传视频内容如同发布文字、图片一样的简单容易，网民自然也成为互联网视频内容制作者、提供者、发布者，网民拥有了视频互联网的生活[4]。

在视频分享网站青娱乐上，你可以看到叔叔大爷婶婶们打太极、跳秧哥；可以看到七十多岁的老太太拍下自己可以原地旋转十八个圈的晨练视频；可以看到一些人豪华的家居陈设和精美装修，也可以看到普通百姓的温馨陋室；可以看到成龙李小龙的崇拜者的双节棍，也可以看到可爱的狗狗们和精美的鱼缸；可以看到其乐融融的小康之家的全家福告白；还可以把自己演绎的歌声等影音献给亲人、爱人、朋友[5]。

视频互联网时代，在线浏览视频、在线拍摄视频、在线发布视频，已经成为极其简单容易的事情。视频表述更直接、更真诚，比文字、图片更有说服力度，更具表现力。视频互联网的内容越来越丰富，越来越以人为本，越来越主流化。网络视频的应用也在广泛出新，很多网站已经准备推出企业视频招聘、视频个人简介等业务。

视频互联网已经融入人们的生活，且成为人们生活应用中不可分割的一部分。目前视频后端处理给中小企业门户网站、电子商务平台以及视频娱乐平台形成了技术壁垒。本项目对视频网站发布后端的整合，通过对网络视频在网站上处理的全过程(视频上传、视频压缩、视频格式转换、数据库中数据状态处理，视频发布)进行深入的研究,构建兼容了多个操作系统的 FFMPEG 的编译环境；结合 FFMPEG 的开源代码，优化更新 FFMPEG 对视频的压缩和格式转换，重新定义 API 接口；设计开发可安装在操作系统中的服务，实现无人值守的对新视频数据的扫描、对视频进行压缩、格式转换、视频截图等操作。从而使从事视频网站开发的技术人员可以利用本系统，实现高品质用户体验的视频处理，同时也是护联网视频传播应用中处理视频的一种创新方法。主要表现为：

视频处理方式创新：打破了原来按部就班的处理方式，把前端的上传视频和后续的视频格式处理发布分离，同时又形成了高品质的用户体验

系统形式的创新：采用操作系统下的服务，形成视频处理智能化处理，同时增加了处理的容错性、稳定性和安全性

技术创新：实现 FFMPEG 的重新编译和优化，用系统服务方式解决视频压缩，格式转换，同时实现服务对视频发布状态的实时跟踪。

1.3 课题研究现状

目前视频Web传播的前端都是采用FLV的视频格式，通过自己订制的Flash播放器播放，客户端只要安装了Adobe的flash插件，用户即可播放清晰文件体积小的视频。

影响视频Web传播的主要是技术后端对视频格式压缩和FLV的格式转换。众所周知，网站的发布平台分为Windows和Linux,强大的视频处理利器FFMPEG是基于Linux的开源项目，而对Windows的兼容性很差，这就给Windows系列下的视频Web传播形成了技术壁垒[6]。

国内目前视频处理尚无好的解决方案，通常采用架设另一个Linux的服务器，专门对视频文件处理，用户上传视频和系统后端处理视频脱节，并且还可能需要系统后端的手动处理；这一系列的条件使得视频传播阻力重重：

首先技术人员需要对视频处理的相关的技术非常了解，这就形成了一个高的技术门槛；

整个网站发布有可能是两种操作系统的服务器协作，这就影响了整个系统的稳定性和及时性，整个处理环节的流畅性就不能得到保证；

同时需要加入人工手动处理，形成了人力资源的成本，并且影响了处理的效率，而互联网视频传播的高速发展，使很多视频Web传播无法回避以上的问题，造成了目前视频的传播发展满足不了用户需求，特别是技术壁垒对中小企业来说，形成了高的技术人力成本，致使到目前企业的视频展示呼声很高，能实施的却不多，极大地影响了互联网视频传播的普及性，和互联网视频的发展[7]。

1.4 课题研究的主要内容

项目组根据市场需求及行业特点，确定本项目主要解决的四方面的问题：对视频转换的开源项目FFMPEG的优化、系统服务中调用视频转换时对进程的管理、服务和Web程序对数据的挖掘和处理、异常捕捉和处理。整个项目根据也是围绕这四个方面的问题展开，在开发中可划分为FFMPEG的优化、服务的开发、服务中数据挖掘及和Web程序的交互，因此课题研究的主要内容有：

1.4.1 研究用于支持这种系统的模型CSCW

CSCW代表一类支持异地用户群体通过计算机和通信技术以合作方式共同工作的计算机系统。人们在群体或组织中采用协调、互相合作的手段进行工作，目的在于完成其最高工作目标，而本系统最终目的就是使用这种模型，把一系列的工作，采用CSCW的模型协同工作达到目标。

1.4.2 服务框架在操作系统环境中的集成

本项目将改善固有的Web程序中对视频处理的繁琐性，采用一个可以安装的系统服务达到无人值守自动扫描处理视频的目的，因此需要开发一个直接安装到操作系统中的服务。

1.4.3 服务对进程的管理

在服务中对视频截图需要调用FFMPEG，当Web中积累大量需要截图和转换的视频时，为了提高效率可以开启多个转换和截图进程，但是又要考虑到服务器的承受力和异常率，如何达到一个合理的进程控制，达到高效率的转换，就必须开发一个对进程实时管理和跟踪的功能。

1.4.4 服务中对数据的处理

视频转换队列的生成，需要在Web产生的数据库中挖掘提取，在海量数据库中如何提高数据库的查询速度便成了研究重点。

1.4.5 服务中对FFMPEG的调用

在开发的服务中，调用第三方程序，进行文件的处理。

1.4.6 异常捕捉和日志记录

当进程转换出现异常、进程出现异常、服务器任务管理出现异常时要进行合理的处理，同时要进行转换日志的记录。

1.5 论文的组织

本文共分为六章，各章内容如下：

第一章 为绪论部分，主要介绍了本课题研究的背景、现状意义及课题的主要内容；

第二章 探讨了CSCW相关的概念和特征，研究了几种常用协作理论；

第三章 介绍了基于FFMPEG的视频转换系统的服务环境，对适用于本系统的系统理论的关键技术进行了研究分析；

第四章 介绍FFMPEG的视频转换的服务器端架构和实现原理，同时对系统的层次模型进行了研究；

第五章 介绍具体的各系统模块设计和具体实现，及关键的代码；

第六章 总结全文，对系统优化及进一步工作进行了展望。

第 2 章 计算机支持的协同工作理论及模型分析

计算机技术的发展把人类社会带入信息时代，随着网络技术，尤其是 Internet 的发展，通信技术与计算机及其网络技术相融合，产生了一个新的研究领域——计算机支持的协同工作(Computer Supported Collaborative Work, CSCW)，简称计算机协同工作，它是社会信息化进程发展的必然产物。

CSCW 代表一类支持异地用户群体通过计算机和通信技术以合作方式共同工作的计算机系统。人们在群体或组织中采用协调、互相合作的手段进行工作，目的在于完成其最高工作目标，诸如获得最大利润、获得设备的最大利用率、生产优质产品和圆满完成科研任务等。

2.1 计算机支持的协同工作的概念和特征

2.1.1 CSCW 的概念

计算机网络、通信技术和多媒体技术的飞速发展，不但给社会许多领域带来了深刻的影响，也给计算机许多领域带来了新的机遇和新的课题。德国斯图加特大学理论物理学教授赫尔曼•哈肯(Hermann Haken)在 60 年代研究激光理论的过程中，经过十几年的努力逐步形成了“协同学”的基本理论和观点。赫尔曼•哈肯教授本人还把协同学思想扩展到计算机科学和认知科学，在 1991 年发表了一本重要著作《Synergetics Computers and Cognition---A Top-Down Approach to Neural Nets》（协同计算机和认知——神经网络的自上而下方法)，从而奠定了协同学的基础。1984 年，美国 MIT 的 Irene Grief 和原 DEC 公司的 Paul Cushman 两位研究员正式提出了 CSCW 的概念。这是他们在描述有关如何用计算机支持来自不同领域与学科的人们共同合作的课题时提出来的。我国清华大学史美林教授等把“计算机支持的协同工作”定义为[8]：地域分散的一个群体借助计算机及其网络技术，共同协调与协作来完成一项任务。它包括协同工作系统的建设、群体工作方式研究和支持群体工作的相关技术研究、应用系统的开发等部分。通过建立协同工作的环境，改善人们进行信息交流的方式，消除或减少人们在时间和空间上的相互分隔的障碍，节省工作人员的时间和精力，提高群体工作质量和效率，从而提高企业、机关、团体、乃至整个社会的整体效益和人类的生活质量。如：共享文件系统提供的资源共享能力，电子邮件和多媒体会议系统提供的人与人之间的通信支持功能，工作流和决策支持系统的组织管理功能，一个企业如果有效地利用这些基本工具构造其企业协同管理信息系统，必将提高企业

的管理水平和效益。

CSCW 是一个多学科交叉的研究领域。不仅需要计算机网络与通信技术、多媒体技术等计算机技术的支持，还需要社会学、心理学、管理科学等领域学者共同协作。计算机协同工作将计算机技术、网络通信技术、多媒体技术以及各种社会科学紧密地结合起来，向人们提供了一种全新的工作环境和交流方式。

可以从 CS 和 CW 两个方面来认识 CSCW 这个概念：在计算机技术支持的环境下(CS)，特别是在计算机网络环境下，一个群体协同工作完成一项共同的任务(CW)，它的目标是要设计支持各种各样的协同工作的应用系统。

2.1.2 CSCW 的特征

CSCW 泛指：在计算机技术支持的环境中（即 CS)，一个群体协同工作完成一项共同的任务（即 CW)。它的目标是设计支持各种各样协同工作的应用系统，因而 CSCW 系统是面向群体（Group)、面向问题的综合人机系统，具体有以下三个特征[9]：

1）任务的协调：CSCW 系统有明确的目的性，CSCW 系统中的协作活动与具体的事务处理有着紧密的关系，在同一环节上有时需要多个参与者从不同的背景讨论处理，因而与系统目标有紧密关系的参与者必须是紧耦合的；另一方面，系统必须是可伸缩的，能够自由地接纳更多的参与者，并根据事务处理的要求，对各类资源的利用灵活地规划。

2）共享信息空间：参与协作的各方往往分布在不同的地理位置（当然也可以在同一地方)，因而事务的分析和处理主要是分布式处理方式，这就要求有高效的信息支持和决策支持。CSCW 系统中协作的参与者需要共享信息空间，如数据、声音、图像、背景资料、协作环境等，参与者之间能有效地讨论和处理事务，事务的分布式处理才能真正实现。

3）与现代科学技术的融合：CSCW 是一个多学科交叉的新兴研究领域，对它的研究和设计难度很大，涉及系统结构、高速多媒体通信协议、多用户协作机制、同步机制、人机接口等等各方面的内容，需要计算机科学、社会学、心理学等各方面的专家共同协作，使系统成为能自由地相互交流思想、协同工作的场所。

2.2 CSCW 协作理论分析

CSCW 研究的目标之一是提高协同成员间的协调配合和协同工作水平。因此必须进一步深入了解群体内成员间的协作模式，用以指导协同工作技术和方法研究。CSCW 中对群体协作模式的研究，是利用社会科学的研究成果，进行跨学科研究，概括出人类群体在信息社会环境下的协作模式，用于指导协同工作技术研究。

下面对 CSCW 领域中出现的四种协作模型进行分析，并运用 UML 建立其相关模型，这里使

用 UML 的类图来描述模型的静态结构，它能够表示模型的各种类及类与类之间的关系。

2.2.1 协调理论

协调理论(Coordination Theory)[10][11][12]是 MIT 协调科学中心的 Malone 提出的一种管理一组协同工作的活动及其相关性的科学。协调理论是关于协调的综合学科， 协调可能被定义为：管理在活动之间管理独立性的过程。协同过程的组成元素包括共同的目标、完成目标需要执行的活动、活动的执行者以及活动之间的相关性。协调理论的主要研究内容是如何管理活动之间的相关性。用 UML 建立的协调理论模型如图 2-1 所示。

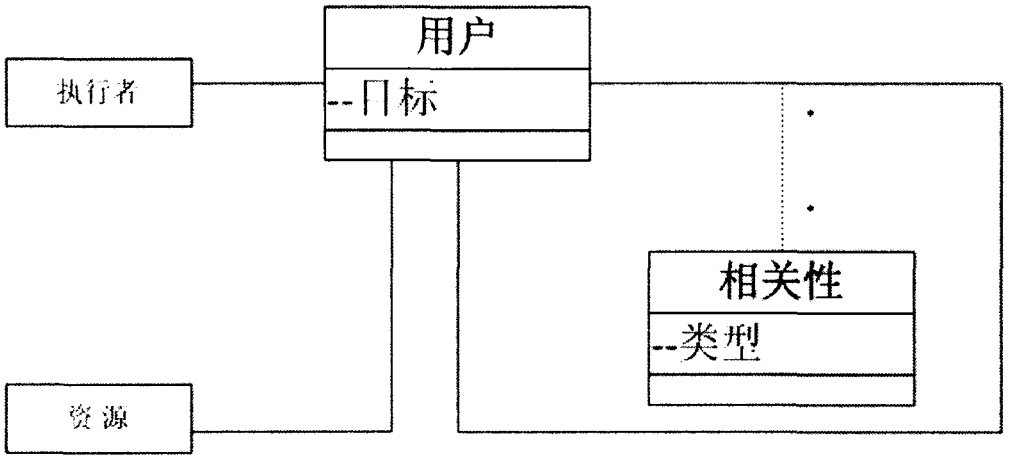


图 2-1 协调理论模型

Fig 2-1 Coordination theory model

2.2.2 活动理论

活动理论(Activity Theory) [13][14]起源于 20 世纪 30 年代，最初提出该理论的是俄国精神学家 Lev Vygotsky(1896-1934)，后来北欧的学者对活动理论加以修正，并进行了公式化的表述[15]。同时这一理论被应用到人机交互设计领域，并引入到过程建模中来。

活动的组成分为项目、目标、规则、团体、任务划分、结果和工具。活动可视为人类从事某一事件的过程集合，即利用工具从某一项目出发，在目标的指引下，在相关规则的约束下通过团体，最后得出所需要的结果。一个活动可以包含几个项目，每一项目可以有一个或多个动机。就其实际应用而言，“团体”相当于一个开发小组：而规则是表明小组成员如何与全局工作相联系，并限制其内部关系；任务划分则是如何将各种活动通过开发小组划分出去。它是一个更具有普遍性的、能够刻画群体协作的模型，把一个任务分解成多个按一定分工具有一定明确目标的由主体和客体组成的活动，任务群体成员根据一定规则，利用合适的工具一步步地执行各个活动，协同完成该任务。主体定义各任务之间的关系，通过活动的执行而完成协同工作。

运用 UML 建立的活动理论模型如图 2-2 所示。

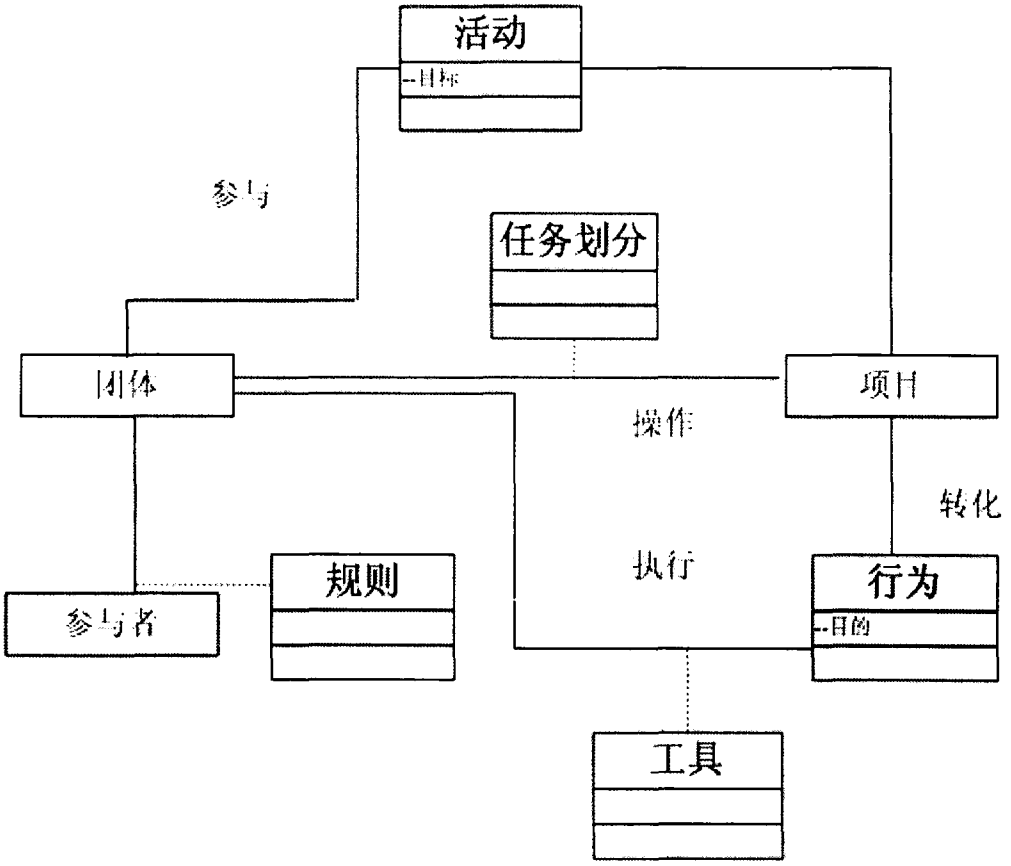


图 2-2　活动理论模型

Fig 2-2 Activity theory model

2.2.3　任务管理器

任务管理器(Task Manager)[16][17][18]是针对协同工作的规范和管理而开发的一种工具。任务管理器的核心概念是任务。其基本思想是：为了完成某一任务，人们利用共享的文档或服务，并且通过交换信息来进行通信。

一个任务，从不同的角度，有不同含义：可以是一个项目，可以是具有相互依赖关系的子任务集，可以是文件夹，作为类似于子任务、文档和消息等共享目标的容器。

资源包括计算机化的资源和非计算机化的资源。前者是指参与同一任务的协同人员所共享的计算机化文档；后者是指类似于车间、机器等非计算机化的广泛对象。

根据不同人员在任务的执行过程中所起的作用不同，将任务管理器的人分为参与者和观察者两类。参与者对任务的属性、文档、服务和消息拥有访问权限；观察者只能浏览与任务相关的信息。参与者还可根据所拥有的访问权限不同，分为责任人和不同的协同工作者。所有相关人员都可交换电子邮件。

运用 UML 建立的任务管理器模型如图 2-3 所示。

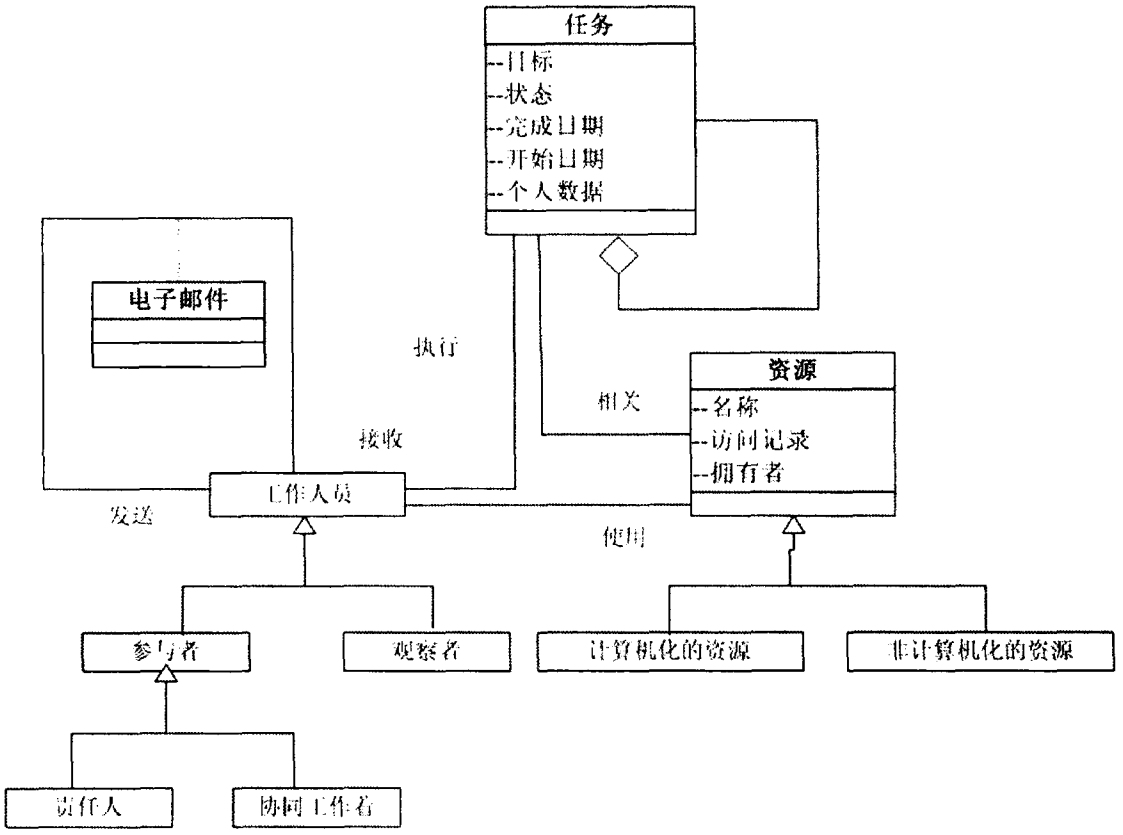


图 2-3 任务管理器模型

Fig 2-3 Task manager model

2.3 本章小结

本章首先对 CSCW 的概念进行了阐述，对活动理论、协调理论和任务管理器常见的 CSCW 协作理论模型进行了分析，运用 UML 建立它们相应的模型，并根据 CSCW 的特征对其进行了分类，为基于 FFMPEG 的视频转换系统的服务环境研究提供了理论基础、模型和依据。

第 3 章 基于 FFMPEG 的视频转换系统的服务环境

3.1 项目目标

3.1.1 系统中涉及的数据类型

基于 FFMPEG 的视频转换系统是为了实现互联网对视频文件传播的需要，在互联网视频分享的过程中，将视频按照一定的规格压缩转换，采用 CSCW 的模型，完成对互联网视频分享需求的服务平台。该服务系统所处理的数据类型主要有以下几种：

视频文件：在互联网中，网民自己通过拍摄、转载、翻录等各种方式制作的视频文件，用户互联网中与亲朋好友及其他网民分享的视频，这也是本系统中要处理的核心数据部分。

图像文件：通过系统对视频中的片断截取的图片文件，用户标示该视频的内容。

视频图像文件的系列化表示：单一的视频和图片在互联网上传播没有办法直接清晰明了的展示核心内容，但是经过系统的处理，将视频、图片和文字配合起来，生成明了的一条视频分享数据，达到最终的分享目的。

用户互动数据：视频在互联网上分享后，网友之间的互动交流数据。

3.1.2 基于 FFMPEG 的视频转换系统的主要工作方式

上述是系统中的数据类型，基于 FFMPEG 的视频转换系统就是对上述各种类型数据的处理，系统主要的工作方式有以下几种：

用户和系统的交流，即数据的来源及采集。互联网用户通过 Web 的形式将自己的视频数据文件传输给本系统，系统将其原始数据有序的存放在本系统架构的服务器平台上[19]。

基于 FFMPEG 的视频转换系统对原始数据的处理。用户上传的视频文件格式不一，文件体积过大，不利于在互联网上传播，系统对各种格式的视频文件进行统一转换，转换成适合在互联网上传播的 FLV 格式，同时采集视频中的某一桢的图片，把用户的原始数据系列化。

系统中异常数据的处理。用户上传的文件损坏，或者文件类型不正确，系统会在转换和处理的过程中对异常数据进行判断，及时释放占用的系统资源，同时将数据整理好之后给用户一个反馈。

系统和用户的交流。数据经过以上两个工作方式的处理后，其他互联网用户在请求该

数据时，系统会按照互联网的标准进行响应。

3.1.3 基于 FFMPEG 的视频转换系统服务的组成部分

基于 FFMPEG 的视频转换系统是一个用户支持互联网视频分享的软件支撑环境，其主要有以下三个部分组成：

A、网络协同服务平台，网络协同应用服务平台是基于 FFMPEG 的视频转换系统服务环境建设的核心。它的主要功能是构建一个支持多种视频格式转换和视频处理、采用同步异步两种协同方式，使用多种协同共享工具、交互式的数据处理环境平台，支持对互联网视频分享的数据格式转换及数据系列化提供各种支撑功能，以满足需求的应用形式，支持人机协同、计算机协同的工作方式。平台支持的 RMVB、AVI、WMV 等、语音、视频等，使用的协同共享工具包括 FFMPEG、IIS、网页等。网络协同应用服务平台以互联网应用的形式进行用户的组织和管理，提供在线视频分享功能。平台应该拥有严格的数据转换和异常处理的处理机制，同时过滤非法的信息，保护互联网的公共信息安全。

B、计算机对视频数据转换处理的子系统，此模块为视频数据提供格式转换服务，返回的视频数据按照序列存放到网络协同的服务环境中。凡是符合本系统兼容的视频格式（目前主要是：RMVB、RM、AVI、WMV、ASF、DATA、MPEG）数据或者音频数据，都可以按照系统规范的要求进行上传转换，从而为互联网视频传播提供统一的视频文件规范。在本系统中主要提供 FFMPEG 为核心的视频转换服务，视频数据经过转换后，和其他图片文字等系列化后，以实时的方式，将处理好的数据在网络协同研究应用服务平台上传播，并由该平台转发给互联网的用户。

C、网络协同操作系统服务的开发和集成规范，基于 FFMPEG 的视频转换系统服务器端的集成是采用可以与操作系统系统无缝结合的方式，将整个系统中协同的部分用开发的可安装的操作系统服务的形式连接。例如将数据的状态扫描、调用 FFMPEG、数据序列化整理都集中在各个计算机的启动服务中处理。

3.2 功能需求

3.2.1 FFMPEG 的视频格式转换

系统的核心是对视频的处理，把用户采集的各种视频格式转换成利于互联网传播的 FLV 格式。因此本系统的核心功能之一就是对视频文件的格式处理，同时达到文件压缩的功能，而 FFMPEG 作为开源的一个视频处理项目，具有强大的功能，也在本项目中起到了关键性的作用。但同时，FFMPEG 也有其缺点，主要体现 FFMPEG 是基于 Linux 开发的开源项目，源代码和 Windows 下最常见的 Visual Studio 提供的 C/C++编译器不兼容[20]，因

此它不能使用 MSVC++编译。要想使用 FFMPEG，最先要解决的问题就是在 Windows 下配置一个类似 Linux 的编译环境，将 FFMPEG 编译为二进制库以后，再利用其进行进一步开发。

MSVC++并不严格的遵循 C 标准，所以整个编译过程必须使用一个合适的系统来完成[21]，从而使 FFMPEG 达到最大的格式支持和对 Windows 系统的兼容。

3.2.2 网络协同服务对数据系列化的处理

项目要达到无人值守，方便快捷地提供适对视频的处理的目标，如何使一系列的服务器实现对原始数据采集后，调用 FFMPEG 对视频压缩、视频格式转换和视频截图，将是重要的实现内容。本项目提出并构建智能服务化的系统，实现操作系统服务的形式集成服务器端的框架，实现系统服务中对第三方调用的技术，完成服务对视频处理的进程管理和队列服务，同时保证视频处理过程中数据的安全性和正确性。从而实现对互联网视频分享数据的系列化，进行视频传播的需求。

实现的内容主要包括可安装服务的开发设计、可安装服务中对第三方调用的技术、数据安全研究等。

3.2.3 服务对数据库的扫描和状态的跟踪

对视频上传数据的跟踪是本项目的对高品质用户体验追求的表现方式。由于本项目的服务对象网络视频的终端用户，对用户上网的体验，研究一套使用方便，操作简单的高品质用户体验尤为重要。本项目提出采用安装在操作系统的服务处理上传数据挖掘和视频文件的客户体验模式，借助服务开发设计、服务对上数据的跟踪、FFMPEG 对视频的处理实现高品质的用户体验，在服务器协同的情况下，实现视频分享过程中的数据处理与分发。

实现的内容主要包括服务对数据库的处理技术、数据挖掘查询研究、数据库技术。

3.2.4 安装配置管理和异常处理

本系统的运行具有实时性和不间断性，如何保证系统运行的长期稳定，数据安全和系统安全将是本项目需要研究的重要内容之一。本项目将研究开发一套安装配置管理和异常处理系统，实现对和数据库结合的连接表处理，对视频转换、数据状态更新操作的日志等进行监控与管理。

实现的内容主要包括配置管理、转换管理、服务报警管理、服务异常处理。

3.3 项目中协同工作的关键技术分析

在本项目的 CSCW 研究中，主要技术基础是计算机及计算机网络技术，主要的动力来源于视频协同处理的应用要求。其关键技术主要包括[22][23][24]：

3.3.1 人机交互接口研究

人机交互在本项目中处于一个比较重要的环节[25]，互联网的视频传播是互联网终端将原始数据提交给协作的计算机组，数据经过处理后，将数据分发给互用。而 CSCW 系统的人机交互接口和单用户系统不一样，它应能体现群体活动及多用户控制的特点。在CSCW系统中，人机接口带来了单用户接口所没有的问题，其中之一就是要支持由多用户系统特有的协作活动所带来的复杂管理。在多媒体网络环境下，多用户、多媒体信息和多窗口的应用程序接口技术，要求能提供支持协同工作的友好用户接口，以便于应用集成、应用管理及用户使用，所以接口技术是 CSCW 系统的关键技术之一。另外，为了支持协同工作，CSCW 的多用户接口必须允许用户知道其他用户的活动，因此，CSCW 的交互接口不是一般意义的人机交互接口，它有一些基本要求，如：它必须能够实时地显示正在合作的信息并把这一信息发布给各参与者，而支持不同的观点和信息表示是 CSCW 多用户接口要解决的又一个问题，一方面允许协作成员对协作信息和结论发表自己的见解；另一方面能根据协作成员的不同权限、级别和层次来表示共享信息。

3.3.2 协作控制机制

在每次的协作过程中需要遵循一定的规则[26]，否则会引起协作的困难。协作控制机制讨论协作过程中产生的各类协作事件间的逻辑关系。CSCW 应用系统中的协作控制机制要求考虑到群体成员在协作时的行为习惯和心理状态，向各成员提供协作所需要的信息。目前的许多 CSCW 应用系统提供一定的信息交流手段和工具，但没有体现协作规则，而由群体成员人为协调各自的行为。这方面的研究内容主要为规则的抽象和协作规则在 CSCW 系统中的实现。在协作控制机制中，协作机制的实现是构造各种 CSCW 应用系统的基础，目前常用的有透明协作（Collaboration-transparent）和有意识协作（Collaboration-aware）两种：其中透明协作方法通过利用远程指针使多个用户能够同时观察某个应用程序的输出结果，通过把向该应用程序输入数据的控制权从一个成员转向另一个成员的方法，来实现协同操作。但是当合作成员需要特定用户组（群）的反应或应用数据时，这种协作机制就缺乏必要的功能，这时候就需要才用有意识协作的方法。

3.3.3 同步机制

在协同系统中，需要向协作的主体之间提供一个统一的工作环境。多媒体通信同步是实现 CSCW 系统中协作成员之间和面对面之间交互的基础和关键[27]。在 CSCW 中，主要研究并实现 MHEG（Multimedia and Hypermedia Coding Expert）提出的同步机制，即脚本同步、条件同步、时空综合同步和系统同步，着重研究同步机制的分层协议。如视频采集时同时响应多个视频的上传、实现多条线程同步处理视频的转换和压缩。

3.3.4 CSCW 应用系统开发环境和应用系统集成技术

CSCW 的应用领域十分广泛，良好的 CSCW 应用系统开发环境可缩短应用系统开发周期，降低应用系统的开发成本。CSCW 应用系统开发环境就是在解决计算机协同工作关键技术的基础上，形成协同工作的应用编程接口（API，Application Programming Interface），为各种 CSCW 应用系统提供一个功能完善的开发环境[28]。CSCW 应用系统开发就是在此基础上进行剪裁，选择适当的协作模式和控制机制，构造 CSCW 的应用系统。

3.3.5 应用共享技术

应用共享技术[29]是指由一个群体的各成员通过各自的机器共同控制在一台机器执行的程序。其目的是扩展已有的大量的单用户应用程序，使之可由多个用户共同控制，实现协作。应用共享的基本方法是把单用户应用程序的显示输出分发到各用户的机器上进行显示，并按一定的策略合并各用户的输入对应用程序进行控制。

3.4 技术路线

3.4.1 对本系统所需要的协同环境的软硬件的部署

视频分享在互联网上的应用是一个相当复杂的工作，为了保证投入的有效性，我们采用合理的服务器硬件架构，在满足需求的前提下，尽量节约成本。首先需要一个专业的可以发布网站的 Windows 系列的 Web 服务器，用于在互联网环境下采集用户的视频信息；其次需要一个共享的存储设备，作为文件服务器，存储这些视频音频文件；然后需要一台 Windows 环境的视频转换服务器，提供本项目的核心服务，视频数据的处理同时对视频分享数据进行系列化整理；最后，需要一个 SQL 数据库服务器，提供数据交互的存储。

3.4.2 基于协同架构的系统结构体系

项目中采用系统的工作理论，根据对 CSCW 的深入研究，结合项目需求，将视频采集、视频转换和数据的处理，采用系统中的人机交互接口、协作控制机制、同步机制等等，构架起一个协同工作的视频转换服务系统。

3.4.3 基于 FFMPEG 的视频转换系统的服务器端框架集成

采用于可以与系统无缝集成的 Windows 服务方式，将整个功能集成起来。整个计算机协同的网络环境基于 Internet 网络上，采用开放式连接，采集用户数据，通过系统服务，进行数据处理，满足视频分享的互联网应用。

3.4.4 基于 FFMPEG 的视频转换系统的安全机制

采用遵循国家标准的策略，在“网络协同应用服务环境”的安全算法中，将采用通过国家认证的安全标准、算法和软硬件工具来加强系统的安全性。对于某些暂时没有国家标

准的算法和通过认证的工具，系统将进行相应的开发。总体而言，系统将从用户认证、IP地址限制、网络数据加密和密钥分发、通讯端口动态指定等方面保证系统安全。

完善其对异常文件的处理机制，提高对损坏文件、病毒文件等的处理。及时记录日志，便于查看记录。

3.5 本章小结

本章主要讲述了基于 FFMPEG 的视频转换系统的项目目标、主要的功能需求和实现的技术路线。

第 4 章 基于服务器端的架构解析和实现原理

4.1 系统服务框架

服务环境集成框架建立在客户机/服务器结构的分布式系统的基础上，它提供了一个异步 CSCW 系统的服务器的框架[30][31]，适合支持具有异步通信要求的 CSCW 系统。该集成框架有效地支持 CSCW 系统的管理和常用工具的集成。它所具有的可扩展性，使得可以很方便地增加对新的视频转换工具和协作方法的支持。集成框架如图 4-1 的结构：

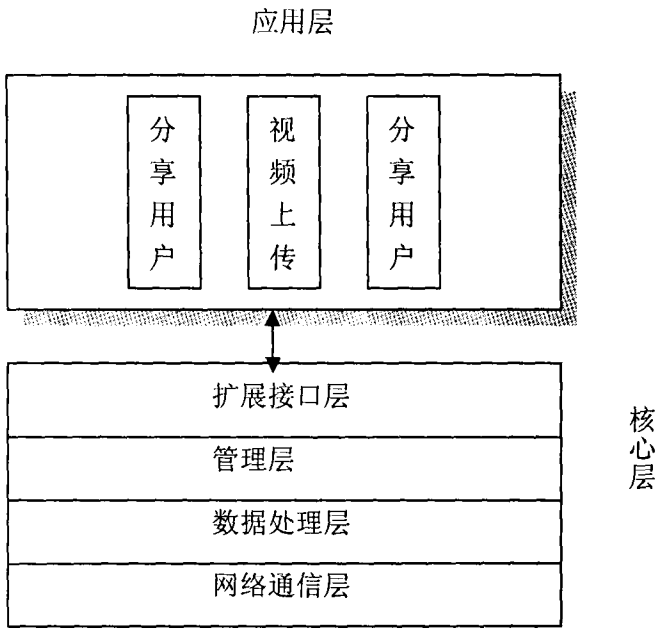


图 4-1 服务器端体系结构

Fig 4-1 Architecture of the server

4.1.1 核心层

网络通信层不仅仅为用户提供了通用的网络通信服务，如 TCP、UDP 等，还允许用户对此进行扩展以满足不同的网络通信需求。根据功能的不同可将协议分为两类：

传输协议，用来传送原始的分享数据，如视频文件数据、音频文件数据等，如 TCP 协议、UDP 协议，这是必需的；

算法协议，对接收的数据进行进一步的处理，如排序，分发整理等。可以根据 客户端发送数据的不同具体处理。数据处理层主要对系统接受的数据进行处理。系统中的数据

处理主要分为 2 类：A、文件数据处理，视频格式转换压缩；B、系列数据即将用户请求分享的数据的其他属性，如视频文件名、视频性质等等进行处理，设置当前分享数据的状态。管理层主要对客户端上传的连接和数据增加进行管理。扩展接口层提供客户端数据传输的可扩展接口和客户端使用 IE 请求连接的并上传数据的接口。

4.1.2 应用层

这层应用提供了一组通用功能，可协助提高用户在登录网站分享视频的时候的用户操作体验，例如视频文件上传进度条提示、视频标签的设定等。

4.2 系统服务中 CSCW 的层次模型

视频转换系统是一个分布式的系统，但又不同于传统的分布式系统，要建立一个通用、高效的 CSCW 系统结构是非常困难的，目前只存在针对具体应用背景分析和设计的 CSCW 系统。现在对协同工作模型的研究主要基于视频文件处理系统和网站数据协同编辑系统。下面讨论一种可行的 CSCW 系统服务的标准化模型设想。

4.2.1 系统的层次结构模型

基于 CSCW 系统的层次结构模型是从 OSI 层次模型的角度出发[32][33][34]，根据 CSCW 信息模型理论建立在多点通信基础上的，其结构示意图如下：(图 4-2)

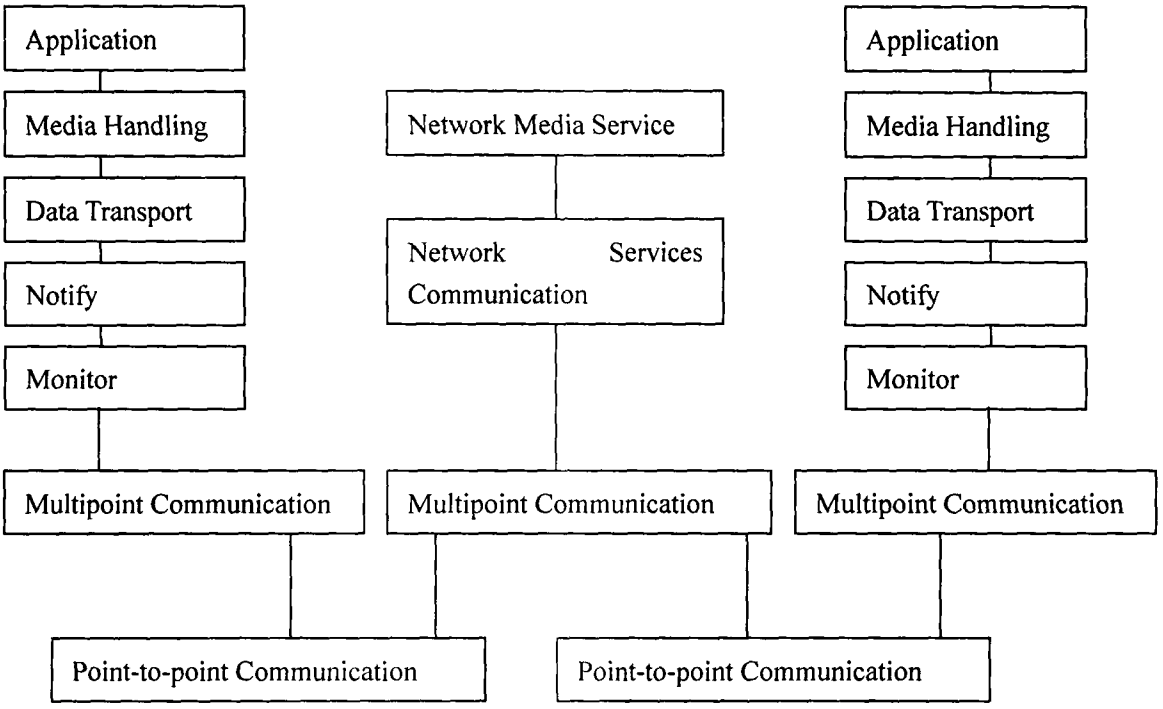


图 4-2 CSCW 层次结构模型

Fig4-2 Layer Structure of CSCW

模型中的大多数层都属于 OSI/RM 的第七层，而模型中的最低层（点对点的通信服务层）的功能包含了 OSI 模型的最低三层（物理层、数据链路层、网络层）；其中本模型的

第二层（多点通信层）提供服务与OSI 模型的传输层、会晤层和表示层基本一致。比如，在多点通信中，连接管理（传输层服务）和同步（会晤层）将在同一层中相互作用，而不能明确分开。然而，因为 OSI 主要考虑点对点通信，因此它把多点通信放在了应用层（Application layer)。

所以，除了点对点通信层，上述 CSCW 系统的分层结构模型可以看作 OSI 应用层的子层。

4.2.2 层次结构模型的特点

上述的 CSCW 的分层结构系统模型[35][36]，其优点在于：

CSCW 应用如果按这种方式结构，那么设计的软件模块就可以在不同的产品 中重复使用，所有应用层以下的模块将是通用的和可移植的，这对 CSCW 系统模型的标准化是非常有意义的。协同工作中的几个关键方面都可以用模型中的不同层次来表示，使得在合适的层间可以交换数据（因为各层之间设定了清晰的界面)。例如，两个不同的多点会议系统可能提供完全不同的特征集，但是这并不影响它们各自互换文件、图像等。对某些应用来说，某些层可能是空的。另一方面清楚知道各层功能应用的用户可以给他们的产品增加不同的功能，例如，如果网络提供了以上模型的服务支持，传真机生产厂家想要建立实用的标准多点协议，那么他们可以使用网络服务来完成多拷贝的分发。

该模型可以提高现有或将来系统性能，利用现有的 CSCW 系统的某些模块，匹配到该模型的某层中，再增加其它层的服务，那么，这样的产品既能符合兼容性，又能有新的功能。例如存储——前向转发、媒体转化和目录功能等都不是新概念，该模型为它们提供了框架。

本模型建立在OSI 七层模型基础上，是基于OSI 分层清晰和可以利用目录管理系统服务等考虑，但是目前OSI 分层结构不能支持多路通信、以及同步等远程协作应用必须具备的通信机制，因此必须在应用层内加入适当的功能来弥补。本论文将在第五章对可靠通信进行详细研究。

为了建立易于标准化 CSCW 模型系统，应该从目前网络发展的情况来分析，面向对象的信息共享对于网络发展前景来说，具有更大的优点，如：

①系统模型易于构造，清晰明了；

②发送者和接收者处于同等地位，而无主次之分，清除了用户被动应答的心理烦恼，这也是远程协作与心理学、社会学相关的一种体现；

③信息模型可建成统一的实现模式，易于标准化，为将来实现全球性的协作通信提供了基础；

④分层结构模型是信息共享模型的一种，它考虑 CSCW 系统按时空矩阵划分的四种情况。

4.2.3 交互、活动、协作三层结构的协作模型

本节在层次模型的基础上研究了一种三层结构的 CSCW 系统的协作模型[37][38]，它是在上述层次模型理论基础上的实现模式。其结构如图 4-3 所示

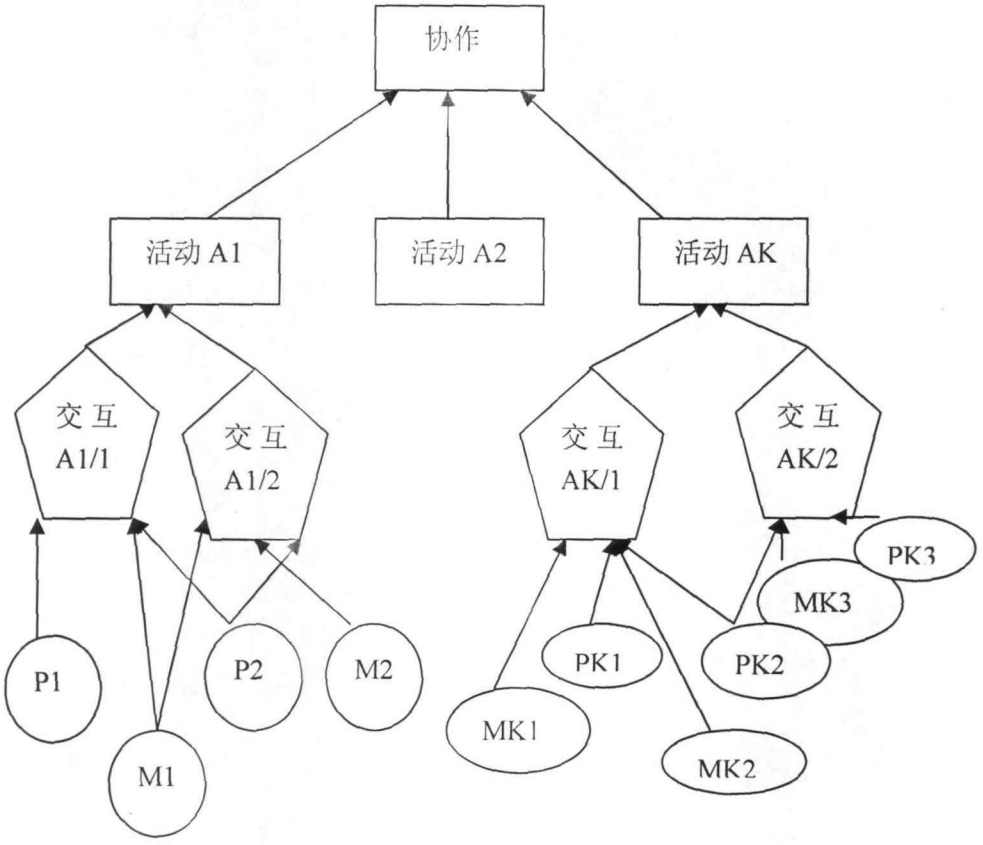


图 4-3 协作模型的三层结构

Fig 4-3 three layers of the cooperation model

在该模型中包括交互、活动、协作三层，其中的交互、活动、协作分别指：

交互——指需协作的成员之间对通信媒体的使用。

活动——指的是由若干个存在关联的交互所组成的集合。

协作——是指由一系列时间上连续的相互之间存在输入/输出依赖关系或因果依赖关系的活动所组成的集合。

4.3 处理多客户端请求

服务环境集成框架作为客户机/服务结构系统的服务器方，必须能够处理多个客户同时向服务器发出请求的情况。这些请求，既可以是对 CSCW 系统的管理指令，也可以是用户之间的数据或状态、属性和视频传送。

该系统在整个数据处理和分享的过程在进程和线程处理上分为两种，第一种是采集到用户的视频数据后，服务器端的集成框架内，对视频文件进行转换和压缩时，进城和线程的管理；第二种是扩展接口层即网站发布的 IIS 中，用户并发时，进程和线程的管理。而第二种我们将放在本章的第三节连接方案选择中讨论，本节将重点对 FFMPEG 转换视频时对进程和线程管理进行探讨。

在实现该系统时，我们认为服务器中文件转换的请求处理模块可在三种方法之间进行选择：单进程单线程的方法、多进程多线程和单进程多线程的方法。

4.3.1 进程和线程

进程（Process）是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的一个独立单位。程序只是一组指令的有序集合，它本身没有任何运行的含义，只是一个静态实体。而进程则不同，它是程序在某个数据集上的执行，是一个动态实体。它因创建而产生，因调度而运行，因等待资源或事件而被处于等待状态，因完成任务而被撤消，反映了一个程序在一定的数据集上运行的全部动态过程。

线程（Thread）是进程的一个实体，是 CPU 调度和分派的基本单位。线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

线程和进程的关系是：线程是属于进程的，线程运行在进程空间内，同一进程所产生的线程共享同一内存空间，当进程退出时该进程所产生的线程都会被强制退出并清除。线程可与属于同一进程的其它线程共享进程所拥有的全部资源，但是其本身基本上不拥有系统资源，只拥有一点在运行中必不可少的信息(如程序计数器、一组寄存器和栈)。

80 年代后期大多数操作系统中还没有线程的概念（即一个进程中只允许一个线程的执行）。事实上，大多数操作系统以进程一词表示可执行实体，而线程是一个较新的词。由于每个进程都有独立的地址空间，两个进程间要进行通信就必须建立共享存储区或共享文件。

采用两个进程来实现并行性并不总是有效。以 UNIX 为例，当一个进程创建另一个进程时，系统必须将第一个进程地址空间内的所有内容拷贝到新进程的地址空间中。对于一个大地址空间，这种操作是很费时间的。更何况两个进程还必须建立一种共享数据的方式。这在某些操作系统中可以简单快速地实现，而在另一些操作系统中则并非如此。

在现代的操作系统中，人们提出了一种可以实现并行性的有效的途径——多线程进程(multithread processes) 。

一个线程是程序的一个执行单元，是进程内的一个可调度实体。它表示了一个独立的执行单位。正如进程在逻辑上表示了操作系统所必须完成的作业一样，线程表示完成该进

程的许多可能的子任务之一。线程驻留于进程的地址空间中，在线程执行时将使用进程的地址空间进行存储。若同一进程中存在多个线程，则它们共享地址空间和所有的进程资源。

利用多线程进程实现并行性避免了用两个进程实现并行性的缺点。线程只需很少的附加开销，并且线程的创建也比进程要快(由于这个原因，有时线程也称为“轻型进程”)。同时，由于进程中的所有线程除栈和寄存器内容之外，共享同一内存，故不再需要特殊的数据传送机制。一个线程只需简单地将输出写入内存，另一线程就可以读出作为输入。这使线程之间的通讯简单、快速。

编写多线程应用程序时必须特别小心，这是因为进程中所有线程对进程的整个地址空间都有存取权限。如果不加以限制，多个线程对共享的数据结构进行存取时，有可能引起数据结构的不一致性；当多个线程调用公用函数时，也会带来重入性问题：即当线程 A 调用函数 F 正在执行时，时间片恰好用完，操作系统调度线程 B 执行，而 B 也要调用函数 F，如果函数 F 使用了静态变量，函数 F 的这次执行就会破坏线程 A 中函数 F 的执行现场，使线程 A 不能正确执行下去。

上述问题可以使用操作系统提供的信号量、互斥量等同步机制来加以解决。

对于用多个进程实现并行性并采用消息、共享内存或管道等显式方式进行通信的应用程序，则不会产生上述问题。一个进程不可能有意无意地破坏另一进程的地址空间。每个进程对自己的私用地址空间保持控制权。

采用多进程或一个进程内的多个线程来实现并行性都是很有用的技术。应用程序的目标决定了对任一特殊程序，用哪种结构更有效。目前流行的各个操作系统，如 UNIX, Windows NT, Windows XP 等均同时支持进程和线程。在服务器内，采用多个线程来运行并共享同一地址空间和资源一般来说是有利的。

进程和线程作为操作系统中的执行单位，从逻辑上看它们是非常相似的。下面仅以单进程单线程方法、多进程方法和单进程多线程为例，论述服务器的结构。

4.3.2 单进程方法

在这种方法中，服务器系统只有一个进程。每当客户机的一个请求到来时，服务器上的这个进程立刻处理这个请求，处理完毕后，送回执行结果。依照这种方式不断循环。服务器结构流程图如图 4-3 所示。

这种结构非常简单，也很容易实现。但是，有一个问题不能忽视：如何及时接收请求？对这一问题有两种实现：同步方式和异步方式，不同的选择会影响服务器的整体结构。

1. 同步方式。所谓同步方式，就是在进行网络读写操作时，进程将阻塞，直到读写完成，才能继续执行下去。它使进程严格地顺序执行，不会产生资源冲突等复杂问题。对

于同步通讯方式，可以有两种方法实现网络通信：

阻塞方式。这是最简单的网络通讯方式，只要调用“读取”功能，则进程阻塞，直到读到数据。采用这种方式的“接收请求”模块可用伪码表示如下：

```
for(每一个请求)
{
Read(客户请求);
Call 请求处理模块;
}
```

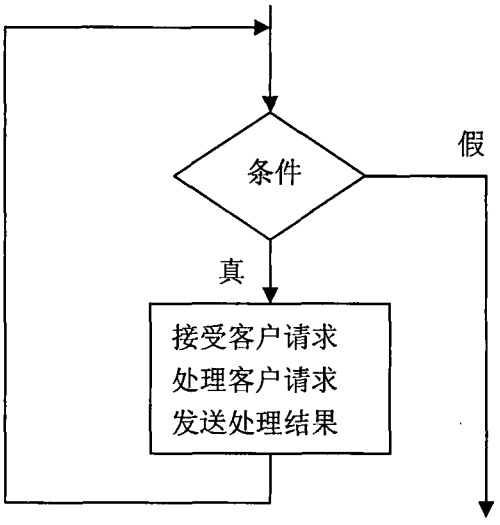


图 4-4 单进程

Fig 4-4 Single process

这种方式对单进程的服务器软件是行不通的。因为服务器只有一个进程，却要处理多个转换的请求，不能因为等待一个视频转换产生请求就放弃了对其它视频转换请求的处理。但在多进程的服务器中，如果让一个进程只处理一个视频转换的请求，这种阻塞读方式却是非常简洁、高效的。

查询方式。即服务器检查一个视频分享数据是否有请求到达，如果没有，则转而检查下一条数据。这种方式的“接收视频转换请求”模块可用伪码表示如下：

```
for(每一个视频转换数据)
{
if(有视频转换请求到达)
    Call 请求处理模块;
}
```

这种方式使单进程的服务器可以处理多个客户机发送的请求，并且结构十分简单，可以适用于一般的系统。

但是，它有一个很大的缺点，即“忙等”现象。即使当前没有视频分享数据向服务器发送请求，服务器也在占用CPU资源对所有的客户进行不停的轮询。这种低效的处理方法会影响操作系统中其它进程的正常处理。如果不进行改进采用其它的进程协调措施，这种结构是无法实用的。

2. 异步方式。这种方式常用于使用事件驱动机制的系统中，程序通过调用一个立即返回的函数启动一个操作。当程序调用一个异步函数时，操作不能立即完成也不会引起错误。异步操作调用只对启动操作负责，它不等待操作完成。操作系统监视此操作的完成，并发送一个消息报告异步操作的成功或失败；程序也可以设置网络模块在网络可以进行读或写时，向进程发送事件，程序就可以在事件处理函数中进行读写操作。

这种方式使服务器进程摆脱了同步方式中的“忙等”现象，明显提高了效率。但是，因为它使进程不一定是顺序执行，也带来了一定的实现的复杂性，实现的思路也与同步方式的明显不同。采用这种方式的单进程服务器的结构可用伪码表示如下：

```
假设对“视频数据转换请求连接”事件的处理函数是“视频转换处理函数（）”;
while(有新的视频数据产生)
{
    把视频数据送给相应的处理函数;
    视频转换处理函数（）
    为该条视频数据产生一新的视频转换请求处理函数的实例;
设置该视频数据的“转换请求”事件的处理函数是该视频转换处理函数（）;
}
视频转换处理函数（）
{
     读取视频分享数据的视频转换请求;
     处理请求;
     送回结果;
更改视频数据状态;
}
```

可以看出，使用异步方式的服务器虽然功能类似于同步查询方式的服务器，但程序结构明显不同。它的大部分代码都集中在各个事件的处理函数中。

采用这种结构的服务器既能有效地处理多个客户的请求，并且避免了“忙等”现象。这种服务器结构比多进程的要简单得多，所以可以用于很多系统。

但是，因为整个服务器只有一个进程，它对多个客户请求的处理仍然是顺序的，是先来先服务的方式，而不是并发的，类似于批处理操作系统中对用户作业的处理。这就使处理粒度比较大，如果服务器正在处理一个需要时间较长的客户请求，就会使其它客户请求的响应时间明显加长。

这种服务器比较适用于对客户请求的处理时间不是很长的系统中。

4.3.3 多进程方法

和上述单进程方法相对应的就是多进程方法。多进程服务器中同时存在多个进程，以处理多个视频数据的不同视频转换请求。

当用户提交一条视频分享数据时，集成在服务器端的服务会立即侦测到，然后就会产生出一个新进程。这个进程始终在系统中运行，负责处理视频数据的视频转换、压缩、截图等请求，直至视频分享数据处理完毕，服务更改了视频数据的状态时这个进程才结束。

当视频文件本身发生意外（如用户上传的视频文件损坏、或格式不正确）时，服务器必须能够及时响应，终止处理视频转换请求的那个进程的运行，以释放该进程所占用的系统资源。因此，服务器上的调用 FFMPEG 转换视频的模块往往采用面向连接的服务方式以便及时检测到这种事件的出现。也可以让集成在服务器端的系统服务模块定时互相发送一些信息，以检测对方的状态。

在多进程服务器上，有两种进程。

第一种进程的代码可以简写如下：

```
while (TRUE)
{
从服务侦测接收新的视频转换请求;
if(系统资源允许)
   产生出一个新进程;
}
```

产生出的新进程(第二种进程)的代码可以简写如下：

```
{
    while (TRUE)
{
```

```
        接收视频转换请求;
        处理视频转换请求;
        发送处理结果;
    更改视频数据状态;
    }
    }
```

可以看出，采用这种结构的服务器，使用传统的同步和异步结合的通讯方式是非常合适的。每个进程只处理一条视频数据的请求，如果用“阻塞”式侦测转换请求的方式，在视频数据请求时，进程将被阻塞，这样，它就不占用系统的CPU资源，不会影响其它进程的执行。并且这种程序结构非常简单，所以它适用于很多场合。

多进程的服务器和单进程的服务器的区别在于多进程的服务器对多个视频转换请求的处理是并发的，类似于分时系统中对用户作业的处理。这样，即使某个客户的一个请求要占用很长的时间，处理其它客户请求的进程仍将继续响应各自客户的请求。这种服务器对客户请求的处理粒度等于操作系统调度各个进程的时间片的大小。

4.3.4 多线程方法

当用户提交一条视频分享数据时，集成在服务器端的服务会立即侦测到，服务器将产生出一个新线程如图4-3所示。这个线程始终在系统中运行，负责处理该条视频数据的转换请求，直至文件处理完毕，服务更改视频数据的状态这个线程才结束。

采用这种方式的多线程服务器的结构可用伪码表示如下：

```
while (TRUE)
{
从服务侦测接收新的视频转换请求;
if(系统资源允许)
    产生出一个新线程;
    加入线程池;
    根据线程优先级处理视频转换请求;
}
```

多线程技术使程序的响应速度更快，因为用户界面可以在进行其他工作的同时一直处于活动状态，当前没有进行处理的任务可以将处理器时间让给其他任务，占用大量处理时间的任务可以定期将处理器时间让给其他任务，可以随时停止任务，可以分别设置各个任务的优先级以优化性能。在以下情况下，最适合采用多线程处理：耗时或大量占用处理器

的任务阻塞用户界面操作，各个任务必须等待外部资源（如远程文件或 Internet 连接）。

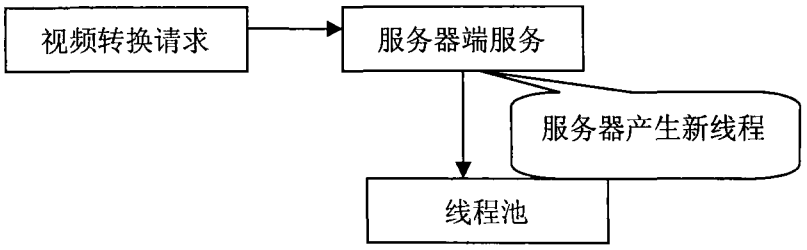


图 4-5 线程图

Fig 4-5 Thread Chart

4.4 连接方案选择

在基于 FFMPEG 的视频转换系统中，视频分享数据采集是通过 Web 机制，互联网用户通过 IE，将自己的视频文件发送上来，每个视频文件都有特定的名称、标签、关键字等。并且，服务器端系统的整个工作过程中，各个服务处理层要经常性的接收用户的视频文件数据和视频数据中对视频转换、截图、压缩的请求，以完成各种协同工作。这样，就存在一个服务器如何与用户连接的问题。考虑到网络通信的兼容性和可靠性，视频文件数据和其他数据等应该采用 TCP/IP 协议的面向连接的 TCP 协议进行通讯。这样有以下两种方案：

4.4.1 建立一条连接

服务器与用户建立一条连接如图 4-6 所示，所有的用户都使用这条连接进行通讯。这样所需的 TCP 连接的数量较少，占用的系统的网络资源也较少。

但是，因为是互联网是一个开放型平台，不可能每次只允许一个用户连接，因此使用一个连接是不可取的；而服务器作为接收方必须有一个分发器，把收到的数据分发到相对应的转换工具、数据系列整理等各个工具，在实现上增加了一定的复杂性，降低了系统响应速度，也大大降低了效率。

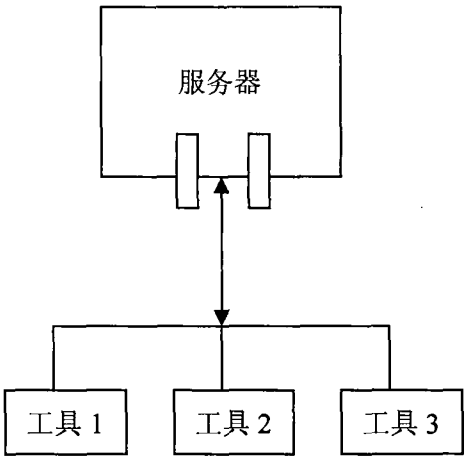


图 4-6 单连接

Fig 4-6 Single Connection

4.4.2 建立多条连接

服务器与互联网用户终端的每一个用户都建立一条连接如图 5-5 所示，不同视频分享数据分别从不同的 TCP 连接传送。这样，编程比较简单，提高了系统的响应速度和并发数，满足了互联网开放性的需求，很大的提高了传输效率，所以如果强调系统的实时性的话，这种方式是最佳的选择。但是，这种方案使用了较多的 TCP 连接，占用的系统的网络资源也较多。

在视频分享过程中，各个客户端之间分享的数据是视频，因此数据传送量较大，要求一条视频分享数据的损坏不能影响其它的分享数据分发，这一点不同于客户端集成框架中以客户端设计为主、客户机之间协同为辅的情况，因此在框架中，我们采用了与客户机终端建立多条连接的方案以增加系统的可靠性。

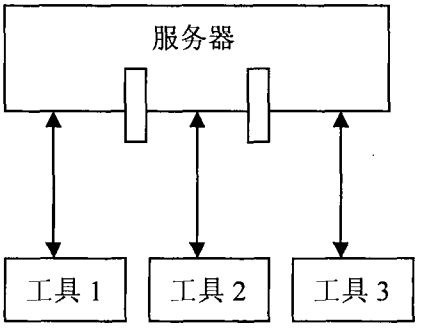


图 4-7 多连接

Fig 4-7 Muti connection

4.5 框架的可扩展性

开放性系统应该可以与其它系统或应用相连，特别是各种协同工具，使它们可同时运行，相互通信和交互数据。当互联网终端增加了新的视频格式或新的转换压缩时，应该能很方便地在服务环境框架中增加对它的支持。另外，因为服务器软件与用户界面的关联较

少，只强调其功能性和可靠性、安全性，比较容易也应该具有较好的可移植性、扩展性和高可用性，也可以很快增加除 FFMPEG 以外的视频转换工具。

4.6 本章小结

提出了用于服务环境的一种集成框架的结构。该结构可以支持异步 CSCW 系统的服务器软件的快速开发提供了开放性的支持，可以容易地在系统中增加对新的视频转换工具的支持。

为了完成处理多用户请求的任务，本文深入探讨了服务器如何处理多个客户机的请求的处理方法，提出了单进程方法、多进程和多线程方法三种方法，分析了这三种方法的优缺点和适用的场合，最后分析了互联网终端和服务器的连接方式，并最后选择了多连接方式作为我们系统的通信方式。

第 5 章 各组成的系统模块设计和实现

5.1 开发环境和设计目标

5.1.1 开发环境

本文的实现设计是采用的系统平台和开发工具如下：

操作系统：Microsoft Windows 2003 Server

开发工具：Microsoft Visual Studio 2005

数据库： Microsoft SQL Server 2000 + Sp4

5.1.2 设计目标

根据目前国内外在 CSCW 方面发展状况、新理论和遇到的一些问题，结合当前软件发展的新动向、新技术和网络环境的差别，根据实际需求，采用成熟的技术方案以确保系统的可用性、高响应性和扩展性，使用质量优异、先进的技术手段，以降低系统总体开发成本和提高可靠性；同时以系统满足不断增长的系统需要而具备的可持续升级能力为技术手段，保证以最小的投入取得最佳效果。具体来说，要从以下几个方面进行考虑：

服务器系统要采用可扩展的框架：为此服务器端也采用模块化的设计。首先 对现有的协同功能进行分析和归类，按需要进行不同的处理方式分为几类，每一类开发一个服务器模块。在服务器处理时不以具体功能来分，而是以不同功能类型来分开处理。比如有些模块可能都只需要访问数据库，则用同样的数据库访问模块来处理。

可靠性：指本系统能可靠、稳定工作，并在发生异常事故的情况下能顺速的 恢复到正常状态继续运转，系统易于维护，有详细的日常运行日志报告，能随时发现错误，能在适度范围内进行 bug 的修正。

服务器客户端结合性：服务器要结合客户端设计，采取一些措施来减轻客户端对服务器的压力。一方面增强客户端请求本地化处理的能力，如优化与客户端本地存储模块的同步，同时在服务器的辅助下加强客户端对数据处理的能力。

安全性：采用各种安全措施来保证服务器系统及其数据的安全，如提供数据库访问模块来隔离数据库，对传输数据进行加密等等。

5.2 系统模块

5.2.1 系统构件图

Service Server 是集成环境视频转换服务和视频分享数据管理的核心，用于实现互联网视频数据提交，服务器环境对视频文件进行转换和视频分享数据的系列化整理。如 5-1 图示，Service Server 由视频分享网站的数据连接器、数据管理器、视频文件数据处理模块和异常处理模块，各组件的具体结构和功能如下：

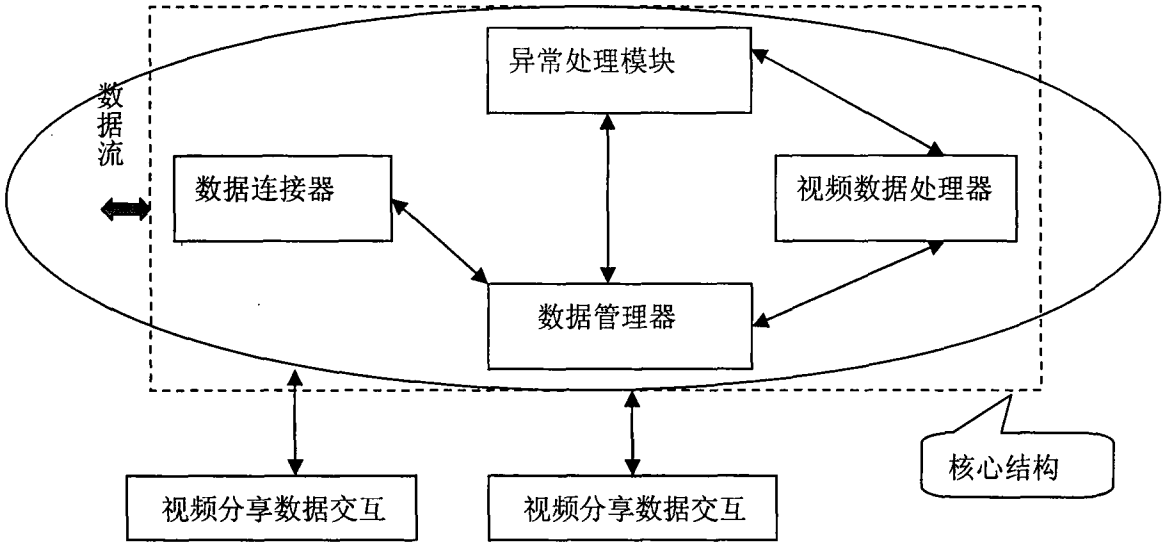


图 5-1 Service Server 构件图

Fig5-1 Component of Service Server

5.2.2 数据连接器

数据连接器即网站的视频分享入口，是 Service Server 和 Client 之间的物理连接，用于接收和发送视频分享数据。一方面 IIS 服务器监听网络，接收 Client 发来的视频数据，然后数据管理器放入对应的文件库和 DB 数据库；另一方面，来自互联网的 Client 发送请求分享读取视频数据，数据管理器取出对应数据并发送给 Client。数据连接管理器包括连接对象、文件传输管理器、线程池管理器三个部分。其中，连接对象用于与 Client 建立 web 连接，在 Client 和数据管理器中传递数据。建立连接时使用的端口号由 IIS 管理器管理。连接对象以多线程方式运行，有效提高接收和发送数据的效率。线程在线程池管理器中进行维护，通过 IIS 中的链接池配置可以改变线程池中线程数。

文件传输管理器，则是在接收用户上传文件的请求后，根据目前 IIS 并发的线程设置，确定是否立即响应请求，当 Server 资源达到设定的上限时，将请求放入等待队列。

5.2.3 数据管理器

数据管理器实现视频分享数据的接收、存储和数据状态的动态管理；支持多线程的数据传输，提供将用户的视频等文件放入文件存储设备，同时动态管理视频分享数据的状态。数据连接器根据数据的状态来分发分享数据；数据管理器动态的将原始数据状态设定为视频文件处理的请求，发送给视频数据处理器。文件经过视频数据处理器处理之后，数据管理器更改数据状态，形成可以在互联网传播的视频分享数据。

5.2.4 视频数据处理器

视频数据处理器一方面取数据管理器中的视频文件处理请求队列中的视频文件数据进行转换压缩，再根据视频文件类型调用 FFMPEG 组件进行视频文件处理，完成后将处理结果传送数据管理器；另一方面，为了增加执行效率，开展多进程多线程转换，根据视频处理服务器的系统资源，同时取几条视频请求进行处理，每条请求处理完毕后收回线程。

同时对于不同视频类型有不同的处理过程，需根据实际情况调用 FFMPEG 的不同组件。下面以实际应用为例，简单叙述处理过程：视频数据理器根据数据管理器中的文件处理队列，首先查询服务器空闲资源，若全部空闲，则接收系统可承受的进程请求；如视频数据处理器同时开始处理三条转换需求，例如第一条的请求文件是 FLV 的，则视频数据处理器调用 FFMPEG 的压缩组件，仅对视频进行压缩处理；第二条为 AVI 的，则视频数据处理器调用 FFMPEG 的转换压缩组件对视频进行处理；第三条为 RMVB 的，同时有截图的请求，则调用 FFMPEG 的转换压缩截图组件对数据进行处理。每个线程处理完毕后，回收相应的系统资源，同时将数据结果返回给数据管理器。具体处理如 5-2 图示：

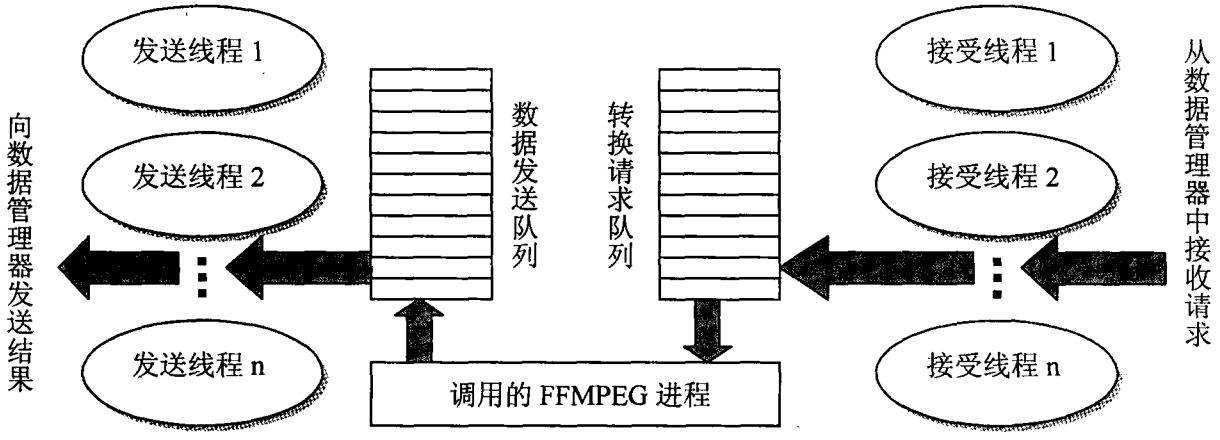


图 5-2 多线程视频处理机制

Fig5-2 Multithread Video handles mechanism

5.2.5 异常处理模块

异常处理模块主要实现检测和测试 Service Server 运行过程中的状态；跟踪转换过程中的文件完整性；向数据处理中心发送对报警处理记录；记录 Service Server 中文件处理的日志。该模块和视频处理器、数据管理器的协作如 5-3 的图示：

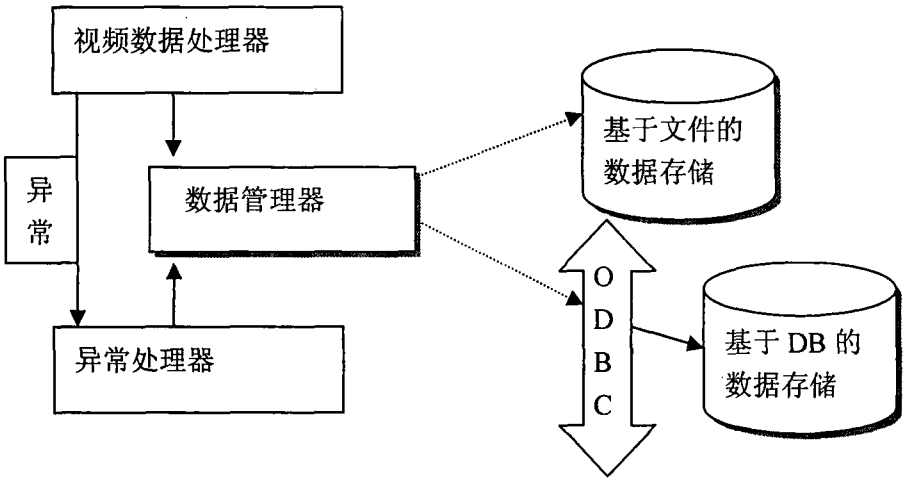


图 5-3 协作方式

Fig 5-3 the storage method of cooperation

5.3 系统实现

本文开发了一个简单的基于 FFMPEG 视频转换的服务环境，在本节中给出了系统实现的一部分关键代码。

5.3.1 数据连接器

数据连接器是连接互联网客户端的核心模块，其主要的运行流程如下：

A、通过 IIS 网站的发布，接收客户端的传送的视频文件数据。

B、分发视频数据。

C、IIS 线程池的管理[39]。

数据连接器主要提供了大文件上传的网络通信方式，及时捕捉上传进度，给互联网用户以好的用户体验，其中的操作由具体协议如 TCP 和 UDP 等实现，其主要的代码如下[40]：

```
///客户端的传输进度条
var r = "传输: {0}K 还未完成";
        var s = "您的文件已经上传完成";
        var t = "上传失败";
        function progressBar()
        {
            this.totalSize = 100;
            this.sizeCompleted = 0;
```

```
this.percentDone = "0%";
this.setSize = function(totalSize, size)
{
    var oProgressInfo = document.getElementById("progressInfo");
    var oProgress = document.getElementById("progress");
    if (oProgress == null || oProgressInfo == null)
        return;

    if (totalSize <= 0)
        return;

    this.totalSize = totalSize;
    this.sizeCompleted = size;
    if (size < 0)
        this.sizeCompleted = 0;
    else if (size > this.totalSize)
        this.sizeCompleted = this.totalSize;

    var sizeLeft = 0;
    var progressInfoText = "";
    sizeLeft = this.totalSize - this.sizeCompleted;

    this.percentDone = Math.round(size / this.totalSize * 100) + "%";
    oProgress.style.width = this.percentDone;

    if (sizeLeft > 0)
        progressInfoText = r.replace("{0}", sizeLeft);
    else
        progressInfoText = s;

    oProgressInfo.innerHTML = progressInfoText;
}
this.UploadError = function()
{
    var oProgressInfo = document.getElementById("progressInfo");
    var oProgress = document.getElementById("progress");
    if (oProgressInfo != null)
        oProgressInfo.innerHTML = t;
    if (oProgress != null)
        oProgress.style.width = "0";
}
this.UploadComplete = function()
{
    var oProgressInfo = document.getElementById("progressInfo");
    var oProgress = document.getElementById("progress");
```

```
            if (oProgressInfo != null)
                oProgressInfo.innerHTML = s;
            if (oProgress != null)
                oProgress.style.width = "100%";
        }
    }
    var iTimerID = null;
    var xmlHttp = XmlHttpPool.pick();
    var url = "progress.aspx?UploadID=<%=Request.QueryString["UploadID"]%>"
    var pb = new progressBar();
    function LoadProgressInfo()
    {
        try
        {
            xmlHttp.open("GET", url + "&t=" + Math.random(), true);
            xmlHttp.send(null);
            xmlHttp.onreadystatechange = function()
            {
                LoadData(xmlHttp);
            }
        }
        catch(e)
        {
            alert(e)
        }
    }
    function LoadData(xmlhttp)
    {
        if (xmlhttp.readyState == 4)
        {
            iTimerID = window.setTimeout("LoadProgressInfo()", 500);
            try{
                eval(xmlhttp.responseText);
            }
            catch(e)
            {
                alert(e)
            }
        }
    }
    function ClearTimer()
    {
        if (iTimerID != null)
        {
            window.clearTimeout(iTimerID);
```

```
                iTimerID = null;
            }
        }
        function UploadCancel()
        {
            location.href = location.href;
        }
```

服务器端的数据接收[41]：

```
protected void Page_Load(object sender, EventArgs e)
    {
        string uploadId = Request.QueryString["UploadID"];
        string scriptText = "";
        string scriptUploading = "pb.setSize({0}, {1});";
        string scriptClearTimer = "ClearTimer();";
        string scriptUploadComplete = "pb.UploadComplete();" + scriptClearTimer;
        string scriptUploadError = "pb.UploadError();";

        string length = "";
        string read = "";

        Openlab.Web.Upload.Progress progress = HttpUploadModule.GetProgress(uploadId,
Application);
        if (progress != null)
        {
            // 如果正在接收数据，利用脚本来通知前端进度条
            //
            if (progress.State == UploadState.ReceivingData)
            {
                length = (progress.ContentLength / 1024).ToString();
                read = (progress.BytesRead / 1024).ToString();
                scriptText = string.Format(scriptUploading, length, read);
            }
            else if (progress.State == UploadState.Complete)
            {
                scriptText = scriptUploadComplete;
            }
            else
            {
                scriptText = scriptUploadError;
            }
        }
        else
        {
            //scriptText = scriptUploadError;
```

```
            }
            Response.Clear();
            Response.Write(scriptText);
            Response.End();
        }
```

数据的接收：

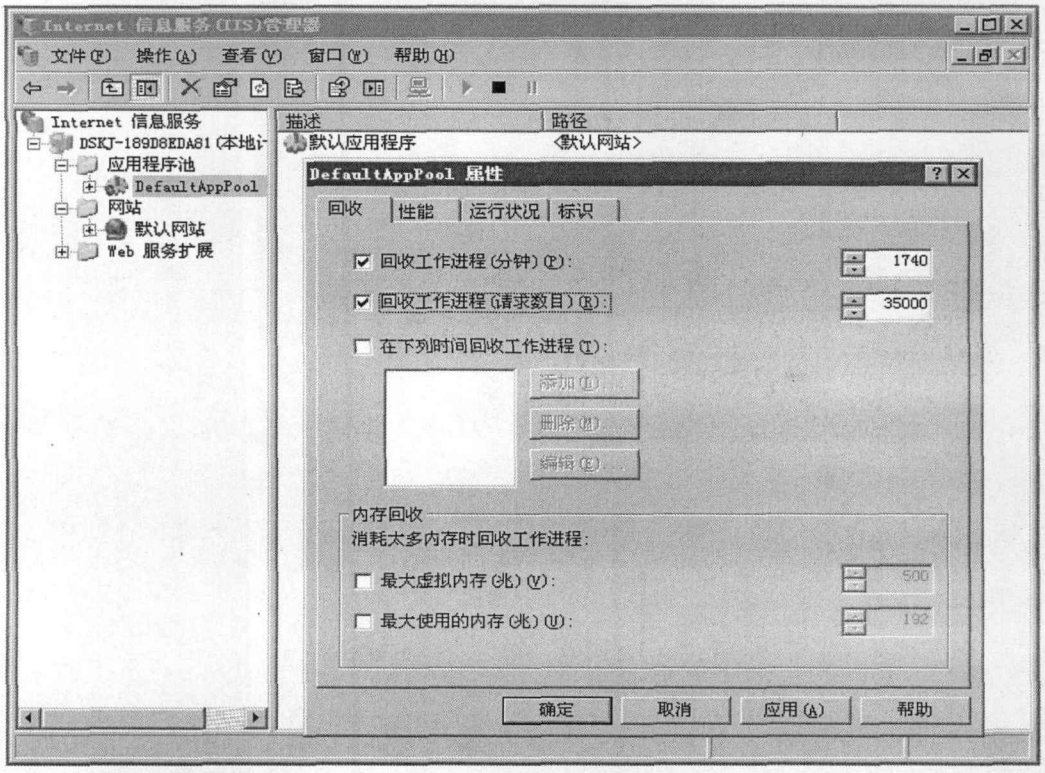
```
protected void Page_Load(object sender, EventArgs e)
        {
            string uploadId = Request.QueryString["UploadID"];
            string scriptText = "";
            string scriptUploading = "pb.setSize({0}, {1});";
            string scriptClearTimer = "ClearTimer();";
            string scriptUploadComplete = "pb.UploadComplete();" + scriptClearTimer;
            string scriptUploadError = "pb.UploadError();";

            string length = "";
            string read = "";

            Openlab.Web.Upload.Progress progress = HttpUploadModule.GetProgress(uploadId,
Application);
            if (progress != null)
            {
                // 如果正在接收数据，利用脚本来通知前端进度条
                if (progress.State == UploadState.ReceivingData)
                {
                    length = (progress.ContentLength / 1024).ToString();
                    read = (progress.BytesRead / 1024).ToString();
                    scriptText = string.Format(scriptUploading, length, read);
                }
                else if (progress.State == UploadState.Complete)
                {
                    scriptText = scriptUploadComplete;
                }
                else
                {
                    scriptText = scriptUploadError;
                }
            }
            else
            {
                scriptText = scriptUploadError;
            }
            Response.Clear();
            Response.Write(scriptText);
```

```
            Response.End();
        }
```

IIS 对线程的管理主要通过起配置实现：



5.3.2 数据管理器

数据管理器是核心的组件，负责使用和协调其它组件完成功能集，因此数据管理器与其它组件的关联比较多。视频数据的新增、视频分享数据的状态管理等都是由此组件完成。同时它实现了文件数据的存储和基于 DB 的数据存储。

文件数据存储代码如下：

```
        protected void btnUpload_Click(object sender, EventArgs e)
        {
            int groupId = 0;
            if (Request.QueryString["groupId"] != null && Request.QueryString["groupId"].ToString() !=
"")
            {
                groupId = int.Parse(Request.QueryString["groupId"].ToString());
            }
            string picUrl = "20060105.gif";
            if (this.myPic.PostedFile.FileName != "")
            {
                string filePath = myPic.PostedFile.FileName;
                string fileType = filePath.Substring(filePath.LastIndexOf(".") + 1);
                fileType = fileType.ToLower();
                if (fileType != "jpg" && fileType != "jpeg" && fileType != "gif" && fileType != "png"
&& fileType != "bmp")
```

```
                {
                    this.labPic.Visible = true;
                    this.labPic.Text = "*上传的文件超过了规定,请选择png或者gif和jpg格式的文件!";
                    this.labPic.ForeColor = System.Drawing.Color.Red;
                    return;
                }
                else
                {
                    this.labPic.Text = "*可不选";
                }
                picUrl = this.Session_UserID.ToString() + DateTime.Now.Year.ToString() +
DateTime.Now.Month.ToString() + DateTime.Now.Day.ToString() + DateTime.Now.Hour.ToString() +
                    DateTime.Now.Minute.ToString() + DateTime.Now.Second.ToString() + "." + fileType;
                if (myPic.PostedFile.ContentLength > 204800000)//判断图片是否大于k
                {
                    this.labPic.Visible = true;
                    this.labPic.Text = "*你上传的图片太大，请转换后上传";
                    this.labPic.ForeColor = System.Drawing.Color.Red;
                    return;
                }
myPic.PostedFile.SaveAs(ConfigurationManager.ConnectionStrings["FileServer"].ConnectionString +
ConfigurationManager.ConnectionStrings["ImagePath"].ConnectionString + picUrl);//把文件保存在此路径
中
                myPic.Dispose();//////////////////释放掉占用的内存
            }

            if (this.uploadFile.PostedFile.FileName != "")
            {
                string filePath = uploadFile.PostedFile.FileName;
                string fileType = filePath.Substring(filePath.LastIndexOf(".") + 1);
                fileType = fileType.ToLower();
                if (fileType != "wmv" && fileType != "asf" && fileType != "asx" && fileType != "rm" &&
fileType != "rmvb" && fileType != "mpg" && fileType != "mpeg" && fileType != "mpe"
                    && fileType != "3gp" && fileType != "mov" && fileType != "mp4" && fileType != "m4v"
&& fileType != "avi" && fileType != "dat" && fileType != "mkv" && fileType != "flv"
                    && fileType != "vob" && fileType != "mp3" && fileType != "wma")
                {
                    this.labFlv.Visible = true;
                    this.labFlv.Text = "*上传的文件格式不符合规定,请选择NextSee支持的文件格式";
                    this.labFlv.ForeColor = System.Drawing.Color.Red;
                    return;
                }
                else
                {
                    this.labFlv.Visible = false;
```

```
            }
            string fileUrl = "NSVideo" + this.Session_UserID.ToString() +
DateTime.Now.Year.ToString() + DateTime.Now.Month.ToString() + DateTime.Now.Day.ToString() +
DateTime.Now.Hour.ToString() +
                DateTime.Now.Minute.ToString() + DateTime.Now.Second.ToString() + "." + "flv";
            string oldfile = "NSVideo" + this.Session_UserID.ToString() +
DateTime.Now.Year.ToString() + DateTime.Now.Month.ToString() + DateTime.Now.Day.ToString() +
DateTime.Now.Hour.ToString() +
                DateTime.Now.Minute.ToString() + DateTime.Now.Second.ToString() + "." + fileType;
            if (uploadFile.PostedFile.ContentLength > 2048000000)//判断图片是否大于k
            {
                this.labFlv.Visible = true;
                this.labFlv.Text = "*你上传的文件太大，请转换后上传";
                this.labFlv.ForeColor = System.Drawing.Color.Red;
                return;
            }
            string strFilePath =
System.Configuration.ConfigurationManager.ConnectionStrings["FileServer"].ConnectionString +
System.Configuration.ConfigurationManager.ConnectionStrings["FileShareDirectory"].ConnectionString;
            uploadFile.PostedFile.SaveAs(strFilePath + oldfile);//把文件保存在此路径中
            uploadFile.Dispose();//////////////////释放掉占用的内存

            string viopian = string.Empty;
            if (this.rioPa.Checked)
            {
                viopian = this.rioPa.Text;
            }
            else if (this.rioPb.Checked)
            {
                viopian = this.rioPb.Text;
            }
            else
            {
                viopian = this.rioPc.Text;
            }
            string origin = string.Empty;
            if (rioOriginy.Checked)
            {
                origin = rioOriginy.Text;
            }
            else
            {
                origin = rioOriginz.Text;
            }
            string viostring = this.txtTaga.Text.Trim();
```

```
                viostring = viostring.Replace(', ', ',').Replace(' ', ',');
                string vioaddress =
System.Configuration.ConfigurationManager.ConnectionStrings["VideoFilePath"].ConnectionString +
oldfile;
                string vioname =
System.Configuration.ConfigurationManager.ConnectionStrings["VideoFilePath"].ConnectionString +
fileUrl;

                DALNS_Video videoDAL = new DALNS_Video();
                MNS_Video videoModel = new MNS_Video();
                videoModel.Vio_Title = this.txtTitle.Text.Trim();
                videoModel.Vio_Content = this.txtContent.Text.ToString().Trim();
                videoModel.Vio_Origin = origin;
                videoModel.Vio_Chanle = this.rbtnVideoChanle.SelectedItem.Text;
                videoModel.Vio_String = viostring;
                videoModel.Vio_Pian = viopian;
                videoModel.Vio_Address = vioaddress;
                videoModel.Vio_Name = vioname;
                videoModel.Vio_Imgurl =
System.Configuration.ConfigurationManager.ConnectionStrings["ImagePath"].ConnectionString + picUrl;
                videoModel.Vio_Userid = int.Parse(Session_UserID);
                videoModel.Vio_Addtime = DateTime.Now;
                videoModel.Vio_Admint = 0;
                videoModel.Vio_Commend = 0;
                videoModel.Vio_Dhit = 0;
                videoModel.Vio_Flag = 0;
                videoModel.Vio_Hit = 1;
                videoModel.Vio_Fav = 0;
                videoModel.Vio_IsRed = 3;
                videoModel.Vio_LastSeetime = DateTime.Now;
                videoModel.Vio_ChannelTypeid = int.Parse(rbtnVideoChanle.SelectedValue);

                long i = videoDAL.Add(videoModel, groupId);
                if (i > 0)
                {
                    this.labUp.Text = "视频文件上传成功!";
                    this.likGoon.Visible = true;
                }
                else
                {
                    this.labUp.Text = "上传失败!";
                }
            }
            else
            {
```

```
            this.labFlv.Visible = true;
            this.labFlv.Text = "*请选择支持的文件并上传！";
            this.labFlv.ForeColor = System.Drawing.Color.Red;
            return;
        }
    }
```

5.3.3 视频数据处理器

消息管理器是最核心的组件，负责调用 FFMPEG 对视频进行处理，同时管理 Service Server 的视频转换线程。对视频处理结束后，返回结果给数据管理器。对开源的 FFMPEG 的重新编译过程如下：

FFMPEG 是 Linux 的开源项目，源代码和 Windows 下最常见的 Visual Studio 提供的 C/C++编译器不兼容，因此它不能使用 MSVC++编译，同时为了增加对各种视频格式的兼容性，要增加第三方的支持。所以实现 Windows 下 FFMPEG 的编译，需要由以下步骤完成：

准备 MSys+MinGW 系统[42]

用 SVN 方式获取 FFMPEG[43]

编译 [44]

链接[45]

加入对第三方库的支持[46]

FFMPEG 调用和线程管理代码如下：

```
    // 进程的主入口点
    static void Main()
    {
        System.ServiceProcess.ServiceBase[] ServicesToRun;
        ServicesToRun = new System.ServiceProcess.ServiceBase[] { new Service1() };

        System.ServiceProcess.ServiceBase.Run(ServicesToRun);
    }
    private void ToFlv()
    {
        //配置文件
        string Path = System.IO.Directory.GetCurrentDirectory() +"\\ChgVideoFormatConfig.ini";
        //日志文件
        string strPathLog = String.Empty;

        IniFile fl = new IniFile();
        fl.FilePath = Path;

        string connectionString = fl.ReadValue("DatabaseConnection","name");
```

```
            //判断是否存在该进程
            int ProNum = 0;
            foreach(Process pro in Process.GetProcesses())
            {
                if(pro.ProcessName == "toflv" || pro.ProcessName == "mencoder")
                {
                    ProNum = ProNum + 1;
                    System.DateTime timeNow = System.DateTime.Now;
                    System.DateTime timeOld = pro.StartTime;

                    TimeSpan ts = timeNow - timeOld;
                    if(ts.Minutes >= 30)
                    {
                        //转换时间太久，大于分钟。不正常
    updateErrorFlag(connectionString,fl.ReadValue("Vio_Id","name"));
                        pro.Kill();
                    }

                }
            }
            if(ProNum>=2)
            {
                return;
            }
            SqlConnection con = new SqlConnection(connectionString);
            try
            {
                //转换文件所在的位置
                string ffmpegPath = fl.ReadValue("ffmpegPath","name");
                //保存文件地址
                string SaveFilePath = fl.ReadValue("SaveFilePath","name");
                //保存图片地址
                string SaveImgPath = fl.ReadValue("SaveImgPath","name");
                //网站默认的绝对路径
                string WebPath = fl.ReadValue("WebPath","name");
                //图片大小
                string FlvImgSize  = fl.ReadValue("ImgSize","name");//"240x180";
                //日志文件
                strPathLog = ffmpegPath + "log\\" + System.DateTime.Today.ToShortDateString()
+"Log.txt";
                //转换颜色
                string VideoColor = fl.ReadValue("VideoColor","name");
                //侦数（图片）
                string ImgStartIndex =  fl.ReadValue("ImgStartIndex","name");
```

```
                //水印文件图片名称
                string ImgPngName = fl.ReadValue("ImgPngName","name");
                //视频开始记录时间（单位：秒）
                string VideoBeginTime = fl.ReadValue("VideoBeginTime","name");
                //截图时间（单位:秒）
                string ImgCutTime = fl.ReadValue("ImgCutTime","name");
                //-s  屏幕的大小 320*240
                string screenSize = fl.ReadValue("ScreenSize","name");
                //-b  比特率200
                string bitrate = fl.ReadValue("Bitrate","name");
                //当前时间
                DateTime dtNow = System.DateTime.Now;
                fl.WriteValue("CurrentTime","name",dtNow.ToString());
                con.Open();
                string sql = @"select top 1 * from NS_Video where (Vio_Flag = 0 or Vio_Flag = 2 or
Vio_Flag = 3 or Vio_Flag = 8) order by Vio_Id asc";
                SqlDataAdapter adapter = new SqlDataAdapter(sql,con);
                DataSet ds = new DataSet();
                adapter.Fill(ds);
                string VideoOldAddress = String.Empty;
                string VideoNewAddress = String.Empty;
                string VideoOldName = String.Empty;
                string VideoNewName = String.Empty;
                string filename = String.Empty;
                string arguments = String.Empty;
                string VideoId = String.Empty;
                string ImgOldName = String.Empty;
                string ImgNewName = String.Empty;
                string Vio_Imgurl = String.Empty;
                bool IsCutPicture = false;
                for(int i=0;i<ds.Tables[0].Rows.Count;i++)
                {

                    //原视频的地址
                    VideoOldAddress = ds.Tables[0].Rows[i]["Vio_Address"].ToString();
                    VideoOldName =
VideoOldAddress.Substring(VideoOldAddress.LastIndexOf("/")+1);
                    //新视频的地址
                    VideoNewAddress = ds.Tables[0].Rows[i]["Vio_Name"].ToString();
                    if(VideoOldName.LastIndexOf(".") < 0)
                    {
                        VideoNewName = "_"+VideoOldName+".flv";
                    }
                    else
                    {
```

```
                    VideoNewName =
"_"+VideoOldName.Substring(0,VideoOldName.LastIndexOf(".")) + ".flv";
                }

                //视频的ID值
                VideoId = ds.Tables[0].Rows[i]["Vio_Id"].ToString();

                Vio_Imgurl = ds.Tables[0].Rows[i]["Vio_Imgurl"].ToString();

                bool IsCompanyPeople = false;
                try
                {
                    //视频的状态
                    int VioFlag = ds.Tables[0].Rows[i]["Vio_Flag"].ToString() == "" ? 0 :
int.Parse(ds.Tables[0].Rows[i]["Vio_Flag"].ToString());
                    if(VioFlag == 3 || VioFlag == 8)
                    {
                        IsCompanyPeople = true;
                    }
                    //视频的转换
                    bool IsMencoderFirst = false;          //
                    string fileType =
VideoOldAddress.Substring(VideoOldAddress.LastIndexOf(".")+1).ToLower();
                    if((fileType == "rm" || fileType == "mpg" || fileType == "rmvb" || fileType
== "wmv" || fileType == "vob" || fileType == "3gp" ) && VioFlag == 0)
                    {
                        IsMencoderFirst = true;
                        VideoNewName =
VideoOldName.Substring(0,VideoOldName.LastIndexOf(".")) + ".flv";
                    }
                  if ((fileType == "rm" || fileType == "mpg" || fileType == "rmvb" || fileType
== "wmv" || fileType == "vob" || fileType == "3gp") && (VioFlag == 2 || VioFlag == 3))
                    {
                        //IsMencoderSecond = true;
                        VideoOldName =
VideoOldName.Substring(0,VideoOldName.LastIndexOf(".")) + ".flv";
                    }
                    if(IsMencoderFirst)    //需要用mencoder 来转换 第一次没有水印
                    {
                        filename = ffmpegPath + "WisMencoder/mencoder.exe";
                        arguments = String.Format(@" -vf scale=320:240 -ffourcc FLV1 -of lavf
-lavfopts i_certify_that_my_video_stream_does_not_use_b_frames -ovc lavc -lavcopts
vcodec=flv:vbitrate=200 -srate 22050 -oac lavc -lavcopts acodec=mp3:abitrate=56 {0}{1} -o
{0}{2}",SaveFilePath,VideoOldName,VideoNewName);
                    }
```

```
                else
                {
                    filename = ffmpegPath + "toflv.exe";
                    arguments = String.Format(@" -i {0}{1} -vhook
'{2}vhook/watermark.dll -f {2}{3} -m 0 -t {4}'  -r 25 -s {7} -ss {6} -b {8}
{0}{5}", SaveFilePath, VideoOldName, ffmpegPath, ImgPngName, VideoColor, VideoNewName, VideoBeginTime, scr
eenSize, bitrate);
                }
                System.Diagnostics.ProcessStartInfo ps = new
System.Diagnostics.ProcessStartInfo(filename, arguments);
                ps.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
                System.Diagnostics.Process.Start(ps);
                //在配置文件中，记录下该视频的Vioid。
                fl.WriteValue("Vio_Id","name",VideoId);
                fl.WriteValue("Vio_Id","flag","0");
                //截屏图片，如果已经上传图片就不截图
                ImgOldName = Vio_Imgurl.Substring(Vio_Imgurl.LastIndexOf("/")+1);
                if(!IsMencoderFirst && ImgOldName.Length < 13)
                {
                    IsCutPicture = true;
                    ImgNewName = "NSVideo" +
ds.Tables[0].Rows[i]["Vio_Userid"].ToString() + DateTime.Now.Year.ToString() +
DateTime.Now.Month.ToString() + DateTime.Now.Day.ToString() + DateTime.Now.Hour.ToString() +
                        DateTime.Now.Minute.ToString() +
DateTime.Now.Second.ToString()+".jpg";
                    ps.FileName = ffmpegPath + "toflv.exe";
                    ps.Arguments = String.Format(@" -i {0}{1} -y -f image2 -t {2} -s {3}
-ss {6}
{4}{5}", SaveFilePath, VideoOldName, ImgStartIndex, FlvImgSize, SaveImgPath, ImgNewName, ImgCutTime) ;
                    System.Diagnostics.Process.Start(ps);
                }

                //
                int LastFlag;
                if(!IsMencoderFirst)
                {
                    if(IsCompanyPeople)
                    {
                        LastFlag = 5 ;//
                    }
                    else
                    {
                        LastFlag = 1;
                    }
                    if(IsCutPicture)
```

```
                                {
                                    sql = "update NS_Video set Vio_Flag = '" + LastFlag +
"',Vio_Name='" + VideoNewAddress.Substring(0,VideoNewAddress.LastIndexOf("/")+1)+VideoNewName + "',
Vio_Imgurl='" + Vio_Imgurl.Substring(0,Vio_Imgurl.LastIndexOf("/")+1)+ImgNewName +"' where Vio_Id='"
+ VideoId + "'";
                                }
                                else
                                {
                                    sql = "update NS_Video set Vio_Flag = '" + LastFlag +
"',Vio_Name='" + VideoNewAddress.Substring(0,VideoNewAddress.LastIndexOf("/")+1)+VideoNewName + "'
where Vio_Id='" + VideoId + "'";
                                }
                            }
                            else
                            {
                                if(IsCompanyPeople)
                                {
                                    LastFlag = 3 ;//
                                }
                                else
                                {
                                    LastFlag = 2;
                                }
                                sql = "update NS_Video set Vio_Flag = '" + LastFlag + "' where
Vio_Id='" + VideoId + "'";
                            }

                            SqlCommand cmd = new SqlCommand(sql,con);
                            cmd.ExecuteNonQuery();
                            cmd.Dispose();

                        }
                        catch(Exception ep)
                        {
                            sql = "update NS_Video set Vio_Flag = 4 where Vio_Id='" + VideoId + "'";
                            SqlCommand cmd = new SqlCommand(sql,con);
                            cmd.ExecuteNonQuery();
                            cmd.Dispose();
                        }
                    catch(Exception ex)
                    {
                        using(System.IO.StreamWriter sw = File.AppendText(strPathLog))
                        {
                            sw.WriteLine(ex.Message);
                            sw.WriteLine(System.DateTime.Now);
```

```
                sw.WriteLine();
            }
        }
        finally
        {
            con.Close();
            con.Dispose();
        }
    }
```

5.3.4 异常处理模块

主要针对用户上传的损坏的文件，或者异常进程进行处理，同时记录视频转换日至，其主要实现代码如下：

```
private void updateErrorFlag(string connection,string VioId)
    {
        SqlConnection con = new SqlConnection(connection);
        try
        {
            con.Open();
            string sql = "update NS_Video set Vio_Flag = 4 where Vio_Id='" + VioId + "'";
            SqlCommand cmd = new SqlCommand(sql,con);
            cmd.ExecuteNonQuery();
            cmd.Dispose();
        }
        catch(Exception ex)
        {
        }
        finally
        {
            con.Close();
            con.Dispose();
        }
    }
private void ErrorHandle()
{
            System.IO.StreamWriter sw;
            if(!File.Exists(strPathLog))
            {
                sw = File.CreateText(strPathLog);
            }
            else
            {
                sw= File.AppendText(strPathLog);
            }
```

```
                    sw.WriteLine(String.Format(@"原视频:{0}{1},新视
频:{0}{2}",SaveFilePath,VideoOldName,VideoNewName));
                    sw.WriteLine(System.DateTime.Now);
                    sw.WriteLine();
                    sw.Close();
}
private void ProHandle()
        {
            //配置文件
            string Path = System.IO.Directory.GetCurrentDirectory()
+"\\ChgVideoFormatConfig.ini";
            //日志文件
            string strPathLog = String.Empty;

            IniFile fl = new IniFile();
            fl.FilePath = Path;

            string connectionString = fl.ReadValue("DatabaseConnection","name");
            //判断是否存在该进程
            int ProNum = 0;
            foreach(Process pro in Process.GetProcesses())
            {
                if(pro.ProcessName == "toflv" || pro.ProcessName == "mencoder")
                {
                    ProNum = ProNum + 1;

                    System.DateTime timeNow = System.DateTime.Now;
                    System.DateTime timeOld = pro.StartTime;

                    TimeSpan ts = timeNow - timeOld;
                    if(ts.Minutes >= 30)
                    {
                        //转换时间太久，大于30分钟。不正常
                        updateErrorFlag(connectionString,fl.ReadValue("Vio_Id","name"));

                        pro.Kill();
                    }
                }
            }
            if(ProNum>=2)
            {
                return;
            }
}
```

5.4 本章小结

本章在上一章提出的集成系统结构基础上，利用 Microsoft Visual Studio2005 技术实现了一个简单的可扩展的基于 FFMPEG 视频转换的系统服务环境。

第 6 章 结束语和系统进一步优化展望

6.1 结论

6.1.1 本文的主要工作有

通过国内外资料的收集和整理、分析，对 CSCW 系统概念、特征进行了综述，论述了用于支持这种系统的模型。

结合研究方向，以网络理论知识为基础，探讨了 CSCW 的基于 FFMPEG 的视频转换系统。

在了解基于 FFMPEG 的视频转换服务环境的基础上，提出了用于协同服务环境的一种系统服务集成框架和应提供的功能。这种集成框架为开发基于客户机/服务器结构的 CSCW 系统提供了一个完整的结构，简化了视频分享互联网应用的程序开发。在该结构中提供了开放性的支持、互联互通和协同功能的增强，并深入讨论了服务器处理多个视频转换请求的方法。

6.1.2 系统服务环境具有的技术特点

扩展性支持。提供两种工具扩展方式。

简洁的功能包装。集成框架实现了工具代理，屏蔽了工具与服务器的通讯细节，简化了工具的编程，网络通信模块借助客户机与服务器之间强大的网络通信能力，依靠转发，间接地增强了互联网视频传播的通信能力，从效果上看比较令人满意。

灵活的通讯方式。针对不同的数据采取了不同的通讯方式和协议。

6.2 进一步工作

基于 FFMPEG 的视频转换系统的服务环境的设计与实现涉及多方面的理论、方法和技

术，本文对其中的一些关键问题进行了有益的研究和探讨，但还有许多问题需要进一步的研究。

在服务器集成框架中，现在使用的是 Windows 系统服务的集成结构，对数据状态的侦测在灵活度方面还需要进一步优化。如何进一步有效利用服务器，当转换请求长时间没有地情况下，是系统及时回收服务进程，需要进一步研究。

在视频转换方面，FFMPEG 尚有些最新的视频格式文件不支持，需要在 FFMPEG 的转换支持方面进一步深入。

因作者水平有限，时间紧迫，任务量大，还有较多需进一步改进和完善之处。我将在此基础上，进一步对基于 FFMPEG 的视频转换服务模型、标准化等方面进行深入研究，并结合分布式理论和技术，提出更适合互联网视频分享应用的具有通用性和扩展性的协作模型。

参 考 文 献

[1]风舞博客. 视频编码说明[DB\OL] http://i.cn.yahoo.com/an_fengwu/blog/p_80/

[2]中国互联网络信息中心. 2008 中国互联网调查报告[DB\OL]

http://www.hdcmr.com/article/yjbg/06/05/21801.html

[3] FFMPEG 工程组. FFMPEG 的详细说明[DB\OL]

http://www.ffmpeg.com.cn/index.php/Ffmpeg%E7%AE%80%E4%BB%8B

[4]郑香霖. 发扬 Web2.0 精神[J] 中国经营报，2007.10:4

[5]王昆丽. 视频生活传播[DB\OL]

http://www.donews.com/Content/200607/c0f177da5afa43129eea540d6dd8ff2b.shtm

[6]Bborn. windows 平台编译 FFMPEG 的静态链接库[DB\OL] http://bborn.cn/blog/2008/11/ffmpeg-h264/

[7]Yangyi. 关于操作系统占有率的统计参考[DB\OL]

http://www.chineselinuxuniversity.net/news/32584.shtml

[8]史美林. 谈计算机支持协同工作的应用及展望[J]. 微电脑世界. 2001（52）

[9]倪强, 朱光喜. 计算机支持下的协同工作的研究现状综述[M]. 计算机工程与应用. 2000.4

[10]范玉顺, 吴澄. 基于协调理论的工作流建模方法[J]. 计算机集成制造系统，2001 7(4)： 1-6

[11] V.M.R. Penichet, I. Marin, J.A. Gallud, M.D. Lozano and R. Tesoriero A Classification Method for CSCW Systems Electronic Notes in Theoretical Computer Science[M], Volume 168, 8 February 2007, Pages 237-247

[12][13][14][15][16]Kuutti, K.: The concept of activity as a basic unit of analysis for CSCW research. Proceedings of the Second European Conference on Computer Supported Cooperative Work (ECSCW'91) [J], Amsterdam, 1991

[17] Kreifelts, T., Hinrichs, E. and Woetzel, G.: Sharing To-Do Lists with a Distributed Task Manager. Proceedings of the Third European Conference on Computer Supported Cooperative Work (ECSCW'93)[J], pp. 31-46, 1993

[18] Teege,G.Object-Oriented Activity:A Model for Integrated CSCW Systems[J],The Journal of Collaborative Computing,1996, 5(1): 93-124

[19]Fenng. YouTube 的架构扩展[DB/OL] http://www.dbanotes.net/opensource/youtube_web_arch.html

[20]Flamepjlh.Windows 下 FFMPEG 编译指南[DB/OL] http://bbs.chinavideo.org/viewthread.php?tid=3447

[21]Michael Moy. MSVC++ 内在属性参考[DB/OL]

http://coding.derkeiler.com/Archive/Assembler/comp.lang.asm.x86/2005-05/msg00104.html

[22]李成锴, 詹永照等. 基于角色的 CSCW 系统访问控制模型[J]. 软件学报, 2000；11(7)

[23] 史美林, 向勇, 杨光信. 计算机支持的协同工作理论与应用[M]. 北京：电子工业出版社. 2000.

[24]顾君忠. 计算机支持的协同工作导论[M]. 清华大学出版社. 2002.11

[25]倪强,朱光喜.计算机支持下的协同工作的研究现状综述{M}.计算机工程与应用.2000.4

[26]张桦,于长云.支持协同工作的多用户接口实现技术[J].天津大学学报（自然科学与工程技术版).2000 Vol.33 o.3 P.394-396

[27]王燕涛,黄克正,李旭东.计算机支持协同产品设计研究[J].青岛大学学报.2003.6 Vol.18 No.2

[28]杨武勇,史美林,张少华.SmartBoard.全复制结构下电子白板工具的研究与实现

[29]刘勇军,付萍萍.虚拟教室中共享电子白板的设计实现[J].四川师范大学学报.2004.11 Vol.27 No.6

[30] 张道春,刘　弘. CSCW中一种多层次协同工作模型研究[C].计算机应用研究,2007年3月　第24卷第3期

[31] Shichao Zhang, Ning Gu and Jiangming Yang, An norm-driven state machine model for CSCW systems ,Expert Systems with Applications[M], Volume 31, Issue 4, November 2006, Pages 800-807

[32]朱羽,王珊,唐伟,戴青,范鹏.基于电子白板的远程教育系统的设计与实现[N].河北大学学报(自然科学版）.2003.9 Vol.23 No.3

[33]王燕涛,黄克正,李旭东.计算机支持协同产品设计研究[N].青岛大学学报.2003.6 Vol.18 No.2

[34]杨武勇,史美林,张少华,SmartBoard.全复制结构下电子白板工具的研究与实现

[35]高京.基于 WEB 的 CSCW 系统与远程教育[Z].计算机世界网.2004.5

[36]Michel Tony.Collaborative network management Proceedings[M] - IEEE Military Communications Conference MILCOM, v 2, 1998, p 599-603

[37]张桦,于长云.支持协同工作的多用户接口实现技术[N].天津大学学报（自然科学与工程技术版).2000 Vol.33 o.3 P.394-396

[38]刘润东.UML对象设计与编程[M]，北京:希望电子出版社，2001: 104-113

[39]IIS线程配置[DB/OL] http://www.study-code.com/server/iis/85458.htm

[40] Bob Beaucheminz. ADO.NET本质论，北京: 清华大学出版社著 2003：52-60

[41] Fritz Onion. ASP.NET基础教程—C#案例版，北京: 清华大学出版社著 2003:124-136

[42] Mingw.Linux系统下编译Windows的程式[DB/OL]

http://www.sudu.cn/info/html/edu/network_technology/20071229/231619.html

[43] James.Subversion 进行版本控制[DB/OL] http://www.svn8.com/SVNSY/20090405/4423.html

[44][45][46]Flamepjlh . FFMpeg在windows下傻瓜编译指南[DB/OL].

http://bbs.chinavideo.org/viewthread.php?tid=3447

致　谢

在本论文完成的过程中，得到了我的导师吕丽民教授的悉心指导。吕老师在论文选题，研究工作和论文写作等方面对我进行了细致认真的指导、帮助和督促，使我的论文得以顺利完成，另外吕老师求实的作风、渊博的学识和灵活的思路也使我受益匪浅，在此特对他致以衷心的谢意。

在研究生学习期间，得到了很多老师和同学的帮助，在此，我对他们表示诚挚的感谢。感谢汤一平和古辉老师对我在开题过程中的指导，提出了许多建议，使我少走了很多弯路，感谢我们李君毅、林海元同学，与他们的联系和交流使我能顺利的完成任务，谢谢他们的理解和帮助。

我要感谢我的父母和家人，是他们长期以来无私的支持、关心、教诲和鼓励使我有机会能够继续我的学业，是他们给了我前进的动力，让我朝着更高的目标发展。另外我也要对家人表示深深的歉意，因为我的学业而影响了他们生活质量的改善，给他们带来了很大的压力。

我还要特别感谢企业导师徐晔大哥，在系统开发过程中提出了很多建议和改进方案，同时也是他在我困难时给了我继续前进的动力，给了我希望和信仰。

最后，谨向所有关心和帮助过我的老师、同学们和朋友们表示我最诚挚的谢意！

攻读学位期间参加的科研项目和成果

参加的科研项目

录用和发表的论文

[1] 基于 FFMPEG 视频转换系统的模块研究[J]. 电脑知识与技术 2009.4 录用

专利