摘 要

本文介绍了支持内容管理的文件管理器 **Z-Explorer** 的设计与实现。

本文首先分析了内容管理平台支持的功能和当前基于内容管理的文件管理器存在的不足，并结合课题组项目研究的需要，给出了课题的设计目标和任务。然后介绍了 **Z-Explorer** 相关的技术背景，详细分析了 **Z-Explorer** 的各功能模块及体系结构，讨论了系统实现的核心技术，研究了系统功能的实现细节，并基于 Windows 平台演示了 **Z-Explorer** 的主要功能。本文最后对 **Z-Explorer** 做出了客观性评价，并对未来的工作进行了展望。

论文设计的 **Z-Explorer** 文件管理器能够充分的挖掘信息的价值，有助于用户更好的进行内容管理，具有较好的研究价值和实用意义，有利于促进内容管理的发展。此外，本文对支持内容管理的文件管理器实现技术所进行的探讨，对于该领域的开发者而言，应具有良好的借鉴意义。

关键字：内容管理、文件管理器、数据库、Java

作　　者：张成年

指导教师：吕　强

The Design and Implementation of a File Manager Supporting Content Management

Abstract

This paper introduces the design and implementation of a file manager named Z-Explorer which supports content management.

First of all, we analyze the function supporting in the content management platform and the disadvantages of the file manager based on the content management. And in consideration of the requirement of our seminar's project, we make clear the destination of our task and formulate feasible plan. Then we introduce the technological background of the Z-Explorer.

Thereafter, we analyze every function module of the Z-Explorer and system structure in details. Later, we discuss some key technologies of the system and dwell on some realizations of the systematic functions. Then we demonstrate the functions of the Z-Explorer on the Windows platform. .

In the last part of this paper, we evaluate the Z-Explorer on objectivity and make a prospect for future expedition in this field. The Z-Explorer is so useful that the users can do well in content management. And the value of information can be well mined. So the Z-Explorer is helpful to promote the development of content management.

Furthermore, I believe the discussion about the technologies, which we have employed to implement the Z-Explorer supporting content management, must be helpful to those developers who are exploring in this field.

Keywords: Content Management, File Manager, Database, Java

Written by Zhang Cheng-nian

Supervised by Lv Qiang

苏州大学学位论文独创性声明及使用授权声明

学位论文独创性声明

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成果。除文中已经注明引用的内容外，本论文不含其他个人或集体已经发表或撰写过的研究成果，也不含为获得苏州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。

研究生签名：[signature] 日　期：4.26

学位论文使用授权声明

苏州大学、中国科学技术信息研究所、国家图书馆、清华大学论文合作部、中国社科院文献信息情报中心有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权苏州大学学位办办理。

研究生签名：[signature] 日　期：4.26

导师签名：[signature] 日　期：

第一章 绪论

1.1 引言

在当今的信息时代，数据的容量和复杂性都呈爆炸式地快速增长。Forrester Research 的一个调查显示：企业信息的内容量在以每年 200% 的速度增长[1]。通常情况下，一个企业以文档、Web 页面、多媒体、电子邮件、结构化数据等方式将信息存储在文件服务器、工作站、视频服务器、电子邮件服务器和无数独立的数据库中[2]。由于这些重要的信息往往分别存储在不同的“信息孤岛”中，企业在管理、利用这些宝贵的信息资源时困难重重，主要表现在以下几个方面：

1、对存储在不同机器或系统上的信息进行搜索。

2、如何与不能访问同一数据的同事协同工作。

3、通过不同的方法（例如：Web 浏览器、电子邮件等）从远程访问信息。

4、管理分散数据（授予访问权限、设置安全和备份数据等）。

5、用可靠的方法对数据提供持续的可用性。

因此，不断膨胀的信息量和低下的信息处理效率使得企业迫切需要有效地将信息管理起来。分散、混乱的信息形成不了竞争力，必须将之整合为有组织的信息才能充分发挥其价值。传统的信息系统解决方案难以解决这个问题，信息系统需要实现从数据管理、文档管理到内容管理的转变，以满足急剧膨胀的信息量和信息处理效率的需要。

1.2 课题内容及意义

1.2.1 课题内容

内容管理除了能够提供传统的数据管理和文档管理所具有的功能外，还能够提供许多传统方式所没有的高级管理功能，如全文搜索、版本控制和工作流等，并且利用内容管理开发平台还可以构建基于内容管

理的高级应用程序。目前，一些大的公司如 Oracle、微软、IBM 等都在其核心的数据库产品上提供了对内容管理的支持，并为用户提供了开发内容管理应用程序的平台。但是，这些公司都没有提供一个易于操作、跨平台、功能完善的文件管理器实现其数据库产品所支持的高级内容管理功能，以至于内容管理的优越性得不到充分发挥，信息的价值不能被充分地挖掘，从而限制了内容管理的推广。

根据调研发现，要想充分挖掘信息的价值、提高生产效率，首先要解决如何快速、准确地搜索到文件，协同同事的工作，更好地管理不同类别的文件，对文件进行更安全的管理，以及根据需要为文件赋予更多有意义的信息，使文件更有价值。

鉴于这些原因，结合内容管理开发平台所支持的功能，并根据课题组研究项目的需要，我们开发了一个功能丰富、操作简单、跨平台的文件管理器——Z-Explorer。此版本的 Z-Explorer 不仅实现了一般文件系统的管理功能，还增强了基于内容管理的文件系统的高级功能，具体包括以下几个功能：

1、查看/设置基于内容管理的文件的属性

该部分主要完成对元数据、版本、类别、ACL(访问控制列表)等信息的查看/设置，工作的重点是增强 ACL 控制和类别设置。

2、基于内容管理的高级搜索功能

该部分主要完成根据元数据、内容、类别、文件类型等进行文件搜索和全文检索，同时，要实现正则搜索和关联搜索功能。

3、对基于内容管理的文件的版本控制

该部分主要完成版本化、检入/检出、取消检出、删除版本文件等功能，并实现对文件的锁定和解除锁定功能，侧重实现灵活的版本检入/检出操作。

4、添加 Context 菜单

文件管理器为了能够方便地提供给用户使用，需要在基于内容管理的文件/文件夹的 Context 菜单（鼠标右键弹出的菜单）中添加一些菜单项，每个新添菜单项关联到相应的文件管理器的功能，这样用户就能

够方便地利用文件管理器实现对文件的内容管理。

1.2.2 课题意义

Z-Explorer 为用户提供了内容管理的操作环境，实现并拓展了内容管理的功能，实现了正则查找功能。另外，Z-Explorer 为课题组其它科研项目的研究奠定了基础。概括起来，主要有以下几方面的意义：

1、Z-Explorer 实现了对基于内容管理的文件/文件夹的高级管理功能。通过它不仅可以对文件/文件夹常见属性进行操作，还可以对文件/文件夹的扩展属性进行操作。Z-Explorer 能够利用访问控制列表——ACL 进行安全管理控制、支持正则搜索功能、实现了关联搜索功能，同时提供了更灵活的版本控制功能。这些功能能够帮助企业进一步挖掘信息的价值和提高生产效率，体现了内容管理所支持的丰富的内容管理功能及其优越性。

2、有利于内容管理的进一步推广。在企业中，用户会使用各种各样的操作系统，如：MS Windows、Linux、Unix 等。而目前没有一个基于内容管理的文件管理器能够跨平台，这势必会限制内容管理在企业中的推广。Z-Explorer 是利用 Java 语言开发的，因此，实现了文件管理器的跨平台，能够在所有具有 JRE 环境的操作系统中运行。这能够促进更多的企业转向内容管理，充分利用内容管理的优越性来提高企业的竞争力。

3、Z-Explorer 开发的另一个重要意义在于为以后整合课题组研发的语义文件系统做好准备。目前，课题组已研发的具有一定语义功能的办公文档就是在内容管理开发平台上实现的。因此，必须要有一个新的文件管理器能够实现对这种新型文档的管理，Z-Explorer 也考虑了这方面的应用需求，为两者整合做好了准备。同时， Z-Explorer 也为课题组其它基于内容管理的研发项目的整合奠定了基础。

1.3 国内外的相关研究情况

内容管理发展到今天，所经历的时间并不长，其技术也有待进一步发展，价值也未被充分地挖掘。市场上内容管理软件产品品种繁多，技

术参差不齐。

但是，内容管理市场发展迅速，据专门从事通信软件市场调查的WinterGreen Research, Inc 调查分析，内容管理市场在 2003 年创下10 亿美金的销售业绩，并且一般认为其市场规模在 2009 年突破 21 亿美金的大关[3]。如此巨大的市场促使全球许多公司正在加大在内容管理方面人力和物力的投入。

在国外，IBM 提供了一套可靠的、易升级、强劲的企业内容管理体系架构，同时也提供了强劲的、安全的和高扩展能力的服务。IBM 已经开发了一系列内容管理器产品的构件，如 Content Manager On Demand、IBM 企业信息门户、MQSeries Workflow 等。

Oracle 公司也在其核心数据库上支持了内容管理，提供了基于内容管理的开发平台，2004 年 12 月 9 日 Oracle 公司发布企业级内容管理工具——Oracle Files 10g[4]，并宣布今后将跨越数据库加大内容管理领域的投入。

微软也在内容管理领域投入了大量的资金，为用户提供了一些内容管理功能，Microsoft Content Management Server 允许内容提供者创建、管理并发布其自己的内容，同时允许 IT 部门快速部署具备伸缩能力的动态站点。另外，曾经被盖茨称为“Longhorn 圣杯”的 WinFS 计划也是为了实现对信息进行内容管理。通过 WinFS，微软公司希望无论信息的格式如何、存储在什么地方，都能迅速地发现和处理这些信息。但由于一些技术原因，WinFS 计划在短期内尚不能实现[5]。

在国内，TRS(托尔思)开发了信息检索和知识管理、内容分发服务器等多款内容管理产品，在国内外领先推出实用化的知识管理和数据挖掘产品。

海量科技的嵌入式数据库搜索引擎（DESE）产品采用了先进的智能分词、概念抽取、自动摘要和全文检索等多项技术，同时结合数据库自身的检索机制，形成基于数据库的企业级搜索引擎，实现了数据的信息搜索、权限管理、数据维护一体化等[6]。如大家熟悉的 CSDN 论坛的全文检索，就是海量科技提供的产品。

目前，内容管理有两种基本的体系实现方法：一是以传统文件系统作为内容的存储库，这种体系只适用于松散控制环境下较小组的情况；另一种是以数据库作为内容的存储库，这种体系是建立在数据库管理系统之上，能够提供更多的功能和稳定的性能。

在 Sourceforge 上有一个文档管理系统的开源项目——KnowledgeTree。它采用传统层次文件系统作为内容的存储库，在传统层次文件系统上实现了支持内容管理的文档管理系统，该系统是完全基于 Web 的，具有丰富文档管理功能，主要集成了如下功能：知识管理、文档版本化控制、层次文档管理、对普通文件格式的支持、扩展的 metadata、创建客户文档类型、应用程序管理的文档链接、方便的文档发布、用户代理和归档等。KnowledgeTree 管理文档的操作界面如图 1-1：

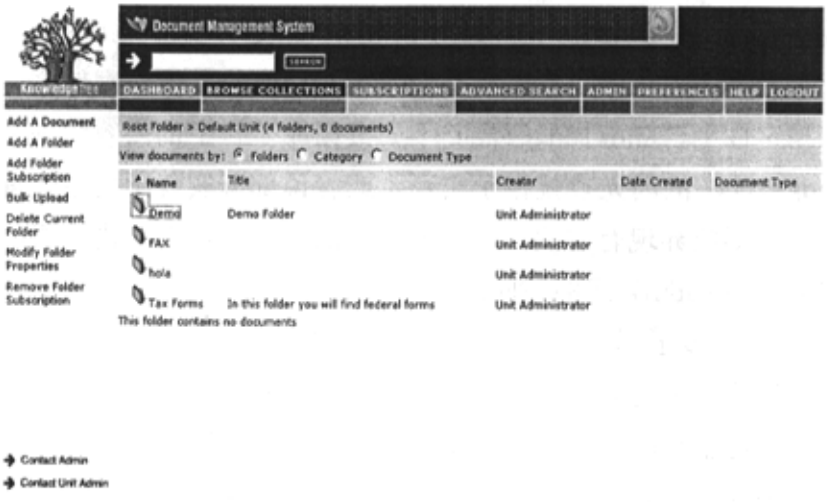

图 1-1 KnowledgeTree 操作界面

KnowledgeTree 提供了开放式的框架，因此它提供开放式的接口来支持开发者所开发的新模块，这样开发人员就能很容易的在其中嵌入自己的应用模块。KnowledgeTree 本身有些功能也是通过嵌入第三方工具来实现的，如：利用第三方工具 FileIndexer 和 CVS(Concurrent Version System) 实现全文检索和版本控制，所有嵌入的第三方工具都存在与 KnowledgeTree 集成的问题，由于 KnowledgeTree 刚刚发展，因此在集成方面做得还不完善，导致了系统稳定性差。

KnowledgeTree 通过对每种类型的文档创建索引为用户提供全文检索功能，这种索引性能不高、可重用性差。它的版本控制存在执行效率低的问题。

KnowledgeTree 是一个文档管理系统，它只对几种常见类型的文档进行管理，不能管理流媒体等类型的文件。另外，KnowledgeTree 发展时间不长，系统还不够完善、稳定性差。

本文使用的 Oracle 9iFS 是一个基于数据库的文件系统，它以数据库为存储库，因此它充分利用数据库管理系统的强大功能，将数据库的安全性、可管理性和可搜索性延伸到文件系统，实现了内容管理功能。

由于 KnowledgeTree 和数据库平台上的内容管理系统在体系上有所不同。KnowledgeTree 是建立在传统层次文件系统上的，而后者是建立在基于数据库的文件系统上的。KnowledgeTree 设定有指定的上传文档的文件夹，在数据库中记录了文档的完全路径等信息，并没有存储文件。在功能和稳定性上，KnowledgeTree 不如数据库平台上的内容管理系统，KnowledgeTree 的闪光点就在于实现了或者说部分实现了内容管理的功能，同时又不放弃现有层次文件系统之上的资源，结合了内容级、文件名级和文件 metadata 级的访问。

另外，由于数据库平台上的内容管理系统能够充分地利用数据库管理系统提供的强大功能，而 KnowledgeTree 却不具备这样丰富的技术支撑，所以在功能实现、性能、各功能的结合上、设计规范以及开发文档等各方面 KnowledgeTree 不如数据库平台上的内容管理系统。

Oracle 9iFS 安装好后，为用户提供了两种访问内容的方式：浏览器和示例文件管理器。通过浏览器对内容进行访问时，存在操作效率低、不方便、文件传输速度低等缺点。而示例文件管理器操作简单、效率高，但其只提供几个简单的用于演示 Oracle 9iFS 高级内容管理的功能，如：版本控制、安全控制等。由于示例文件管理器提供的功能并不完整，如版本控制功能只允许用户将版本化文件的最后一个版本检出，这完全不能满足版本化操作要求，另外，也没有为用户提供登录窗口，每个用户都被赋予管理员权限，因此它并不能作为一个企业的内容管理工具。

本文开发的 Z-Explorer 文件管理器借鉴了 KnowledgeTree 和示例文件管理器的功能，并结合课题组项目的需要，为用户提供了丰富的内容管理功能和简单的操作界面，并实现了跨平台。

1.4 本文的组织结构

本文内容是以如下方式组织的：

第一章 绪论，这部分对论文内容作了概括性介绍。给出了课题内容、课题的意义以及国内外该领域的研究情况。

第二章 技术背景，这部分简要的介绍了课题涉及的相关技术，指出系统与这些技术的关系。

第三章 Z-Explorer 的总体设计，这部分对文件管理器的功能进行了详细的分析，提出了文件管理器要实现的功能，并对 Z-Explorer 体系结构做了合理的设计。

第四章 Z-Explorer 的设计与实现，这部分介绍了重要模块的设计与实现，重点介绍实现过程中用到的一些核心技术。

第五章 平台搭建与实例演示，这部分介绍了 Z-Explorer 的运行环境、部署方法，并通过一些实例演示，介绍了 Z-Explorer 的主要功能。

第六章 结语，我们总结了本文完成的主要工作以及这些工作成果的价值和实际意义，指出了课题的一些不足之处和可以开展的后续工作，并对系统进行了展望。

第二章 技术背景

2.1 内容管理

随着社会信息化的推进，信息量呈急速膨胀趋势，传统的数据管理、文档管理已经无法满足企业用户的需求。因此，内容管理应运而生，它能够很好地帮助企业用户高效率地整合企业信息，充分挖掘企业信息的价值。因此，良好的内容管理解决方案将是现代企业、组织实施信息化战略的一个重要保障。

2.1.1 什么是内容

内容是记录在某种媒体上的意义或意图，并通常用来表示或传达某种含义，而这种含义与记录它的媒体无关[7]。

内容的成熟需要经过三个阶段，分别是数据、信息和知识。数据是未经组织的数字、词语、声音、图像等[7]。数据本身是没有意义的，只有在特定的上下文环境中或是与其它数据的联系中才有意义。信息是以有意义的形式加以排列和处理的数据[7]。也就是把数据置于具体的上下文环境中并和其它数据一起经过组织，数据就成熟为信息。信息不是孤立存在的，信息与信息之间有复杂的关系。知识是用于生产的信息（有意义的信息）[7]。信息经过加工处理、应用于生产，才能转变成知识。而内容是介于信息和知识之间，当把信息加以组织然后为了特定的目的以某种方式发布出来后，信息就成为了内容。

2.1.2 什么是内容管理

内容管理是一项新兴的技术。对于内容管理，目前业界还没有一个统一的定义，不同的机构有不同的理解：

Gartner Group 认为[8]：内容管理从内涵上应该包括企业内部内容管理、Web 内容管理、电子商务交易内容管理和企业外部网(Extranet)信息共享内容管理（如 CRM 和 SCM 等）。

Merrill Lynch 的分析师认为[9]：内容管理侧重于企业员工、企业

用户、合作伙伴和供应商方便获得非结构化信息的处理过程。内容管理的目的是把非结构化信息出版到 intranets, extranets 和 ITE(Internet Trading Exchanges)，从而使用户可以检索、使用、分析和共享。商业智能系统(BI)侧重于结构化数据的价值提取，而内容管理则侧重于企业内部和外部非结构化资源的战略价值提取。

TRS 认为[10]：内容管理不是某种单独的创新技术，而是许多先进技术的综合应用，它涵盖企业内联网(Intranets)、因特网(Internet)和企业外联网(Extranets)应用，大大突破了传统信息流管理软件、办公自动化软件以及文档管理软件的应用范围、使用效果和商业价值。内容管理解决方案重点解决各种非结构化或半结构化的数字资源的采集、管理、利用、传递和增值，并能有机集成到结构化数据的商业智能(BI)环境中，如 ERP, CRM 等。电子商务和 XML 是内容管理市场发展的源动力，内容管理解决方案的终极目标是实现内容价值链的最优化。

综合上述的观点，我们认为，内容管理是协助组织和个人，借助信息技术，将数据、文档、视频、音频等多种形式整合在一起，对它们进行分析处理，再提供给需要这些数据的用户，并在企业个人、组织、业务、战略等诸方面产生价值的过程。

2.1.3 内容管理技术应用的逐步深化

内容管理从二十世纪八十年代出现至今，随着它在企业信息化应用中的地位不断提升，其核心技术的研发在不断深入，内涵也更加丰富。如图 2-1 所示[11]，内容管理中技术应用的深化主要表现为更多智能手段的引入。目前的内容管理解决方案具备了检索、工作流、版本控制等基本功能，但处理的智能性还不高；同时，虽然文档自动分类技术、文本挖掘、Web 挖掘、Portal 等技术已经有了一定的应用，但必定会有更智能化的内容处理技术出现，如语义分析、自然语言理解、智能代理等。课题组研发的语义文件系统正是对内容管理技术的进一步发展。

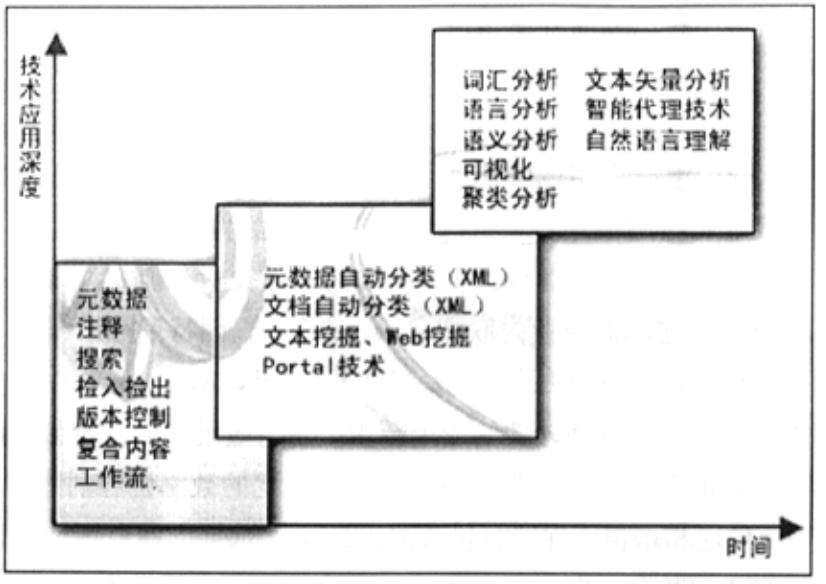

图 2-1 内容管理技术应用的逐步深化

2.1.4 基于数据库的文件系统与内容管理的关系

目前，实现内容管理的主要策略是将非结构化和半结构化的数据统一存储到数据库中。利用数据库技术实现对这些信息资源的组织和管理，通过数据库平台延伸对内容管理的支持，在保留文件系统功能的基础上实现了基于内容的管理，而且还将数据库强大的数据管理功能延伸到文件系统中，如可用性、可靠性、安全性、统一管理和海量存储等。

基于数据库的文件系统实际上就是利用数据库技术实现内容管理的一种新型文件系统，而且在此基础上可以实现语义文件系统。本课题所选用的 Oracle 公司的网络文件系统——Oracle 9iFS （9.0.1）就是基于数据库的文件系统。在 9.0.3 之后的版本，Oracle 公司将 Oracle 9iFS 改称为 Oracle CM(内容管理)。

2.2 Oracle 9iFS 简介

2.2.1 Oracle 9iFS 概述

Oracle 9iFS 是 Oracle 公司开发的基于 Oracle 数据库的新型文件系统。它是对 Oracle 数据库革命性的扩展，不仅具有 Oracle 数据库的

可靠性、可用性和可伸缩性，还具有标准文件系统的常见特性和易用性，更支持了基于内容管理的新特性。

Oracle 9iFS 将企业所有数据统一成单个、统一的信息库，企业的所有数据都存放在 Oracle 数据库中，包括文档、多媒体、web 页面、电子邮件、结构化数据等。Oracle 9iFS 在系统内支持超过 150 种常见的文件格式，当企业应用依赖于一个特殊结构的文档时，开发者利用 Oracle 9iFS Java API 能够很容易定义一种新文档类型，如课题组正在研发的具有一定语义功能的文档。同时，Oracle 9iFS 的可扩展性也为企业构建其它基于内容管理的应用程序提供了方便。

Oracle 9iFS 为企业提供了比一般文件系统更高级的信息管理功能。它提供了从操作系统到数据库的无缝数据传输，把来自传统文件系统的信息与数据库中的文件和数据统一了起来，创造性地把关系型数据库的安全性、可管理性和可搜索性延伸到文件系统。

2.2.2 Oracle 9iFS 体系结构

Oracle 9iFS 被设计成三层架构[12]来提供较好的性能、可扩展性和可靠性，三层结构分为：(1)数据库层；(2)协议服务器、扩展层和信息库(Repository)组成的中间层；(3)客户端。图 2-2 显示了 Oracle 9iFS 的体系结构：

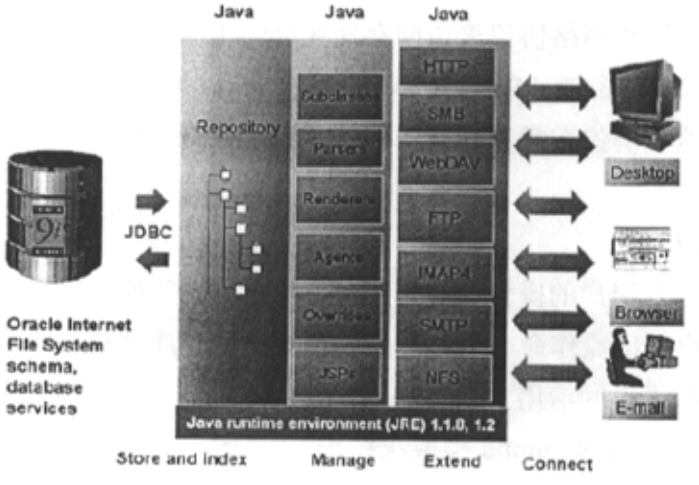

图 2-2 Oracle 9iFS 体系结构

1、数据库层

Oracle 9iFS 将所有数据(文件内容、元数据、用户和组等信息)存储在 Oracle 数据库中，而不是存储在本地磁盘上。

2、中间层

在中间层中包含了信息库、扩展层和协议服务器。

- 信息库

信息库是管理存储在 Oracle 数据库中数据的 Java 类集合，将存储在数据库中的行和列的内容表示成标准的文件系统的文件和文件夹。信息库运行在自己的 Java 虚拟机上，通过使用标准的 Oracle JDBC 调用读写 Oracle9i 数据库中的大对象(LOB)的方法来获取和存储文件。

- 扩展层

支持强大的扩展功能是 Oracle 9iFS 一个重要的特性，在扩展层中包含了基本的功能类，如：子类(Subclasses)、分析器(Parsers)、解析器(Renderers)、代理(Agents)、覆盖(Overrides)等，Oracle 9iFS 信息库中的所有类在 Java API 开发包中予以公开，企业可以利用这些开发包为企业构建特殊的应用。

- 协议服务器

中间层的最后一层是协议服务器，是客户端与 Oracle 9iFS 信息库通信的协议服务器集合。Oracle 9iFS 是一个更容易访问的文件系统，可以通过许多种网络协议来访问存储在 Oracle 9iFS 中文件和文件夹，包括：HTTP、SMB、WebDAV、FTP、IMAP4、NFS、WCP、SMTP 和 NTFS 等协议。利用这些协议在常见的操作系统上都能对 Oracle 9iFS 中的文件进行访问。

3、客户端

客户端是用户的操作界面，客户端可以通过多种通信协议与信息库交流，用户发出操作命令后，命令被传递到信息库并执行相应操作，最终将操作结果返回给用户。

2.2.3 Oracle.ifs.beans 类层次结构

Oracle 9iFS 是一个 Java 应用，通过面向对象的方法管理存储在数据库中的所有对象。Oracle 9iFS Java API 为开发者提供了开发内容管

理应用程序的接口，Oracle 9iFS Java API 中的类按照功能组织成包，共组成 25 个包，其中 oracle.ifs.beans 包为用户创建应用程序提供了主要的类。其中 LibraryObject 类是 oracle.ifs.beans 包中所有类的基类，其它类都是从它继承而来，它提供了所有类的基本功能。LibraryObject 类包含了三个基本的子类：PublicObject 类、SchemaObject 类和 SystemObject 类。其它类在这三个子类的基础上进行扩展，类层次结构如图 2-3：

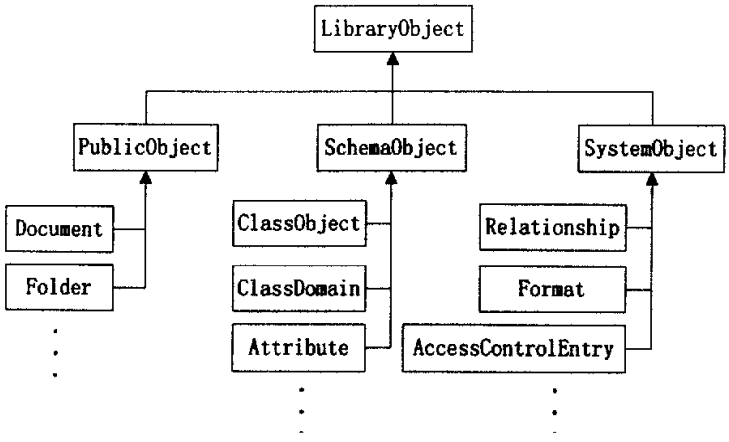

图 2-3 oracle.ifs.beans 类层次结构

1、PublicObject 类

PublicObject 是继承 Lirary0bject 的抽象子类，PublicObject 中定义的属性和方法在文档、文件夹、类别等类中被实现。PublicObject 类管理信息库中文档、文件夹等永久信息，供终端用户应用程序使用。

2、SystemObject 类

SystemObject 类是为管理 PublicObject 类提供支持的辅助类，不能够被终端用户直接操纵，例如，继承 SystemObject 类的 Relationship 类对象管理着 PublicObject 类对象之间的关联关系。只有具有管理员权限的用户才能修改 Relationship 类来改变 PublicObject 类对象之间的关联关系。

3、SchemaObject 类

SchemaObject 类也是一个抽象类，Oracle 9iFS 用继承它的子类来管理内容类型的对象，并用它来记录每一个内容类型的元数据。

2.2.4 Oracle 9iFS 访问控制

1、Oracle 9iFS 访问控制列表结构

在 Oracle 9iFS 中，利用访问控制列表(Access Control List——ACL)实现了更细粒度、更灵活的安全控制。ACL 是访问控制项条目(Access Control Entries-ACEs) 的集合，每个 ACE 又对应一组许可绑定(Permission Bundles)。在 Oracle 9iFS 中每个 ACL 是独立定义的，可以应用到多个 PublicObject 对象上，PublicObject 对象与 ACL 通过一个属性进行关联。图 2-4 显示了 ACL 结构模型：

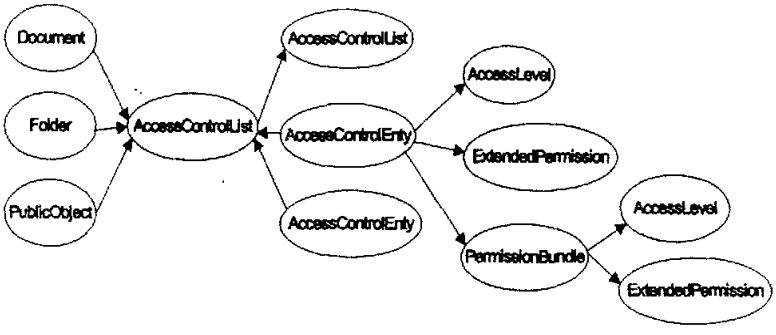

图 2-4 Oracle 9iFS ACL 结构模型

2、Oracle 9iFS 访问控制

对文件内容的安全访问是最基本的要求，Oracle 9iFS 从两个层次来管理文件的访问控制：信息库层访问控制和对象层访问控制。

- 信息库层访问控制

在 Oracle 9iFS 的 DirectoryUser 对象中存放着能够访问信息库的所有用户，当用户试图访问信息库中的信息时，用户首先必须提供登录名和口令，Oracle 9iFS 验证用户是否有权访问信息库，只有合法的用户才能建立与信息库的连接，访问信息库。

- 对象层访问控制

通过信息库层的访问控制验证后，当用户试图对一个 PublicObject 对象进行操作时，如：文件或文件夹，Oracle 9iFS 首先利用 ACL 对当前用户的操作进行验证，只有通过 ACL 验证，用户的操作才能被执行，否则操作被拒绝。

2.2.5 Oralce 9iFS 高级搜索

1、Oracle 9iFS 高级搜索

由于 Oracle 9iFS 是基于 Oracle 数据库的，因此它能够很容易的利用数据库的一些搜索功能。Oracle 9iFS 在 oracle.ifs.search 包中定义了丰富的搜索功能类，利用它们可以开发出比传统文件管理器更丰富的高级搜索功能。

2、搜索对象模型

在利用 oracle.ifs.search 包中的搜索功能类组装查询条件时，与利用 SQL 关键词组装 SQL 查询语句类似，并且它们之间存在着一定的对应关系，请参考附录二。

在 Oracle 9iFS 搜索对象模型中，使用四个主要的 Java 对象来创建查询、执行查询、操纵结果和保存搜索标准，对应的搜索功能类为：SearchSpecification、Search、SearchResultObject 和 SearchObject。图 2-5 显示了创建查询时要使用的搜索功能类及它们的相互关系。利用这些搜索功能类组装查询条件时，最终由 Oracle 9iFS 自动的将其解析成能够在数据库中执行的 SQL 语句。

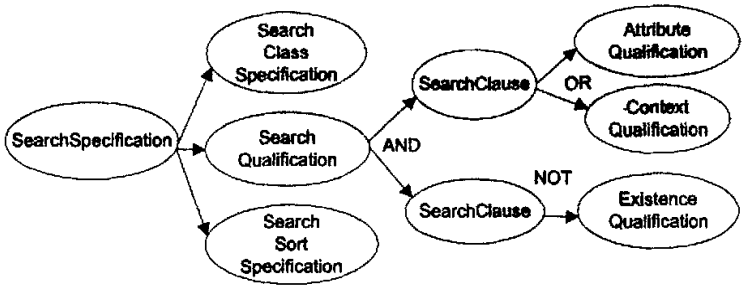

图 2-5 搜索功能类的相互关系

2.3 Oracle 9iFS 中 Java 类

Oracle 9iFS 是一个 Java 应用，采用 Java 面向对象开发基础的类层次结构，通过面向对象的方式将存储在数据库中的 Schema 表封装成 Java 类，而存储在 Schema 表的数据将以一个个实例呈现给用户，用户通过对封装好的类和方法进行操作就能实现对存储在数据库中的文件

进行操作，此时用户不需要知道数据在数据库中的组织方式。

在 Oracle 9iFS 中，一个封装好的 Java 类对应着数据库中的一个 Schema 表。当创建一个 Oracle 9iFS 系统时，将会在数据库中创建多个对应 Java 类的 Schema 表和实例，Oracle 9iFS 的所有数据将存放在这些表中。下面说明 Java 类与 Schema 表对应关系、类的继承和类的关联关系在数据库中的表现形式。

Schema 表中每列对应着 Java 类对象的一个属性，而每条记录则相当于类的一个实例。类的继承在数据库中的表现形式为：当创建一个子类继承父类时，则为子类创建一个表，表中 ID 列与父类表中的 ID 列保持一致，在子类中新增的属性，也就是在子类表中新增相应的列，如图 2-6 中的父类表 ODM_PUBLICOBJECT 与子类表 ODM_DOCUMENT，每创建一个子类实例时，则分别在父类和子类中创建一条记录，其中 ID 值保持一致。而类的关联关系是通过表与表的字段进行关联来实现，如图 2-6 中表 ODM_PUBLICOBJECT 中 Owner 与表 ODM_DIRECTORYUSER 中 ID 的关联。

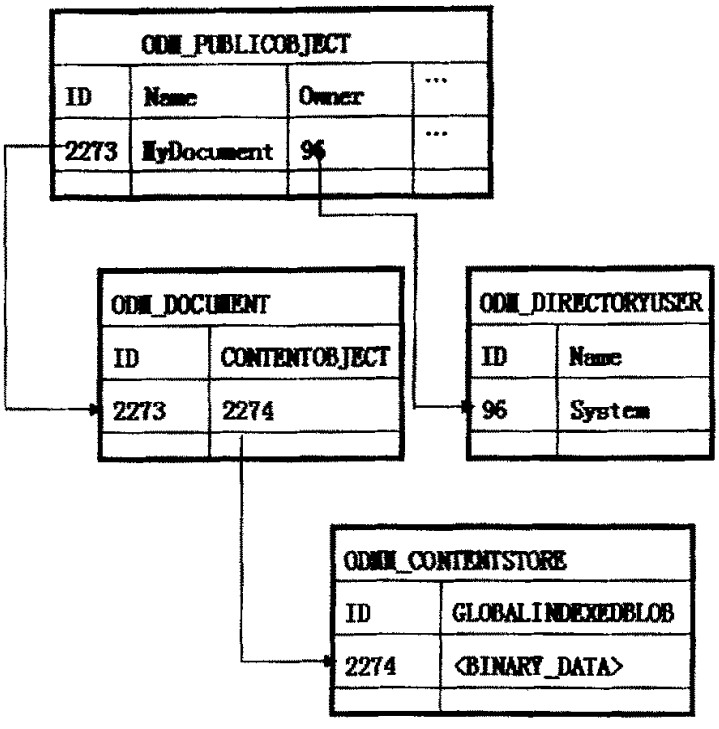

图 2-6 Oracle 9iFS 部分 Schema 表间关系示意图

2.4 Java 中的正则表达式简介

2.4.1 正则表达式

正则表达式最早是由数学家 Stephen Kleene 于 1956 年在对自然语言的递增研究成果的基础上提出来的。具有完整语法的正则表达式被使用在字符的格式匹配方面，后来被应用到熔融信息技术领域。自从那时起，正则表达式经过几个时期的发展，现在的标准已经被 ISO(国际标准组织)批准和被 Open Group 组织认定。正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。一个正则表达式就是由变通字符（如字符 a 到 z）以及特殊字符（称为元字符）组成的文字模式。

2.4.2 Java 中的正则表达式

SUN 公司从 JDK1.4 版本之后提供了 java.util.regex 正则表达式 API 包，JAVA 程序员可以免去找第三方提供的正则表达式库的周折。java.util.regex 是一个用正则表达式所订制的模式来对字符串进行匹配工作的类库包，它包括两个类：Pattern 和 Matcher。Pattern 是一个正则表达式经编译后的表现模式[13]；Matcher 对象是一个状态机，它依据 Pattern 对象做为匹配模式对字符串展开匹配检查。

2.5 Shell 扩展的基础知识

Windows Shell 提供了丰富的资源，这些资源不仅包括命令行方式，还包括图形方式的功能调用，例如资源管理器、计算机管理等包含的各种功能都是 Windows Shell 的组成部分，所以，利用这些资源来设计系统可以达到良好的效果。Shell 资源按照应用方式来划分，大概可以分为两类：一是调用 Shell 资源来构建自己的系统；二是利用 Shell 提供的机制，对 Shell 进行扩展。

本文主要利用 Shell 的扩展，“Shell 扩展”从字面上分两个部分，Shell 与扩展。Shell 指 Windows Explorer(资源管理器)，而扩展则指

当某一预先约定的事件(如在以.doc 为后缀的文件图标上右键)发生时由Explorer调用执行的代码。因此一个Shell扩展就是一个为Explorer添加功能的 COM 对象。所谓的 Shell 扩展就是能够添加某种功能到Windows Shell 的COM对象[14]。

Shell 扩展是个进程内服务器(运行在 Explorer 进程内),它由资源管理器来调用并响应系统外壳内发生的事件。Shell 扩展程序实现了一些接口来处理与 Explorer 的通信,在响应系统外壳内发生的事件之前,资源管理器查找这些注册的模块并加载它们。VC 的 ATL(活动模板库)是设计 Shell 扩展最快捷的方法。本文就是采用 ATL 模板来开发 COM 对象的。

Shell 扩展的类型很多,每种类型都在各自不同的事件发生时被调用运行,Shell 扩展常见的扩展类型有:Context 菜单扩展处理器、属性页扩展处理器、拖放目标扩展处理器、放置目标扩展处理器、提示信息扩展处理器,表 2-1 做了详细描述[14]。在本文中主要利用 Context 菜单扩展处理器来扩展文件右键的 Context 菜单。

表 2-1 常见的 Windows Shell 扩展接口及描述

类型	何时被调用	描述
Context 菜单扩展处理器(ContextMenu)	用户右键单击文件或文件夹对象时	允许在外壳对象的Context菜单中增加新的菜单项
属性页扩展处理器(Property Sheet)	要显示一个文件对象的属性框时	向文件类属性对话框中加入另外的属性表页。也适用于控制面板应用
右拖拽(Right drag and drop)	用户用右键拖放文件对象到文件夹窗口或桌面时	允许在右拖拽后出现的Context菜单中增加新的菜单项
左拖拽(Left drag and drop)	用户拖动Shell对象并将它放到一个文件对象上时	决定在外壳内用鼠标左键拖拽一个对象到另一个对象上时做什么
提示信息扩展处理器(Infotip)	用户将鼠标盘旋于文件或其它Shell对象的图标上时	当鼠标移到某个文件类型文档上时显示简短文本信息

第三章 Z-Explorer 的总体设计

3.1 系统目标

经过对内容管理功能的分析，并结合课题组科研项目的需要，我们研发的 Z-Explorer 文件管理器要达到以下几个目标：

- 支持内容管理并能充分发挥内容管理的优越性

文件管理器不仅要实现标准文件系统的管理功能，还要实现基于内容管理的一些特有的高级功能，如更多的属性操作、全文搜索、正则搜索、类别搜索、版本控制、ACL 安全控制等功能，要充分体现出内容管理的优势。

- 为课题组研发的语义文件系统提供支持

课题组正在研发一种新型的语义文件系统，该文件系统是基于内容管理的文件系统上开发的，为用户提供了更多的语义管理功能，进一步挖掘了信息的潜在价值。为此，必须有相应的文件管理器来管理语义文件系统中的文件，Z-Explorer 一个重要的目标就是满足这方面的需求。

- 提供简单的接口

文件管理器由客户端模块、接口映射模块和服务器端模块三个部分组成。服务器端模块封装了具体的操作逻辑，并为客户端模块提供调用的接口，为了能够简单地将这些接口映射到客户端，在设计这些服务器端接口时，要力求简单。

- 友好的用户界面

绝大多数用户熟悉传统文件系统的文件管理器的使用方法，为了符合用户的操作习惯、降低 Z-Explorer 的操作难度，Z-Explorer 文件管理器力求与传统文件管理器的界面风格保持一致，使用户能够很容易地学会使用 Z-Explorer 文件管理器。

- 跨平台

为了满足使用不同操作系统的用户需求，促进内容管理的进一步推

广，Z-Explorer文件管理器要实现跨平台性。因此我们采用Java语言开发，这样Z-Explorer可以在所有具有JRE环境的操作系统上运行。

3.2 Z-Explorer文件管理器的功能分析

根据对内容管理功能的分析，并结合Oracle 9iFS的开发平台，Z-Explorer主要增强以下几个核心功能：查看/设置文件属性、高级搜索、版本控制、类别设置和安全管理。经过对这些功能的分析总结，并根据这些功能的相互关联关系，将系统分为三个功能模块：查看/设置文件属性模块、高级搜索模块和版本控制模块。

3.2.1 查看/设置文件属性模块

基于内容管理的文件一般存储在数据库中，每个属性对应着数据库中表的一列，每个文件相当于表中的一条记录，这样能够充分利用数据库表为文件定义更多的属性。在基于数据库的文件系统中，充分的利用了这一特性，能够为文件定义更多的、更有价值的属性，如课题组正在研发的语义文件系统，正是为文件定义更多的具有语义性的属性。

传统文件管理器是无法对这些属性进行管理的，因此，Z-Explorer必须解决这个的问题。在查看/设置文件属性功能模块中，除要实现对文件名称、作者、创建时间等常见属性的操作外，还要实现对文件的版本、ACL和类别等属性进行操作。

3.2.2 高级搜索模块

基于内容管理的文件存储在数据库中，这样它充分的发挥了数据库强大的查询功能，它支持了比传统文件系统更有价值的搜索方式，如全文搜索、类别搜索和关联搜索等，同时还可以指定搜索结果的对象类型、是否包含子对象、是否包含版本化文件等。在Z-Explorer中必须实现这些高级的查询功能。

另外，目前很多高级用户都要求具有正则搜索功能，这样能够灵活的设置更复杂的搜索标准。但本课题使用的后台数据库是Oracle 9.0.1版本，在这个版本中不支持正则表达式查找功能。为了满足用户这方面的需求，我们要在Z-Explorer中要实现正则查找的功能，提高用户的

检索效率。

3.2.3 版本控制模块

基于内容管理的文件系统支持版本控制功能。版本操作是基于内容管理的一个重要功能，通过版本操作用户可以为不同阶段的文档保存一个版本，从而保存文档创作的历史轨迹，在需要的时候，用户可以检出历史版本，对其进行修改。同时，通过检入/检出功能可以实现多个用户协同编辑一个文档。因此，Z-Explorer 要实现版本操作功能，包括对文件版本化、检出、检入、重命名、删除等功能。

另外，为了进一步实现对文件/文件夹的安全控制和用户的协同工作，Z-Explorer 还要实现对文件/文件夹的锁定和解除锁定的操作，这样被锁定的文件/文件夹就不会被他人修改，直到文件/文件夹被解除锁定。

3.3 Z-Explorer 体系结构

Z-Explorer 是利用 Oracle 9iFS 提供的 Java API 开发包开发的，根据 Oracle 9iFS 的体系结构和应用程序部署规则，我们将系统的体系结构设计成三个模块：服务器端模块(SFM)、接口映射模块(ZFS)和客户端模块(CFM)，系统的体系结构如图 3-1 中的阴影部分。

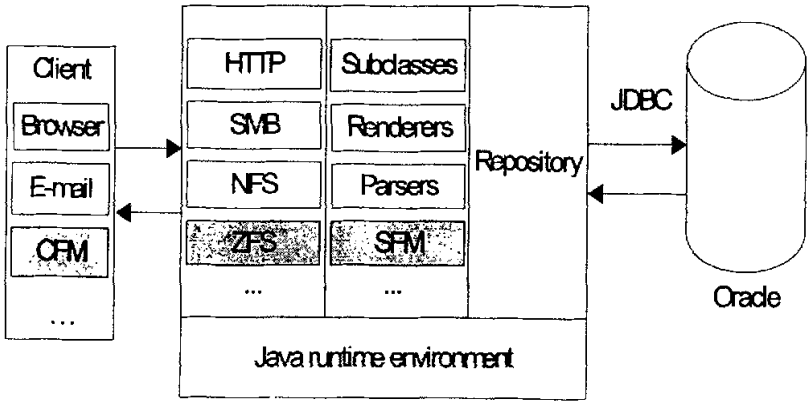

图 3-1 Z-Explorer 文件管理器的体系结构

1、服务器端模块(SFM)

服务器端模块部署在 Oracle 9iFS 服务器上，该模块实现对 Oracle

9iFS 中文件的具体操作逻辑，并为客户端模块提供了这些功能的调用接口，关于接口的详细说明参见本文附录一。这些接口通过接口映射模块映射到客户端，客户端模块通过本地系统调用的方式调用映射到客户端的接口，从而实现对服务器端接口的调用。SFM 接收 CFM 传过来的操作命令并调用相应的接口，然后对 Oracle 9iFS 中文件进行操作，并将执行结果返回给 CFM 模块处理，CFM 模块向用户显示必要的结果信息。SFM 模块接口在设计时，要力求简单，以降低 ZFS 模块接口映射的难度。

2、接口映射模块(ZFS)

Z-Explorer 客户端模块最终部署在传统的文件系统中，服务器端模块部署在 Oracle 9iFS 服务器中，因此必须将服务器端模块封装的内容管理接口挂载到客户端模块所在文件系统的虚拟文件系统上，供客户端模块进行本地调用。这部分功能由课题组其他人员完成，已不属于我的课题内容，为此下文不在详述。

3、客户端模块(CFM)

CFM 模块定义了可视化的用户操作界面。在此操作界面中，将 Z-Explorer 的功能通过可视化的方式提供给用户使用，实现了用户的操作逻辑。用户的操作请求，通过调用挂载到本地文件系统的接口，并通过接口映射模块传递到服务器端模块，SFM 模块接收到操作命令后，根据操作命令执行相应的操作，然后再通过 ZFS 模块将结果传递到 CFM 模块，最后，CFM 模块将操作结果显示给用户。

3.4 开发环境的选择及配置

3.4.1 开发环境的选择

1、Oracle 9iFS 开发平台

通过对内容管理功能的分析，我们发现 Oracle 公司提供的 Oracle Internet File System（Oracle 9iFS）支持了内容管理的基本功能，而且 Oracle 9iFS 也是一个非常好的二次开发平台，它提供了丰富的开发包和完整的开发文档，并且可扩展性好。

课题组正在研发的语义文件系统也是在 Oracle 9iFS 平台上实现

的，为了给语义文件系统提供可视化的用户操作界面，必须将Z-Explorer与语义文件系统整合起来，从而在Z-Explorer中支持对语义文件系统中文件的操作，因此我们也必须选择Oracle 9iFS开发平台。

最后，Oracle公司在内容管理方面雄厚的实力和强大的科研资金的投入，从Oracle 9iFS升级到Oracle CM和Oracle File 10g，这些无不反映了Oracle公司在内容管理领域快速成长的事实。因此，我们有理由相应Oracle 9iFS的开发平台，它能够保证我们科研的持续性。

2、JBuilder9开发工具

在系统的开发过程中，我们选择了JBuilder9开发工具，主要考虑到以下两个原因：

首先，Z-Explorer最终要实现跨平台的重要目标，而Java是实现跨平台的最好的开发语言，由它开发的系统可以在所有具有JVM环境的操作系统中运行。

其次，Oracle 9iFS开发平台遵循标准的Java开发模式，Oracle 9iFS平台本身以及提供的开发包都是用Java开发的。同其他Java应用类似，Oracle 9iFS在它的环境中包含面向对象开发基础的类层次。Oracle 9iFS Java API不仅支持了标准的文件操作功能，还支持了对内容管理的操作功能，为开发人员提供了强大的Java开发接口。

因此，为了利用Java语言开发Z-Explorer，并提高开发效率，我们采用了JBuilder9这个可视化开发工具。

3.4.2 开发环境的配置

Z-Explorer文件管理器是利用Oracle 9iFS提供的Java API开发包开发的，根据Oracle 9iFS应用程序部署规则，服务器端模块(SFM)和接口映射模块(ZFS)必须部署在Oracle 9iFS服务器上，因此，首先要安装Oracle 9iFS服务器。开发工具采用的是JBuilder9，开发文件管理器时，用到了一些Oracle 9iFS提供的Oracle 9iFS Java API开发包，在开发前必须将这些开发包加载到Z-Explorer工程中。除此之外，还要配置JDK的环境变量。Oracle 9iFS所有文件都存放在Oracle数据库中，当前使用的数据库版本为Oracle 9.0.1。

1、Oracle 9iFS 服务器[15]和 Oracle 数据库的安装[16]

在本开发环境中，根据安装文档，将 Oracle 数据库和 Oracle 9iFS 9.0.1 服务器独立的安装在两台 Windows 2000 server 平台上，并对 Oracle 9iFS 服务器进行适当的配制。

2、JBuilder9 环境配置[17]

当 Oracle 9iFS 服务器安装好后，一些参数存放在$ORA_HOME\ 9iFS \settings 目录下的配制文件中，在应用程序与 Oracle 9iFS 创建会话时会用到这些参数，因此，必须将$ORA_HOME\ 9iFS \settings 路径加到 JBuilder9 中的 JDK 环境中。

Z-Explorer 用到了一些 Oracle 9iFS Java API 开发包，需将这些开发包加载到工程中，用到的开发包有：

$ORA_HOME\ora90\9iFS\lib\repos.jar, $ORA_HOME\9iFS\lib\utils.jar, $ORA_HOME\9iFS\lib\adk.jar, $ORA_HOME\j2sdk1.4.1_02\lib\servlet.jar, $ORA_HOME\classes12.jar。

第四章 Z-Explorer 的设计与实现

从第三章中我们知道，Z-Explorer 被分成三个功能模块：查看/设置文件属性模块、高级搜索模块和版本控制模块。这三个功能模块是根据各功能之间的联系和用户的操作习惯划分。Z-Explorer 也是主要围绕这三个功能模块进行设计与实现。另外，每个功能模块都按照服务器端模块、接口映射模块和客户端模块的体系结构设计。下面分别介绍它们的详细实现过程。

4.1 共用类的设计与实现

在 Z-Explorer 中定义了一些共用类，如：ZfsBase、BaseUtilities、PublicObjectUtilities 等类，这些类定义了所有功能模块用到的方法和函数，我们将这些共用类统一放在 base 包中，为以后共用这些类提供方便。

在 Z-Explorer 中，执行每个功能时首先要创建一个会话，然后才能利用这个会话执行相应的操作。在 ZfsBase 类中定义了启动 Oracle 9iFS 的 LibraryService 服务和创建会话的方法，Oracle 9iFS 中 LibraryService 的一个实例相当于创建会话的“工厂”，在创建会话之前，先运行起 LibraryService 服务，运行该服务的方法是 startService()。然后这个 LibraryService 服务就像运行起来的“工厂”进行“生产”会话，生产会话的方法为 establishSession()。

还有一些如 disconnectSession()、cleanup()等公共方法也都在这些公共类中定义，这样在其它模块中就可以共用这些方法。

4.2 查看/设置文件属性模块

基于内容管理的文件系统中文件的属性不仅包括了标准文件系统中文件的属性，还包括大量其本身特有的属性和用户自定义的属性。文件的属性可分为四种相对独立的信息：普通属性信息、版本信息、类别

信息、安全控制信息。这部分功能实现了对这些属性的查看和修改，在用户的操作界面中，我们将这些信息显示在不同的 Tab 页中，这样有利于功能的实现，更有利于用户对属性进行操作。下面详细介绍每种属性操作的具体实现。

4.2.1 文件普通属性的管理

1、文件普通属性的查看

Oracle 9iFS 中文件的普通属性都是从它们的父对象 PublicObject 继承的，在 PublicObject 对象中定义了所有文件的公共属性，如名称、创建者等。要想获取这些属性，首先要通过 findPublicObjectByPath() 获取 PublicObject 对象，然后利用该对象提供的方法来获取公共属性。由于 Oracle 9iFS 中这些公共属性较多，为了更好地将这些属性显示给用户，在用户操作界面中，我们采用 JBuilder 的 JdbTable 控件来显示属性信息。

2、文件普通属性的修改

有些文件属性的值在文件创建后就永远不能被修改，如文件创建日期、创建者等。因此，这些属性的值在 **Z-Explorer** 中不允许被修改。另外，对于已版本化的文件，只能修改版本化文件中最后一个版本的相关属性值。这里需要注意的是，文件历史版本的属性和内容都是只读的，用户不能对它们修改。

下面是修改文件描述属性值的示例代码：

```
//获得当前操作的文件 PublicObject 对象
PublicObject po = folderUtil.findPublicObjectByPath(objectPath);
if(po.isVersioned){
  //文件已被版本化,对最后一个版本属性进行修改
  versions[versions.length-1].setDescription(newValue);
}
else
{
  //文档未版本化,直接对文件的属性进行修改
  po.setDescription(newValue);
}
```

3、文件属性的增加和删除

在 Oracle 9iFS 中可以容易的为某类文件（如 Document 文件）新

增属性。新增属性时，首先取得文件的 ClassObject 对象，然后根据用户指定的属性名称和属性类型创建一个 AttributeDefinition 属性定义对象，最后将其添加到 ClassObject 对象中，这样就为文件创建了新的属性。主要代码如下：

```
ClassObject co = session.getClassObjectByName(Type);
AttributeDefinition attdef = new AttributeDefinition(session);
attdef.setAttributeByUpperCaseName(Attribute.NAME_ATTRIBUTE,
    AttributeValue.newAttributeValue(AttributeName));
attdef.setAttributeByUpperCaseName(Attribute.DATATYPE_ATTRIBUTE,
    AttributeValue.newAttributeValue(AttributeType));
co.addAttribute(attdef);
```

删除文件属性操作比较简单，只要调用文件的 removeAttribute() 方法就可以实现。

4.2.2 显示已版本化文件的版本信息

已版本化的文件可能有多个版本，每个版本都有相应的属性信息，如版本的创建时间、检入/检出时间、创建者和版本的描述等信息。在 Z-Explorer 的用户界面中，用两个 JdbTable 控件来显示版本相关的信息：一个显示文件目前拥有的版本序列，另一个显示特定版本的属性信息。用户可以将历史版本的内容检出，存在副本中，然后就可以对副本内容进行修改，修改结束后作为一个新的版本检入。这里用户还可以将历史版本的内容复制到本地磁盘。查看指定版本属性的操作类似于 4.2.1 中显示属性的操作，下面将介绍如何将指定版本复制到本地磁盘和删除指定版本操作的实现。

- **将指定版本复制到本地磁盘**

这部分功能实现的基本思路是：获取指定版本的内容，然后将内容存储到 FileDialog 对话框指定的本地磁盘文件中。为了使服务器端接口便于映射到客户端，将版本内容读成一个 Java 类型的输入流 (InputStream)，利用 Java 类型的参数将内容传递到客户端。图 4-1 显示了将指定版本内容复制到本地磁盘的具体过程。

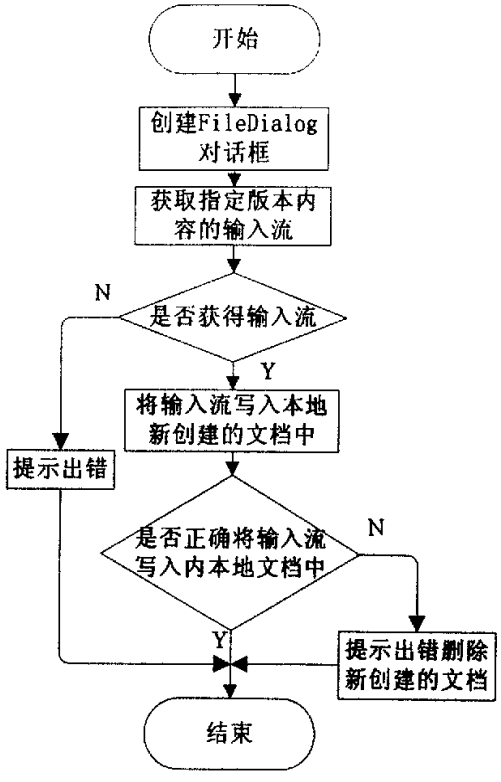

图 4-1 将指定版本内容复制到本地磁盘的流程图

在将指定版本内容复制到本地磁盘的过程中，有两个重要的操作：其一是如何将版本内容转换成 Java 类型的输入流，其二是如何将输入流写到本地磁盘新创建的文件中。

这里我们首先获取指定版本的 Document 对象，然后利用 getContentStream()方法将 Document 对象的内容读到 Java 类型的 InputStream 输入流中。

已获取的版本内容输入流传到客户端后，利用下面的代码将其写到指定的本地磁盘文件中。

```
if (input != null) {
  FileOutputStream fout = new FileOutputStream(fileNew);
  int b = input.read();
  while (b != -1) {
    fout.write(b);
    b = input.read();
  }
}
```

4.2.3 查看/设置文件的类别属性

在基于内容管理的文件系统中，通过类别可以更容易地管理企业中的文件。在为文件设定类别后，可以将其存放在文件系统中任意位置。以后只要通过类别搜索，就可以很容易地将相同类别的文件搜索出来进行管理。因此，在 Z-Explorer 中要为用户提供类别设置和查找功能。

一个文件可以被设定一种或多种类别。当文件被设定类别后，该文件的类别对象就会拥有一些自己的属性信息。Z-Explorer 提供了查看这些类别属性信息的界面。

在显示文件类别属性时，首先要判断当前文件是否已被版本化，未版本化文件可以直接获取类别类型数组，若文件已被版本化，应取得文件的最后一个版本，然后再获取类别类型数组，从类别类型数组中再逐个获取类别的属性。图 4-2 显示了公共对象与类别的关联关系(ODM_INTERMEDIAAUDIO 为 ODM_CATEGORY 的一个子类)。每当为文件指定类别时，就会在 ODM_CATEGORY 对象中添加一条 ASSOCIATEDPUBLICOBJECT 记录（该记录只是文件的 ID 号），并在 CATEGORY 的子类 INTERMEDIAAUDIO 中添加一条记录。图 4-2 说明 ID 为“28706”文件已被设置了一个类别类型为 INTERMEDIAAUDIO 的类别。

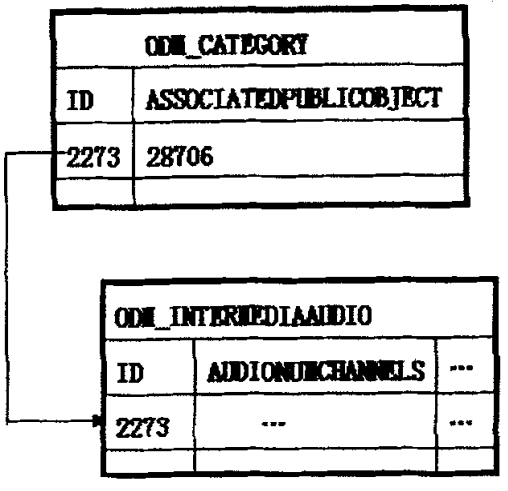

图 4-2 公共对象与类别的关联关系

在查看文件类别属性时，首先用 getCategories()方法获取文件的类别数组，然后利用 getEffectiveClassAttributes()方法获取文件每

一类别的具体属性值。在删除和添加文件类别时，主要用到 categories[i].delete()和 addCategory()两个方法。

4.2.4 查看/设置文件的 ACL

基于内容管理的文件系统通过ACL实现了对文件更细粒度的安全访问控制。ACL 是由一个或多个访问控制条目（Access Control Entry——ACE）组成，每个 ACE 由被授权人和与之对应的一组许可权限组成，如图 4-3。

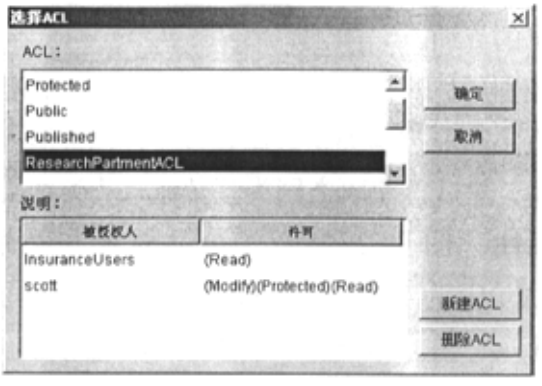

图 4-3 ACL 对象信息

在基于内容管理的文件系统中，每个文件都拥有一个 ACL，在文件被创建时，系统会为其自动分配一个默认 ACL，授权用户可以查看/修改文件的 ACL，并允许创建新的 ACL。文件 ACL 和 ACE 的查看比较容易，只要通过 getAcl()和 getAccessControlEntrys()即可实现。下面主要介绍修改和新建 ACL 的实现方法。

1、修改文件的 ACL

在修改文件 ACL 时，首先要从系统中取出所有能够供用户使用的 ACL，当没有满足用户要求的 ACL 时，用户可以创建一个新的 ACL，并将其赋给文件，图 4-4 显示了修改 ACL 的流程。

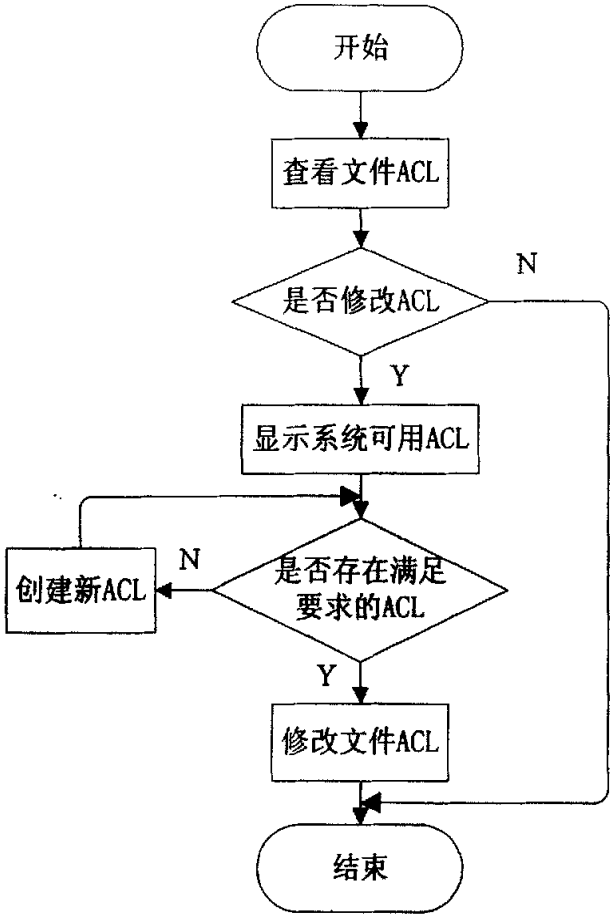

图 4-4 修改文件 ACL 的流程图

修改文件 ACL 的具体实现方法如下：

- **显示文件系统中所有当前用户能够使用的 ACL**

在系统中有两类 ACL：一类提供给用户使用的 ACL，另一类提供给系统内部使用的 ClassACL。为了获取用户能够使用的 ACL，首先通过 getSharedAccessControlListCollection()方法获取系统中所有 ACL，然后再用 getClassAccessControlListCollection()方法获取所有 ClassACL，最后在所有 ACL 中去除 ClassACL 并将剩下的 ACL 提供给用户使用。

- **将文件的 ACL 修改成新选择的 ACL**

修改文件的 ACL 时，首先要根据已选择的 ACL 名称来获取 ACL 对象，

然后重新设置文件的 ACL。如果修改文件夹的 ACL，则允许用户选择是否修改文件夹中所有文件的 ACL。如果只修改文件夹的 ACL，则只需对文件夹对象调用 setAcl()方法，修改其 ACL。但如果要修改文件夹中所有文件的 ACL，则需要通过递归调用的方法完成修改动作，其方法如下：

```
//修改文件夹类型的 ACL
public void changeFolderAcl(…) {
  //获取文件夹中的所有文件，将所有文档和子文件夹都设为相同的访问控制
  PublicObject[] items = folder.getItems();
  int length = (items == null) ? 0 : items.length;
  for (int i = 0; i < length; i++) {
    遍历文件夹中的每个对象，如果是子文件夹，则设置子文件夹的 ACL
    并递归调用此方法来设置子文件夹中所有文档和文件夹的 ACL。
    if (items[i] instanceof Folder) {
      设置当前文件夹的 ACL，并递归调用 changeFolderAcl()方法
    }else {
      遍历的对象是文档类型，直接修改 ACL
    }
  }
}
```

2、新建 ACL

用户可以利用系统默认的 ACL 来管理其他用户对文件的访问，但更多的时候，用户需要利用满足自己管理要求的 ACL 来管理其他用户对文件的访问。因此，在 Z-Explorer 中，为用户提供了创建 ACL 的方法。创建 ACL 的操作步骤有些烦琐，首先要创建一个 ACL 定义，并指定 ACL 定义的相关属性值，再创建一个或多个 ACE 定义，为每个 ACE 定义设置授权对象属性(DirectoryUser 和 DirectoryGroup)和权限属性，然后将 ACE 定义指定给 ACL 定义，最后调用创建 ACL 的接口来创建 ACL，这里不再详述。

4.3 高级搜索模块

Z-Explorer 文件管理器要想实现复杂的高级搜索功能，首先，在客户端模块必须提供给用户设置复杂查询条件的接口，使用户能够方便地设置复杂的查询条件。在服务器端模块将用户设置的查询条件组装成一个完整的搜索类，然后自动转换成能够在数据库中执行的 SQL 查询语

句，并在数据库中执行搜索，最后搜索结果将以可视化的形式显示给用户。

4.3.1 用户界面实现

搜索界面为用户设置复杂查询条件提供了可视化的接口。为了便于用户有条理的设置查询条件，根据查询条件的相互关联关系，通过六个 Tab 页面为用户提供设置接口：名称和位置、日期和大小、高级、内容、类别和关联查找，如图 4-5 所示：

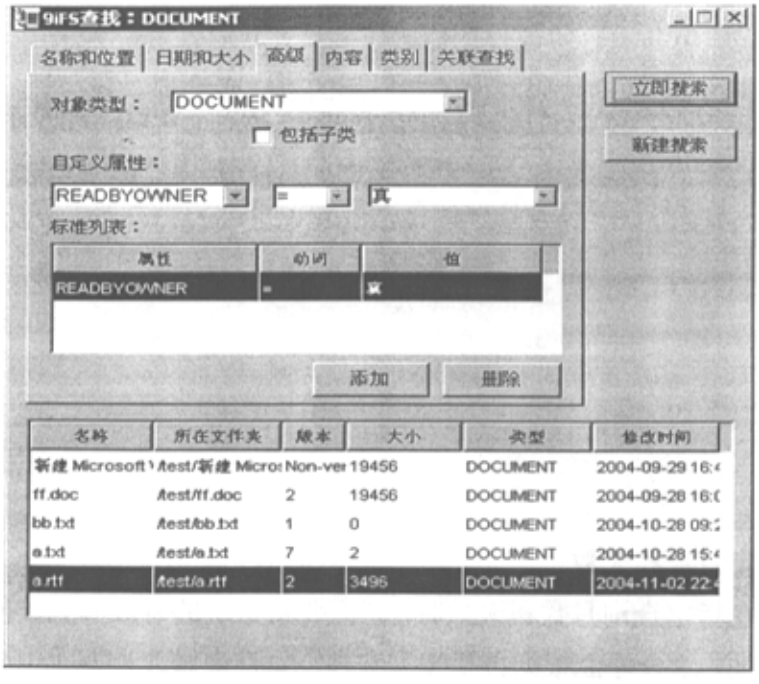

图 4-5 文件查找界面

下面介绍在实现搜索操作界面过程中用到的关键技术。

1、名称和位置/日期和大小页面实现

在这两个设置搜索条件的页面中，为用户提供了设置公共属性查询条件的操作界面。这些属性是所有文件都具有的，因此，我们将这些属性查询条件的设置固定在一些控件中，而不需要动态的生成，如名称、说明、所有者、搜索路径、日期和文件大小。而更多属性查询条件的设置，可以在其它 Tab 页中进行设置。

在属性操作界面实现的过程中，设置日期条件的操作界面比较复杂，界面中向用户提供了设置文件的修改、创建或截止日期查询条件的接口，同时提供了多种设定日期查询条件的方法，如：指定具体时间段、指定月数或天数，这就给日期合法性验证增加了难度。

• 具体日期控制

具体日期的控制是通过 JBuilder9 的 MaskFormatter 和 JformattedTextField 两个控件实现的，这两个控件保证日期的输入格式的正确性。但并没有验证日期值的正确性，如输入月份大于 12、日期大于 31 日等，下面的代码实现了对日期的进一步验证。

```
//根据输入取出年、月、日的数值，验证年和月的合法性
if (year < 1 || month < 1 || month > 12 || date > 31) {
  //输入不合法
}
//年和月合法，进一步验证输入的日期是否超过指定月最大日期
Calendar cal = Calendar.getInstance();
cal.set(year, month - 1, 1);
int maxDay = cal.getActualMaximum(Calendar.DAY_OF_MONTH);
//验证用户输入的日期是否超过了指定月份的最大日期号
if (date < 1 || date > maxDay) {
  //输入不合法
```

• 指定月数和天数

用户在指定查询月数和天数时，则需要将指定的月数和天数换算成具体的日期，这样才能根据日期搜索文件。如查询修改日期在 100 天以内的文件，这时就需要将 100 天换算成具体的日期，换算方式如下：

```
while ( (currentDate - countDate) < 0) {
  countDate -= currentDate;
  //年和月份做相应的变化，同时获取下一个月的最大日期号 nextMonthMaxDate
  ...
  currentDate = nextMonthMaxDate;
}
```

2、高级页面实现

图 4-5 中高级 Tab 页面中，允许用户指定搜索结果的文件类型，在传统文件系统中没有这样的功能。Z-Explorer 能够动态地获取系统中所有文件类型，并能够动态获取各种类型的属性；用户可以指定搜索的

结果类型，并根据指定类型的扩展属性设置搜索条件，这样就能够更快速、更准确的搜索文件。

在 Z-Explorer 中，允许用户搜索的对象类型为 Document 和 Folder 及它们的子类对象类型。我们通过 getClassObjectCollection()方法获取系统中所有对象类型，然后利用 isSubclassOf()方法逐个判断，剔除那些不是 Document 和 Folder 及其子类的对象类型，最后将结果绑定到对象类型下拉框中。当用户选择某对象类型时，Z-Explorer 利用 getExtendedClassAttributes()方法动态地从系统中搜索出该对象类型的扩展属性，供用户设置查询条件。

当用户选择一种对象类型，并选择其中一个属性设置查询条件时，首先根据属性的类型，设置该属性比较操作符。系统中每种属性值类型有对应的比较操作符，日期/数值类型的操作符有“>”、“>=”、“=”、“<”、“<=”，字符类型的操作符有“=”和“包含”，布尔类型的操作符只有“=”。用户在操作符下拉框中选择一种操作符，然后设置查询条件的值，设置好就可以添加到搜索标准列表中，作为一个查询条件，用户可以同时设置多个这样的查询条件，以精确的查找文件。

3、内容页面实现

在 Oracle 9iFS 中，新创建的文档没有被索引，若要全文搜索这些新建文档，必须更新文档的索引。在内容 Tab 页面中提供了“更新索引”的功能按钮，点击此按钮并在弹出对话框中输入连接数据库的字符串、Oracle 9iFS 在数据库中对应模式的用户名及密码，然后更新全文索引，其部分代码如下：

```
Statement stmt = dbConnection.createStatement();
CallableStatement proc = null;
stmt.addBatch("update odmz_context_router set contentprocedure=contentproc
    edure");
stmt.addBatch("commit");
stmt.executeBatch();
proc = dbConnection.prepareCall("call ctx_ddl.sync_index('ifs_text')");
proc.execute();
```

4、类别页面实现

类别页面的实现类似于高级页面的实现，但两者有一个重要的区

别：类别页面中类别属性绑定的是类别的所有属性，包括父类属性；而在高级页面中只绑定对象的扩展属性。这是因为高级页面中对象的父类属性查询条件已在名称和位置/日期和大小 Tab 页面中提供了设置查询条件的接口，而类别属性没有在其它页面中提供类似的设置查询条件的接口，因此，要通过 getEffectiveClassAttributes()方法获取类别的所有属性。

5、关联搜索页面实现

在关联搜索页面中，用户可以设置与当前搜索对象类型一些属性相关联的关联对象的查询条件。当在高级页面中指定搜索结果类型时，该类型中可能有部分属性关联着另一个对象类型，如文件的创建者属性，该属性关联着另一个用户对象，创建者的信息都存储在用户对象的属性中，如果当前要根据创建者的一些信息搜索文件时，关联搜索就派上了用场，能够满足这样的要求。

在关联搜索页面中，将当前搜索对象中所有与其它对象关联的属性动态的绑定到关联属性下拉框中，在关联对象的下拉列表框中列出所有可以被关联的对象。这样，就可以将属性与关联对象关联起来，并设置关联对象的查询条件。设置关联对象属性的查询条件与高级页面中设置查询条件的方法类似。

6、正则查询条件与非正则查询条件

在 Z-Explorer 中，充分利用了 Java 的正则查找工具包，为高端用户提供了正则搜索的功能。在设置搜索标准的用户操作界面中，用户可以输入两种类型的搜索条件：正则查询条件和非正则查询条件。用户输入查询条件时不需要区分它们，Z-Explorer 能够根据用户输入的查询条件自动地识别它们，并通过不同的变量传递给服务器端模块。

4.3.2 服务器端非正则查询条件搜索功能的实现

服务器端程序接收到客户端的正则查询条件和非正则查询条件后，首先根据非正则查询条件从文件系统中搜索满足非正则查询条件的结果。服务器端程序根据非正则条件的不同调用不同的 Search 类方法来创建搜索对象。通过 Search 类提供的方法创建搜索对象时，开发人员

不需要知道数据库中表的结构和组织方式。在执行组装好的搜索对象时，系统会自动的将搜索对象转换成 SQL 语句，在 Oracle 数据库中执行查询，搜索出满足非正则查询条件的结果。然后再经过正则条件的过滤得到最终搜索结果，图 4-6 显示了构造查询语句的大致流程。

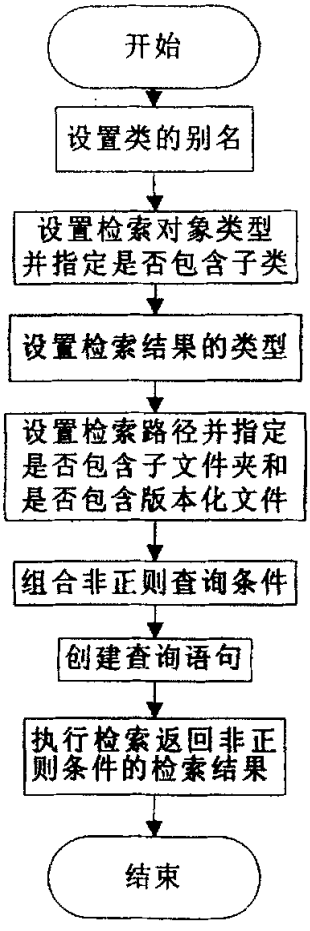

图 4-6 构造非正则查询条件语句的流程图

创建查询语句实例同编写 SQL 语句一样，要设置 SELECT、FROM 和 WHERE 条件等部分，下面详细的介绍构造查询语句的过程：

1、创建搜索标准的 SELECT 和 FROM 条件实例

查询语句实例的 SELECT 和 FROM 条件存放在 SearchClassSpecification 实例中，因此首先创建一个 SearchClassSpecification 实例，调用实例的 addResultClass()和

addSearchClass()方法来设置 SELECT 和 FROM 条件。

• **创建别名**

这里的别名类似 SQL 语句中 SELECT 后面的搜索字段的别名。在创建搜索标准实例时，所有对对象的访问都是通过对象的别名进行访问，而不是对象的实名。在创建别名时，有一个 SearchClassCount 整型变量，每增加一个别名时，变量值加 1，以确保别名的唯一性。

• **设置搜索结果的类型**

设置搜索结果类型相当于设置 SQL 语句中 SELECT 要搜索的字段，也就是指定搜索返回结果的对象类型。这里指定的搜索结果类型最后转换成 SQL 语句在数据库搜索时，也就将该结果类型所有属性对应的数据库表字段作为 SELECT 要搜索的字段。设置方法为：

```
m_SearchClassSpec.addResultClass(aliasName);
```

• **设置搜索的范围**

SearchClassSpecification类中addSearchClass()方法是用于指定当前搜索从哪些对象中进行搜索，相当于SQL语句中FROM关键词所指定的搜索范围。这里可以指定从多个对象中进行搜索，类似于对数据库中多个表进行搜索。通过addJoinCondition()方法还可以设置嵌套的子查询语句，从而实现更复杂的查询。addSearchClass()方法使用如下：

```
// className 是类名称，aliasName 是上面创建的别名，recBeh 指定是否包含子类
SearchClassSpecification.addSearchClass(className,aliasName,recBeh);
```

• **设置检索路径并指定是否包含子文件夹和版本化文件**

在基于内容管理的文件系统中执行搜索时，与标准文件系统一样可以指定在哪个文件夹中进行搜索，除此之外，它还允许用户指定在搜索结果中是否包含版本化文件。核心代码如下：

```
searchUtils.addFolderRestrictCondition(folder,isIncludeSubFolder,alias,
isIncludeVersion);
```

2、创建搜索标准的 WHERE 条件实例

在基于内容管理的文件系统中，一个文件是由多个实例对象组合而成的，实例对象通过关联属性连接起来，实例对象的属性值存放在 Oracle 数据库的不同表中(如 2.3 节所述)。

在这些实例中存在着两种类型的类对象关系：继承关系(如

Document 与 PublicObject 之间)和关联关系(如 PublicObject 与 DirectoryUser)。设置这两种关系的查询条件的方法是不同的，在对对象本身属性设置查询条件时，只需设置属性名称、比较操作符和匹配值；而在对关联对象属性设置查询条件时，不仅要设置关联对象的属性名称、比较操作符和匹配值，还要将关联对象与被关联的关联字段连接起来。另外，在设置文档内容作为全文搜索条件时，需要以一种单独的方式来处理，在设置 WHERE 查询条件时，只有这三种不同设置方式，下面分别对它们进行介绍：

- **设置对象本身属性查询条件**

对象本身属性包括父类属性及其扩展属性，在匹配到满足查询条件的属性时，就等于找到了要查找的对象。这类查询条件可以直接调用 SearchUtilities 类中的 addAttributeCondition()方法创建查询条件。该方法的包含四个参数，分别为：alias 变量是对象的别名，attributeName 变量是对象的属性名称，operator 变量是条件操作符，value 变量是查询条件值。该方法原型是：

```
addAttributeCondition(alias, attributeName, operator, value);
```

- **设置关联对象属性查询条件**

在对关联对象属性设置查询条件时，除了创建一个关联对象属性查询条件，还要指定搜索对象与该关联对象之间的关联关系，即通过什么属性进行关联。设置这类查询条件时，首先，要设置关联对象的属性查询条件，方法同上面设置对象本身属性查询条件。然后，通过 addJoinCondition()方法设置搜索对象与关联对象之间的关联关系。这个方法中有四个参数，分别为：alias 参数是搜索对象别名，duAlias 参数是关联对象别名，aliasAttribute 参数和 aliasAttribute 参数指定这两对象通过搜索对象的 aliasAttribute 属性与关联对象的 duAliasAttribute 属性进行关联。方法原型如下：

```
addJoinCondition(alias, aliasAttribute, duAlias, duAliasAttribute);
```

- **设置文档内容查询条件**

在前面已经介绍过利用 Oracle Text 为文档类型的文件创建内容索引，创建好文档的内容索引，就可以进行全文搜索。利用 Oracle 9iFS

Java API 提供的方法可以很容易的创建基于内容管理的全文搜索条件，其方法如下：

```
//创建一个 ContextQualification 的实例，并设置内容搜索条件
ContextQualification cq = new ContextQualification();
cq.setQuery(query);
//此时相当于设置了 ContentObject 对象的一个查询条件
//下面设置 ContentObject 对象与 Document 对象之间的关联字段
addJoinCondition(docAlias, Document.CONTENTOBJECT_ATTRIBUTE,
                    contentObjectAlias, contentObjectID);
```

- **组合查询条件**

上面创建的所有 WHERE 条件被暂时存放在一个 m_SearchQualifications 向量中，这里需要将向量中的所有查询条件组装到一个 SearchQualification 实例中，在多于两个查询条件时，采用“与”连接的方式将它们连接起来，具体实现如下：

```
SearchQualification searchQual = null;
if (m_SearchQualifications.size() == 0){
  //没有任何查询条件
  searchQual = null;
}
else if (m_SearchQualifications.size() == 1){
  //只有一个查询条件
  searchQual =(SearchQualification) m_SearchQualifications.elementAt(0);
}
else{
  //有多个查询条件，将多个查询条件连接起来
  Enumeration e = m_SearchQualifications.elements();
  searchQual= (SearchQualification) e.nextElement();
  while (e.hasMoreElements() ){
    SearchQualification scl = (SearchQualification) e.nextElement();
    searchQual = new SearchClause(searchQual, scl, SearchClause.AND);
  }
}
```

3、创建查询语句

创建好查询条件实例后，就可以利用这些实例创建一个完整的查询语句实例，方法如下：

```
//创建一个查询语句。
AttributeSearchSpecification searchSpec;
//如果当前查询包含内容查询，则创建一个内容查询语句，
```

```
//否则创建一个属性查询语句
if (m_ContextQualification != null){
  ContextSearchSpecification cs = new ContextSearchSpecification();
  cs.setContextClassname(m_ContentObjectAlias);
  searchSpec = cs;
}
else{
  searchSpec = new AttributeSearchSpecification();
}
//设置查询的类，类似于 SQL 语句中 from 关键词指定在哪些表中查询
searchSpec.setSearchClassSpecification(m_SearchClassSpec);
//设置查询条件，类似于 SQL 语句中 where 关键词指定的条件
searchSpec.setSearchQualification(searchQual);
```

4、执行查询

执行查询时首先要获得一个已建立好的会话，利用会话执行创建的查询语句，检索结果存储在 Search 对象中，然后，就可以从 Search 对象中逐个取出搜索结果。执行搜索的方法如下：

```
Search search = new Search(session, seasession);
```

4.3.3 服务器端正则查询条件搜索功能的实现

由于当前 Oracle 数据库版本还不支持正则表达式查询功能，我们只能利用 Java 提供的正则查询功能间接的支持用户的正则查询。在 4.3.2 节中搜索出了所有满足非正则查询条件的结果，如果此次搜索用户设置了正则查询条件，那么还要利用 Java 提供的正则查找功能对结果逐个验证，剔除那些不满足正则查询条件的结果，最终返回满足正则查询条件的结果给客户端程序处理，图 4-7 示意了一次完整搜索的过程：

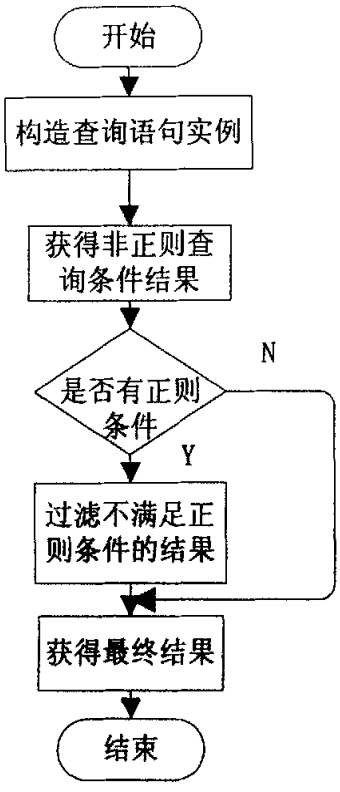

图 4-7 搜索流程图

剔除那些不满足正则查询条件的方法：对满足所有非正则条件的结果逐个验证，如对象的名称查询条件是正则查询条件，利用 Java 正则查找功能逐个验证搜索结果中对象的名称，剔除不满足正则条件的结果，只有名称符合正则条件的对象才是最终结果，代码实现如下：

```
//遍历非正则条件的搜索结果，验证其是否满足正则条件
search.next();
   ...
if (是否合法) {
  //合法的结果，将其添加到最终结果中
}
```

4.4 版本控制模块

在版本控制模块中实现了对文件的版本化、检入/检出、取消检出、重命名、删除、锁定和解除锁定等功能，这些功能用户使用较频繁，因此，每个功能将对应 Context 菜单中的一个菜单项，用户可以快速的调

用这些功能。下面主要介绍文件的版本化、检入/检出的实现，其它功能的实现方法比较类似。

4.4.1 版本化文件过程的描述

对文件进行版本化操作，也就是为此文件创建三个管理文件多个版本的对象(Family、Version Series、VersionDescription)，当通过检入/检出为版本化文件创建一个新的版本操作时，实际上就是为文件创建两个新的对象(VersionDescription、Document)[18]，用户也可以随时删除任一指定的版本(最新版本除外，版本化的文件最新的一个版本不能被删除)。如果用户欲删除已版本化的文件，则可通过“删除版本化文件”功能实现，执行此操作后，将彻底删除该文件的所有对象：Family、Version Series、VersionDescription 和 Document，图 4-8 示意了版本化文件的过程。

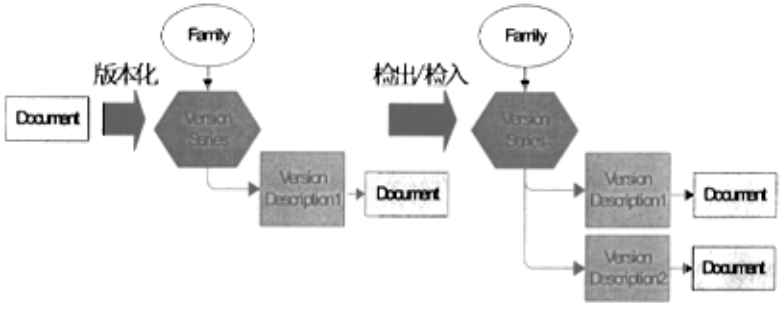

图 4-8 版本化文件过程

4.4.2 版本化操作的实现

文件版本化时，首先要进行合法性检查，包括检查用户是否有权限执行版本化操作、文件是否已被锁定、是否已版本化等。只有通过合法性检查，才能对文件进行版本化，版本化流程如图 4-9：

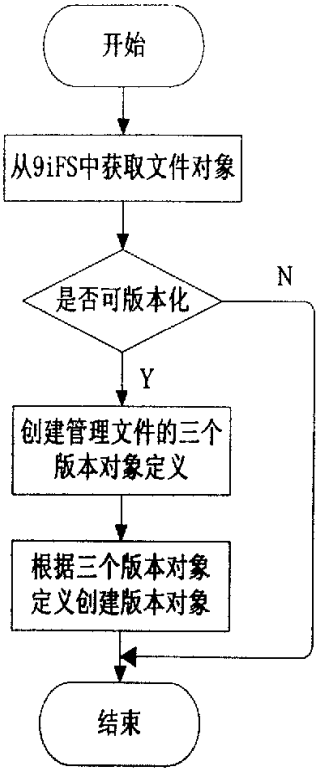

图 4-9 文件版本化的流程图

对文件进行版本化操作的步骤如下：

1、根据文件的路径名从系统中获取文件对象

```
//获取文件对象
FolderUtilities folderUtil = new FolderUtilities(session, logger);
PublicObject po = folderUtil.findPublicObjectByPath(objectPath);
```

2、判断版本化操作合法性

在判断版本化操作的合法性时，除对文件本身的合法性进行检查外，如文件是否被锁定、是否已版本化等，还要检查文件的所有父文件夹的合法性，如果父文件夹被锁定，则文件不能被版本化。

3、创建版本化对象的定义

通过上面的合法性检查后才可以对文件进行版本化，首先创建管理文档的三个版本化对象的定义，实现的示例代码如下：

```
//创建一个 VersionDescription 对象的定义，并设置属性值
VersionDescriptionDefinition vdd = new
VersionDescriptionDefinition(session);
```

```
......
//创建 VersionSeries 对象的定义，并设置属性值
VersionSeriesDefinition vsd = new VersionSeriesDefinition(session);
......
//创建 family 对象的定义，并设置属性值
FamilyDefinition fd = new FamilyDefinition(session);
......
vsd.setFamilyDefinition(fd);
```

4、创建文件的版本化对象

```
VersionDescriptionDefinition vdd = new
VersionDescriptionDefinition(session);
```

4.4.3 检入/检出

文件版本化后，就可以通过检入/检出为文档创建一个新的版本，保存文件创作的历史轨迹。新建一个版本时，首先要检出文件，然后对检出的文件进行修改，最后将修改好的文件检入，生成一个新的版本。通过检入/检出创建一个新版本的示意图如图 4-10：

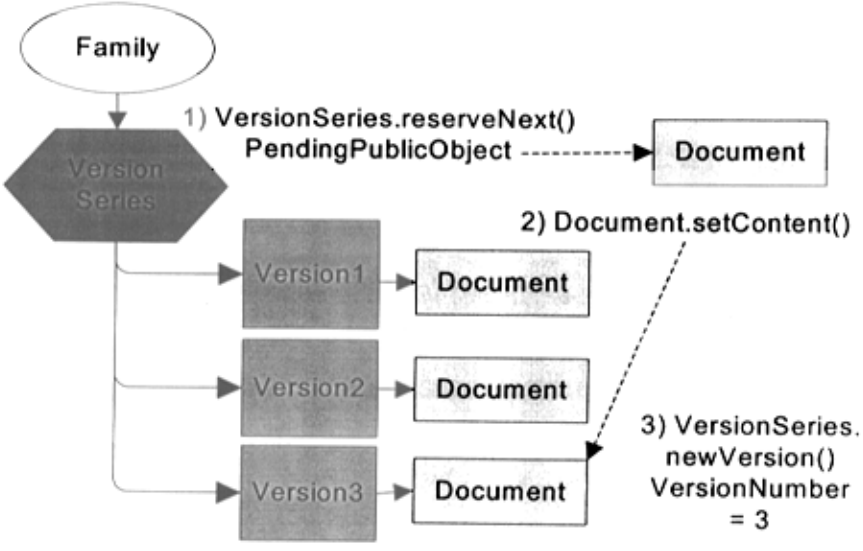

图 4-10 创建一个新版本的示意图

4.4.4 检出功能的实现

在检出版本化文件时，也要进行合法性检查，通过合法性检查后，才能调用 reserveNext ()方法将文件检出，文件检出时将为文件创建一个副本(PendingPublicObject)对象，创建好副本后，调用

setPendingPublicObjectContent()方法将副本的内容设置成指定检出版本的内容。

4.4.5 检入实现

用户对检出文件内容的修改将保存在副本中，当用户欲将修改后的内容作为一个新的版本保存起来时，可以调用检入操作将文件的最新内容保存到文件的新版本中。在执行检入操作时，和其它操作一样，要检查检入操作的合法性，包括检查文件是否已被版本化、是否已被检出、当前用户是否是检出该对象的用户等。只有通过合法性检查，才能调用newVersion()方法执行检入操作，检入操作是在已版本化的文件中添加一个新的版本，也就是在Family对象中新创建一个Document对象和一个VersionDescription对象，检入过程的流程如图4-11：

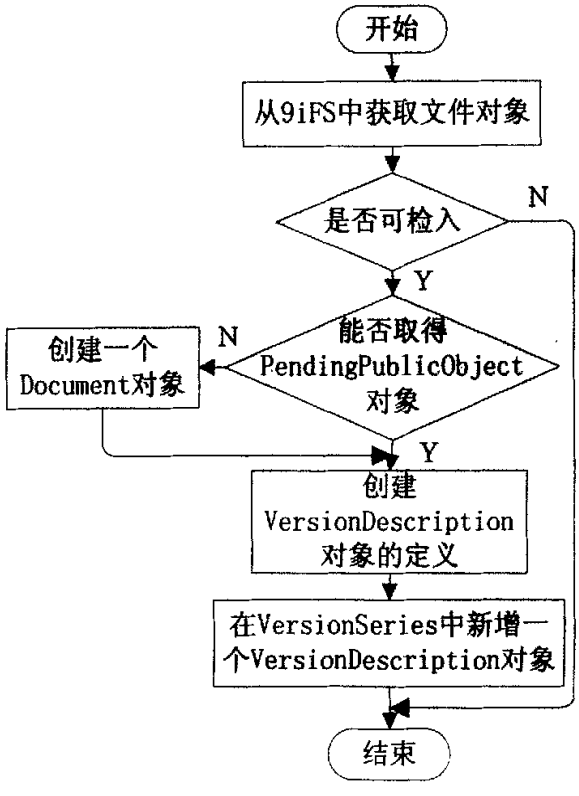

图4-11 检入操作流程图

检入操作的详细步骤如下：

1、获取要检入文件对象、并检查检入操作的合法性。

2、通过合法性检查后，获取检出文件的副本文档。在对检出文件执行检入操作时，首先获取 PendingPublicObject 对象作为即将新建版本的文档对象，如果检出的版本中没有内容时，获取 PendingPublicObject 时将会取得一个空对象，此时需要创建一个空白文档对象作为新建版本的文档对象。

```
Document doc = (Document) verUtils.getPendingPublicObject(po);
if(doc == null)
  doc = docUtils.createDocument(objectName,Document.CLASS_NAME);
```

3、创建新版本的 VersionDescription 定义，并设置其属性和新版本的文档对象，与版本化文件操作不同的是，此时不需要创建 VersionSeries 和 Family 对象，具体代码如下：

```
//创建新版本文档的 VersionDescription 对象的定义并设置其属性
VersionDescriptionDefinition vdd = new VersionDescriptionDefinition
(session);
.....
vdd.setPublicObject(doc);
```

4、调用 newVersion()方法为文件创建新的版本，覆盖已检出的版本。

```
VersionDescription vd = vs.newVersion(vdd);
```

4.5 Context 菜单的实现

4.5.1 重要的成员函数

Context 菜单的扩展就是要实现一对 COM 接口：一个提供特定行为，另一个用于初始化目的。因此，编写 Context 菜单句柄(Context Menu Handler)必须实现 IcontextMenu 和 IshellExtInit 两个接口[19]。除了 Iunknown 接口所定义的函数之外，Context 菜单句柄还需要用到 QueryContextMenu、InvokeCommand 和 GetCommandString 三个非常重要的成员函数。

1、QueryContextMenu 函数

每当系统要显示一个文件对象的 Context 菜单时，它首先要调用该函数。为了在 Context 相关菜单中添加菜单项，在该函数中调用

InsertMenu 函数。

2、GetCommandString 函数

当鼠标指针移到一个 Context 菜单项上时，在当前窗口的状态条上将会出现与该菜单项相关的帮助信息，此信息就是系统通过调用该函数获取的。

3、InvokeCommand 函数

当用户选定了某个 Context Menu Handler 登记过的菜单项后，该函数将会被调用，系统将会传给该函数一个指向 LPCMINVOKECOMMANDINFO 结构的指针。在该函数中要执行与所选菜单项相对应的操作。

4.5.2 Context 菜单的实现

1、初始化接口

当 shell 扩展被加载时，Explorer 将调用已实现的 COM 对象的 QueryInterface()函数以取得一个 IShellExtInit 接口指针，该接口仅有一个方法 Initialize()[20]，其函数原型为：

```
HRESULT IShellExtInit::Initialize (LPCITEMIDLIST pidlFolder,LPDATAOBJECT
            pDataObj,HKEY hProgID );
```

Explorer 使用该方法传递各种各样的信息。PidlFolder 是用户所选择操作的文件所在的文件夹的 PIDL 变量。一个 PIDL[指向 ID 列表的指针]是一个数据结构，它唯一地标识了在 Shell 命名空间的任何对象，一个 Shell 命名空间中的对象可以是或者不是真实的文件系统中的对象。pDataObj 是一个 IDataObject 接口指针，通过它可以获取用户所选择操作的文件。hProgID 是一个 HKEY 注册表键变量，可以用它获取 DLL 的注册数据。

在初始化接口中获取被右击选择的文件名，这里文件名的存放格式与拖放文件到带 WS_EX_ACCEPTFILES 风格的窗口时使用的文件名格式是一样的，因此可以使用同样的 API(DragQueryFile())来获取文件名。首先获取包含在 IdataObject 中的数据句柄：

```
if(NULL==lpdobj){
  return E_INVALIDARG;
}
```

```
FORMATETC feFmtEtc ={CF_HDROP,NULL,DVASPECT_CONTENT,-1,TYMED_HGLOBAL};
STGMEDIUM mdmSTG = { TYMED_HGLOBAL };
// 在数据对象内查找 CF_HDROP 型数据
if(FAILED(lpdobj->GetData(&feFmtEtc, &mdmSTG))){
  return E_INVALIDARG;
}
// 获得指向实际数据的指针
HDROP hDrop = (HDROP)GlobalLock(mdmSTG.hGlobal);
// 检查是否取得数据的指针.
if(NULL==hDrop){
  return E_INVALIDARG;
}
int nDropCount = ::DragQueryFile(hDrop, 0xFFFFFFFF, NULL, 0);
// 取得第一个文件名并把它保存在类成员 m_szFile 中
if(1==nDropCount){
  ::DragQueryFile(hDrop, 0, m_pzDropFile, MAX_PATH);
}
GlobalUnlock(hDrop);
//释放资源
::ReleaseStgMedium(&mdmSTG);
return S_OK;
```

由于 Shell 扩展运行在 Explorer 进程内，要是扩展代码崩溃了，Explorer 也会随之崩溃，在 Win 9x 上，这种崩溃可能导致需要重启系统，因此要进行错误检查，特别是指针的检查。要是返回 E_INVALIDARG,Explorer 将不会继续调用以后的扩展代码；要是返回 S_OK，即初始化了扩展，Explorer 将再一次调用 QueryInterface()获取另一个下面就要添加的接口指针：IcontextMenu。在 IcontextMenu 中定义了三个方法，它们分别为：QueryContextMenu、GetCommandString 和 InvokeCommand。利用这些方法可以添加 Context 菜单项、在状态栏上提示帮助信息、并执行用户的操作。

2、修改 Context 菜单

在 QueryContextMenu 方法中定义了要添加的菜单项，QueryContextMenu 方法的原型如下：

```
HRESULT CFolderShlExt::QueryContextMenu(HMENU hmenu,UINT indexMenu,UINT
idCmdFirst, UINT idCmdLast, UINT uFlags)
```

下面通过在 Context 菜单中添加一个“将文件版本化”的菜单项来

演示 QueryContextMenu 添加菜单项的方法。

```
if(!(uFlags & CMF_DEFAULTONLY)){
  ::InsertMenu(hmenu, indexMenu, MF_BYPOSITION,
  idCmdFirst, _T("将文件版本化"));
  if ( NULL != m_CMbmp ){
    SetMenuItemBitmaps(hmenu, indexMenu, MF_BYPOSITION, m_CMbmp, NULL);
  }
  return MAKE_HRESULT(SEVERITY_SUCCESS, 0, USHORT(1));
}
return MAKE_HRESULT(SEVERITY_SUCCESS, 0, USHORT(0));
```

对于 Context 菜单扩展而言，只有一个值是重要的：CMF_DEFAULTONLY，该标志告诉 Shell 命名空间扩展保留默认的菜单项，这时 Shell 扩展就不应该加入任何定制的菜单项，这也是为什么此时要返回 0 的原因。如果该标志没有被设置，就可以修改菜单了(使用 hmenu 句柄)，并返回 1 告诉 Shell 添加了一个菜单项。

3、在状态栏上显示提示帮助

下一个要被调用的 IContextMenu 方法是 GetCommandString()。该方法是用户在浏览器窗口中右击文件，或选中一个文件后单击文件菜单时，状态栏会显示提示帮助。GetCommandString()函数将返回一个帮助字符串供浏览器显示，GetCommandString()的原型为：

```
HRESULT CSimpleShlExt::GetCommandString(UINT idCmd, UINT uType,
        UINT *pwReserved, LPSTR pszName, UINT cchMax)
```

4、执行用户的选择

IContextMenu 接口的最后一个方法是 InvokeCommand()。当用户点击新添加的菜单项时该方法将被调用，其函数原型是：

```
HRESULT IContextMenu::InvokeCommand ( LPCMINVOKECOMMANDINFO lpici);
```

CMINVOKECOMMANDINFO 结构带有大量的信息，但在此只关心 lpVerb 和 hwnd 两个成员。lpVerb 参数有两个作用：它或是可被激发的 verb(动作)名；或是被点击的菜单项的索引值。hwnd 是用户激活扩展的菜单所在的浏览器窗口的句柄。

当选择一个菜单项的操作功能时，将调用 InvokeCommand()方法，首先检查当前选中菜单是哪一项操作，然后利用 WinExec()来执行

Z-Explorer 的相应操作，程序实现如下：

```
// 如果 lpVerb 实际指向一个字符串，忽略此次调用并退出.
if ( 0 != HIWORD( pCmdInfo->lpVerb ))
  return E_INVALIDARG;
switch ( LOWORD( pCmdInfo->lpVerb )){
  // 点击的命令索引
  TCHAR szCmdLine[2000];
  case 0: {
    strcpy(szCmdLine,TEXT("java Main 0 "));
    strcat(szCmdLine,m_pzDropFile);
    WinExec(szCmdLine,SW_HIDE);
    return S_OK;
  }
  break;
  ......
}
```

5、注册 Shell 扩展

Z-Explorer 在部署时用到两种 COM 组件，其一是用于在文件的 Context 菜单上添加菜单项，其二是用于在文件夹的 Context 菜单上添加菜单项。每种 COM 组件写好后要将其注册到注册表中，这样才能在相应的文件/文件夹的 Context 菜单中添加自定义的菜单项，下面以注册向文件的 Context 菜单上添加菜单项的 COM 组件为例，说明 COM 组件的注册方法。

在开发添加 Context 菜单的 COM 组件时，ATL 自动生成注册 COM DLL 服务器的代码，但这只是让其它程序可以使用新创建的 DLL，为了告诉浏览器使用新的扩展，需要在文件类型的注册表的“HKEY_CLASSES_ROOT\txtfile”键下注册扩展。

在“txtfile”键下，有个名为 ShellEx 的键保存了一个与文本文件关联的 Shell 扩展的列表。在 ShellEx 键下，ContextMenuHandlers 键保存了 Context 菜单扩展的列表。每个扩展都在 ContextMenuHandlers 下创建了一个子键并将其默认值设为扩展 COM 的 GUID。所以，需要创建下面 Shell 扩展的键：HKEY_CLASSES_ROOT\txtfile\ShellEx\ContextMenuHandlers\CMDocumentShlExt，并将其值设为 ATL 创建的 GUID 值，具体实现如下：

```
HKCR{
    NoRemove txtfile{
        NoRemove ShellEx{
            NoRemove ContextMenuHandlers{
                ForceRemove CMDocumentShlExt = s 'GUID'
            }
        }
    }
}
```

每一行代表一个注册表键，"HKCR"是 HKEY_CLASSES_ROOT 的缩写。NoRemove 关键字表示当该 COM 服务器注销时该键不用被删除。最后一行有些复杂，ForceRemove 关键字表示如果该键已存在，那么在新键添加之前应先删除该键，这行脚本的余下部分指定一个字符串，它将被存为 CMDocumentShlExt 键的默认值。

第五章 平台搭建与实例演示

本文将 Z-Explorer 文件管理器部署在 Windows 操作系统平台上进行实例演示。

5.1 运行环境

硬件环境：能运行 Windows 2000 Server、Oracle 9.0.1 数据库和 Oracle 9iFS 9.0.1 的计算机。

软件环境：Windows 2000、Oracle 9.0.1、Oracle 9iFS 9.0.1、具有 JRE 运行环境。

5.2 平台搭建

文件管理器的运行需要用到 Oracle 数据库、Oracle 9iFS 等软件和 JRE 环境，它们可根据自带安装文档进行安装。Z-Explorer 则以 Jar 包的形式发布，在 Windows 环境中进行部署时，只要将 Jar 包所在路径添加到 ClassPath 的系统环境变量中，然后在 Shell 扩展中实现对 Z-Explorer 的功能调用。

5.3 实例演示

Z-Explorer 文件管理器设计时考虑到了用户的操作习惯，整个界面风格类似于传统文件管理器的界面风格，用户无需培训就能够很快地学会使用，下面通过部分截图来演示它的一些功能。

5.3.1 Context 菜单

本文将 Z-Explorer 部署在 Windows 平台上进行演示，因此，必须通过 Shell 扩展在 Oracle 9iFS 的文件/文件夹上添加 Context 菜单，由于对文件和文件夹提供的功能不同，因而 Context 菜单也有差别，其区别如图 5-1 所示，左图是文件夹的 Context 菜单，右图是文件的 Context 菜单。

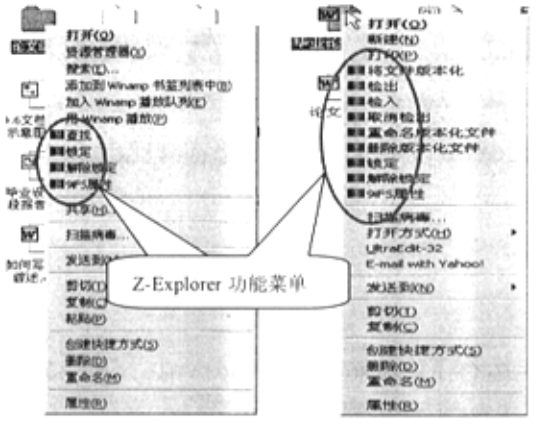

图 5-1 Z-Explorer 的 Context 菜单

5.3.2 登录

Z-Explorer 所管理的文件都存放在一个公共的信息库中，用户要访问文件时，首先要通过信息库的合法性验证，只有合法的用户才能调用 Z-Explorer，对存储在信息库中的文件进行管理。在首次使用 Z-Explorer 的功能时，为用户提供图 5-2 的登录界面，登录的信息将缓存在客户端，以便下次直接从缓存中读取登录信息进行登录，缓存信息直到客户端电脑关机时删除。

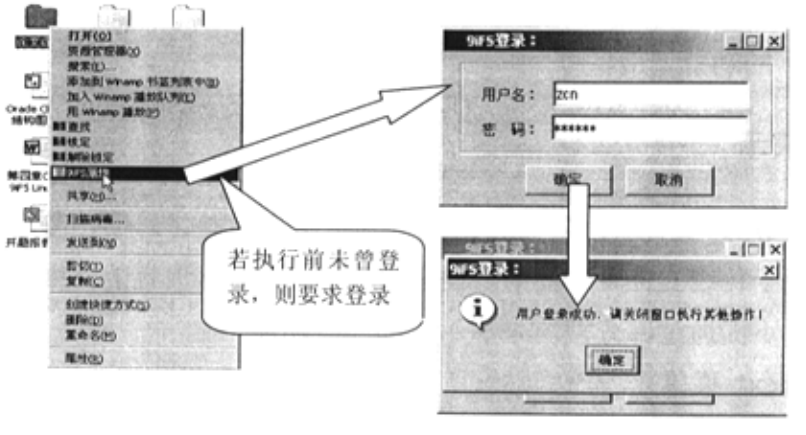

图 5-2 用户登录过程

5.3.3 新建文件属性

在 Z-Explorer 属性操作界面中，用户可以利用“新建属性”和“删除属性”按钮来定义文件的属性。为文件新建属性时，首先点击“新建

属性”按钮，在弹出对话框中输入新建属性的名称，并指定属性的类型，然后点击“确定”按钮，这样就为文件创建了新的属性，图 5-3 为文件新建了一个“总页数”属性。

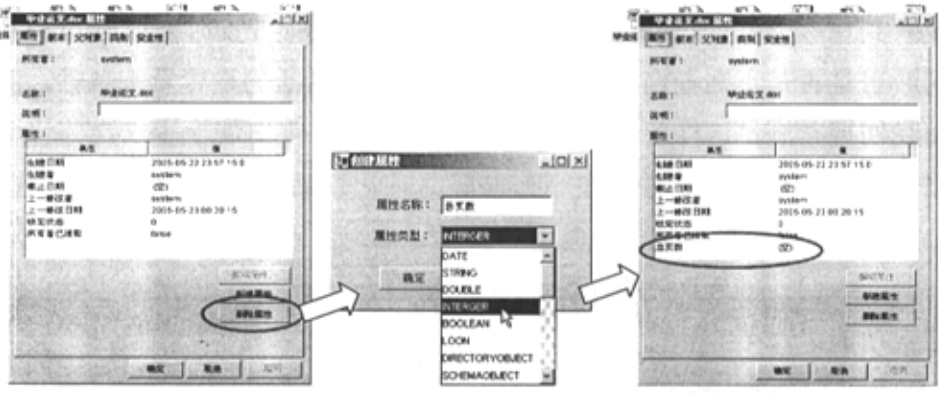

图 5-3 添加文件属性

5.3.4 高级搜索

Z-Explorer 提供了除传统文件系统提供的搜索功能外，还提供了全文搜索、正则搜索、类别搜索和关联搜索。并且在一次搜索中，可以进行组合条件搜索。下面分别演示这些高级的搜索功能。

1、全文搜索

全文搜索时，在查找操作界面的“内容”Tab 页中输入全文搜索的关键字，然后执行搜索，就可以将内容包含关键字的所有文件搜索出来，如图 5-4 所示：

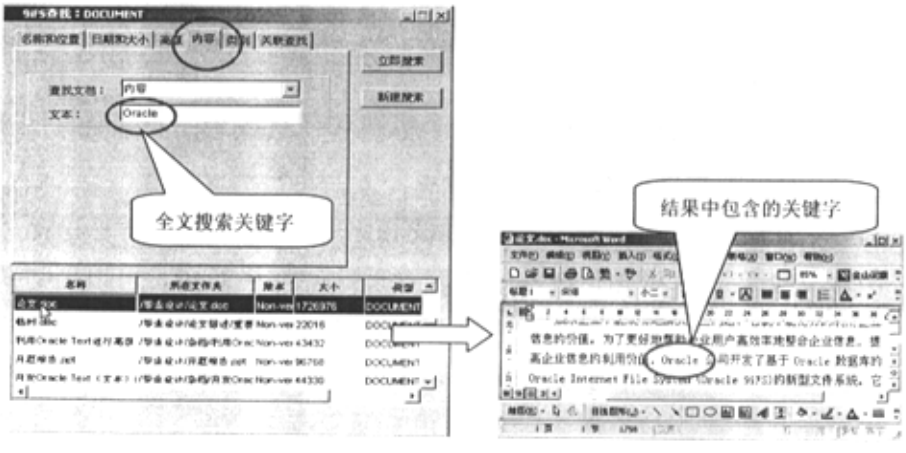

图 5-4 全文搜索

2、正则搜索

Z-Explorer 在“名称和位置”Tab 页中为高级用户提供了正则搜索功能，用户可以对字符串类型的属性构建复杂的正则查询条件，准确地搜索到目标文件。Z-Explorer 根据查询条件进行搜索时，能够自动的判断出是否是正则查询条件。因此，用户可以灵活的设置查询条件。假如我们搜索满足这样条件的文件：名称中“文件”出现至少一次并包含“管理器”,而且“文件”在“管理器”之前出现后缀名以“doc”或“txt”结尾的所有文件。其正则查询条件是：(.)*(文件)+(.)*管理器(.)*(doc|txt)。图 5-5 中的左图演示了正则搜索功能，右图演示了非正则搜索功能：

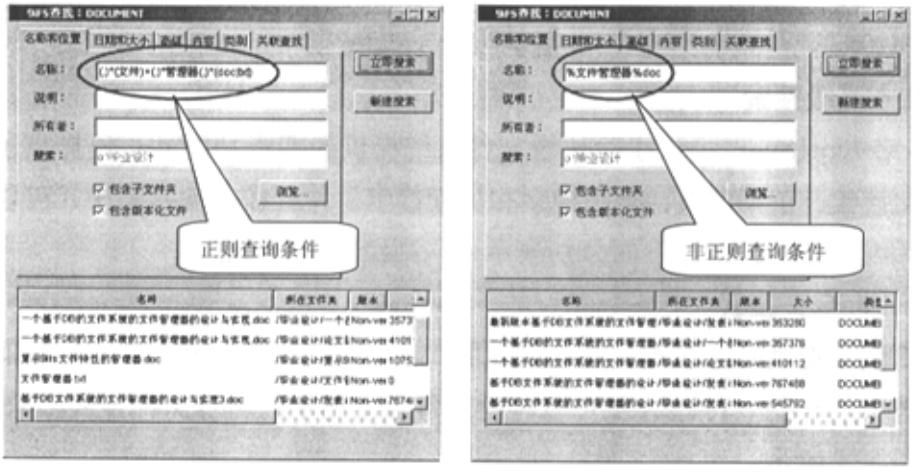

图 5-5 正则搜索与非正则搜索

这里需要说明一点，Z-Explorer 中查询条件有一个规定：“_”和“%”分别代表非正则查询条件中的任意一个字符和任意多个字符，而“？”和“*”作为正则查询中的专用字符。

3、类别搜索

由本文 4.2.3 节可知，在基于内容管理的文件系统中，可以通过类别属性对文件进行分类，这样存放在不同位置的文件可以通过类别将同类文件搜索出来。图 5-6 演示了文件类别的设置和搜索过程。

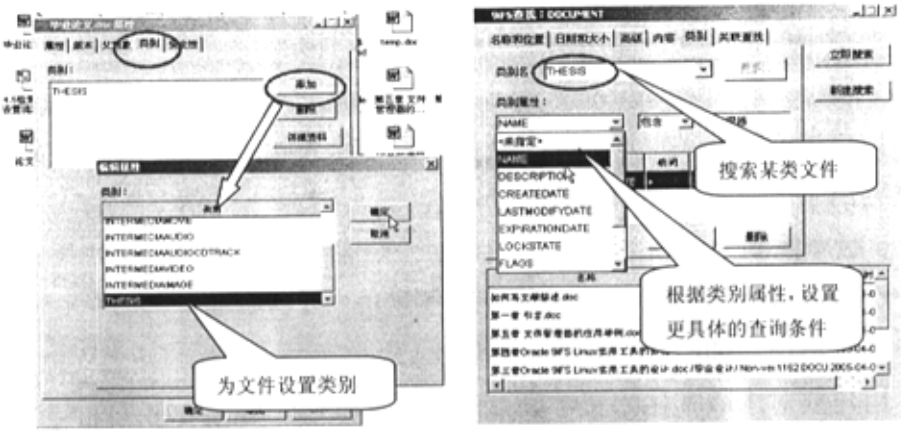

图 5-6 类别搜索

图 5-6 中左图显示了在 Z-Explorer 属性设置窗口为文件设置类别的过程，首先在属性设置窗口选择“类别”Tab 页，然后点击“添加”按钮，在弹出的“编辑属性”窗口为当前文件选择一种类别，再点该页面的“确定”按钮，最后点击属性设置窗口中的“应用”按钮。为文件设置好类别后，就可以在图 5-6 中右图的查找窗口的“类别”Tab 页设置类别的查询条件。

4、关联搜索

关联搜索能够为用户进一步提供搜索功能，如某用户需要根据邮件发送者的一些个人信息(如：爱好、生日)来搜索满足条件的邮件。图 5-7 演示了根据文档的关联对象——CONTENTOBJECT 的字符集属性搜索字符集为“UTF8”的文档。

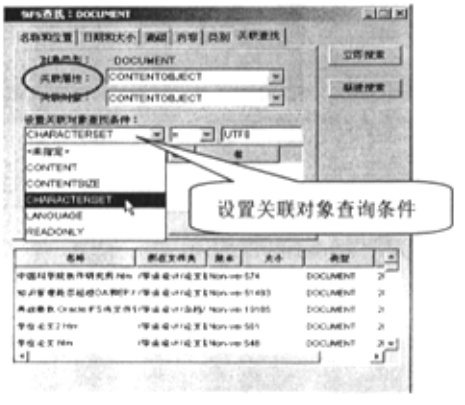

图 5-7 关联搜索

上图中的关联属性有两层含义：一、此属性是对象类型“DOCUMENT”的一个属性；二、该属性与一个同名的对象相关联。关联属性下拉框列出了当前搜索对象类型属性中所有与其它对象关联的属性，关联对象下拉框列出了所有可被关联的对象。

5.3.5 版本控制

Z-Explorer 提供了全面的版本控制功能，可以通过扩展的 Context 菜单进行调用，如图 5-1 中的右图所示。

用户要对未版本化的文件进行版本管理时，首先要选择“将文件版本化”菜单项对文件进行版本化，在弹出的版本化对话框中可以输入对当前版本的一些描述信息，然后点击“确定”按钮，对文件完成版本化，如图 5-8 所示。

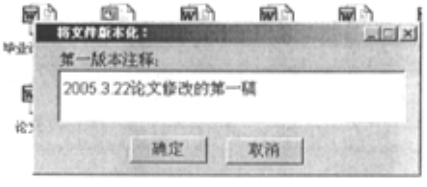

图 5-8 将文件版本化

版本化成功后，就可以通过检入/检出为文件创建新的版本。图 5-8 中左图是检出操作界面，图中显示当前文件有六个版本，用户可选择任一版本，然后点击“检出选中版本”按钮将其检出。另外用户还可以在属性操作界面中查看文件的版本信息，并可点击“复制到…”按钮将选中版本的内容复制到本地磁盘。同时，用户也可以点击“删除”按钮将选中的版本删除，但 Z-Explorer 不允许删除最新的版本，操作界面如图 5-9 中右图所示。其它的“取消检出”和“重命名版本化文件”等功能在此不再熬述。

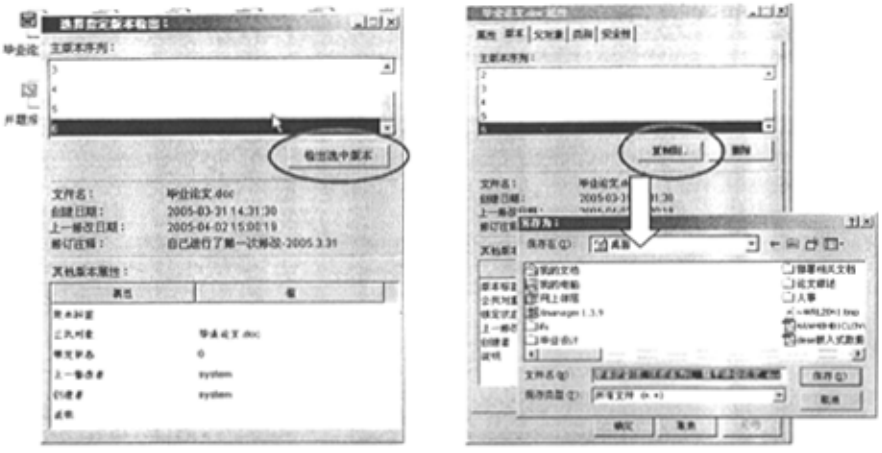

图 5-9 版本操作

5.3.6 安全管理

下面我们主要演示通过 ACL 实现对文件的安全管理的过程。首先打开 Z-Explorer 设置属性的操作界面，在“安全性”Tab 页中可以看到当前文件已被设置了一个默认名称为“Published”的 ACL，此时可以点击“更改”按钮，在弹出的对话框中选择一个 ACL，若当前没有合适的 ACL，用户可以新建 ACL，这里我们选择名称为 yaomingRead 的 ACL，然后点击“确定”按钮。设置过程如图 5-10 所示。

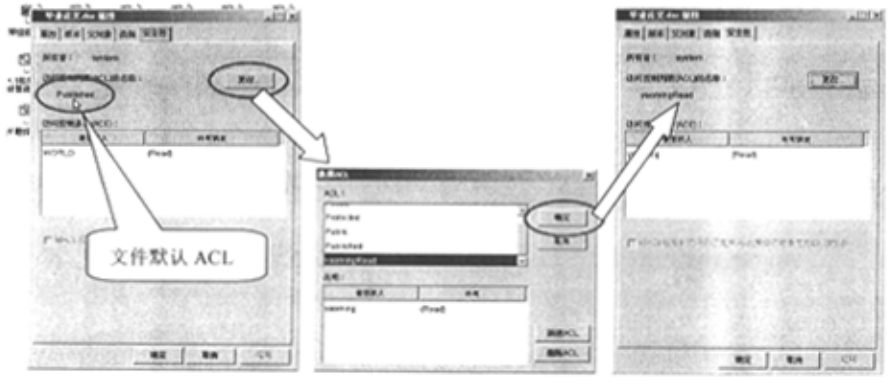

图 5-10 设置文件的 ACL

图 5-11 显示了两个不同用户欲访问上面刚刚被修改了 ACL 的文件，此时 ACL 只允许“yaoming”一个人具有读该文件的权限。图 5-11 中左图显示了用户“liuxiang”试图访问该文件时被拒绝；右图显示了用户“yaoming”具有读权限，但在试图修改文件时被拒绝。

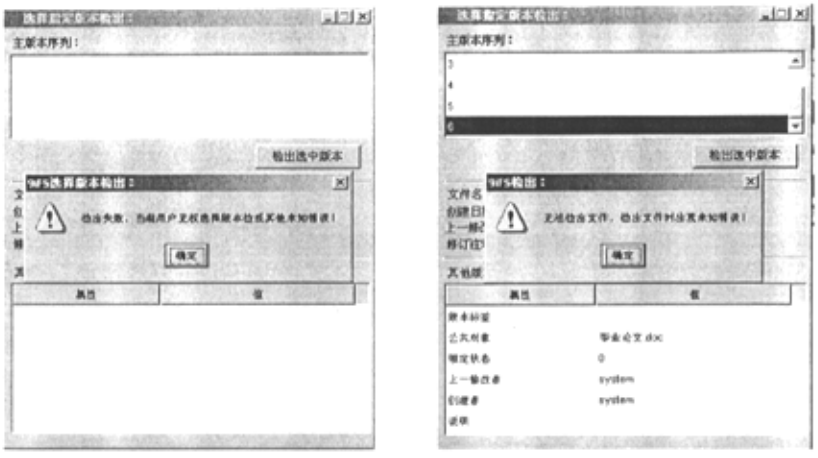

图 5-11 通过 ACL 验证用户的合法性

由于篇幅有限，这里只演示了 Z-Explorer 一些主要功能。

第六章 结语

6.1 总结

本课题针对用户对内容管理的迫切需求和当前内容管理环境下用户管理文件手段的不足，结合课题组科研需要，构建了一个功能丰富、操作简单、跨平台、支持内容管理的文件管理器——Z-Explorer。

本文分析了内容管理领域要解决的基本问题，研究了目前内容管理开发平台所支持的功能，总结出一个优秀的 Z-Explorer 文件管理器需要具有高级搜索、版本控制、类别设置、ACL 安全控制等功能，并能跨平台，这些是实现 Z-Explorer 重点和难点。本文所有的工作都是围绕着如何实现这些功能和将 Z-Explorer 集成到 Context 菜单供用户使用而展开的。

Z-Explorer 文件管理器最终比较完整地实现了全文搜索、正则搜索、关联搜索等高级搜索功能，提供了更安全的安全控制机制，实现了类别设置和查找功能，同时也为用户提供了更灵活的版本控制功能，并为课题组其它项目的研发奠定了基础。

本文实现的 Z-Explorer 文件管理器另一个重要的特色是具有较出色的跨平台性。它是利用 Java 语言开发的，能够在所有具有 JRE 环境的操作系统上部署和运行。但这里需要说明的是，由于 Z-Explorer 是通过 Context 菜单提供给用户使用的，因此实现 Z-Explorer 与 Context 菜单关联时，不同的操作系统对应的实现方法有一定的区别，需要单独加以实现。但是这部分工作量并不大，不影响 Z-Explorer 本身的跨平台特性。

Z-Explorer 实现了基本的内容管理功能，体现了内容管理的优越性，具有良好的研究价值和实用意义。

6.2 展望

目前，内容管理还处于起步阶段，许多技术尚不成熟，智能化程度

不高。今后需要将 Z-Explorer 与课题组正在研发的语义文件系统整合起来，提高对内容进行语义管理的功能。同时，随着课题组项目研发的进展，在 Z-Explorer 中可以进一步整合协同工作、中文分词等功能。尽管 Z-Explorer 文件管理器目前还不够完善，但是我们相信，随着研发工作的不断推进，Z-Explorer 的功能将更加丰富和完善。

参考文献

[1] http://industry.ccidnet.com.

[2] Dennis Dawson,Tom Grant,Alison Stokes.在 Internet 时代管理数据:Oracle Internet File System(9iFS)商务白皮书[EB].http://www.oracle.com/global/cn/documentation/9iAS_ifs_bwp.pdf.2001.

[3] WinterGreen Research,Inc.Content Management Market Opportunities, Strategies,and Forecasts 2004-2009［EB/OL］.http://www.giichinese.com.tw/chinese/wg17238_content_management.html.2004.

[4] http://www.oracle.com/technology/global/cn/products/ifs/index.html.

[5] http://tech.sina.com.cn/focus/Longhorn/index.shtml.

[6] http://www.hylanda.com/home.htm.

[7] 世界银行.《1998 年世界发展报告——知识促进发展》[EB/OL]. http://www.kmcenter.org/ .1998.

[8] http://www.gartner.com/Init.

[9] http://www.ml.com.

[10] http://www.trs.com.cn.

[11] 夏敬华.内容管理创造数字财富[EB/OL].http://www.trs.com.cn/resources/hyzs/t20031022_2683.htm. 2003.10.

[12] 飞思科技产品研发中心.Oracle9i 数据库高级管理[M].北京:电子工业出版社,2002.7.

[13] java.util.regex 的帮助文档[EB].http://java.sun.com.

[14] Ben Ezzell.Windows 2000 环境下 Visual C++编程从入门到精通.北京:电子工业出版社,2000.10.

[15] Francine Hyman, Sudhanshu Garg, Scott Harrison.Oracle® Internet File System Installation Guide Release 9.0.1.1.0 for Microsoft Windows NT/2000[EB].http://www.oracle.com.2001.9.

[16] Janelle Simmons,Harish Akali,Warren Briese.Oracle9i Database Installation Guide Release(9.0.1) for Windows[EB].http://www.oracle.com.

2001.6.

[17] Francine Hyman.Oracle® Internet File System Setup and Administration Guide, Release 9.0.1[EB].Oracle 9iFS and Oracle9i database documentation sets. 2001.7.

[18] Alison Stokes, Dennis Dawson. Oracle® Internet File System Developer Reference Release9.0.1.1.0[EB].Oracle 9iFS and Oracle9i database documentation sets.2001.9.

[19] Geroge Shepherd,David Kruglinski.Visual C++ .NET 技术内幕.北京:机械工业出版社,2003.

[20] 周鸣扬.Visual C++界面编程技术[M].北京:北京希望电子出版社,2003.

攻读学位期间公开发表的论文

[1] 张成年，赵雷，吕强. ADO.NET 连接池中非正常断开连接的异常控制[J]. 计算机工程与设计，2005.9

后记

仰首是春，俯首是秋。三年回眸，弹指一瞬，此时的总结却又翻开了人生新的篇章。

曾经有过各种疑惑，经历之后，积淀下来的是收获。收获的不仅仅是学业，更为重要的是学业背后的成长，其中有对生活的感悟，对事业的追求和对友情、亲情的珍爱。

感谢母校苏州大学，感谢她为我提供了一个接受教育的机会，感谢她美丽的校园，广博的图书，渊博的老师，使我的身心接受了幸福的洗礼。

导师吕强教授思理之密、洞察之深，更是每每指引我超过许多纷杂与困惑而切中问题的肯綮。感谢您几年来对我的悉心关怀和指导，是您带给我们很多新鲜的教育方式，使我们在轻松之余得到了提高。然而弟子不才，高先生之期望逃矣！

衷心的感谢杨季文教授对我工作的严格要求和生活的帮助，这才使得我三年来迅速地成长。

诚挚的感谢王红玲老师、王岩老师和朱雯老师在论文写作期间提出很多有益的建议和大量审稿工作。这里还要感谢赵雷老师、孔芳老师及其他各位老师三年来在学习和生活上的帮助。

感谢周志军、张步忠、倪伟恩、费勤、大刚、阿金、小祥、高彦明、张江、阿德、小周、宏亚、周建美等同学给予我热诚无私的帮助。感谢一起生活的师弟师妹，本来枯燥求学过程因为有了他们才有了许多欢笑，在此致以最诚挚的谢意！

父母家人多年来的默默支持，更是肯感激之辞难表万一。读书生活使我不能向父母尽反哺之情，不能尽兄姐之亲情，这使我每每愧疚于心。但我深深知道，论文的字里行间都渗透着他们对我殷切的期望。

感谢女友顾明珠五年来一如既往的支持，因为有了你最真的爱，我前进的脚步才迈得更踏实有力！

三年的研究生生活是我人生旅程中难以忘怀的重要驿站，我珍惜，我留恋！

张成年谨记

2005年4月20日于苏州大学

附录一 服务器端模块的接口定义

Z-Explorer 客户端模块的代码实现了用户操作界面的控制，服务器端模块的代码实现了具体的操作逻辑，并为接口映射模块提供了简单的接口。这些接口根据功能不同被封装在不同包中：base 包、propertyserver 包、searchserver 包、versionserver 包。base 包中定义了公共接口。propertyserver 包中定义了对属性操作的接口。searchserver 包中定义了实现搜索功能的接口。versionserver 包中定义了实现版本化功能的接口。这里我们列出了服务器端模块定义的接口，并给出了部分接口原代码。

本系统共开发了 58 个类，代码共有 19231 行代码，如果除去自动生成代码，直接手写的代码大约有 10000 行。另外，编写 Context 菜单扩展的代码总量为 800 行左右。根据课题组工作安排，在 2004 年 4 月到 2005 年 1 月系统开发阶段，每半个月都要对自己的工作做一个总结，并形成文档发布在内部的 BBS 论坛上，这期间共写了 18 份总结文档。另外，在系统开发过程中我对工作细节做了详细的记录。

一、服务器端定义的接口

● base 包中定义的接口

1. 启动服务：

```
public LibraryService startService()
```

2. 创建会话：

```
public LibrarySession establishSession(LibraryService service)
```

3. 关闭会话：

```
public void disconnectSession(LibrarySession session)
```

● propertyserver 包中定义的接口

4. 修改文件的 ACL：

```
public boolean changeDocumentAcl(PublicObject po, AccessControlList acl)
```

5. 修改文件夹的 ACL：

```
public boolean changeFolderAcl(Folder folder, AccessControlList acl)
```

6. 创建新的 ACL：

```
public int createAcl(String aclName, String[][] aces)
```

7. 获取指定 ACL 的 ACE：

```
public String[][] getAce(String objectPath)
```

8. 获取用户可以使用的 ACL：

```
public String[][][] getAcls()
```

9. 删除指定 ACL：

```
public boolean deleteAcl(String aclName)
```

10. 修改文件/文件夹的属性：

```
public boolean setChangedProperties(String objectPath, String[][]
        hasChangedProperties, String[][] changeCategoryName)
```

11. 获取指定版本内容输入流：

```
public InputStream copySpecificVersion(String objectPath, int
        specificVersion)
```

12. 修改文件/文件夹的类别类型：

```
public void changeCategory(PublicObject po, String[][] changeCategoryName)
```

13. 获取所有类别的名称：

```
public String[] getCategoriesName(String objectPath)
```

14. 获取所有类别：

```
public String[] getCategory()
```

15. 获取指定文件/文件夹的属性：

```
public String[] getProperties(String objectPath)
```

16. 获取指定版本的属性：

```
public String[] getVersionProperties(String objectPath, int i)
```

- **searchserver 包中定义的接口**

17. 获取系统中所有文件/文件夹类型：

```
public String[][] getDocumentAndFolderObjectType()
```

18. 获取搜索结果：

```
public String[][] getFile()
```

19. 获取对象的扩展属性：

```
public String[][] getObjectExtendedAttribute(String objectType)
```

20. 获取指定类别的属性：

```
public String[][] getCategoryTyepProperty(String categoryTypeName)
```

21. 获取关联对象类型：

```
public String[] getRelateObjectType(String relateObjectTypeName)
```

- **versionserver 包中定义的接口**

22. 对文件进行版本化：

```
public int versioning(String objectPath, String revcomment)
```

23. 检出文件的指定版本：

```
public int checkoutSpecialVersion(String objectPath)
```

24. 检入版本：

```
public int makeCheckin(String objectPath, String comment)
```

25. 设置检入版本的内容：

```
public int setPendingPublicObjectContent(String objectPath, int
                specificVersion)
```

26. 取消检出：

```
public int makeCancleCheckout(String objectPath)
```

27. 改变版本化文件的名称：

```
public int changeVersionName(String objectPath,String newName)
```

28. 删除文件的指定版本：

```
public int deleteSpecificVersion(String objectPath, int specificVersion)
```

29. 锁定指定文件：

```
public int makeLock(String objectPath)
```

30. 解除文件的锁定：

```
public int makeUnlock(String objectPath)
```

二、部分接口的实现代码

1. startService()

```
public LibraryService startService(){
    String serviceName = getServiceName();
    String schemaPassword = getSchemaPassword();
    LibraryService service = null;
    try {
        service = LibraryService.startService(serviceName,schemaPassword);
    }catch (Throwable e) {
        log("无法启动服务:");
    }finally{
        return service;
    }
}
```

2. establishSession()

```
public LibrarySession establishSession(LibraryService service){
    String userName = getUserName();
    String password = getUserPassword();
    LibrarySession session = null;
    try{
        CleartextCredential cred = new CleartextCredential(userName,
                                        password);
        session = service.connect(cred, null);
```

```
    }catch (Throwable e){
        log("无法创建会话:");
    }finally{
        return session;
    } }
```

3. disconnectSession()

```
public void disconnectSession(LibrarySession session){
    try{
        session.disconnect();
    }catch (Throwable e){
        log("断开会话时出错:");
    }
}
```

4. changeDocumentAcl()

```
public boolean changeDocumentAcl(PublicObject po, AccessControlList acl){
    ...
    try {
        po.setAcl(acl);
        return true;
    }
    catch (IfsException e) {
      log("设置文件 ACL 时出错:");
    }finally{
        return false;
    }
}
```

5. changeFolderAcl()

```
public boolean changeFolderAcl(Folder folder, AccessControlList acl) {
    ...
    try{
        //所有子文件夹和子文件都设为相同的访问控制
        PublicObject[] items = folder.getItems();
        int length = (items == null) ? 0 : items.length;
        for (int i = 0; i < length; i++) {
            // 如果是文件夹，则递归调用 changeFolderAcl()方法
            if (items[i] instanceof Folder) {
                Folder f = (Folder) items[i];
                f.setAcl(acl);
                changeFolderAcl(f, acl);
            }else {
```

```
                // 如果是文件，则修改文件的 ACL
                changeDocumentAcl(items[i], acl);
            }
        }
        return true;
    }
    catch (IfsException e) {
        log("设置 ACL 时出错:");
     } finally{
        return false;
    }
}
```

6. getDocumentAndFolderObjectType()

```
public String[][] getDocumentAndFolderObjectType() {
    ...
    try {
     ClassObject documentType = session.getClassObjectByName("DOCUMENT");
     ClassObject folderType = session.getClassObjectByName("FOLDER");
     Collection  col =  session.getClassObjectCollection();
     for(int i = 0; i < col.getItemCount(); i++)
     {
      classObject =  session.getClassObjectByName(col.getItems(i).toString());
      if(classObject.isSubclassOf(documentType)||
              classObject.isSubclassOf(folderType))

      {
       documentAndFolderCount++;
      }
     }
     documentAndFolderType = new String[2][documentAndFolderCount];
     int hasDocumentAndFolderCount = 0;
     for(int i = 0; i < col.getItemCount(); i++)
     {
      classObject =  session.getClassObjectByName(col.getItems(i).toString());
      if(classObject.isSubclassOf(documentType))
      {
        documentAndFolderType[0][hasDocumentAndFolderCount] =
            col.getItems(i).toString();
        documentAndFolderType[1][hasDocumentAndFolderCount] = "0";
        hasDocumentAndFolderCount++;
      }
      else if(classObject.isSubclassOf(folderType))
      {
```

```
            documentAndFolderType[0][hasDocumentAndFolderCount] =
                col.getItems(i).toString();
            documentAndFolderType[1][hasDocumentAndFolderCount] = "1";
            hasDocumentAndFolderCount++;
          }
        }
        return documentAndFolderType;
      }
      catch (Throwable e) {
        return null;
      }

    }
```

7. getFile()

```
public String[][] getFile() {
    ...
    //根据客户端模块传递过来的查询条件，创建搜索语句
     alias = searchUtils.addSearchClass(className, isIncludeObjectSubclass);
     //设定搜索路径查找条件
     Folder folder = folderUtils.findFolderByPath(searchCondition[1][3]);
     ...
     searchUtils.addFolderRestrictCondition(folder, isIncludeSubFolder, alias,
                                            isIncludeVersion,
                                            true);
     // 设定名称查找条件
     if (this.searchCondition[0][0] == "1") {
       av = AttributeValue.newAttributeValue(this.searchCondition[1][0]);
       searchUtils.addAttributeCondition(alias,"NAME",
                       AttributeQualification.LIKE,av);
     }
     //设定说明查找条件
     ...
     //设定日期查找条件
     ...
     //设定文件大小查找条件
     ...
     //设定类别查找条件
     ...
     //设置高级条件查找
     ...
     //设定所有者查找条件
     if (this.searchCondition[0][2] == "1") {
       String duAlias = searchUtils.addSearchClass(DirectoryUser.CLASS_NAME,
```

```
        isIncludeObjectSubclass);
    searchUtils.addJoinCondition(alias, PublicObject.OWNER_ATTRIBUTE,
                                 duAlias, null);

    av = AttributeValue.newAttributeValue(searchCondition[1][2]);
    searchUtils.addAttributeCondition(duAlias, PublicObject.NAME_ATTRIBUTE,
                                      AttributeQualification.LIKE, av);
  }

  //设定内容查找条件
  if (this.searchCondition[0][9] == "1") {
    searchUtils.setTextCondition("contentName", searchCondition[1][9],
                                 alias);
  }

  //设置关联查找
  if (searchRelateObjectCondition != null) {
    ...
    for (int i = 0; i < this.searchRelateObjectCondition[0].length; i++) {
      if (searchRelateObjectCondition[5][i].equals("0")) {
        relateDuAlias = searchUtils.addSearchClass(
            searchRelateObjectCondition[0][i], false);
        searchUtils.addJoinCondition(alias,
                                     searchRelateObjectCondition[0][i],
                                     relateDuAlias, null);
        //设置关联对象的查找条件
        for (int j = i; j < searchRelateObjectCondition[0].length; j++) {
          if (searchRelateObjectCondition[0][j].equals(
              searchRelateObjectCondition[0][i])) {
            searchRelateObjectCondition[5][j] = "1";
            //设置属性的名称
            propertyName = searchRelateObjectCondition[1][j];

            //设置操作比较符
            int operator = 0;
            for (int k = 0; k < 6; k++) {
              if (k != 5 &&
                  this.arrayOperator[k].equalsIgnoreCase(
                  searchRelateObjectCondition[2][
                  j])) {
                operator = k;
                k = 6;
              }
```

```
                else if (k == 5) {
                  operator = 8;
                }
              }

              //设置条件值
              if (searchRelateObjectCondition[4][j].equals("1")) {
                //日期类型
                SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
                Date date = format.parse(searchRelateObjectCondition[3][j]);
                av = AttributeValue.newAttributeValue(date);
              }
              else if (searchRelateObjectCondition[4][j].equals("3")) {
                //DOUBLE类型
                av = AttributeValue.newAttributeValue(java.lang.Double.
                    parseDouble(searchRelateObjectCondition[3][j]));
              }
              ...
              searchUtils.addAttributeCondition(relateDuAlias, propertyName,
                                                operator, av);
            }
          }
        }
      }
    }
    searchUtils.addOrderByClause(alias, "NAME", false);
    Search search = searchUtils.createSearch(isIncludeObjectSubclass);
    return getSearchResultProperties(search);
  }
  catch (Throwable e) {
    return null;
  }
}
```

8. versioning()

```
public int versioning(String objectPath, String revcomment) {
  ...
    try {
    PublicObject po = folderUtil.findPublicObjectByPath(objectPath);
    Folder parent = folderUtil.findFolderByPath(parentPath);
    VersionSeries vs = verUtils.getPrimaryVersionSeries(po);
    int checkLocked = s_common.checkLocked(folderUtil, objectPath, po, vs);
    if (checkLocked != 0) {
      return checkLocked;
```

```
        }
        Family docFamily = verUtils.makeVersioned(po, parent, revcomment);
        if (docFamily == null) {
          return -1;
        }else {
          return 1;
        }
      }catch (Throwable e) {
        return -1;
      }finally {
        cleanup();
      }
}
```

附录二 SQL 语句关键词与搜索功能类的对应关系表

oracle.ifs.search 包中的功能类用于组装搜索标准，与利用 SQL 关键词组装 SQL 查询语句一样，搜索功能类也必须按照一定的规则组装搜索标准，每个功能类与 SQL 关键词存在着一定的对应关系，下表说明了 SQL 查询语句关键词与搜索功能类的对应关系。

SELECT<result> FROM <list>	SearchClassSpecification
WHERE	SearchQualification
(	SearchClause
<expression>	AttributeQualification, FolderRestrictQualification, ContextQualification, ExistenceQualification, FreeFormQualification, Or PropertyQualification
<join>	JoinQualification
(	SearchClause
<expression>	同上<expression>
<join>	JoinQualification
<expression>	同上<expression>
ORDER BY <list>	SearchSortSpecification