基于元数据的档案集管理程序的设计与实现

专业：计算机软件与理论
硕士生： 刘云赟
指导老师： 倪德明 副教授

摘 要

本论文讨论了大规模数据集备份的情形下，利用嵌入归档文件头部的自描述元数据信息对散落的归档文件集合实施有效管理的方案，并进行了详细设计与实现。

在通常的备份归档系统中，会遗留下大量散落的归档文件，只有对归档文件实施有效的管理才能使其成为有意义的可恢复数据。并且随着归档集合规模的增大，对归档的操作变得愈加复杂，需要灵活简便的工具来帮助管理人员实施管理。

论文探讨了对档案文件集合的分类管理需求，建立了由元数据构建得到的多级分类的树状档案集合模型。论文讨论了元数据集合、元数据划分、元数据分类树等模型的若干性质，给出了由元数据节点组织为分类树的方法。对这些模型给出了半形式化的描述。

在元数据分类树的基础之上，论文定义了通过树结构视图在归档集合上进行的选择扩展、级联删除、迁移、验证、搜索等操作并给出了相关算法。还讨论了针对不同的归档类型的情形下一致的集合操作语义。

在模型讨论基础之上，本文设计了元数据的分散存储、树结构的组织生成。实现了分类树类与归档管理类等可重用组件，设计实现了一套命令行工具框架和一组命令行 UI 类体系，使得创建、扩展命令行工具更简便清晰，输入输出结构化信息更具语义特性。

本文在归档集的分布式元数据管理基础上适当引入集中元数据缓存的思想，设计实现了元数据与分类树缓存机制，利用归档极少改动的特点在大规模归档集的情形下能够保持良好的性能。

笔者实现了满足管理需求的命令行工具，采用一致的命令行子命令、参数与选项格式，并且产生结构化输出，适合脚本批处理应用。定义统一的档案文件集合操作接口，使得扩展的插件可以对不同类型的档案进行一致的操作。

对元数据分类树模型的讨论、管理操作的定义与实现、不同归档类型的一致集合操作的讨论为归档系统的通用性的提出奠定了基础。元数据存储设计、缓存机制设计、命令行工具的构建，给备份系统的归档管理提供了一个简便易行、可扩展的方案。命令行输出的结构化设计为脚本的编写提供了可能，并在此基础上封装实现了图形界面工具，使得管理人员可以通过图形界面管理远端主机的命令行工具与归档集合。

关键词： 归档管理，元数据，分类树

The Design and Implementation of Archive Set Management Program Based on Metadata

Major: Computer Software and Theory
Name: Liu Yunyun
Supervisor: Associate Prof. Deming Ni

ABSTRACT

The thesis discussed the available management solution of large set of archive files using metadata, and finished the detail design and implementation.

In regular backup-archiving systems, a large number of archive files would be produced and leaved. It is could become useful and ready-to-recoveried data that only when performing efficient managements on archives. It has a increasing complexity as the increasing of archive set. A flexible utility is needed to help archive manager.

This thesis discussed the demand of archives management, build a module of transferring from metadata set to multi-leveled classified tree. The characters of modules as metadata set, metadata classification, and classified tree were discussed. Thesis presented the algorithm of organizing metadata nodes to a classified tree, and presented the form description of these modules.

Based on the metadata classified tree, thesis defined the operations as select、expansion, delete, migrate, validate, and search, and discussed the identical set operating semantics under different archive types.

Based on the discussion of modules, thesis designed the distributed storage of metadata, and the organization of tree structure, implemented reusable components as classified tree class and management class. It designed a suit of command line utility framework and a suit of command line IO class framework, which make building command line more convenient.

Introducing thoughts of centralized management properly into the distributed

storage of metadata, the article implemented a metadata cache system. Since the archives nearly never be modified, the program could perform well when data set increasing.

Author implemented command line utilities that meet the management demand. The utilities uses identical sub-commands, options, and arguments format, and produces structural output, suitable for batch scripting application. An identical set-operating interface was defined and it allows functions extensible to operate different types of archives.

It is the discussion of classified tree module, the implementation of management operations, and the discussion of identical interface to different archive types that established the basement of archive management generalization. It is the metadata storage design, cache design, and utilities building that provided a available and extensible solution. It is the structural command output that make possible to scripting, and the further development. A GUI tool was implemented based on command line tools, and the GUI tool could manage remote command line tools and archives.

Key words: Archive Management, Metadata, Classified Tree

论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，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，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。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。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：2008年 5月 26日

学位论文使用授权声明

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅，有权将学位论文的内容编入有关数据库进行检索，可以采用复印、缩印或其他方法保存学位论文。

学位论文作者签名：　　导师签名：

日期：2008年 5月 26日　　日期：2008年 5月 26 日

第一章 引言

1.1 背景

以信息技术尤其是网络技术的飞速发展，使得公司与企业组织的运作越来越依赖于 IT 系统。大量有关生产、销售的业务信息数据维系着整个组织的生存和发展，它们是珍贵无比的无形资产。这些数据一旦因为存储系统遭受意外而被丢失、篡改、物理损害或者遭遇其他不可避免的自然灾害，造成大量数据丢失，对于所有的组织来说，都无疑是一场灾难，因为它除了会给企业带来重大的经济损失以外，甚至有可能动摇企业的生存基础。所以，加大保护数据安全的力度，已经成为了越来越多企业的共识。

归档管理是备份中的一个重要环节，也是和最终用户直接交互最多的环节。在大量、长期的备份工作中，会留存下大数据量的归档集，他们可能来自于不同的主机、操作系统，具有不同的数据资源类型、备份类型、时态特性。散落无组织的归档数据是无意义的，不可管理的，难于恢复的。只有通过精心设计的技术手段，将归档数据集有效的组织起来，使用一定的视角和方法学将其纳入特定的管理体系，才能使得大量的归档数据集可控、可管理，恢复时候方能发挥巨大作用。所以，日益膨胀的数据量和精准的备份/恢复操作要求都对归档集的管理提出了新的考验。如何在复杂大量的归档集上进行有效的数据管理，是存储/备份领域的热点问题。

1.2 现有工作基础

中山大学软件研究所和广州威腾网络科技有限公司合作的数据备份系统项目，着力研究数据备份领域的关键问题和技术，并建立了理论模型、备份通用模型和存储模型等。并研制诞生了优秀的企业数据容灾备份产品 NetBunker[3]。在

这些成果的基础上，开发了新的个人数据备份工具 ProRecovery，并提供了相应的归档管理方案。本文内容即是在此归档管理方案的研制过程中的一些工作。

ProRecovery 采用插件式结构处理流式备份数据，在磁盘上写入备份归档。保存的归档的头部写入了由威腾公司定义的归档文件的通用头结构 UDS（Unified Data Structure）记录了关于此归档的来源类型时态基本属性等翔实的元信息，并制定了此头结构的类型、大小、读写方式等基本规范。

1.3 本文的工作

对元数据在数据管理中的应用的研究，多见于电子商务文件管理、电子政务、单位档案管理等领域[1][2]。而在备份领域鲜有专门提及，但是实际使用相当广泛。

日常用的归档软件如 rar,zip,tar，仅具有为文件压缩打包功能，成为散落无组织的数据。大型备份存储软件（如 Veritas）管理界面略显繁琐不明晰，并且缺少对时态信息的有效表达。一些小型备份软件（如 SQLsrv）有强烈的时态信息，但只能管理单一来源的数据。

ProRecovery 采用的 UDS 头结构为管理多来源、多类型、时态特征明显的大归档集提供了可能。本文即在此结构基础之上，进行适当的扩展，作为本文研究的基本数据结构。

本文的主要工作有：

1. 对归档环境中的元数据、元数据分类树做了半形式化的归纳描述。
2. 对由分类树表示的归档集合上进行的各种操作进行了总结，并给出了相应的算法描述。提出了归档系统管理的通用特性。
3. 设计了实际操作归档集合的软件工具 archmng，进行了如下工作：
   a) 类体系设计，设计了分类树数据结构、归档管理类等可重用组件，设计实现了一套命令行工具框架和一组命令行 UI 类体系，使得创建、扩展命令行工具更简便清晰，输入输出结构化信息更具语义特性。
   b) 缓存策略设计。在归档集的分布式元数据管理基础上适当引入集中元数据缓存的思想，设计实现了元数据与分类树缓存机制，利用归档极少改动的特点在大规模归档集的情形下能够保持良好的性能。

4. 实现了 archmng 命令行工具，设计了结构化的命令行输出，使得工具适用于脚本编写、图形界面封装。
5. 在命令行基础上封装实现了图形界面管理工具 archmng_win，完全利用命令行工具获得功能，而不具体操作归档。并且可以执行远程命令行。

本文在厘清了若干基本内容与现状（第二章）之后，在第三章针对档案集管理的需求提出了元数据模型，总结了一般的元数据组织模型。对元数据的组织、管理、元数据树进行了建模。重点探讨了元数据分类树的若干性质，并提出了元数据树的分类方法和生成算法，以及形式化了分类、删除、迁移、验证、搜索等常用操作。最后指出了对于不同归档类型的操作一致性并讨论了其形式化意义。第四章叙述了在前述模型指导下程序工具的详细设计和实现，分析了元数据的存储与组织形式，并设计了元数据现场组织与缓存的策略，介绍了一个通用的元数据树管理命令行框架并设计了其插件结构，实现了其图形界面的包装。第五章对讨论进行了总结并指出档案管理系统的一般特性。

第二章 基本概念、背景与现状

2．1 数据备份的发展历程

从数据备份的发展过程而言，一句话可以概括为，备份的发展贯穿了计算机的发展史，备份发展史是计算机发展史的缩影，它经历了三个阶段[4]：

1. 单机备份：多以“单机处理，软盘交流”的个人行为为主，无计划，随机性很大，备份没有得到重视；备份的形式通常就是对要备份的文件复制到硬盘的另一个位置或者软盘上，数据得不到有效的保护。
2. 局域网环境备份：数据共享显得日益重要，数据备份得到了应有的重视，成为系统管理员或应用操作员日常工作的一部分；备份的数据是网络中整个企业的数据，介质以磁带为主，不再是以前的盘对盘的备份，但依然是系统管理员的手工作业，通常是通过备份工具或是备份命令脚本进行人机交互执行，效率不高，成为系统管理员工作领域中繁重的负担；还没有出现商业化备份软件，研究性备份系统开始诞生。
3. 基于 Internet/Intranet 的应用环境下的备份：典型表现是整个企业业务流程依赖于 Internet/Intranet 网络环境；网络应用更加复杂化，数据种类增加，异种数据库、多种文件系统和操作平台呈现在异构的企业计算机网络中；数据量剧增，存储空间增大；数据存储位置变得分散；出现 NAS 和 SAN 存储机制，备份技术在结构上也得到了长足的发展,从 LAN Free 备份到无服务器备份，基本工作机制也逐步得到更新。

网络环境和应用系统的日趋复杂化，计算机数据处理出现了许多的问题：

- 数据管理工作难以形成制度化，数据丢失现象难以避免；
- 数据分散在不同的主机、应用系统上，管理分散，安全性得不到保障；
- 难以实现数据库数据的高效在线备份；
- 运行着的系统使得维护人员寸步难离，业务人员工作效率下降；
- 存储介质管理困难；

- 历史数据保存困难；
- 非计算机系统因素的隐患。

尤其应用环境发生了革命性的变化，数据备份工作随之变革的结果就是企业级备份软件的出现。备份软件针对上述问题，需要适应新环境下的备份需求：

- 需要克服数据的分散给备份带来的困难；
- 需要应付种类繁多的数据类型；
- 需要大容量的存储介质并对存储介质进行更好的管理；
- 备份不能影响企业应用系统的可用性；
- 备份日志信息应能有效管理并在此基础上的提供报表分析功能；
- 备份操作应该简单易行。

2.2 基本定义

世界著名的数据备份专家 W. Curtis Preston 在其名著《Unix Backup and Recovery》一书中对备份作出如下定义[5]：

备份 ＝ 拷贝 ＋ 管理

这个简洁的公式，表明了数据备份中的主要问题，所谓拷贝，就是对数据执行移动，复制等，所谓管理，就是对备份数据进行索引。

在参考文献 6 中，对网络数据备份系统给出了更加详细和准确的定义[6]，即：

备份＝拷贝＋变换＋传输＋存储＋管理

换句话说，备份系统的功能就是对各种需要备份的资源使用对应的方式进行拷贝，然后根据需要对原始数据进行一系列数据变换，通过不同层次的网络传输协议和通道技术传输到介质，最后以适当的存储结构存储在选定的介质上，并且整个流程以及相关的资源都处于备份中心的集中管理调度之下。

在 ProRecovery 系统中，为了方便界面的陈述和对备份机制与流程的规范化，我们对备份语境下的若干基本概念做了界定：

【定义】备份对象是指一系列备份目标资源，如文件系统、数据库等，他们通常是一个选定的可备份资源子集。

在一个企业 IT 基础设施构架的环境中，首先，备份对象可能来自于不同的

很多主机，典型情况下都是处在同一个局域网或者 VPN 内的主机。其次，备份对象具有多种类型，常见的例如文件系统，例如数据库表，数据库日志，Outlook 邮件等。再次，一个特定类型的备份对象，是由用户选定的这个主机中具有这个类型的资源一个子集。例如用户可以选定名为 DataSrv 的主机上文件系统中/home/dataop 目录、/etc/conf 目录这两个目录一起作为一个备份对象。

【定义】在备份操作中，实际的备份对象表现为一个逻辑上具有组织结构的集合被操作（选择、运算、存储），称作备份集。备份集可以有全备、差备等类型。

备份对象与备份集的差异在于，备份对象指的是备份前存在于主机上需要被操作的数据，而备份集是将备份对象的数据在指定的操作中与结果后形成的数据集。关于备份集的全备差备增备的类型差别在后节有详细讨论。

【定义】归档文件是备份集经过备份操作后存储下来的实际物理形式，具体可以表现为磁带记录、光盘内容、磁盘文件等。

归档文件是动态的备份集最终形成的持久化形态。在 ProRecovery 系统中，根据其针对个人备份的需求，一律采用了磁盘文件的归档形式。下文中除非特别指出，也一律指磁盘文件形式的归档文件。根据所存储的备份集类型，归档文件相应的也有全备差备之分。

2.3 数据备份的分类

按照备份操作的方式分类，备份可分为完全备份和增量备份，差异备份[7]。

完全备份用于复制所有选定的文件，并且在备份后标记每个文件（即清除存档属性）。也就是说，正常备份是复制所有要备份的文件，并且在备份后清除所有文档的“存档”属性，这样在下次备份时，备份软件会认为这些文件都还没有被备份。

增量备份则是另一种形式，它仅备份自上次正常或增量备份以来创建或更改的文件，并且将这些文件标记为已经备份。举例来说，当第一次进行完全备份后，所有文档的“存档”属性被清除，而在新建或修改这些文件后，文件重新被加上“存档”属性，增量备份就是备份这些文件，并且在备份后将这些文件的 “存档”属性

清除，以保证下次不再备份这些文件。

差异备份用于复制自上次正常或增量备份以来所创建或更改的文件。它不将文件标记为已经备份（即没有清除存档属性）。完全或增量备份去掉了文件的“存档”属性，在新文件创建或旧文件被修改后，文件重新被加上了“存档”属性，差异备份就是备份这类文件。在备份完毕后，差异备份并不会清除这类文件的“存档”属性，这样的话，在下次运行差异备份的时候，只要在此期间上次差异备份的文件没有被更改，则它们还会包含在备份集中，将被再次备份。

三种方式各有各的优势，正常备份和差异备份恢复容易，增量备份则占用空间小和花费时间短。

按照备份数据对现有应用产生的影响程度，可分为离线备份和在线备份。离线备份(Off-line Backup)，它是在进行备份操作时，服务器不再接受来自用户或应用对数据的更新。离线备份可以很好地解决在备份过程中数据的完整性的问题，是防止破坏、敌意病毒袭击、应用失误等的有效方式。在线备份(On-line Backup)，就是用户和应用正在更新数据时系统能够进行备份。在线备份最大的难点是如何保持数据的完整性。为了保护数据的完整性，可以采用两种技术：锁和快照。锁技术就是系统在备份某一文件时拒绝对该文件和目录的任何修改命令，因而锁技术会对数据可用性产生一定的影响，在应用持续的同时进行备份不可避免地降低了系统性能。快照是通过内存作为缓冲区(快照 Cache)，由软件提供系统磁盘存储的即时数据映像。目前，在线备份大多采用快照技术。

根据备份数据所处的位置可分为本地备份和远程备份。本地备份，即通过存储网络将数据备份在局域网范围内的备份。这种方式可利用现有的各种资源和技术来达到高速的备份。它的性能可以很好地满足实际需要，如数据一致性、容错等要求，而且性能是最高的。远程备份，在数据高可用性的网络环境中，异地数据备份是一个必不可少的手段。当本地应用受到灾难性破坏时，通过远程备份和容灾，来保护和恢复数据，使损失降低到最小。在远程备份过程中，数据传输要跨越校园网、城域网甚至广域网，这时，网络的传输速度对备份性能的影响是关键的因素。延迟也是影响远程备份性能的重要因素，特别是广域网的数据库备份，可以通过异步的方式来解决。另外，在远程备份中还要解决数据的完整性和一致性问题，解决这个问题比较成熟的技术是采用快照和镜像相结合的方法。

在 ProRecovery 中归档类型均采用全备与增备的方式。

【定义】对一个特定的备份对象的一次全备与到下一次全备之间的所有增备形成一个生命周期。

在一个生命周期内的归档文件，是可以将备份对象恢复到某一个确定时刻所需要的归档集合的闭包。生命周期之外的归档对于本次恢复无用。

2.4 已有的数据备份产品简介

在国际上，目前有成熟的企业级备份软件，有相当多大规模的独立软件供应商，而且从事备份软件研发的公司和机构组织也有很多。目前备份软件市场主要被 CA、VERITAS、Legato 三大公司所占领，另外还有 IBM、HP、COMPARE 这些硬件厂商也推出了自己的备份软件。其中 CA、VERITAS、Legato 和 IBM 四家公司的备份系统在目前来说占有了相当大的市场份额。现对它们逐一做个简单介绍[4]：

(1)CA 公司的产品主要有两个系列：ARCserveIT 系列和 BrightStore 系列。这两个系列提供高性能、易于管理性和卓越的可靠性。

(2)VERITAS 主要有 NetBackup 与 BackupExec 等产品。NetBackup 软件是一个功能强大的企业级数据备份管理软件，它为 Windows NT、UNIX 和 NetWare 环境提供了完整的数据保护机制，具有保护企业中从工作组到企业级服务器的所有数据的能力。 Backup Exec 软件是一种多线程、多任务的存储管理解决方案，专为在单一的或多节点的 Windows 2000/NT 企业环境中进行数据备份、恢复、灾难恢复而设计，使用于单机 Windows 2000/NT 工作站、小型局域网以及异构的企业网络。

(3)Legato 研制开发的 NetWorker 数据存储管理系统采用多服务器网络环境平行作业处理技术，完全支持关键备份服务器集群、 NAS 文件服务器的本地和三向备份以及 LANFree 备份，并支持多种应用和数据库的在线备份和恢复。

(4)IBM 的 Tivoli 管理环境是一个用于网络计算管理的集成的产品家族。它可以将数据的备份和归档拷贝存放在离线的存储介质上，从而保护组织的数据免受硬件故障和其它错误操作的破坏。

其他的备份存储厂商还有 BakBone，昆腾等。以及相关的归档软件如 Peta-Serve、StorNext-SAN、Amass、DiskExtend、SAM/FS、Veritas HSM 等。

这些国外的比较成熟的备份软件产品占据备份的高端市场，提供大而全的功能，对资源的配置条件要求较高，对我国当前的中小企业的来说购买这些产品的成本投资比较大，不适合国内中小企业的具体情况。

国内推出了自主开发的具有一定的市场占有率的备份软件的软件企业规模有限。除了一些自由软件之外，大部分企业都选择前述的三大公司的备份软件产品。国内在该领域的软件大部分都是较小规模公司制作的备份系统，优势和强度以及技术含量并不明显。

国内在备份软件开发领域起步较晚，在有效的备份服务器和介质服务器的设备管理方面的研究还有待走到深入的阶段。

2.5 ProRecovery 简介

ProRecovery 数据备份软件是广州威腾网络科技公司（VITON）自主研发，独立定义了备份数据存储结构和一套算子运行框架的数据备份软件，可应用于本地和网络数据备份。该软件最大的特点是把数据备份和还原操作插件化，插件接口协议化，提供了一部分数据处理功能，实现了算子的升级和校验。

2.5.1 ProRecovery 系统运行框架

本文所讨论的归档由 ProRecovery 数据备份系统产生，讨论实现的工具作为 ProRecovery 的辅助管理工具。首先介绍该系统的体系结构和运行框架[8]。

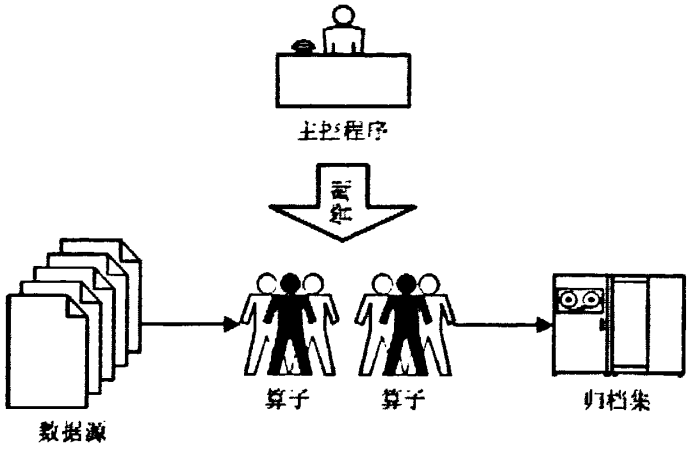


图 2-1

ProRecovery 是一个基于流式数据的数据备份系统，备份系统的处理对象是一条数据流，数据流由读算子从数据源读出，经过主控程序的规划，依次流过数据处理算子链，受到算子的数据处理"指令"，数据经过处理后成为一个备份集，最后由写算子写入备份介质的一种机制。

2.5.2 ProRecovery 备份系统的备份集结构

ProRecovery 系统的数据备份集结构为 UDS(Universal Data Structure)结构[9]，简单来说，UDS 结构就是备份集的数据结构，其结构图如下：

GH(自描述信息)	*GB(全局信息)*	*DATA(用户数据)*	*AX(辅助文件)*

图 2-2

如图所示，一个完整的备份集由 GH，DATA 和 GT 组成，其中，GH 代表 UDS 的头结构，存放用于备份恢复代理和变换模型需要的数据；备份恢复代理在调用变换模型的时候，填充好代理相关的元数据部分，而与变换模型相关的部分，由变换模型根据相关信息进行填充。具体说来，与变换模型相关的元数据包括：所使用的算子的名称、版本号、参数，加载算子的顺序等。

DATA 是实体数据，是指那些描述数据流信息和变换处理信息的数据，实体数据是备份恢复的数据对象本身，是无结构的流序列。

GT 表示尾结构，用于变换模型，主要存放算子处理后的结果信息，包括副作用数据和一些统计信息等。尾结构在算子对数据块进行处理后，由变换模型填写。

GH 和 GT 统称为元数据，元数据使数据流在一定程度上实现自描述，他的最主要作用是恢复数据时的索引重建。而本文所讨论的元数据，就位于 UDS 的 GH 部分。

第三章 元数据管理模型

3.1 元数据概述

3.1.1 元数据的通用定义

根据《电子文件归档与管理规范》这一国家标准，电子文件的元数据 Metadata 被定义为[10]："描述电子文件数据属性的数据，包括文件的格式、编排结构、硬件和软件环境、文件处理软件、字处理软件和图形处理软件、字符集等数据。"因此，元数据实际上就是对数据进行著录而得到的著录信息，这些著录信息专门用于电子文件的管理，以保证电子文件的真实性、可靠性，元数据就是为了提高电子文件的凭证性而提出来的。

元数据最本质，最抽象的定义为：data about data (关于数据的数据)。它是一种广泛存在的现象，在许多领域有其具体的定义和应用。在软件构造顶域，元数据被定义为：在程序中不是被加工的对象，而是通过其值的改变来改变程序的行为的数据。它在运行过程中起着以解释方式控制程序行为的作用。在程序的不同位置配置不同值的元数据，就可以得到与原来等价的程序行为。

随着数字信息环境的发展，元数据的性质、范围和作用也不断深入。对元数据最为权威的衍生定义有两个[11]：

第一个是国际标准组织制定的国际标准《DsO / IEcl1179_1区信息技术一 元数据注册— —第一部分 ：数据元素的说明及标准化框架》中所定义的："元数据是定义和描述其他数据或过程的数据"。

第二个是国际著名的元数据标准化机构——都柏林元数据机构制定的 《都柏林核心元数据应用》中所定义的："元数据是关于数据的结构化数据"。

3.1.2 元数据的通用性质

从元数据在组织信息资源的功能上区分，元数据有以下四种类型[12]：

管理型元数据：用来管理与支配信息资源的元数据 ，如信息收集、版权与翻版跟踪、排架信息、数字化标准选择、版本控制等.

描述型元数据：用来描述与识别信息资源的元数据，如记录编 目、寻找帮助、专题索引、资源链接 、用户注释等。

保存型元数据：与信息资源保存管理有关的元数据 ，如资源的物质条件、数字资源的保存行为(数据更改与迁移)。

技术型元数据：与系统怎样运行有关的元数据 ，如硬件与软件、数字化信息的格式、压缩比率、定标例程 、系统响应跟踪、数据验证与安全(如加密键、密码)等。

2001 年颁布的国际标准 X5015489《信息与文献——文件管理》所定义的电子文件管理元数据是[11]：“在文件管理领域，元数据是指自始至终地描述文件的背景、内容、结构及其管理的数据”。电子文件管理元数据是一个结构化的标准体系，其目的是对电子文件进行组织、管理、发现、识别、选择、定位、开发、利用和评价，追踪电子文件在管理和使用过程中的变化，有助于实现电子文件信息资源的凭证价值 、集成整合与长期保存。

电子文件管理元数据通过对电子文件的 “背景、内容、结构、管理”的控制，来确保电子文件具有如下特性[11]：

1. 真实性：即具有背景、结构和内容的文件其原始特征自始至终地保持一致，文件就是文件的本身；
2. 可靠性 ：即文件作为可靠凭证的性质，文件作为证据的权威性和可信赖性；
3. 完整性：即文件是完全的，并且未作任何改变；
4. 可使用性 ：即定位、检索、显示和说明文件的性质。

而电子文件只有具备了以上这些特征之后，才能被称为真正意义上的具有档案凭证价值的电子文件。电子文件管理元数据的 “控制”机理，是使电子文件免于失去 “真实性、可靠性、完整性和可使用性”的保证，是防范威胁这些本质特征的风险产生的关键措施。

3.2 归档的分类管理需求与特点

在局域网多数据源的环境中，各种备份集生成的归档文件来源于不同的主机、具有不同的数据源类型，包含不同的数据源集合，从属于各自的生命周期，生命周期中的归档具有相互依赖的关系。所以归档文件除了自身的备份数据外，还隐含有丰富的描述自身属性的信息。传统的备份环境下的人工标签、介质实体的分类存放等就表达了这样的自描述信息。在归档文件中即利用元数据来记录其自身信息。相较于电子公文、企业商务文件、多媒体影音资料、图书馆资料等元数据常见应用领域，备份归档管理有其固有的特点。通过分析这些管理特点，可以将元数据更有目的的应用于归档管理之中。归档管理的元数据特点分析如下：

1. 自描述性。归档文件的元数据信息获取应该仅仅需要归档文件本身，而不需要依赖外部的数据来源，才能在大规模的归档管理中灵活扩展，方便迁移与维护数据一致性。因此元数据应该在归档文件内部提供，使得归档文件具有自描述的性质。

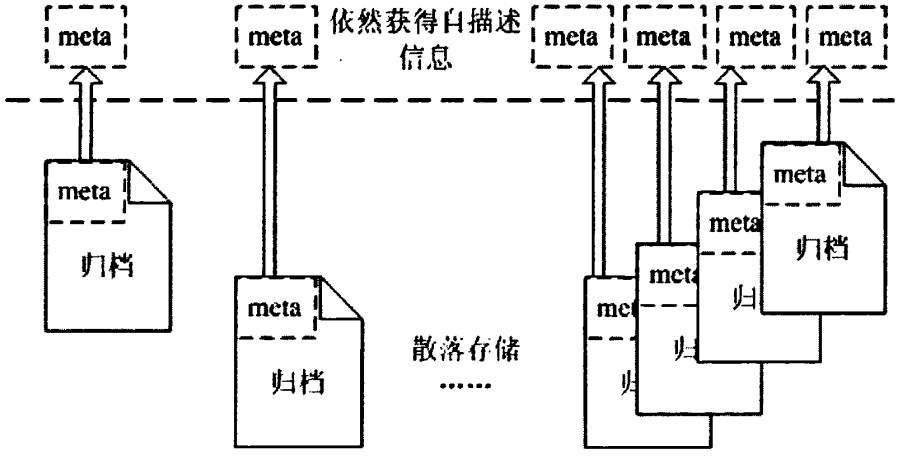


图 3-1

2. 只读性。备份归档的意义在于保留一个特定历史时刻数据的状态，所以归档文件在产生之后通常是一次写入极少更改。“极少的更改”一般发生

在标记删除、归档合并、内部资源增减等操作。因此归档数据具有相当的持久稳定性。无需考虑大量的写入操作，而需要考虑大量的读操作，故会引入缓存策略。

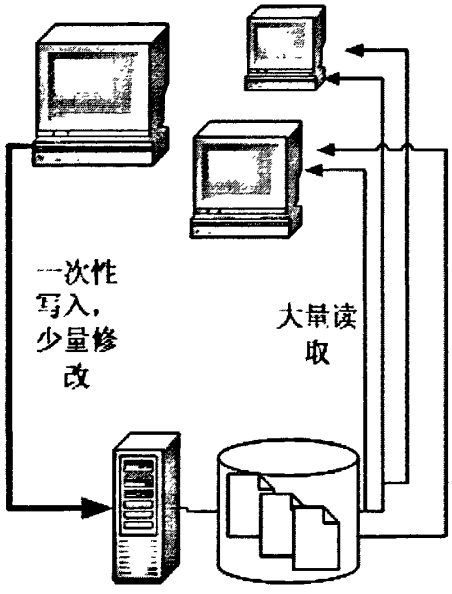


图 3-2

3. 真实有效性。真实性包含两个方面：一是归档数据需要保证真实可靠，恢复之后是有效可用的。二是归档自带的元数据自描述信息确实是反映了其后归档数据的真实情况。因为有了上述的只读性，所以可以方便的在归档产生时以及少数的修改时产生\维护校验码从而随时进行一致性校验，有效的防止归档文件与数据的仿冒、篡改。
4. 结构规范。归档元数据的挑选、格式、数据类型、含义都是经过事先仔细定义的，归档文件、读写双方都遵守这个定义。
5. 级联性。全备与差备的归档之间有相互依赖的关系。在元数据中应当体现这种关系。对于删除、迁移等操作来说，应该考虑级联操作的因素。使得一个生命周期的归档文件集合的有效性具有原子性。

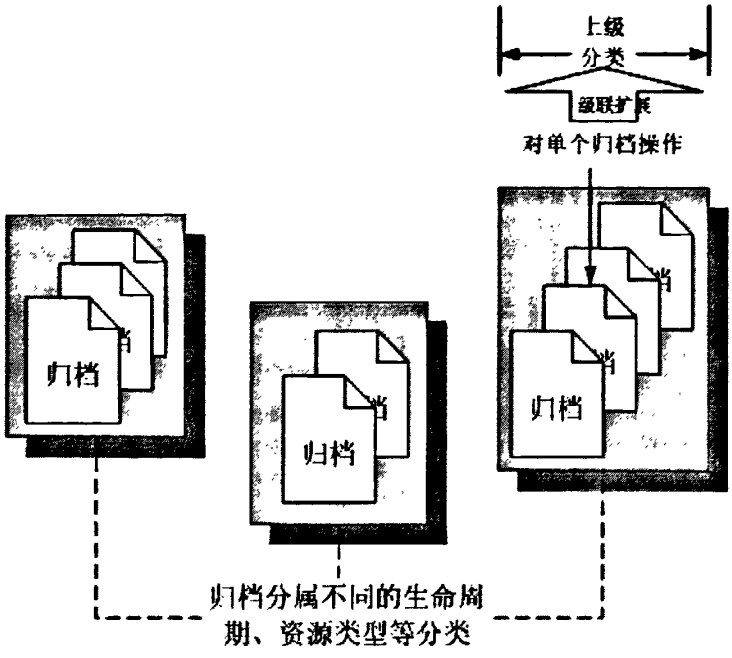


图 3-3

6. 树形结构[4]。任意系统中的所有备份对象皆可做按划分关系做分组，一个分组内部可以按照进一步的指标划分。有限次数的分组的结果是：所有备份对象呈树状分布，树的每一个结点都表示一个分组。

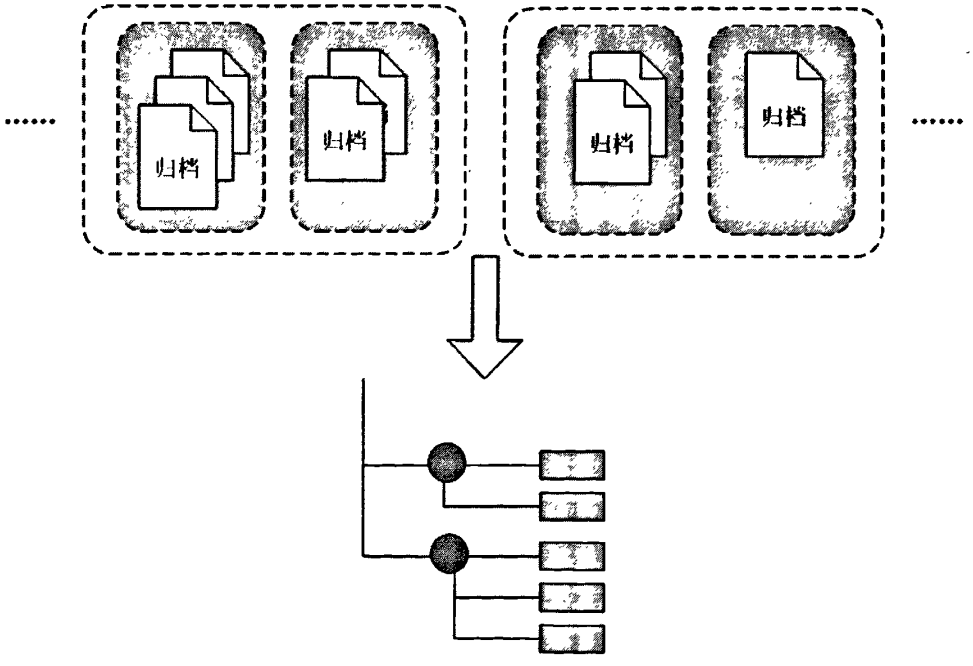


图 3-4

3.3 档案管理中的元数据

在归档文件中也利用了元数据来描述档案自有的信息。作为电子数据嵌入在归档文件头部。具体的组织格式和读写方式在后面章节详述。下面给出这样组织元数据的若干形式化描述：

【定义】元数据是由元数据项（简称项）组成的集合，一个元数据项是一个<key, value>二元组。key 表示项的名称（键），value 表示项值。

元数据 M(a)可表示为集合$\{<k_1, v_1^a>, <k_2, v_2^a>, ..., <k_n, v_n^a>\}$。$k_1, k_2, ..., k_n$是指定的一组元数据项键，$v_n^a$表示在 a 归档中 n 处项的值。

例如可以指定一个元数据项键为 host，值为 192.168.0.100，可以按照习惯记作 host=192.168.0.100。另几项可记作 res_type=oracle，time=20080124 等等。这些项集合成为一份归档文件的元数据。

应该注意这种表示方法暗示在一份归档元数据内各元数据项是没有顺序的。在大多数情况下为了方便讨论，可以规定所有 M 项键遵守同一个有序排列，不妨指定为$<k_1, k_2, ..., k_n>$，则元数据 M(a)可简化表示为有序 n 元组$<v_1^a, v_2^a, ..., v_n^a>$。例如我们指定元数据项键顺序为 time, host, res_type, ...，那么元数据值可记为 20080124, 192.168.0.100, oracle, ...。

3.4 元数据分类树

任意系统中的所有备份对象皆可做按划分关系做分组， 分组的结果是：所有备份对象呈树状分布，树的每一个结点都表示一个分组。该树简称“分类树”。

3.4.1 定义与性质

【定义】分类指标组是元数据项键集的 m 个元素子集形成的一个有序 m 元组<ki1, ki2, ..., kim> (1<m<n, i1...im∈{1...n})。

【性质】按照项键的有序组，总可以从元数据全集开始，将由前一分类指标划分来的子集按照当前指标划分为若干子集。

例如，全集 S 可以按照每个归档 ki1 处的值划分为 S1∪S2∪...∪Sn，进而每个子集如 S1 又会按照其中每个归档在 ki2 处的值划分为 S11∪S12∪...∪S1m。如果将一个集合与其划分为的所有子集视为偏序关系，则这个划分过程就会得到一个偏序关系的集合如{a≺b, a≺c, b≺d}，再加之指定最初的全集 S 作为根，则形成了一个树状结构。

【性质】归档集在按照选定的分类指标组进行分类划分得到树状的结构。

明确了以上概念和性质之后，可以指出元数据树具有如下的性质：

1. 深度固定，元数据树的每一分支都具有 m 的深度
2. 宽度不定，每一节点的出度各不相同
3. 树中除了 root 与叶结点之外的每一级都是一个指标投影下的各元归档数据的象集
4. 叶结点全为（且仅在）元数据节点，非叶节点为按照某元数据项的分类聚合

设元数据 M(a)为{<k_1, v_1^a>, <k_2, v_2^a>, ..., <k_n, v_n^a>}，亦可简化为（如前节讨论）<v_1^a, v_2^a, ..., v_n^a> 。在如后者给定项键顺序<k_1, k_2, ..., k_n>的情形下，多份元数据可形成 n 目关系 R: Range(k1)×Range(k2)×...×Range(kn)，挑选的分类指标为<ki1, ki2, ..., kim> (1<m<n, i1...im∈{1...n})。设 Ij 为 R 在 kj 列上投影的象集，其中元素不妨表列为 i_1^j, i_2^j, ...。

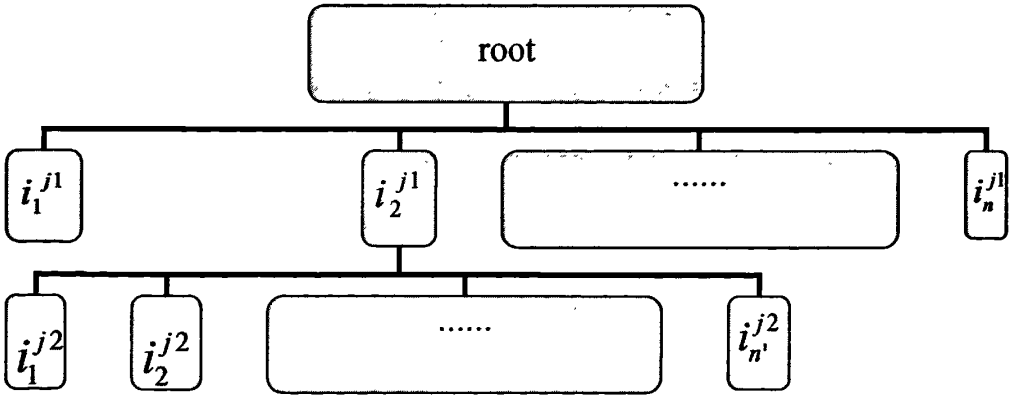


图 3-5

3.4.2 用集合与偏序来表示分类树

假设有树：

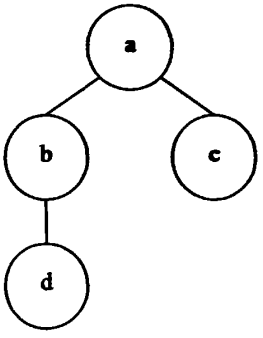


图 3-6

如果将父亲-孩子关系看作便偏序关系≺，并指出树根的话，则如图的树可表示为<a,{a≺b, a≺c, b≺d}>，表示根为 a，然后 b 是 a 的孩子，c 是 a 的孩子，d 是 b 的孩子。应该注意，此方法暗示：兄弟节点（象）之间的次序是不重要的。按照此种表示方法，一棵元数据树可以表示为：<root, {root≺host1, root≺host2, host1≺oracle, ……}>

3.4.3 元数据树生成算法

由归档元数据的集合，依照下面的算法，可以在指定的分类指标下构建出一棵分类树：

```
树初始为<root, Ø>
初始化各分类指标的象集为 I1=I2=...=Im=Ø
递归形式（非递归形式）
遍历每份元数据 M(a)
  遍历每个分类指标 ki
    如果 vai∉Ii 则 在 vai-1 下创建节点 vai，并加入 Ii
    M(a)进入 vai 为祖先的子树
```

3.5 管理操作

对归档集的管理是通过附着于其上的元数据集合进行分类管理实现的。而对元数据进行管理，即是对由元数据节点生成的元数据树进行若干种确定的操作。管理人员从视图概念上操作的是元数据树的子树，实际映射为的是此子树所包含的所有叶节点元数据所在的归档文件。管理操作是档案集管理的关键部分，直接提供给管理人员对归档集合控制的能力。本文总结的若干种常用的档案管理操作，是对档案管理系统能力的整理归纳与约束。

管理操作可以分类为如下几类：归类划分、选择扩展、删除、迁移、合并、一致性校验、搜索等。

- 归类划分 Catalog:: S->∪Si, i∈(1...n)

其中 S 为归档集合，Si 表示划分为的若干子集。

将归档集 S 按照一定的规则（不同的主机、资源类型、生命周期）划分成树状，划分树的叶结点为所有归档，每一层结点都是按照某一标准形成一个划分的子集。通过归档的时态信息，可以得到某备份对象可恢复到哪些时间点。划分结果显示为文本或图形界面。

归类划分操作通常不是由管理人员手动触发的，而是由管理软件整理好呈现给管理人员。具体算法如前元数据分类树生成算法所示。

例如，散落的归档文件有：

图 3-7

每个归档存储了形如下的元数据：

```
- host = 192.168.0.5
- res_type = Oracle
- time = 2007-10-10
- res_id = Oracle_bizdata
```

```
- life_cycle = viton06
- arch_id = viton0601
                ------
```

根据分类树的构建算法可以得到如下的分类树：

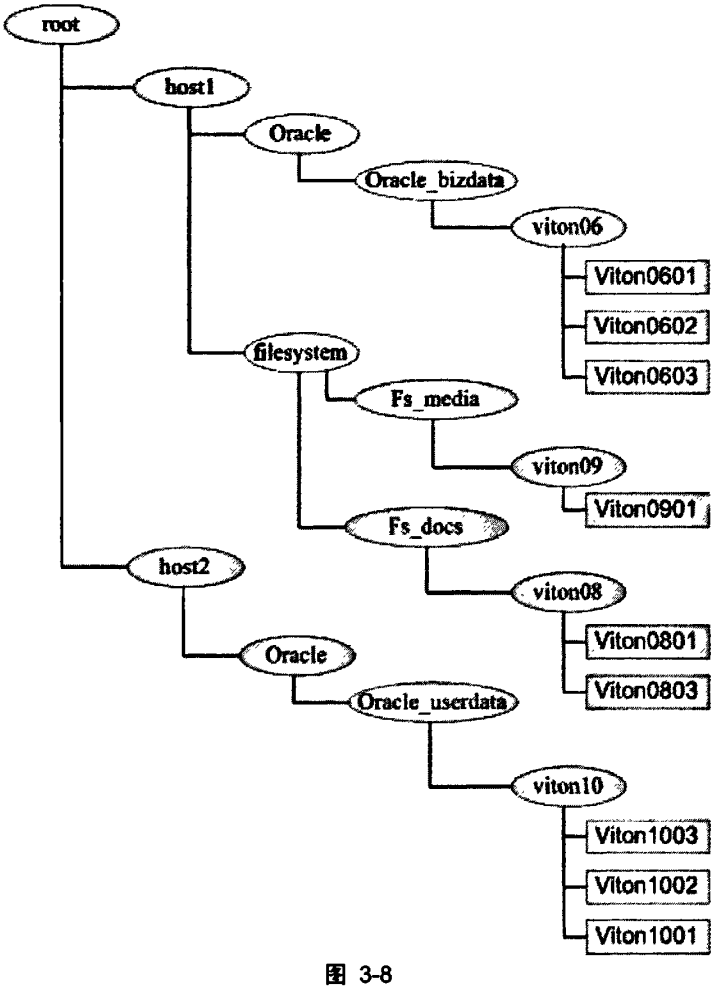


图 3-8

途中方形框代表具体的归档文件，内容为其归档 id。每一级的项键（分类指标）依次为主机、资源类型、备份对象、生命周期、归档 id。椭圆节点为由元数据在对应分类指标上的投影象集。

本操作没有副作用。

- 选择扩展 Select:: Path->expansion level->S'

在归类划分的基础上，可以选择某个具体归档，或某个划分级别所产生的某

个子集。选择了归档或归档集之后，可以向上扩展到指定的层级，如指定某个归档扩展到其所在的生命周期，以便整体恢复或迁移，或删除全备之后同一周期内的差备也会级联删除等。

选择扩展操作即是在元数据分类树上对子树进行操作，给出一个选定的子树（Path），和扩展级别（expansion level），得到新的子树所包含的叶结点归档的集合（S'）。

例如，如前所示的分类树中，我们观察路径为 host1/filesystem/的局部子树，假定有路径 host1/filesystem/fs_docs/viton08 选定了这个生命周期，则选择情况如下图所示：

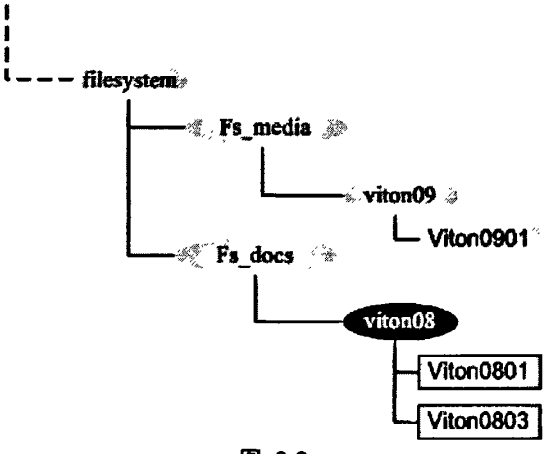


图 3-9

黑底白字的节点 viton08 被路径指定选中，黑方框的叶结点为实际被选中的归档文件 viton0801 与 viton0803。现在若再指定在这个选择基础上向上扩展两级，即选择的节点由路径指定的 viton08 沿树向上溯两级，到达 filesystem 节点。此时的选择情况为：

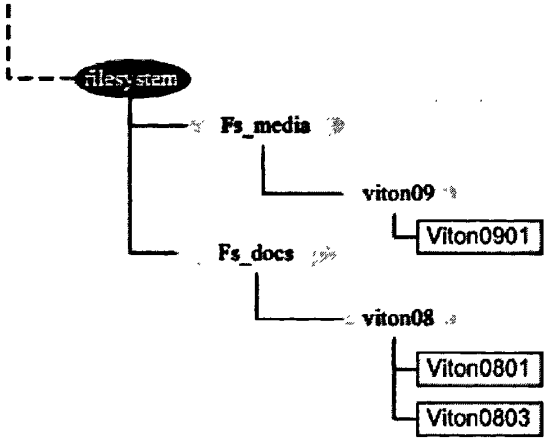


图 3-10

可以看到此时实际被选中的归档增加了 viton0901。

选择扩展的具体算法如下所示：

```
解析 Path 为若干段
按照 expansion level 将 Path 尾部去掉对应数目的段，组合为
新的子树路径 Path'
设 Path' 子树的所有叶节点集合 L 为∅
遍历 Path' 子树（深度优先与广度优先均可），若为叶节点则添加入
L
```

通常情况下此操作不会单独发生，而是作为删除、迁移等操作的级联扩展功能的子操作，由其他操作触发。

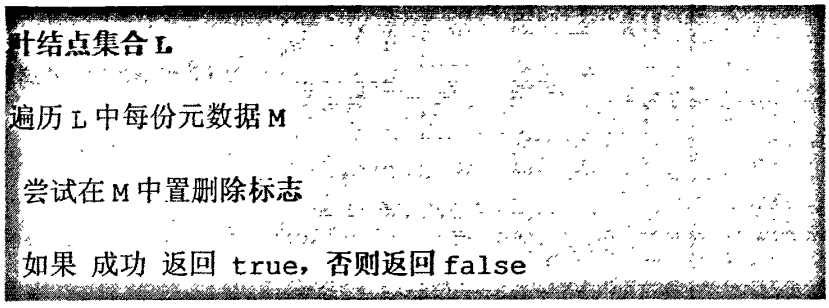
本操作没有副作用。

- 删除 Delete:: Path->expansion level->bool

对指定归档文件进行标志删除或实际删除，根据被删除归档的扩展选择结果可以级联删除。

给出欲删除的起始子树路径 Path，与扩展级别 expansion level（含义同选择扩展操作），得到删除操作是否成功的信息。具体算法如下：

```
根据 Path 与 expansion level 进行选择扩展操作，得到扩展后
```



本操作有副作用。

- 迁移 Migrate:: S m->expansion level->S'm，m∈(1...n)

Sm 表示归档集合 S 中某一个具体的档案节点。将归档文件迁移到另一存储位置，根据扩展级别级联迁移。

- 合并 Merge:: S1->S2->expansion level->Sm, i∈(1...n)

Sm 表示归档集合 S 中某一个具体的档案节点。对于文件系统的数据源此操作有效，可以按照时态信息合并两个归档与其中的文件系统树，并且采取合适的新旧覆盖策略，达到结果为最新最全的面貌。

- 一致性校验 Validate:: Path->bool

检验归档文件数据的校验结果是否与元数据中的校验码一致。防止归档数据有存储错误、被破坏篡改、仿冒等。

算法即为获得 Path 子树的所有叶结点的集合后遍历，进行验证比较操作。略去详细描述。

此操作无副作用。

- 搜索 Search:: S->string keyword expr->result

在指定的路径表示的子集 S 中，对元数据集合进行关键字为 keyword 的搜索。因为元数据文本量少的情况，一般只考虑精确字符匹配。返回匹配的归档文件。

此操作无副作用。

3.6 不同归档类型的操作通用性

【性质】上述的若干种操作，总是作用于一个（由 Path 表达的）集合 S。作用于集合 S 的操作总是将对应的 f 施加于 S 中每一个元素。

【定义】若定义操作一个归档文件的对应一阶函数为 f:: m->m'

【定义】可定义将函数施加于每一个元素的二阶函数为

Map:: (m->m')->S->S'即 f->S->S'={f(m), Map(S-{m})}

【定义】这些操作可统一形式化陈述为二阶函数 F:: (m->m')->S->S'即 f->S->S'=Map(f)(S)，表示将操作 f 施加于 S 中每一个元素得到 S'。

【定义】将作用于 S 中具体一个归档文件的一组若干种操作记为二阶函数 op:: operation_name->f。如 $op_{validate}$ 就得到了对一个归档文件的验证操作。将作用于 S 的一组若干种操作记为三阶函数 OP:: operation_name->F。如 OP_{search} 就得到了对 S 的搜索操作。

下面引入归档类型的概念：

【定义】归档类型：凡是能够引起备份档案间依赖关系的改变/操作内部数据方式改变的全备差备差异、资源类型差异等差异，都区别出一种特定的归档类型。在不致引起混淆的情况下也可以称为备份类型。一个归档文件 m 的归档类型记做 type(m)。

上述的操作定义都是在假定 S 中的每个 m 都是同一备份类型的情况下讨论的，不需考虑个体 m 的操作差异性。如果引入备份类型的概念并根据实际情况合理的假定 S 中的每个 m 并不具有同一个备份类型，以上操作需要重新讨论。op:: operation_name->bck_type->f。即 $op_{search}^{some\_type}$ 才得到一个针对 some_type 的 search 操作函数。

现在来观察 OP。对于一个集合 S，依然只是需要指定操作名称就应该可以得到 S'。即 OP:: operation_name->F 的类型定义不变。下面将类型逐步替换：

OP::operation_name->F

= operation_name->map($op_{operation\_name}$)(S)

= operation_name->($op_{operation\_name}$ ->S->S')

= operation_name->($op_{operation\_name}$ ->S->{ $op^{type(m)}_{operation\_name}$ (m), Map(S-{m})})

由此可见需要对 map 进行重新定义：

map:: f->S->S'={ $f^{type(m)}$ (m), Map(S-{m})}

相应的

f:: bck_type->m->m'

【定义】备份类型操作绑定：设有备份类型 t，则 t 与其所有对应的单归档操作簇 *op'* 形成 t 类型的操作绑定。

实践上，t 类型的定义者（注册者，实现者）也需要提供 *op'* 这一组函数的实现。

在任意一个归档管理系统中，如果存在若干种归档类型，则为了满足管理操作需要必定会注册对应多的归档类型 t，并由第三方开发者提供 t 的定义与相应的操作函数实现，来满足归档管理系统的统一接口，从而使得最终用户在一致的操作习惯和语义下获得不同的功能。

第四章 系统详细设计与实现

4.1 管理程序需求与结构

4.1.1 管理程序需求

在上述的模型之上，我们设计与构建实现了档案集管理的程序工具。

档案集管理程序是为了满足大集合归档管理、元数据模型而实现的软件工具。管理程序具有读取单个归档文件元数据、扫描归档纪元数据构建分类树、对分类树进行若干种管理操作等。

除了满足上述要求，归档管理程序还应该满足工程上的若干便利：

1. 脚本化。具有明晰规范的子命令与选项参数输入结构，方便管理人员用作脚本开发。在备份管理环境中，繁杂多样的数据来源与机械的操作成为备份管理员的主要负担。提供编写脚本的能力是一个备份管理工具能够切实有效的减轻管理员负担与灵活性的重要因素。
2. 插件化。提供利用插件扩展操作类型与操作功能的能力。
3. 结构化输入输出。便于进一步分析输出构建脚本与图形化界面。图形界面工具完全依赖于与命令行工具的输入输出沟通。输入输出的结构化与可解析是能够进行进一步封装和二次开发的关键。

4.1.2 管理程序的基本结构

管理工具的基本结构如下所示：

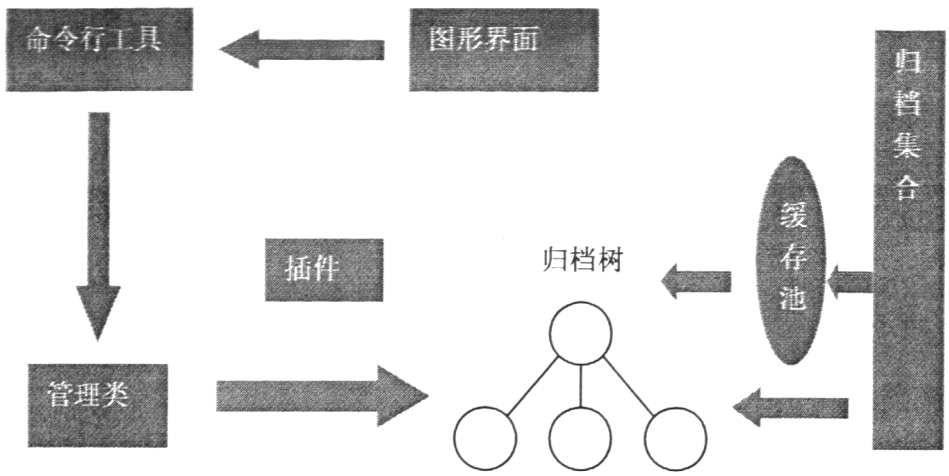


图 4-1

归档集合为已有的磁盘文件，归档读取部件由 C++写成，归档树与管理类为 php 写成，利用了其动态语言的特性灵活构建分类树。缓存部分的归档元数据缓存由 sqlite 存储，C++读写；分类树缓存部分由 php 维护其序列化字符串。命令行工具为 php 写成。图形界面由 VB 构建。

其中归档树和管理类是最为重要的两个部分。归档树由如 3.4 节讨论的算法从归档文件集合中生成（现场或缓存）。

4.2 元数据项的制定

在数据仓库、电子档案管理等领域，元数据的选取指定都有一系列的国际规范。在备份领域是由各厂商根据产品需求自行制定。对于面向个人数据备份的 ProRecovery 来说，元数据应该能够充分准确的反应备份发生时关于数据、配置、操作等一系列信息而又不致繁冗，是的归档体积不致膨胀又能够在脱离外部环境的情况下自描述自身信息。其包含的元数据项应该具有丰富性、确定性、有效性。

丰富性：丰富的项表达了大量的自描述信息，才能为多维度的管理提供可能

确定性：具有哪些项、每项的意义、格式是什么，都应该是确定的，读写双方共同遵照的。

有效性：元数据项应该按照管理目标提供有效的可资利用的数据，应该避免无益于归档管理的冗余信息。

根据以上的管理需求，我们可以充分的、审慎的挑选数据项，作为元数据。应该指出，对于不同的备份系统以及不同目的档案管理系统，管理需求有所变化，因此挑选的元数据也不是一成不变的。一般来说，元数据项集合越丰富，能够适应的管理需求范围就越广，集合越确定，能够实施的有效管理就越准确。两者此消彼长，（需要根据实际管理需要）求得平衡。

ProRecovery 产品继承沿用了（中大软件所与威腾公司）出品的 NetBunker 所使用的 UDS 头结构。此结构定义了一组确定的元数据项，包含主机、备份类型、主机用户名、资源用户名、设备号、作业号、创建时间、过期时间等等一系列丰富的信息。

4.3 元数据的存储与组织

元数据的存储组织方式可分为集中式和分布式两种[13]。集中式的元数据组织采用元数据仓库、元数据数据库等具体方式让各个数据节点的元数据保存维护于一个集中的逻辑/物理节点。分布式的元数据组织让元数据存在于其描述的数据节点处，随数据节点一同迁移。为了体现归档自描述的特性，ProRecovery 的归档元数据与归档文件绑定在一起。

归档文件是由 ProRecovery 的写算子流式写下的二进制文件。元数据信息在所有备份数据之前被写下，嵌在归档文件本身的头部，随着归档文件一同产生、迁移、消亡。元数据长度固定，便于读取并与归档数据相区别。

根据元数据与元数据项的模型，我们可以很自然的构建出一个结构体来表示一份归档元数据。如

```
struct header{
char host[64];
int bck_type;
...
};
```

为了方便读写二进制数据，UDS 在元数据结构体的基础之上构建了一个联合体来做结构数据与二进制数据的转换。示意如：

```
union GeneralHeader{
char data[GHSIZE];
struct{
    version_t version;
    short_ID_t ba_tag;
    ...
};
};
```

通常将 GHSIZE 定为 4096 即 4k 大小，用于作为一个数据块在写算子中首先被写下嵌入归档头部。

写入步骤为：将一个 GeneralHeader 变量的各个结构体项填入实际值，然后通过访问其 char data[]的 union 成员将所有数据一次性写入文件。

读取步骤为：读取文件头 4k 到一个 GeneralHeader 变量的 data[]成员，即可通过结构体各成员访问元数据项。

4.4 元数据树类

元数据树类需要提供将散落的归档元数据根据分类标准组织成树状结构的服务。包括了 GHReader 与 ArchTree 两个部件。

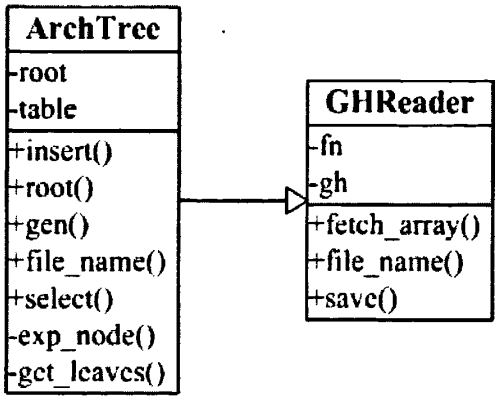


图 4-2

GHReader 部件提供读取单个归档 UDS 数据的能力。ArchTree 类是一个树状

数据结构类，在 GHReader 读取单份元数据的基础之上构建分类树，并提供分类树的遍历访问、选择扩展操作。构建分类树的算法如 3.4 节所述。PHP 的弱类型与动态语言特性为构建分类树带来了极大的便利，生成树的主要代码示意如下：

```
class ArchTree{
private $root;
private $table;
.....
public function insert($gh,$all=false){
    if(is_object($gh)){
        if($all || $gh->del!=="true"){
            $this->root[$gh->host][$gh->res_type][$
gh->res_id][$gh->life_cycle][$gh->arch_id]=$gh;
            $this->table[$gh->arch_id]=$gh->file_na
me();
        }
    }
}
.....
}
```

选择扩展的主要方法代码为：

```
public function select($sel_path,$exp){
    return $this->get_leaves($this->exp_node($sel
_path,$exp));
}

private function exp_node($sel_path,$exp){
    $path_phrases=explode("/",$sel_path);
    $tail=count($path_phrases)-1;
    for($i=$tail;$i>$tail-$exp;$i--){
        unset($path_phrases[$i]);
```

```
    }
    $node=$this->root();
    foreach($path_phrases as $phrase){
        $node=$node[$phrase];
    }
    return $node;
}

private function get_leaves($node){
    if(is_object($node)){
        return array($node);
    }elseif(is_array($node)){
        $set=array();
        foreach($node as $sub){
            $set=array_merge($set,$this->get_leaves
($sub));
        }
        return $set;
    }else{
        return array();
    }
}
```

4.5 缓存策略

理想状况下，分类树的构建可以采取每一次都将归档集的头部扫描，现场生成分类树。但是一来这样效率是不高的，二来因为归档极少被修改的特性，这样也是不必要的。可以通过缓存的方法将已有的归档集扫描信息、构建过的分类树临时保存下来，供下次快速取用。

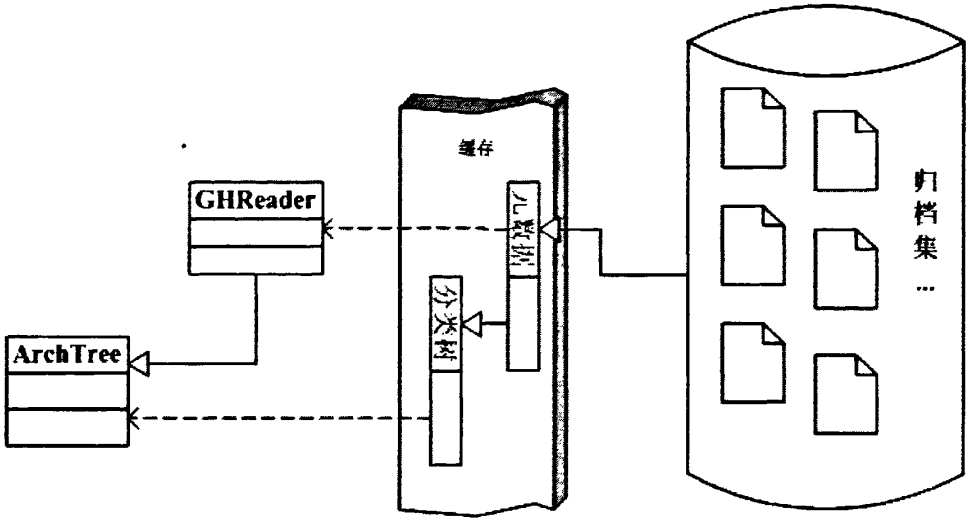


图 4-3

缓存记录了上次扫面缓存的时间。缓存的具体数据分为两部分，元数据的缓存与分类树的缓存。元数据缓存维护一张记录上一次对归档集合元数据扫描的结果表，可以从其还原出上一次扫描得到的各元数据表。分类数缓存维护了上一次根据元数据扫描结果构建的分类树的序列化字符串，程序代码从中可以直接快速生成分类树。

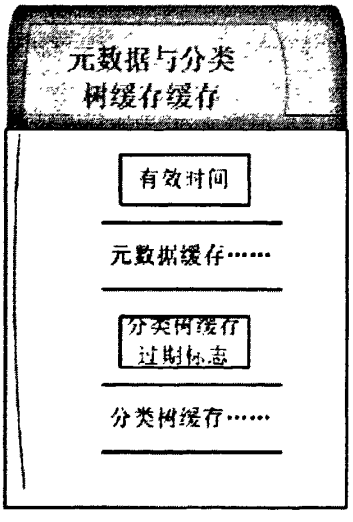


图 4-4

缓存策略叙述如下：

在 GHReader 被上级部件（通常就是 ArchTree）调用时，首先通过缓存记录的上次扫描时间与指定归档的修改时间检查是否在上次扫描之后此归档发生过改动，如发生改动，则重新读取此归档元数据，结果更新至缓存，并向分类树缓存写入标志表明分类树缓存未跟上元数据缓存的更新，最后将新的扫描结果返回。如未发生改动，则直接返回缓存中的该归档元数据。

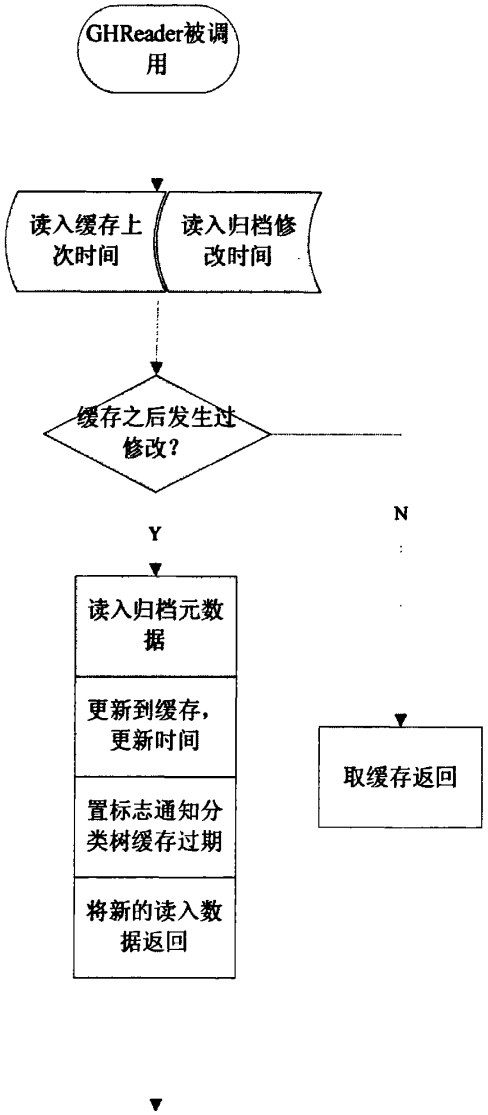


图 4-5

在 ArchTree 被上级部件（通常就是管理类）调用构建的时候，ArchTree 会首

先检查分类树缓存是否跟上了元数据缓存，如没有，则会从 GHReader 重新构建分类树（注意此时 GHReader 也是从缓存读取，如果在上一次元数据更新分类树未更新之后又发生了归档改动，已然可以触发元数据缓存的更新过程）。如果标识检查为最新，则比照缓存中的文件列表与当前文件列表以及上次扫描时间与各归档文件修改时间，以检查是否发生过归档改动，如发生则从重新更新元数据开始从头更新分类树。如果未发生改动，则最后才可以直接返回缓存结果。

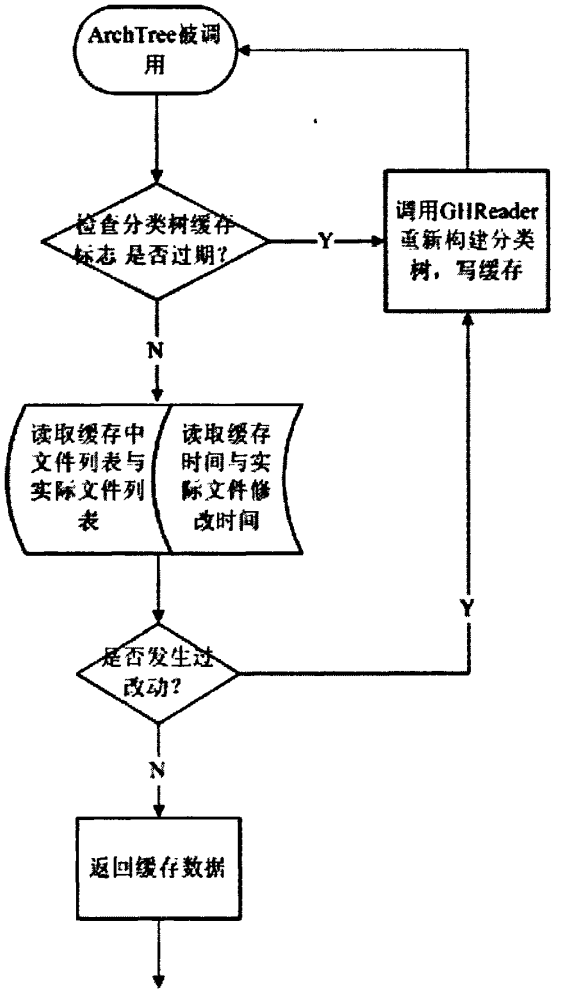


图 4-6

元数据与分类树被访问读取的频率远远高于归档发生改动的频率，因此，绝大多数情况下都是从缓存不加更新的直接返回数据，大大节省了扫描元数据的磁盘 IO 开销与构建分类树的计算成本。

实践上，分类树的缓存采用 PHP 的序列化功能实现，通过 serialize()和 unserialize()函数，将序列化字符串一次性装入恢复出分类树结构。元数据缓存通过 C++调用 sqlite 库来维护本地数据库文件。

4.6 管理类

管理类是对分类树与单个归档文件实施特定若干种管理操作的类。

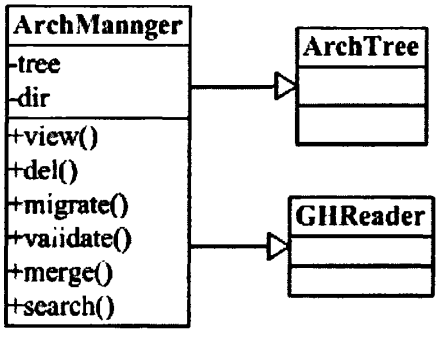


图 4-7

如图所示，管理类提供了前述若干种管理操作的方法，除了 select 操作已经内化于 ArchTree 中。管理类会构建并实例化一个 ArchTree 分类树来对归档集合进行操作，并通过调用 GHReader 来对具体的单个归档文件操作。

在 4.9 节中将讨论为了更好的实现图形界面对命令行界面的封装，通过指定"-xml"选项开关将命令行文本输出为 XML 格式。这一格式的生成也是由 ArchManager 完成。

4.7 命令行界面

4.7.1 需求约束

命令行工具是在上述数据与模型操作类的基础之上提供予归档管理人员的

初步用户界面。命令行工具需要满足的目标有：

1. 通用命令行格式，方便管理人员学习记忆。
2. 适合脚本批处理的二次开发
3. 结构容易扩展，能够利用插件增强类型与功能
4. 结构化输入输出，供进一步用作日志分析或图形界面封装。

4.7.2 命令行格式设计

命令行工具格式的选择采用了主命令-子命令-选项-参数的格式，参考了RMAN[14]与 PEAR[15]工具的命令行设计。通用表示为

archmng subcmd –opt1:arg1 –opt2:arg2 …

其中 arg 不一定要存在，即可以单独有-optn。

根据前节所述的分类树管理操作，我们在 archmng 中实现了若干子命令：view, select, del, migrate, merge, validate, search；分别对应查看分类树、选择扩展、删除、迁移、合并、校验、搜索功能。

4.7.3 命令行内部结构

命令行内部结构主要包括命令解析调用部分、子命令实现部分、与 UI 输出部分。

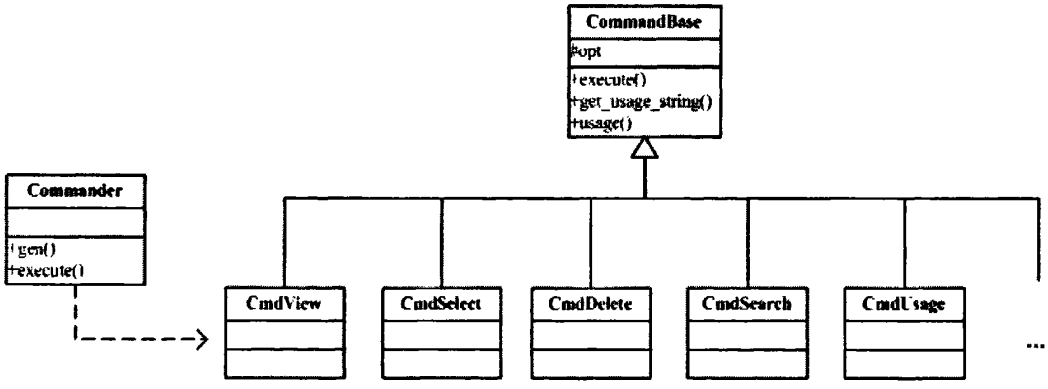


图 4-8

子命令实现采用 OO 方法设计构建了子命令共同基类、子命令子类的继承体系，使得编写拓展新的子命令更加容易。存在一个命令基类为 CommandBase，主要包含了 execute 抽象接口供子命令子类实现与一些用法帮助说明功能相关的接口。一个子命令由一个对应的子类继承 CommandBase 来实现，主要在构造方法中解释基类得到的选项与参数的意义，并在实现其 execute 接口的代码中完成命令具体功能。

Commander 对象为一工厂对象，用来根据传入的子命令串来产生相对应的子类，或可由子命令串直接执行。

如前所述，命令行结构采取了主命令-子命令-选项-参数的格式：

archmng subcmd –opt1:arg1 –opt2:arg2 …

archmng.php 负责了这个解析和调用的过程。大致步骤为：按照空格与冒号解析选项-参数，存入全局变量；按照子命令字符串通过 Commander 工厂产生命令子类并执行。

命令行 UI 组件采用了单根类体系设计。

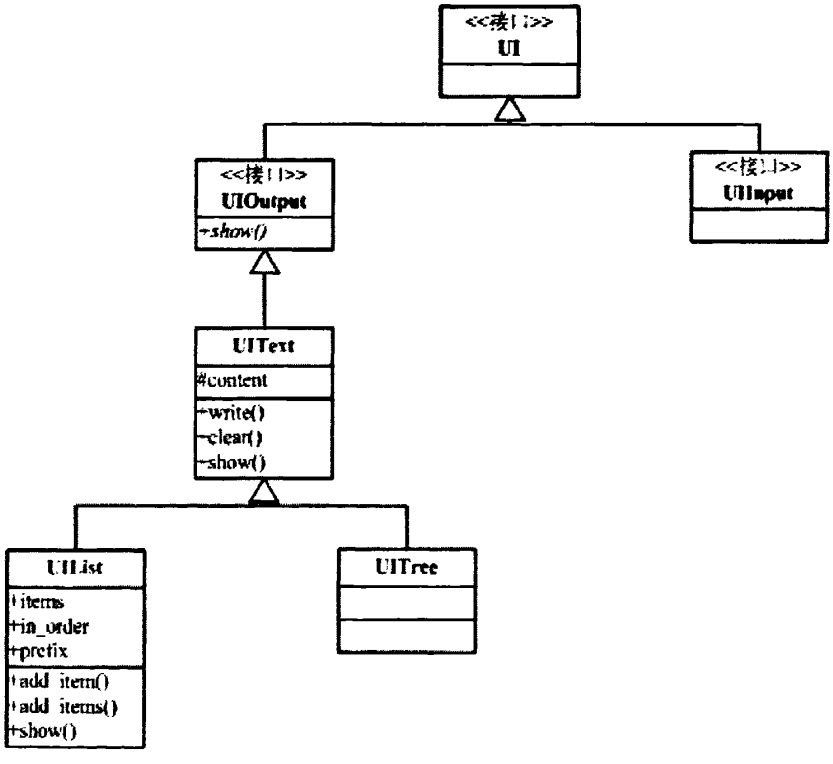


图 4-9

统一的 UI 接口派生出 UIOutput 与 UIInput 两个接口分别用于输出与输入。其中 UIOutput 的虚方法 show 指明了输出内容的方法。UIText 实现了命令行输出最基本的功能：输出文本。派生出的 UIList 与 UITree 都是在其基础上增加了项目、排序、前缀等结构化性质。

管理命令行工具的所有输入输出都在此体系的基础之上进行。因此可以对这个类体系进行替换，适合不同的输入输出场景，如图形界面，远程终端，web 页面等。

4.7.4 子命令功能与命令行输出设计

- 查看 View

列出单个文件的 UDS 头内容：

```
>archmng view -f:archives\01.arch
- host = 192.168.0.5
- res_type = Oracle
- time = 2007-10-10
- res_id = Oracle_bizdata
- life_cycle = viton06
- arch_id = viton0601
```

此输出采用 UIList 输出，列表项开头的“ - ”前缀可以定制。

列出一个目录下所有归档文件的元数据分类树：

```
>archmng view -dir:arch
1.192.168.0.5
2.    Oracle
3.        Oracle_bizdata
4.            viton06
5.                viton0601
6.                viton0602
7.                viton0603
```

```
8.    filesystem
9.      fs_media
10.       viton09
11.         viton0901
12.     fs_docs
13.       viton08
14.         viton0801
15.         viton0803
16.192.168.0.9
17.   Oracle
18.     Oracle_userdata
19.       viton10
20.         viton1003
21.         viton1002
22.         viton1001
```

此输出采用 UITree 控件，每项前缀编号方便选择某项的功能扩展。输出的为一个具体目录下的元数据分类树，按照主机、资源类型、备份集、生命周期四级分类，叶端都是具体的归档文件。

列出一个目录下包括标记删除了的所有归档文件的元数据分类树：

```
>archmng view -dir:arch -all
```

后述的删除操作将会对归档进行标记删除，已标记删除的归档在 view 时不会纳入分类树。如果进行 view 命令时加上-all 选项可将标记删除的归档也纳入进来。

- 选择扩展 Select

选择一个路径，返回其下归档的集合：

```
>archmng select -dir:archives -path:192.168.0.5/f
ilesystem/fs_docs/viton08
- viton0801
- viton0803
```

通常此命令在命令行状态下没有明显的作用，是为了扩展功能配合别的命令使用。

选择一个路径，指定扩展级别，返回从此路径节点出发扩展相应级别后包含的归档集合：

```
>archmng select -dir:archives -path:192.168.0.5/f
ilesystem/fs_docs/viton08 -exp:2
- viton0901
- viton0801
- viton0803
```

上面这个命令，在指定建树目录为 archives 的情况下首先选定了一个路径节点 192.168.0.5/filesystem/fs_docs/viton08

- 删除 del

标记删除归档树上某个指定路径以下所有节点，可以指定扩展级别：

```
>archmng del -dir:archives -path:192.168.0.5/file
system/fs_docs/viton08 -exp:2
>archmng view -dir:archives
1.192.168.0.5
2.    Oracle
3.        Oracle_bizdata
4.            viton06
5.                viton0601
6.                viton0602
7.                viton0603
8.192.168.0.9
9.    Oracle
10.       Oracle_userdata
11.           viton10
12.               viton1003
13.               viton1002
```

```
14.                viton1001
```

上面的结果可以看到 del 之后的子树不会出现在 view 中。如果 view 加上-all 选项可以看到被删除的归档子树，说明是被标记删除的：

```
>archmng view -dir:archives -all
1.192.168.0.5
2.    Oracle
3.       Oracle_bizdata
4.          viton06
5.             viton0601
6.             viton0602
7.             viton0603
8.    filesystem
9.       fs_media
10.         viton09
11.            viton0901
12.      fs_docs
13.         viton08
14.            viton0801
15.            viton0803
16.192.168.0.9
17.    Oracle
18.       Oracle_userdata
19.         viton10
20.            viton1003
21.            viton1002
22.            viton1001
```

强制删除，会删除归档文件：

```
>archmng del -dir:archives -path:192.168.0.9/Orac
le/Oracle_userdata/viton10/viton1001 -force
```

再使用 view –all 命令就看不到强制删除的归档了：

```
>archmng view -dir:archives -all
1.192.168.0.5
2.    Oracle
3.       Oracle_bizdata
4.            viton06
5.                 viton0601
6.                 viton0602
7.                 viton0603
8.    filesystem
9.       fs_media
10.           viton09
11.                viton0901
12.      fs_docs
13.           viton08
14.                viton0801
15.                viton0803
16.192.168.0.9
17.   Oracle
18.      Oracle_userdata
19.           viton10
20.                viton1003
21.                viton1002
```

- 合并 merge

此类操作与归档类型相关，需要使用插件来扩展其能力，由第三方具体实现，在下节详述。

- 一致性校验 validate

校验单个归档文件：

```
>archmng validate -f:archives\01.arch
ok.
```

校验一个目录下所有归档文件，可以通过选项-all 来指定是否包含已标记删除的归档：

```
>archmng validate -dir:archives
9 ok.
```

- 搜索 search

在指定路径范围内搜索关键词，并可以指定扩展级别。，可以通过选项-all 来指定是否包含已标记删除的归档：

```
>archmng search -dir:archives -keywords:Oracle
1.192.168.0.5
2.    Oracle
3.        Oracle_bizdata
4.            viton06
5.                viton0601
6.                viton0602
7.                viton0603
8.192.168.0.9
9.    Oracle
10.       Oracle_userdata
11.           viton10
12.               viton1003
13.               viton1002
```

4.8 插件-扩展管理类与命令行

上述几类操作中，大多数都是将归档作为一个不可再分的整体以及由其组成

的集合来进行操作的。但是有两类在归档管理中可能出现的操作需求不一定约束于这个限制：

一类如 merge 这样的操作，操作对象与结构都还是整体的归档文件，操作形式一致，但是具体的合并方法涉及到归档内部的备份数据的类型与结构。

一类是如表列归档数据内容，抽取部分数据、搜索归档内容等这样的操作，已经不是将归档文件作为一个黑盒部件，而是需要读取内部的备份数据。

这两类操作都不完全是归档集的操作，但确是归档管理可能的需求，所以虽然在管理类与分类树中没有内建提供这些操作，但是我们构建了一种类型注册与插件的机制来让第三方开发者有机会扩展管理类与命令行工具。

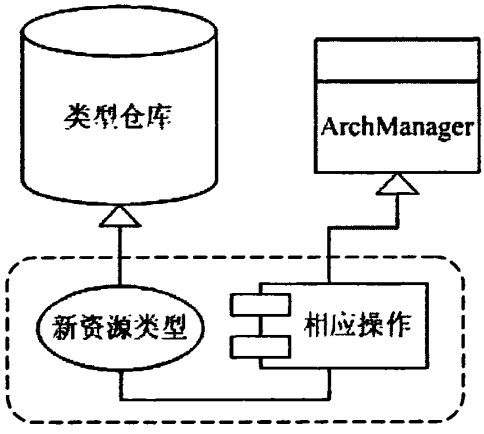


图 4-10

第三方开发者需要做三件事情：

一是向资源类型仓库注册一个新的类型，写明 res_type_id；

二是继承实现已经提供的 ArchMngExt 类，与 res_type_id 关联，提供相应的操作：实现 merge，扩展其他有名操作，指明禁用的操作。

三是继承实现 CmdExt，实现子命令解析到 ArchMngExt 子类的调用。

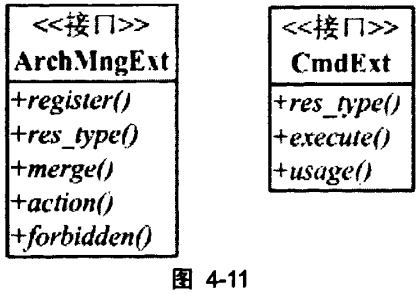


图 4-11

ArchMngExt 通过 res_type 将某子类实例与具体的注册类型关联起来。接口留有虚方法 action(action_name,arg_list)，可以根据操作名实施具体的操作代码。forbidden 方法指明了在现有提供的操作中需要禁用哪些操作。

特别的，其中 register 方法完成了几项工作：将类型 id 与类型仓库注册，通知 Commander 类需要实例化相应 res_type 的 CmdExt 子类。

例如，我们需要新增对邮件备份档案的操作功能。首先扩展 ArchMngExt 类：

```
Class MailArchMng extends ArchMngExt{
(指明 res_type 如 res_mail_arch)
...(实现相应方法)
}
```

邮件档案的合并据有其特殊性，例如按照时间顺序线性的合并，或者按照话题合并，等等，均由第三方实现者定义。邮件档案还可能支持更多的操作例如直接回复邮件、查看联系人等，可以在 action 中调用特殊的实现。

然后实现者需要扩展 CmdExt 类：

```
Class CmdReply extends CmdExt{
(指明 res_mail_arch 类型)
...(解析命令行参数调用 MailArchMng 的 action("reply"))
}
```

通过系统在类型仓库初始化的时候发现 MailArchMng 类，调用其 register 方法从而出测了邮件归档类型、获得扩展功能的实现代码、扩展命令行解析，就可以处理邮件类型的归档了。如：

```
Archmng reply -arch:some/path/to/arch -mail_id:MA
087
```

从而可以在一个指定的具体归档中回复指定 id 的邮件，具体的方式如调用打开邮件程序。

4.9 图形界面工具

在命令行工具的基础之上，作者构建了图形界面工具 archmng_win。使得在 win32 环境下，能够将指定位置的归档集合在一个统一的图形界面中表示出分类树及其详细信息查看，并可支持各种归档操作。特别的，图形界面工具的显示信息和功能完全依赖于与命令行工具标准输入输出的交互。

archmng_win 的操作界面如下图所示：

登陆验证进入

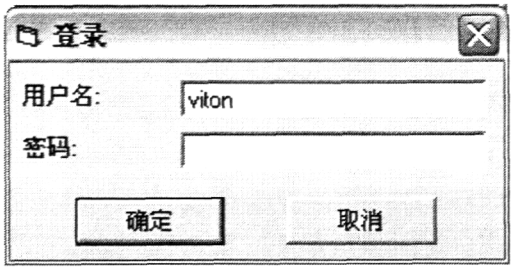


图 4-12

选择归档集合的文件夹：

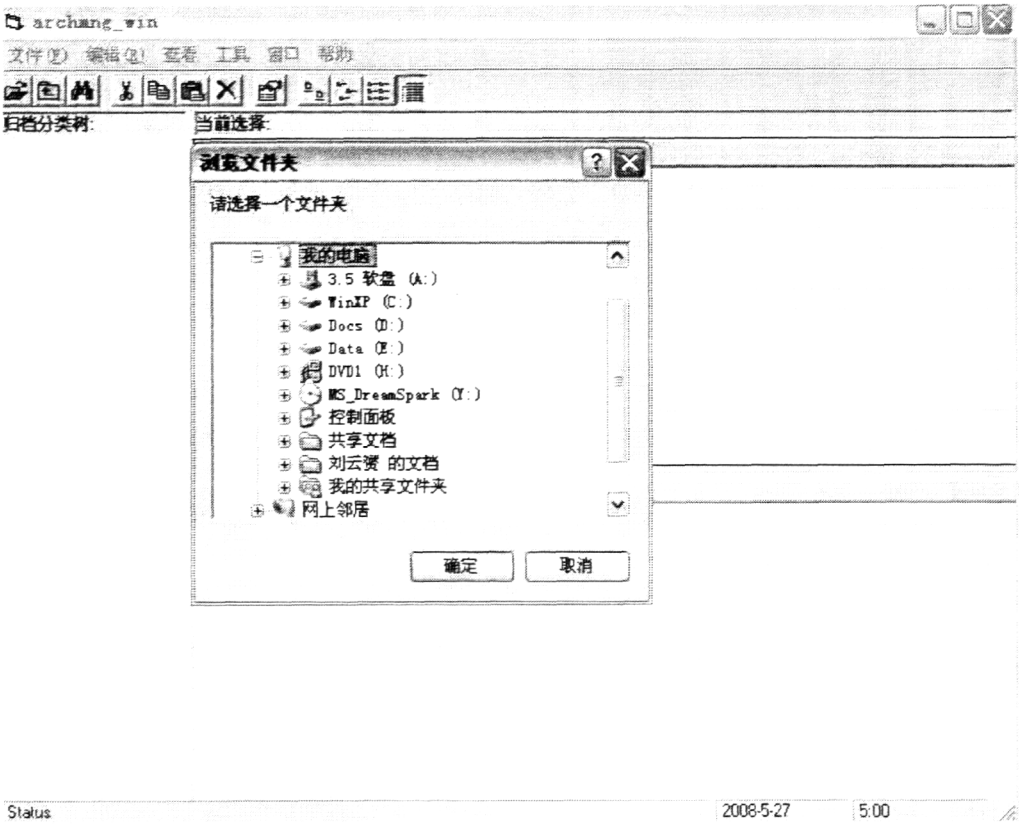


图 4-13

生成了分类树，在分类树上选择节点，可于右侧得到选择的归档叶结点：

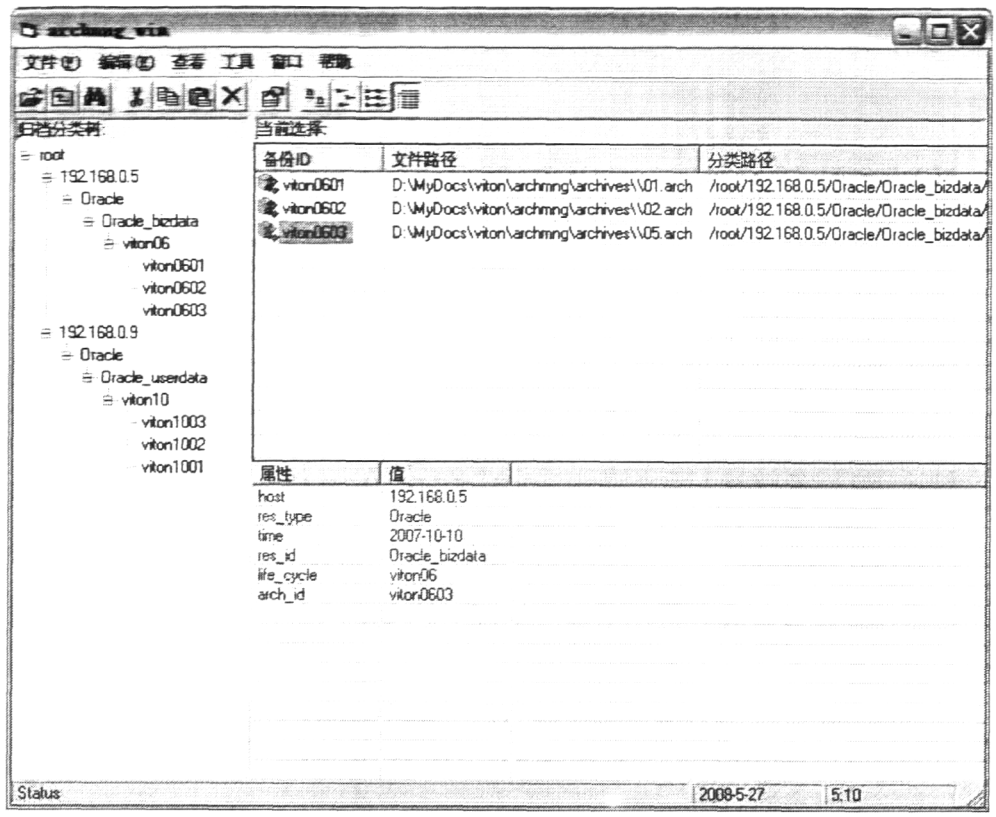


图 4-14

在右侧上部选择具体归档，可看到具体的元数据信息，以及右键菜单操作：

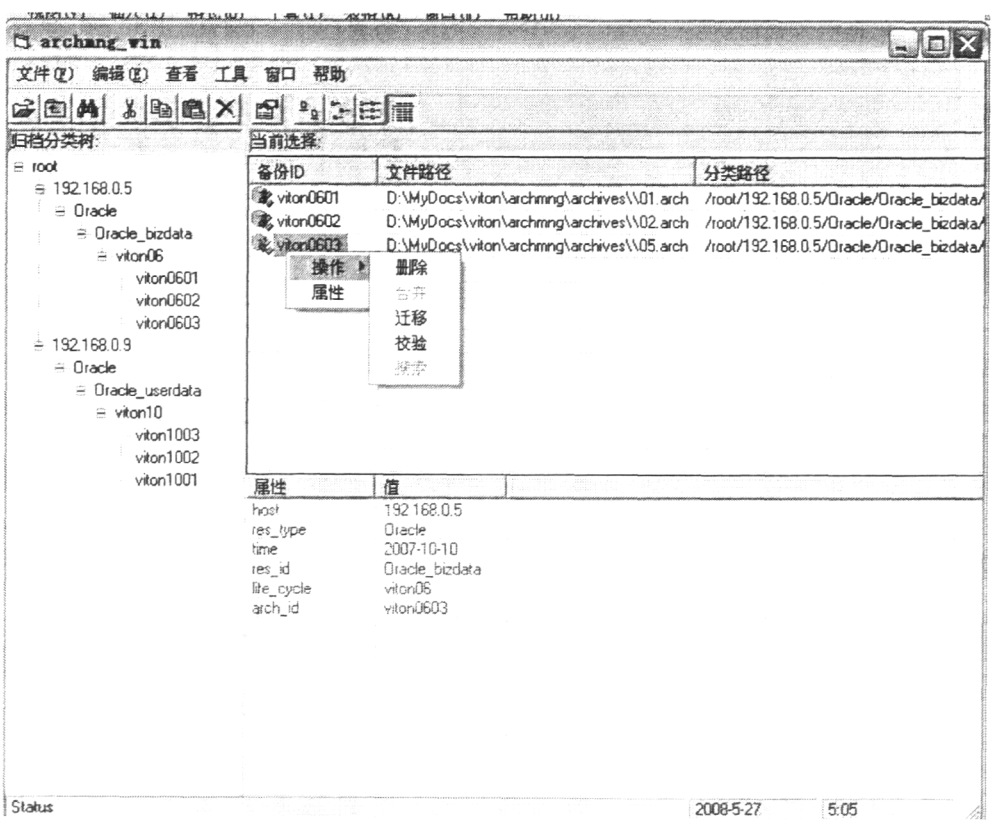


图 4-15

作为命令行工具的封装，图形界面工具没有直接操作归档数据，只与命令行工具进行交互。由图形界面生成命令行参数文本执行，返回的结果解析为图形界面表达。例如，左侧分类树的构建即是对应执行命令"archmng view –dir:some_dir"。在分类树种选择入右侧明细表格对应命令"archmng select –dir:some_dir –path:some_path"。右侧下方的元数据明细信息对应命令"archmng view –f:some_file"。右键菜单中的各项操作分别对应如前所述字面上的各种归档操作命令。

为了使得命令行工具的返回文本更容易被图形界面解析，作者在命令行工具中设计了"-xml"选项开关。不使用此选项开关，执行命令得到如前所是正常的文本结果。如果使用此开关执行命令，将得到适合图形界面解析的 xml 结果文本。如，分类树的 xml 文本输出：

```
>archmng view -dir:archives -xml
<node path="/root" level="0" text="root">
 <node path="/root/192.168.0.5" level="1" text="
192.168.0.5">
  <node path="/root/192.168.0.5/Oracle" level="
2" text="Oracle">
   <node path="/root/192.168.0.5/Oracle/Oracle
_bizdata" level="3" text="Oracle_bizdata">
    <node path="/root/192.168.0.5/Oracle/Oracl
e_bizdata/viton06" level="4" text="viton06">
     <node path="/root/192.168.0.5/Oracle/Ora
cle_bizdata/viton06/viton0601" level="5" text="vi
ton0601">
</node>
.....
</node>
</node>
</node>
</node>
```

选择命令的 xml 文本输出：

```
>archmng select -dir:archives -path:192.168.0.9/O
racle/Oracle_userdata/viton10 -xml
<selected>
<item id="viton1003" filename="archives\06.arch"
/>
<item id="viton1002" filename="archives\07.arch"
/>
<item id="viton1001" filename="archives\08.arch"
/>
</selected>
```

图形界面可以方便的解析如上的命令行结果输出，从而只依赖于命令行工具而不操作具体的归档文件。形成如下结构：

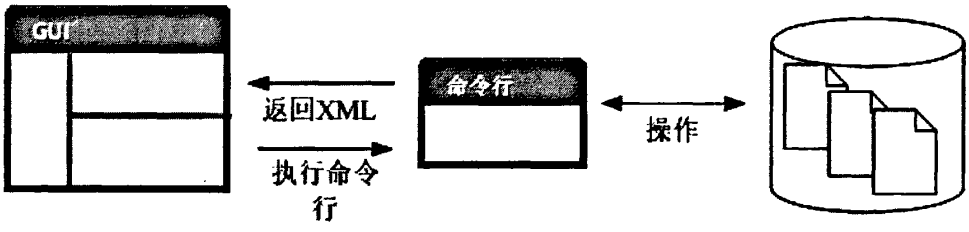


图 4-16

archmng_win 可以指定执行的命令行工具的路径，特别的，可以是 URL 远端路径。archmng_win 与 archmng 之间仅存在文本通信，因此通过 http 服务器交换文本进行通信是容易的。可以做到归档集合与命令行工具在网络中的一台主机中，而图形界面管理工具在网络中的另一台主机，可以远端实施有效的管理。如下图所示：

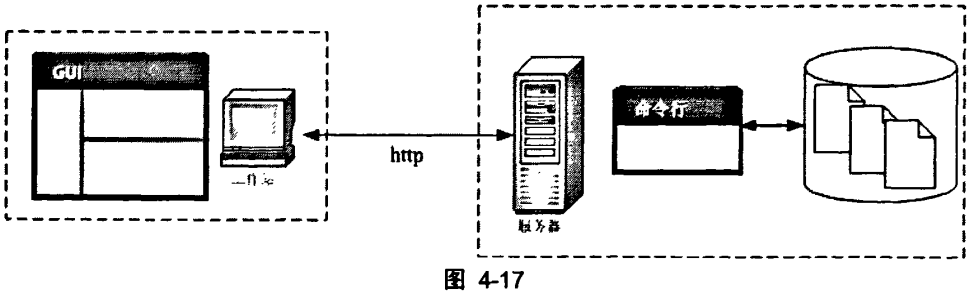


图 4-17

远程一端可架设 http 服务器如 apache，通过一个简单的 php 页面来执行 http 客户端将要传过来的命令字符串，并且将执行结果的文本返回。为了防止未授权的客户端联入，服务器端加入了授权字段。服务器页面代码简要示意如下：

```
<?php
If(!is_authorized($_GET['auth'])){
exit;
}
$archmng="/path/to/local/archmng";
$lines=array();
exec($archmng.$_GET['cmd'],$lines);
foreach($lines as $line)echo "$line\n";
?>
```

本地一端，在 archmng_win 设置对话框将命令路径设为远程，url 填入诸如 http://some_host/cmd.php 。则 archmng_win 生成命令时将发送形如 http://some_host/cmd.php?auth=xxx&cmd=xxx 的 http 请求来得到结果。

第五章 总结与展望

本论文讨论了大规模数据集备份的情形下，利用嵌入归档文件头部的自描述元数据信息对散落的归档文件集合实施有效管理的方案。

5.1 总结

在归档管理中主要面临的问题特点以及对策结论有：

1. 归档文件应自带描述自身信息的元数据，不论在有组织还是在散落的情况下归档管理程序都应该能够重建归档树。
2. 归档元数据的集合具有树状分类特征，通过分类树的视图，能够方便管理人员掌握归档的有效覆盖情况。
3. 对归档集合所进行的操作，都通过分类树来进行。特定的若干种操作具有一致的接口界面，针对不同类型归档的操作在一致接口的情况下可以由第三方扩展。

本文的主要工作成果还有：

- 对归档环境中的元数据、元数据分类树做了半形式化的归纳描述。对归档集合上进行的各种操作进行了总结，给出了相应的算法描述。提出了归档系统管理的通用特性。
- 设计了实际操作归档集合的软件工具 archmng，设计了分类树数据结构、归档管理类等可重用组件，设计实现了一套命令行工具框架和一组命令行 UI 类体系，都能够通用复用于命令行工具构建与输入输出结构化设计上。
- 缓存策略设计。在归档集的分布式元数据管理基础上适当引入集中元数据缓存的思想，设计实现了元数据与分类树缓存机制，利用归档极少改动的特点在大规模归档集的情形下能够保持良好的性能。
- 实现了 archmng 命令行工具，设计了结构化的命令行输出，使得工具适用于脚本编写、图形界面封装。设计实现了处理不同归档类型的插件体系结构，能够由第三方开发者提供更多的扩展功能。

- 实现了 archmng_win 图形界面管理工具，只依赖于与命令行工具的标准输入输出通信，并且可以远端管理命令行工具与归档文件。

5.2 展望

归档管理是备份归档软件用于实际使用时的重要特色功能之一，是体现归档与备份的实际价值与商业价值的入手点。本文对归档管理进行了初步的探讨与建模。在以下方面可以继续做相应的拓展工作：

1. 在缓存策略方面可以进一步改进提高效率，考虑缓存大小，在海量归档数据的情况下缓存大小必须受限。受限时如何提高缓存命中率。
2. 在插件扩展方面应该更进一步进行严格的建模与讨论。目前扩展命令与常用命令还有所区别，虽然做到了对于最终用户的一致性但还没有做到对于开发者的一致性。
3. 构建了命令行工具之后有望通过仔细设计命令行的输出从而在此基础之上：
   a) 进行备份管理实用脚本的编写
   b) 仔细设计结构化输入输出使得与图形界面的交互更具有通用性

参考文献

[1]. David Wallance. Metadata and Archival Management of Electronic Records. Archivaria. 1993.

[2]. A Erlandsson. Electronic Records Management: A Literature Review. International Council on Archives, Paris, 1996

[3]. 威腾数据备份项目组 NetBunker 软件系统概要设计文档, 第一版. 广州威腾网络科技有限公司, 2003.

[4]. 张金波. 数据备份中的关键问题建模与应用，中山大学硕士学位论文，2004

[5]. W.Curtis Preston 著, 李如貂, 钟日红, 王森等译. Unix 备份与恢复. 机械工业出版社，2003 年 1 月.

[6]. 李菁菁. 网络环境下跨平台的通用备份模型的研究与实现. 中山大学硕士学位论文，2005.

[7]. 林峰. 基于再生树模型的备份集管理研究及其应用，中山大学硕士学位论文，2006

[8]. 张巍. ProRecovery 插件系统设计文档. 广州威腾网络科技有限公司，2008

[9]. 潘志宇. 数据备份系统中变换模型的研究与设计 中山大学硕士生学位论文 2006

[10].中华人民共和国国家标准 电子文件归档与管理规范

[11].陆小辉. 元数据在电子文件管理中的应用. 科技广场. 2007 年 9 期 :219-221

[12].沈凤善. 元数据的类型、模型与规范. 牡丹江师范学院学报(自然科学版) 2006 年第 4 期: 22-23

[13].彭丹. 分布式元数据管理. 现代计算机. 总第 265 期: 58-59

[14].Darl Kubn, Scott Scbulze. Oracle RMAN Pocket Reference. O'Reilly Press. 2002

[15].http://pear.php.net

[16].R P King, N Halim, H Garcia-Molina, et al. Management of Remote Backup Copy for Disaster Recovery[J]. ACM Trans on Database Systems, 1991,16(2):338-368.

[17].Schellenberg, T.R.. Modern archives. Principles and techniques. University of Chicago Press, Chicago. 1956

[18].周功业 吴伟杰 陈进才. 一种基于对象存储系统的元数据缓存实现方法. 计算机科学. 2007 Vol.34 No.10 :146-148

[19].杨德志 许鲁 张建刚. BWMMS 元数据分布信息缓存管理. 计算机科学. 2007 年 10 期

[20].JM Skinner, JW Pflugrath, RM Sweet. Development of a GUI for an existing command-line driven program. Proceedings of Summer 1992 Meeting/Share, 1992

[21].MS Kressin, BH Berger, BP Smith. Method for adding a graphical user interface to a command line application. US Patent 5,617,527, 1997

[22].DA Wallance. Managing the Present: Metadata as Archival Description. Archivaria, 1995

致谢

两年的硕士研究生生涯转眼之间就过去了。在此，我要感谢所有在研究生学习阶段给予我指导、在论文完成过程中给予我帮助的老师和同学。

本文课题是在倪德明导师的指导之下得以完成的，我在这里向倪老师表示深深的谢意。两年来，倪老师给予了我很多的指导和支持，是倪老师将我引入了容灾备份与大规模数据集处理这一丰富多彩的领域，正是倪老师的指导和帮助才使我能够顺利完成论文工作。在倪老师的关心和鼓励下，我在专业知识、实践能力等各方面都有所提高。倪老师渊博的学识、踏实的治学态度和严谨的工作作风也令我钦佩不已，受益匪浅。这两年能够得到倪老师的教导是我一生中宝贵的财富，我将铭记于心。

同时，我还要感谢曾经教导过我的其它老师，是他们的授课提高了我各方面的理论知识水平。在此还要特别感谢我的同门，张巍，吉嘉，曾芳，杨志明，是他们给我的建议和帮助，使得我能够拓展思路，及时取得论文的新进展。并在一起进行开发和讨论，度过了难忘的愉快时光。

在这里还要特别感谢我的家人，是他们的支持使我能够顺利的完成两年的研究生学习。还要感谢所有关心、帮助过我的亲人、师长与朋友们。

最后，感谢栽培我的中山大学，我终生以曾经在国内一流名校中学习知识而自豪。