电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

硕士学位论文

MASTER THESIS

论文题目	基于 FastDFS 的目录文件系统的研究与实现
学科专业	计算机应用技术
学　　号	201221060457
作者姓名	周子涵
指导教师	戴元顺　　教授

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

基于 FastDFS 的目录文件系统的研究与实现

周子涵

指导教师 戴元顺 教授

电子科技大学 成都

申请学位级别 硕士 学科专业 计算机应用技术

提交论文日期 2015.03.31 论文答辩日期 2015.05.27

学位授予单位和日期 电子科技大学 2015 年 6 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

RESEARCH AND IMPLEMENTATION OF THE DIRECTORY FILE SYSTEM THAT BASED ON FASTDFS

A Master Thesis Submitted to
University of Electronic Science and Technology of China

Major:	**Computer Applied Technology**
Author:	**Zhou Zihan**
Advisor:	**Prof. Dai Yuanshun**
School:	**School of Computer Science & Engineering**

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：_____ 日期：　　年　　月　　日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：_____ 导师签名：_____

日期：　　年　　月　　日

摘　要

云存储是在云计算的概念上延伸和发展出来的一个新的概念，是一种新兴的网络存储技术。云存储利用集群应用和分布式文件系统等软件，将网络中大量类型不同、容量不同的存储设备连结在一起，作为一个整体协同工作并以一个统一的方式对外提供数据存储和访问服务。当云计算系统中配置了大量存储设备，并且将系统运算和处理的重点放到数据的存储和管理上而不是数据的计算上，那么该云计算系统就变为一个云存储系统。也可以说云存储是一种新的存储方案，它将存储资源统一集中放至云端，这样无论用户身处何处都可以随时通过网络访问云上的数据资源。

FastDFS 是一款开源的高性能分布式文件系统。它的主要功能包括文件存储、文件同步和文件访问（如文件上传和文件下载）等，同时满足大容量和负载均衡的要求。本文将在实验室现有云平台的基础上，探索一种新的解决方案以满足平台对云存储的要求。新解决方案将根据云存储的各项需求，在 FastDFS 的基础上增加以目录服务为核心的新功能，设计基于 FastDFS 的目录文件系统。

本文首先阐述了基于 FastDFS 的目录文件系统的总体设计。从对整个系统的功能和性能的需求分析开始，总体设计包含了概要设计、总体架构、模块设计、并发访问与可靠访问设计、数据安全与隐私保护设计等五个方面。总体设计确定了基于 FastDFS 的目录文件系统的大体框架，在这个大体框架下本文紧接着阐述了整个系统的详细设计与实现。这其中主要包括对目录结构、公共接口和数据安全三个方面的详细设计与实现。这三个方面构成了以目录服务为核心的新功能。本文最后对整个系统进行了测试，结果证明基于 FastDFS 的目录文件系统符合预期设计。

关键词：云存储，分布式文件系统，FastDFS，目录服务

ABSTRACT

Cloud storage is an emerging network storage technologies extending from the concept of cloud computing. Using clustering applications and distributed file system, cloud storage links a large number of different types and different capacity storage devices in the network together. Cloud storage work together in a unified way providing external data storage and access services. When the cloud computing system is configured mass storage device，and focuses on data management and storage rather than calculated data，it becomes a cloud storage system. In other words, cloud storage is a new storage solution. It puts all storage resources on the cloud, and then makes users access storage services no matter when and where.

FastDFS is an open source and high-performance distributed file system. Its main functions include file storage, file synchronization and file access (such as file uploading and downloading). At the same time, it meets the requirements of large capacity and load balance. This thesis will explore a new solution to meet the demand for cloud storage which is based on the existing cloud platform in the laboratory. The new solution will be according to the requirements of cloud storage, on the basis of FastDFS increase with directory service as the core of new features, and design a directory file system based on FastDFS.

This thesis expounds the overall design of based on FastDFS directory file system firstly. On the function and performance of the whole system of requirement analysis, general design includes the profile design, overall architecture, module design, design of concurrent access and reliable access, data security and privacy protection design and so on. Overall design determines the based on FastDFS directory file system’s general framework. Under the general framework, the thesis expounds the detailed design and implementation of the whole system. That mainly includes the directory structure, public interface and data security. These three aspects constitute the directory service which is the core of new features. Finally, this thesis tests the whole system, and the result shows that the based on FastDFS directory file system is in line with expectations.

Keywords: cloud storage, distributed file system(DFS), FastDFS, directory services

目 录

第一章 绪 论

1.1 研究背景

随着Web2.0概念的出现，互联网发展迎来了新的高峰。借此发展起来的众多网站及业务系统需要为用户存储和处理大量数据，且这些数据与日俱增。由此带来的严重问题就是，如何在用户数量迅速增长的情况下快速扩展原有系统。同时随着移动互联网的快速发展，各种智能移动终端和宽带移动网络的普及，越来越多的移动设备接入到互联网中。这不仅带来了机遇和商机，同时更带来了挑战，负责移动设备相关业务的计算机系统会承受更多的负载。为应对这样的挑战，最简单的做法就是增加相关软硬件和人员的投入，但这样同时带来的就是成本的快速上升。由于资源的有限性，电力成本、空间成本、维护成本的提升直接导致计算机系统成本的提升，这就使得如何更有效的利用现有的有限资源满足更多用户的需求成为迫切需要解决的问题[1]。

在问题接踵而至的同时，各种网络计算机设备在功能和性能上都有很大的提升，同时它们的价格也更加低廉，计算机系统在相同的成本投入下可以获得更高的处理能力，因此可以服务更多的用户。除了设备的发展，在技术上，分布式计算更加成熟。通过互联网连接在一起的分散在地理上不同位置的软件、硬件和信息资源被整合到一起，使得人们可以完成规模更大，复杂度更高的计算任务。各种存储技术、虚拟化技术的快速发展和应用，使得计算机系统能够提供更加强大的计算机能力和服务能力。随着对提高资源利用率、资源集中化管理与资源按需使用的迫切需求，云计算应运而生[2]。

云计算作为近几年出现的新兴概念，现在已逐步发展成一种产业。云计算模式为资源的弹性分配、用户的按需使用提供了新的可能，并且在价格和风险控制上都具有相当优势，其已经成为IT业的新标志。云计算的一个基本特点就是将数据集中存储在云端，用户只需关心使用而无需关心数据在实际计算和存储时涉及的具体细节，这降低了用户用于购买和维护软硬件的成本[3]。云计算使得用户对各种资源的使用更加方便和自然。

云存储是在云计算的概念上延伸和发展出来的一个新的概念，是一种新兴的网络存储技术。云存储利用集群应用和分布式文件系统等软件，将网络中大量类型不同、容量不同的存储设备连结在一起，作为一个整体协同工作并以一个统一的方式对外提供数据存储和访问服务[4]。当云计算系统中配置了大量存储设备，并且将系统运算和处理的重点放到数据的存储和管理上而不是数据的计算上，那么该云计算系统就变为一个云存储系统[5-9]。也可以说云存储是一种新的存储方案，

它将存储资源统一集中放至云端，这样无论用户身处何地都可以随时通过网络访问云上的数据资源。

1.2 研究现状

云存储如今正扮演着越来越重要的角色。传统的存储方式不仅成本高且效率低下，越来越不能满足用户日益增长的对存储的需求。如今迈入的大数据时代更是使数据以指数级增长。在这个背景下，包括私有云、公共云以及混合云在内的云存储服务正在快速发展，并得到了越来越多的企业和单位的重视。这三种类型的云存储服务为企业和单位进行大型规划提供了可能和措施，并能够更加灵活的满足部分客户的特殊需求[10]。

私有云是为某个单独的特定客户特别设计定制的云服务，通常部署在企业或单位的内部网络上。私有云主要为企业或单位内部人员提供服务，其搭建工作可由其他公司负责也可由企业或单位内部 IT 部门负责，维护工作则一般由内部 IT 部门负责。

公共云是为多个非特定用户搭建的公共云服务，通常部署在企业或单位的外部网络上，即部署在互联网上。用户通过与服务提供商签订协议来使用云存储，服务提供商负责公共云的搭建和维护并对外提供相应的资源和服务，用户只需要使用就可以。相对于私有云，公共云使用简单方便，并且省去了搭建云服务的人力和物力成本。公共云在数据安全、隐私保护等方面仍有较高风险。目前市场上已有相当数量的公共云存储产品，如国外有 Dropbox、sky drive、google drive 等，国内有百度云、微云等。

混合云即为私有云和公共云的组合。处于对数据的安全和控制的考虑，企业或单位并不会将所有数据都存储在公共云上，而是将一部分涉密数据存储在私有云上。混合云即同时使用私有云和公共云。

云存储是计算机存储领域里的一个新的概念，其致力于解决海量数据的存储问题[11]。云存储目前已成为学术界和工业界的一个研究热点，它的兴起正在颠覆传统的存储系统架构，其正以良好的可扩展性、性价比和容错性等优势得到业界的广泛认同。

云存储良好的可扩展性，容错性以及对用户的透明性，这些特性都与分布式文件系统息息相关。现有的云存储分布式文件系统中国外有 GFS、HDFS、Lustre、PVFS、GPFS、Ceph 等，国内有 FastDFS、TFS 等。它们的许多设计理念类似，但同时也各具特色。

近年来在云存储的发展和应用过程中，又不断有新的问题和观点出现并被广

泛讨论和研究，如云存储数据中心部署、云存储 QoS 控制、用户请求区分、数据副本策略以及调度机制等很多方面。

1.3 研究内容

云存储中非常重要的一个组成部分就是文件系统。这不仅包括系统级文件系统，也包含应用级文件系统。系统级文件系统提供底层的存储保障，应用级文件系统则提供更多的功能。

系统级文件系统并没有多少可以选择的余地，基于 Unix/Linux 操作系统的服务器通常使用 ext3 或 ext4 文件系统，而基于 Windows 操作系统的服务器通常则使用 NTFS 文件系统。相比系统级文件系统，应用级文件系统则有很多，如 GFS、HDFS、TFS、FastDFS 等。他们各自适用于不同的领域，能满足该领域中系统级文件系统不能满足的需求。

FastDFS 是一款开源的轻量级高性能分布式文件系统。它的主要功能包括文件存储、文件同步和文件访问（如文件上传和文件下载）等，同时满足大容量和负载均衡的要求。这些特性都是云存储所需要的。但其不具有目录服务的特点也增加了将其应用于云存储后管理和使用的难度。本文将在实验室云平台的基础上，讨论云存储的新的解决方案或新的实现方式。新解决方案将在 FastDFS 的基础上增加以目录服务为核心的新功能，设计实现基于 FastDFS 的目录文件系统。

1.4 论文结构

本文共分为 6 章，各章节的内容安排如下。

第一章为论文的绪论部分，阐述了论文的研究背景、研究现状、研究内容和本文的组织结构。

第二章介绍了 FastDFS 的相关基础，包括分布式文件系统的基础。同时本章还讨论了如何扩展 FastDFS 的应用环境以满足云存储的需求。

第三章阐述了基于 FastDFS 的目录文件系统的总体设计，其中包括目录文件系统的需求分析、概要设计、总体架构、模块设计、并发访问与可靠访问设计和数据安全与隐私保护设计。

第四章阐述了基于 FastDFS 的目录文件系统的详细设计和实现过程，其中主要包括目录结构、公共接口及数据安全的设计与实现。同时叙述了设计与实现中遇到的问题及解决方案。

第五章结合实验室云平台对整个目录文件系统做了测试。

第六章是全文总结与展望。

第二章 FastDFS 相关基础

2.1 分布式文件系统

2.1.1 分布式文件系统概述

文件系统是操作系统的重要组成部分，它通过抽象由操作系统管理的存储资源向用户提供统一的访问接口，并对用户屏蔽使用存储设备的操作细节[12]。

文件系统根据使用环境和所提供的功能不同，从低到高依次可分为四个层次。第一个层次是单处理器单用户的本地文件系统，如 DOS 的文件系统；第二个层次是多处理器单用户的本地系统，如 OS/2 的文件系统；第三个层次是多处理器多用户的本地文件系统，如 Linux 的文件系统；第四个层次是多处理器多用户的分布式文件系统[13]。

分布式文件系统（Distributed File System，DFS）是指文件系统管理的物理存储资源并不局限于本地节点上，同时还管理通过计算机网络连接的其他节点[14]。分布式文件系统基于 C/S 模式的设计，通常一个网络内可能包括多个可供用户访问存储资源的服务器[15]。同时，分布式文件系统的对等特性也允许一些系统在扮演客户端的同时扮演服务端。

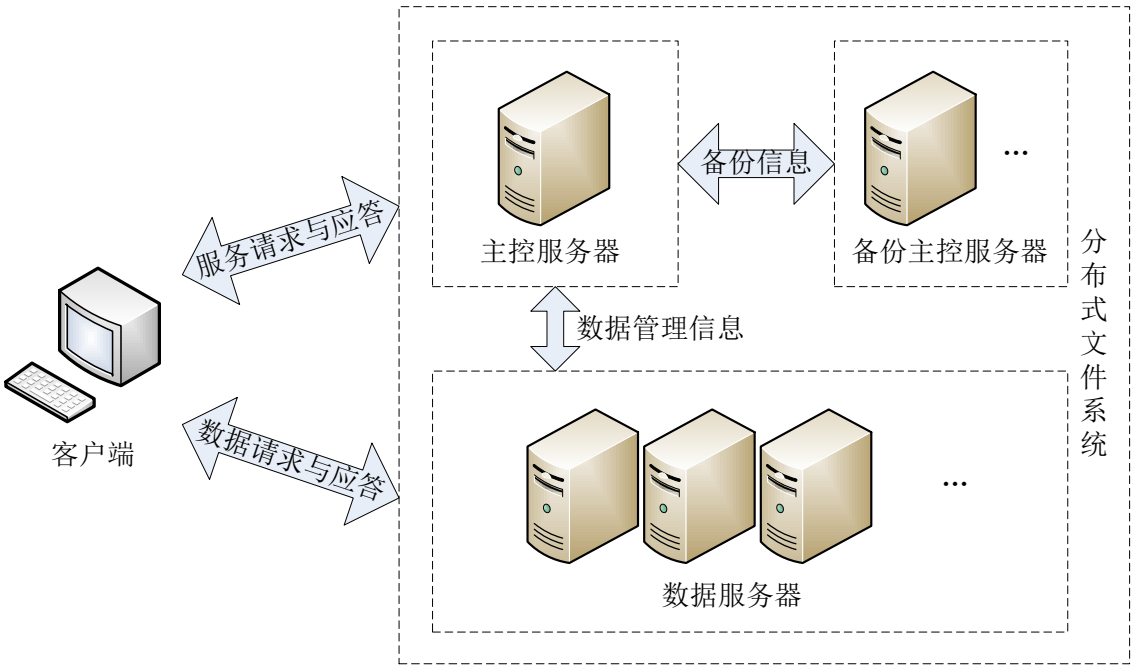

图 2-1 分布式文件系统一般架构图

分布式文件系统的一般架构如图 2-1 所示。分布式文件系统一般包含主控服务器、备份主控服务器和数据服务器三个部分[16]。主控服务器由一台服务器组成，主要负责整个分布式文件系统的管理、调度和控制等。备份主控服务器是主控服务器的备份，可由一台或多台服务器组成，当主控服务器出现问题时由备份主控服务器接替。现在通常将主控服务器与备份主控服务器合并，由多台服务器组成。这样主控服务器之间既能平衡负载也能相互备份。数据服务器主要负责数据的管理和控制。客户端则为用户一端，它通过网络访问分布式文件系统。

早期的分布式文件系统主要以文件的远程访问为主要功能，更多的关注系统的性能和数据访问的可靠性[17]。这时期具有代表性的分布式文件系统有 NFS（Network File System）和 AFS（Andrew File System），它们对以后的文件系统设计有十分重要的影响。随着计算机技术与网络技术的飞速发展，基于光纤网络的 SAN（Storage Area Network and SAN Protocols）和 NAS（Network Attached Storage）得到了发展与广泛应用。当 SAN 和 NAS 两种架构日趋成熟，人们考虑将它们结合起来以充分发挥两者的优势，从而推动了分布式文件系统的发展。

2.1.2 部分分布式文件系统简介

2.1.2.1 GFS

GFS（Google File System）是 Google 公司为了存储海量搜索数据而设计的专用文件系统。它是一个可扩展的分布式文件系统，主要用于大型的、分布式的、需要对大量数据进行访问的应用[18-21]。它对硬件要求不高，只需运行在廉价的普通硬件上即可为大量用户提供总体性能较高的服务。GFS 是 Google 公司针对自身特点设计实现的，是处理整个互联网范围内问题的重要工具。

2.1.2.2 HDFS

HDFS（Hadoop Distributed File System）是 Apache Hadoop Core 项目的一部分，是一个基于 JAVA 的支持数据密集型应用的分布式文件系统[22-25]。HDFS 是一个具有高度容错性的分布式文件系统，能够像 GFS 一样部署在廉价的机器上。HDFS 能提供高吞吐量的数据访问，非常适合具有大规模数据集的应用。

2.1.2.3 TFS

TFS（Taobao File System）是由阿里巴巴公司设计开发的，主要针对海量的非结构化数据的一个高可扩展、高可用、高性能、面向互联网服务的分布式文件系统[26-27]。它以普通 Linux 机器集群为基础，采用了高可用架构和平滑扩容技术，

从而保证了整个文件系统的高可用性和高可扩展性。

2.2 分布式文件系统与本地文件系统的区别与联系

本地文件系统（Local File System）即只管理本地物理存储资源的文件系统，处理器通过系统总线直接访问存储资源。分布式文件系统与本地文件系统都是管理物理存储资源的系统。但它们的区别不仅仅在于管理的存储资源的分布上。在 2.1.1 小节中提到的文件系统分类层次中，高层次的文件系统都是以低层次的文件系统为基础，实现了更多更高级的功能。

随着层次的提高，文件系统在设计和实现方面的难度也会随即提高。现有的分布式文件系统一般还保持着与本地文件系统几乎相同的访问接口和对象模型，这主要是为了向用户提供向后的兼容性[28-29]。但几乎相同的访问接口和对象模型并不能说明分布式文件系统的设计和实现难度没有增加。恰恰相反，正是由于对用户透明地改变了系统结构，掩盖了分布式文件操作的复杂性，才大大增加了分布式文件系统的设计和实现难度[30-31]。

2.3 FastDFS 简介

FastDFS 是一款开源的高性能分布式文件系统。它的主要功能包括文件存储、文件同步和文件访问（如文件上传和文件下载）等，同时满足大容量和负载均衡的要求[32]。

2.3.1 FastDFS 概述及架构

FastDFS 包含三种角色，分别是 client、tracker 和 storage，如图 2-2 所示。

tracker 负责文件访问的调度和负载均衡。storage 负责文件管理，主要包括：文件存储、文件同步和提供文件访问接口。它同样以键值对的形式管理文件的属性。

tracker 和 storage 都包含一台或多台服务器。服务器可以随时加入 tracker 或 storage 所在的集群而不影响集群中其他原有服务器的正常运行。

storage 集群通过分组（volume）来满足大容量的要求。storage 包含一个或多个分组，每个分组中存储的文件是相互独立的，所有分组一起对外提供完整的存储服务。每个分组包含一台或多台存储服务器，同一分组中的存储服务器上的文件是相同的，它们相互备份并且负载均衡。当新的存储服务器加入分组后，分组中已经存在的文件便会自动同步到这台服务器，同步成功后系统会将该服务器的状态改为在线，随后这台服务器便可以提供存储服务。storage 的容量等于所有分

组容量的总和。由于每个分组中的服务器存储的文件相同，所以单个分组的容量等于该分组中容量最小的那台服务器的容量。在 storage 的容量不足时，可以通过扩展一个或多个分组来扩充容量。

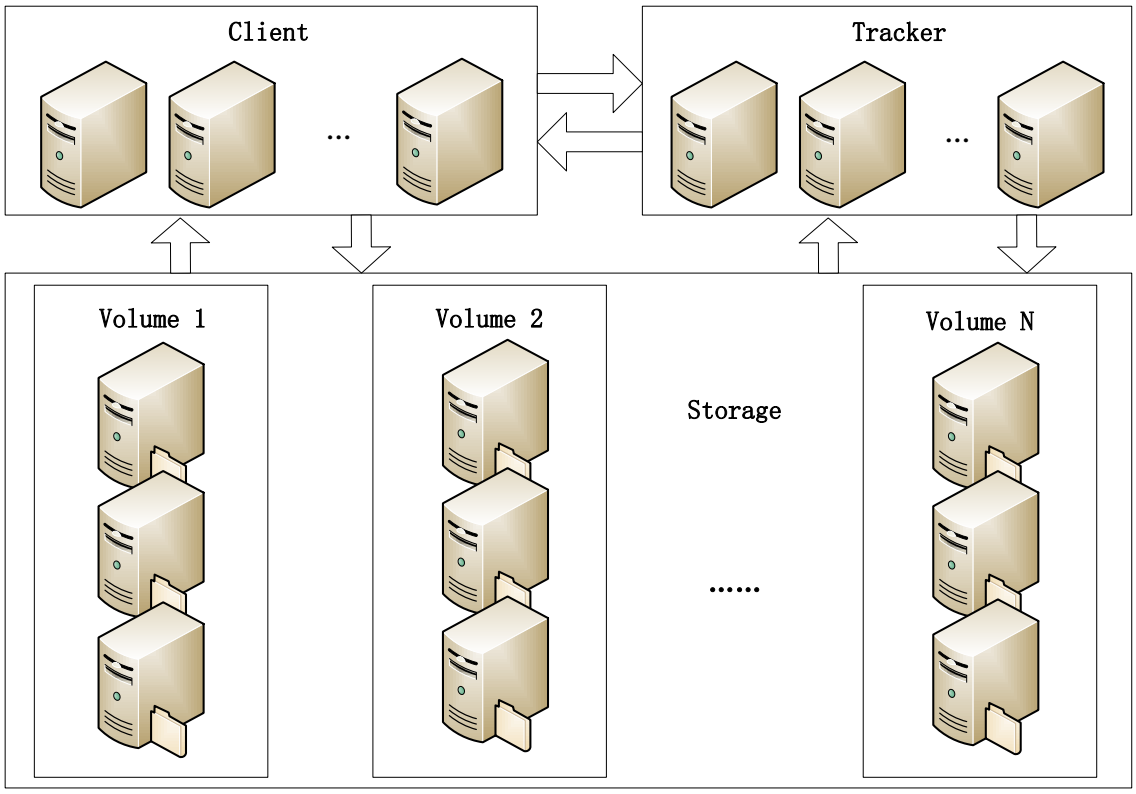

图 2-2 FastDFS 架构图

FastDFS 中文件的标识符分为两个部分：卷（分组）名和文件名。

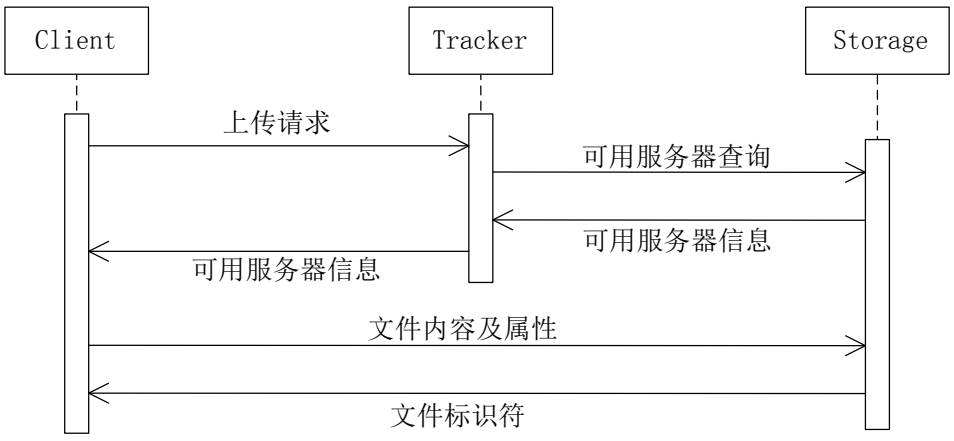

图 2-3 文件上传过程

文件上传时，client 首先询问 tracker，tracker 查询 storage 可用服务器后将相应信息返回给 client，之后 client 直接与 storage 可用服务器通信进行文件上传，完成后 storage 将文件标识符返回给 client 以便之后执行下载操作时使用。整个上传过程如图 2-3 所示。

文件下载时，client 首先发送文件标识询问 tracker，tracker 查询文件所在分组可用服务器并将相应信息返回给 client，之后 client 直接与 storage 可用服务器通信，发送文件标识符给 storage 可用服务器完成文件下载。整个下载过程如图 2-4 所示。

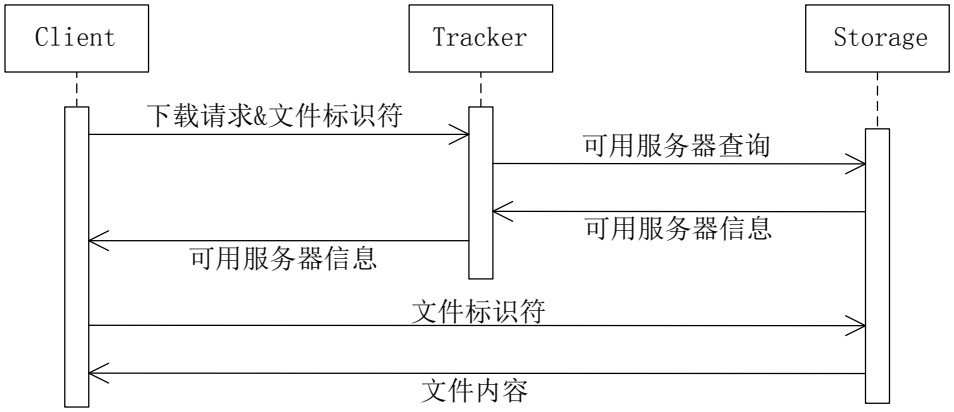

图 2-4 文件下载过程

tracker 在查询 storage 可用服务器时完成了文件访问的调度和负载均衡。

2.3.2 FastDFS 的优势

FastDFS 是一款轻量级的分布式文件系统，适合小文件的存储访问。同时具备良好的冗余备份、负载均衡等性能。

冗余是确保数据安全的有效措施之一。在 storage 中每一个分组中的服务器互为备份，互为冗余。只要分组中的服务器不同时宕机，该分组就能不间断的对外提供服务，并利用同步机制恢复故障服务器中的数据。

分组中的多台服务器在起到冗余备份作用的同时起到了负载均衡的作用。分组中的多台服务器存储的是相同的文件，提供的是相同的服务。因此众多访问该分组的请求可均衡地分摊到分组中的每台服务器，以此提供最佳服务。

在线扩容也是 FastDFS 的重要优势之一。能够在不中断服务的情况下升级容量对整个文件系统的维护是大有裨益的。这不仅使整个系统对外保持了优质的服务，同时减少了维护的开销。

2.3.3 FastDFS 应用环境及局限

尽管 FastDFS 在某些方面表现出色，如冗余备份、负载均衡、在线扩容等，但其并非一款全能的分布式文件系统，在某些应用场景中仍然表现出其局限性。

FastDFS 以文件为管理单元，这样做虽然简单但管理粒度过于单一同时也不够人性化。首先是用户必须自己记录所有上传文件的标识符以便后续下载使用，这就在无形中增加了用户的负担；其次是其不具有对文件的分类管理功能，如果用户期望对上传的文件分类管理就必须自己对上传文件的标识符进行分类管理以达到同样的效果，这同样增加了用户的负担和使用上的不便；最后是浏览文件带来的不便，比如用户希望查看自己都上传了哪些文件，而保存下来的文件标识符不仅含有文件名还含有卷名，在一长串标识符中快速准确的识别文件名是比较困难和麻烦的，这同样增加了用户的负担。

FastDFS 架构中的 client 主要指其他使用 FastDFS 存储服务的服务器，其中并不包含一般的终端用户。云计算时代终端用户对云存储的需求越来越大，FastDFS 作为一款优秀的分布式文件系统，使其为终端用户提供云存储服务是不错的选择。但 FastDFS 本身并不以用户为导向，因此不能满足终端用户的多种个性化需求，这也使得其在这方面的应用受到了局限。

2.4 扩展 FastDFS 应用环境的探究

为了使 FastDFS 能够满足某些实际应用的需要，就必须对其做出一定的修改或添加适当的功能，以此扩展其应用环境。下面将探究一些可行的方法。

2.4.1 目录结构

首先考虑为 FastDFS 添加目录结构。为 FastDFS 添加的目录结构必须是和用户相结合的。简单的将 storage 中服务器的目录结构呈现给终端用户是没有任何意义的。利用这样的目录用户根本无法完成浏览和管理文件的操作，因为用户根本无从知晓自己刚刚上传的文件被存储到了什么位置，存储的名称是否还是文件本身的名称等等，这些都是由 FastDFS 决定的。只有将目录结构与用户相结合，只有为每一个用户建立独立的目录，这样才能真正满足用户的需求，实现目录应有的功能。

在实际使用过程中，用户根本不关心也不需要关心自己上传的文件到底被存储到了什么位置，而只关心文件是不是被以他认为的方式存储了，这就是目录的作用。用户能够按自己的意愿建立目录或文件夹，并将文件上传至自己希望的位置就够了，至于文件到底被存储到了什么位置由系统来处理。系统必须负责处理

用户目录与实际存储位置的映射。

如此一来就可以很好地解决上一节中提到的有关用户浏览管理文件的问题，同时扩展了 FastDFS 的应用环境。

2.4.2 终端访问

其次考虑为 FastDFS 添加终端访问。在为 FastDFS 添加了基于用户的目录结构之后，还必须提供相应的访问方式以使用户可以浏览他的目录和使用 FastDFS 的存储服务。

终端访问方式主要可分为 web 访问和客户端访问。web 访问使用浏览器访问 FastDFS 存储服务。web 访问方式灵活方便，跨平台性好，任何可以运行浏览器的平台都可以无障碍地访问。客户端访问通常基于一个专门开发的应用程序，通过这个专门的程序访问 FastDFS 的存储服务。客户端程序除了具备 web 方式具有的功能外，通常还具有更多的辅助功能及更多的个性化设置，用户体验更好。但客户端访问方式需要为不同的平台开发不同的客户端应用程序。尽管 web 访问方式和客户端访问方式各有优劣，但本质上只是表现形式的不同。无论是使用哪种访问方式访问 FastDFS 的存储服务，服务端使用的都是同一套接口，并不会因为访问方式的不同而设计不同的接口。

终端访问能够使用户对存储服务的访问更加简单方便，并使 client 的范围不再只局限于服务器，而是能够让一般终端用户同样使用 FastDFS 出色的存储服务。

2.5 本章小结

本章介绍了 FastDFS 相关基础并对 FastDFS 的架构和上传下载机制等做了比较详细的介绍。FastDFS 是一款开源的高效的轻量级分布式文件系统。它适合小文件的存储访问，同时具备良好的冗余备份、负载均衡、在线扩容等优势。但它在某些应用环境中仍表现出局限性。为其添加目录结构和终端访问可以扩展其应用环境。

第三章 目录文件系统总体设计

本章将在实验室现有云平台的基础上，探索一种新的解决方案以满足平台对云存储的要求。新解决方案将在 FastDFS 的基础上增加新的功能，设计基于 FastDFS 的目录文件系统（以下简称目录文件系统）。

3.1 目录文件系统需求分析

在对目录文件系统进行设计之前首先要对其进行需求分析，作为云存储的解决方案必须能够满足几项基本的需求。

3.1.1 功能需求

3.1.1.1 网络接入

如图 3-1 所示，云存储作为网络服务，首先应该提供的就是网络接入功能。目录文件系统必须能使用户无论身处何地都能随时通过网络使用目录文件系统提供的资源与服务。

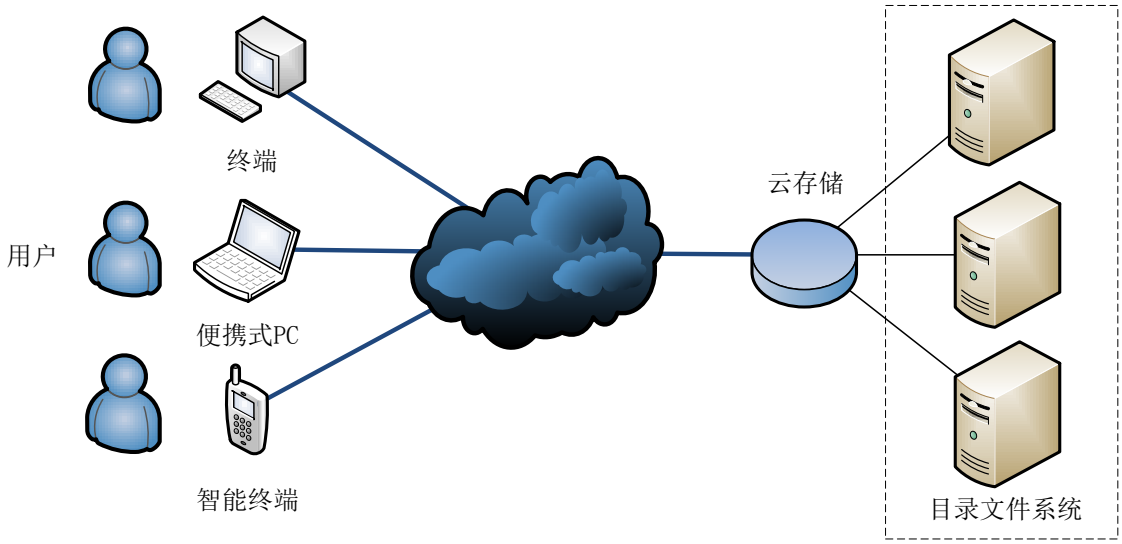

图 3-1 云存储网络接入示意图

3.1.1.2 用户注册与认证

每个用户在通过网络接入目录文件系统之后，其拥有的资源和可以使用的服务是不同的，这就要求目录文件系统能够识别不同的用户，即对用户进行认证。

用户认证可以使用户的各种服务信息与其注册的账户相关联，如用户的存储容量，存储内容，个人性化设置等。同时用户认证也将方便整个目录文件系统的管理和控制。

3.1.1.3 权限控制

每个用户对资源和服务的需求量不同，同时愿意为使用资源和服务而付出的代价也不同，因此这就需要目录文件系统能够对用户的权限进行控制，控制其对资源与服务的使用权限。

3.1.1.4 文件上传与下载

云存储作为存储服务，上传和下载文件是非常基本的功能，也是用户使用云存储的主要方式，是基础中的基础，因此目录文件系统必须具备该功能。

3.1.1.5 目录

提供目录功能将使用户方便且高效的管理其存储的内容，这关系到用户的使用体验与云存储的服务质量。作为为终端用户提供云存储服务的解决方案，目录是必不可少的。目录结构是文件系统的基础结构，它为用户访问和操作文件提供了简单高效的接口。目录结构将文件系统以一种友好且合乎逻辑的形式展现在用户面前。利用目录用户可以方便地浏览和管理文件系统中的众多文件。无论是系统级文件系统还是应用级文件系统，目录结构都是必不可少的[33]。图 3-2 为用户通过目录结构使用云存储的示意图。

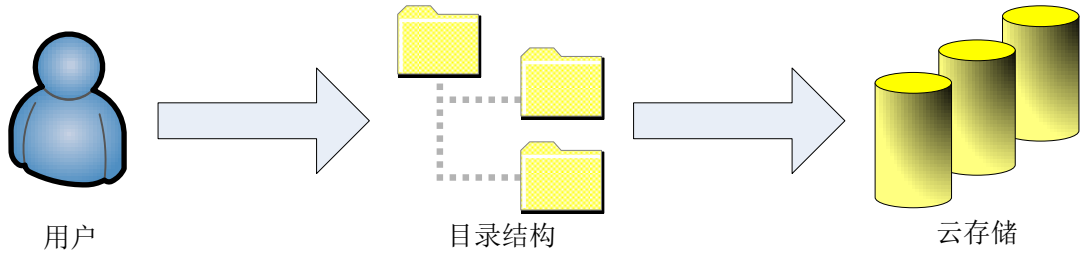

图 3-2 目录结构示意图

目录结构是用户使用云存储最直接和最直观的接口，目录结构的好坏直接影响到云存储体验的好坏。因此基于 FastDFS 的目录文件系统必须具备该功能。

3.1.1.6 公共接口

目录文件系统还应该为相应的终端应用程序开发提供公共接口。目录文件系统提供的云存储服务不仅要能通过 web 端访问，还要能通过专门的客户端程序访问。web 端能提供更好的跨平台性，而客户端则能提供更丰富和精致的功能以及多样的个性化设置。因此提供丰富的接口将方便相应终端程序的开发。

3.1.1.7 安全管理控制

现在人们越来越注重隐私保护，个人目录对人们来说不仅是隐私的一部分，同时也涉及到用户的个人数据，因此需要得到相应的保护。

3.1.2 性能需求

3.1.2.1 网络延时

为提升用户体验，必须在尽可能短的时间内完成用户个人目录的查询、显示及其他功能。云存储作为网络服务，其性能必将受限与网络环境的好坏，但良好的设计可以减少服务对网络的需求。

3.1.2.2 并发访问

云存储必须能够同时为多个用户提供优质服务才能够称得上是云服务。因此，为确保多个用户同时请求目录服务时保证每个用户的服务质量，后台目录文件系统在并发访问上也应该表现良好。

3.1.2.3 可靠访问

目录文件系统持续不间断的提供服务也是云存储的基本需求。冗余备份是确保系统可靠性的基本措施，因此目录文件系统在这方面也应该满足相应需求。

3.2 目录文件系统总体设

3.2.1 概要设计

目录文件系统以 FastDFS 为基础，结合数据库集群实现。考虑到数据的并发访问和安全，数据库集群使用 MySQL-Cluster。目录结构通过设计专门的目录文件实现。数据库主要用于用户注册和用户认证时用户信息的录入和验证，同时通过数据库实现用户与其个人目录间的一一映射，即通过用户在数据库中的信息可以确认其个人目录文件的存储信息。目录结构用于记录用户上传的文件信息及各

级子目录的信息。

终端获取目录时首先需要进行用户认证，向服务端发送用户名和密码。服务端通过核对用户在数据库中的信息完成验证。用户验证成功后再根据用户在数据库中记录的附加信息确认其目录文件的存储位置，之后获取目录文件并返回文件内容给终端。终端获得目录文件内容后显示目录。

当用户进行上传文件、删除文件或修改目录等操作时，终端会请求更新服务端目录文件，并更新本地目录文件。

3.2.2 总体架构

目录文件系统属于应用级文件系统，位于操作系统之上，提供应用级的存储服务。云存储整体架构如图 3-3 所示。

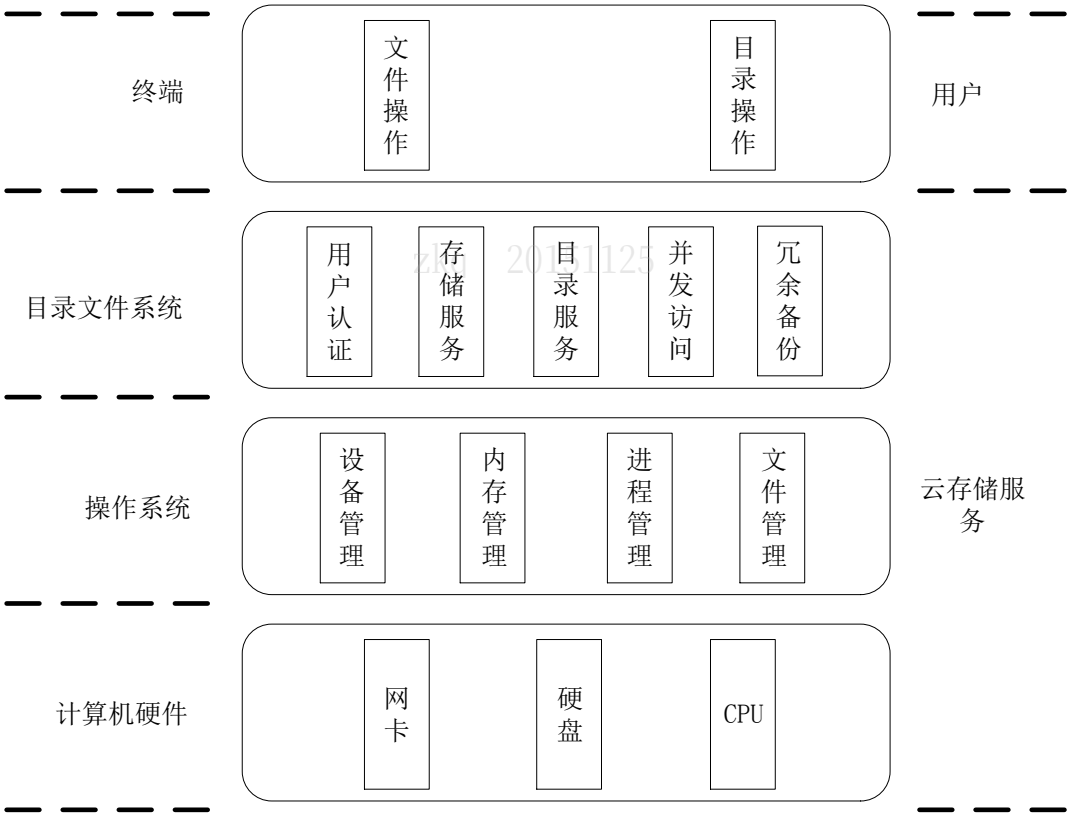

图 3-3 云存储整体架构图

终端通过网络使用目录文件系统，目录文件系统通过操作系统使用计算机硬件资源。用户通过终端使用云存储服务，而目录文件系统、操作系统和计算机硬件一起为用户提供云存储服务。

终端为用户提供文件操作和目录操作。主要包括文件的上传、下载和删除，目录的新建、修改和删除。目录文件系统提供用户认证、存储服务和目录服务。存储服务对应终端的文件操作，目录服务对应终端的目录操作。除此之外还提供并发访问和冗余备份，这将用于保证云存储的服务质量。目录文件系统通过操作系统访问各种硬件资源，进而对外提供各种服务。

目录文件系统可以进一步细分，如图 3-4 所示。目录文件系统可分为数据库、目录结构、tracker 和 storage。其中 tracker 和 storage 为 FastDFS 的组成部分。数据库用于完成用户认证部分的功能，并起到映射用户个人目录结构的作用。目录结构记录用户的全部目录信息，包括所有上传的文件信息及各级子目录信息。tracker 本身已具备处理文件存储服务的并发访问，现在其上添加对目录服务并发访问的处理能力。storage 保持原有功能不变，提供存储服务和冗余备份。

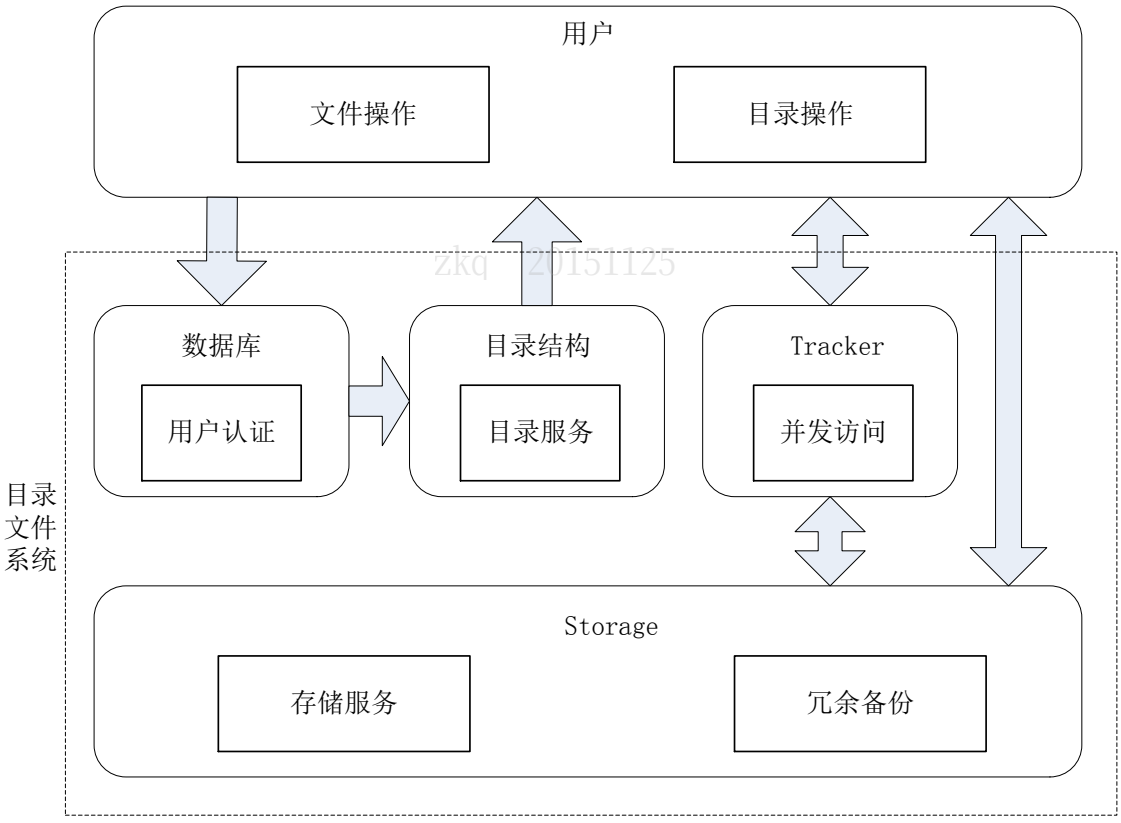

图 3-4 目录文件系统架构图

3.2.3 模块设计

根据目录文件系统的需求分析，将文件系统分为接入模块、用户注册模块、用户认证模块、目录管理模块、文件上传下载模块。

接入模块负责将各类终端接入目录文件系统。用户注册模块负责为用户提供在目录文件系统中注册账户的功能。用户认证模块负责用户登录时的账户认证。目录管理模块负责用户的目录管理，如创建目录、打开目录、删除目录等操作。文件上传下载模块负责处理用户上传文件和下载文件的相关业务。各模块之间的数据流图如图 3-5 所示。

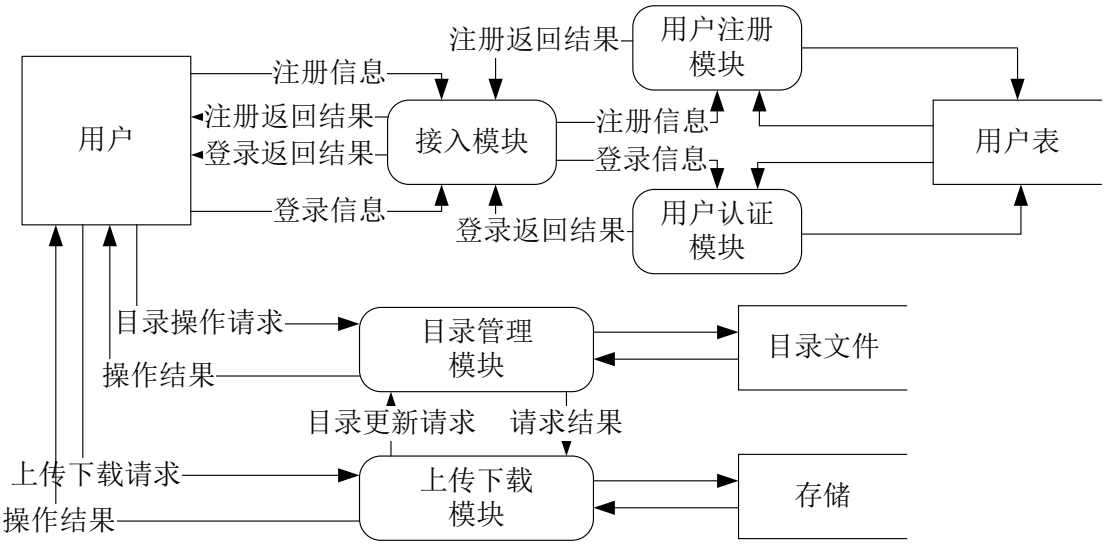

图 3-5 模块间数据流图

用户将注册信息经接入模块传递给用户注册模块，用户注册模块通过查询用户表判定本次注册是否有效。若有效则将新用户信息写入用户表。无论本次注册是否有效用户注册模块都将通过接入模块返回注册结果。用户登录过程的数据传递与用户注册过程类似。

用户在进行诸如创建删除目录等操作时将目录操作请求发送至目录管理模块，目录模块操作目录后向用户返回操作结果。

用户在进行上传下载文件操作时向文件上传下载模块发送操作请求，上传下载模块完成文件的上传下载后向用户返回执行结果。同时，上传文件成功后上传下载模块会向目录管理模块发送目录更新请求，目录管理模块更新目录后返回请求结果。

3.2.4 并发访问与可靠访问设计

FastDFS 在处理文件的上传和下载时对并发访问及负载均衡已有相应考虑，而对于新增功能如目录服务等的并发访问问题则必须做出全新考虑。

首先，为减少系统开销及方便资源共享，服务端在与终端建立连接后将使用

线程来处理用户各项服务请求。线程的创建和切换开销都小于进程，同时线程间可方便的共享某些重要资源，如某些重要变量，而无需像进程一样实现繁琐的进程间通信模块。

其次考虑是否进一步使用线程池来处理用户的各项服务请求。线程池的优势在于减少了频繁创建和销毁线程的系统开销，提高服务器处理并发访问的能力，适合调度频繁周期短的任务。用户在使用网盘进行上传或下载操作时，由于网络因素的限制，根据文件的大小这个过程可能持续较长时间。同时基于对用户习惯的考虑在文件上传或下载完成后也不会立即退出云存储服务。因此考虑到访问云存储服务不是一个短暂的过程，服务线程不会被频繁的创建与销毁，因此为简化设计与实现而不再使用线程池技术。

作为存储服务并发访问的瓶颈通常在于磁盘 IO 和网络 IO。目录服务虽然需要访问服务端本地目录文件，但由于目录文件的容量通常并不会很大，因此在使用目录服务时不会产生很大的磁盘及网络开销。

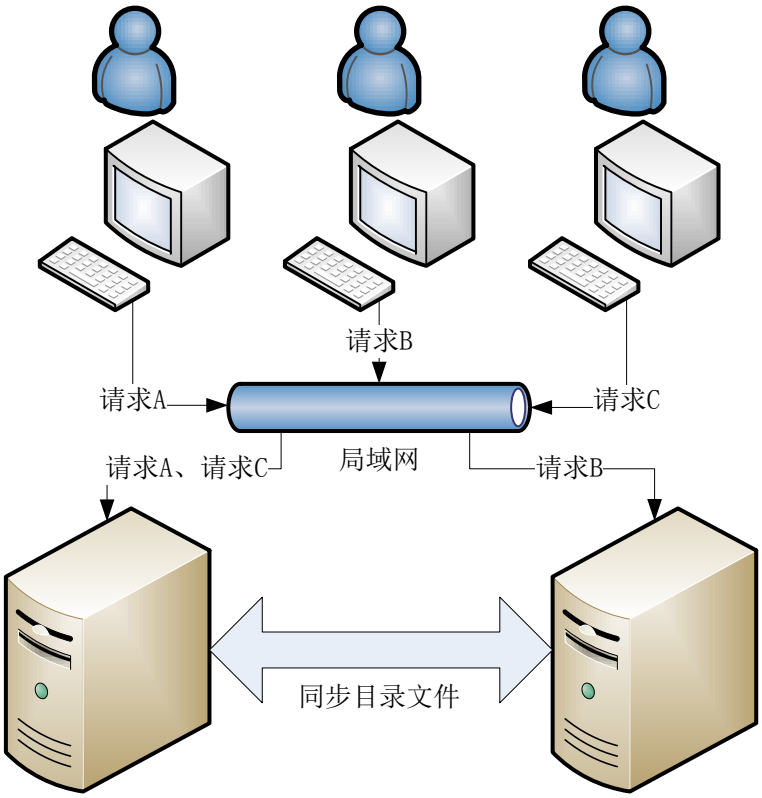

图 3-6 目录文件的负载均衡与冗余备份

FastDFS 将用户上传的文件分组存放，不仅起到了负载均衡的作用，同时满足了冗余备份的需求。目录文件也将由多台服务器同时保存，并保证服务器间目

录文件的同步更新。如图 3-6 所示，不同用户的请求将随机发送到不同的服务器，而由此引起的每台服务器的目录文件变化都会同步至其他服务器。如此既能保证目录服务的均衡访问，也实现了目录文件的冗余备份，提高了整个目录文件系统的可靠性。

3.2.5 安全管理控制设计

目录文件系统的安全管理控制目前由两方面构成。一方面是用户认证，另一方面是接口控制。

用户认证即验证用户的身份，与用户权限控制一同提供基础的数据保护，确保正确的人访问到资源。默认只有用户本人才知晓其登录账户的账户名和密码，通过用户认证后目录文件系统可以确定当前登录用户为其本人。

接口控制即通过接口控制使用者的访问行为。即便当用户试图直接通过调用相应公共接口访问目录文件系统资源时也不能绕过用户认证的过程。举例来说，仅是与服务端建立连接而未通过用户认证，即便这时调用上传或下载的公共接口也不能实现相应功能。用户在进行认证前只能通过公共接口进行注册和登录行为。当用户通过公共接口完成用户认证后即可通过调用其他公共接口访问相关资源。这样一来可以防止用户在未进行认证的情况下通过调用相应接口来进行未授权操作。同时，接口控制也可以防止认证用户非法访问他人目录。

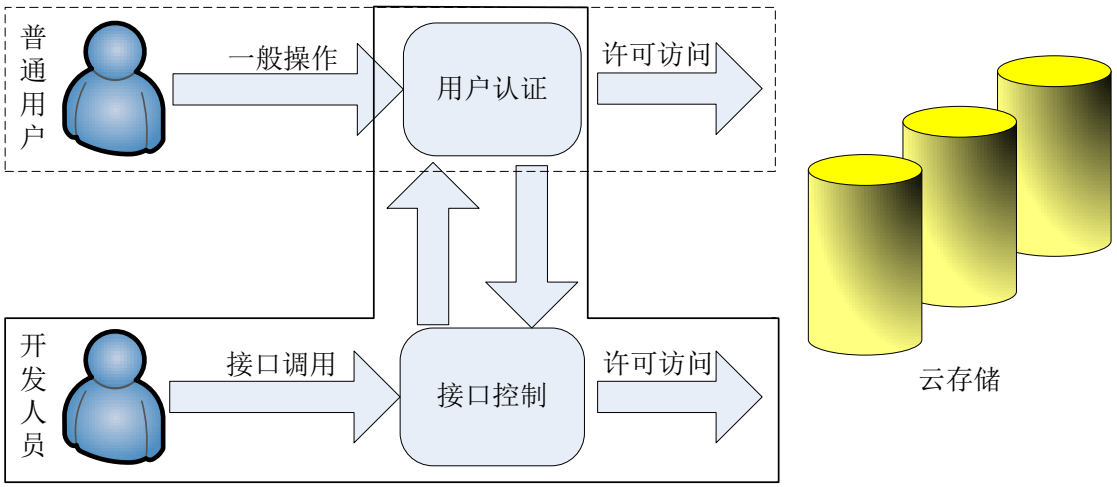

图 3-7 用户认证与接口控制

用户认证和接口控制的关系如图 3-7 所示，它们共同为目录文件系统提供更加完善的数据安全及隐私保护。

3.3 本章小结

本章主要阐述了基于 FastDFS 的目录文件系统的总体设计。从需求分析开始，首先对目录文件系统的功能需求和性能需求做了描述。之后就需求分析中描述的各个需求从概要设计、总体架构、模块设计、并发访问与可靠访问设计、数据安全与隐私保护设计等几个方面描述了总体设计。

第四章 目录文件系统详细设计与实现

4.1 系统环境设置

本章以 4 台装有 ubuntu server 14.04 LTS 的服务器为例，描述基于 FastDFS 的目录文件系统的实现。

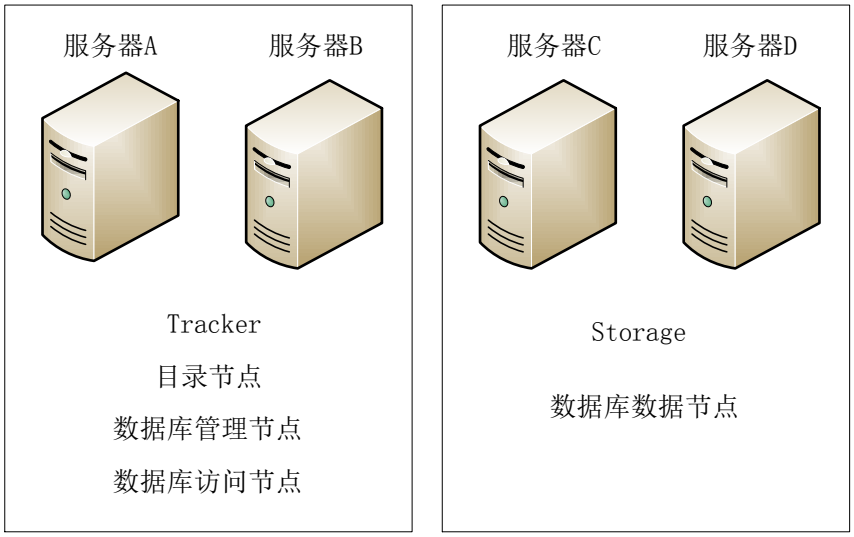

图 4-1 环境配置

如图4-1所示，四台服务器的IP地址分别设置为192.168.2.11、192.168.2.12、192.168.2.13 和 192.168.2.14，且为方便之后叙述实现过程，将四台服务器分别命名为 A、B、C 和 D。服务器 A、B、C 和 D 顺序对应之前设置的四个 IP 地址。四台服务器共同组成目录文件系统。其中服务器 A 和 B 将担任四种角色，分别是 Tracker、目录节点、数据库管理节点及数据库访问节点。服务器 C 和 D 将担任两种角色，分别是 Storage 和数据库数据节点。

4.2 目录结构详细设计

FastDFS 是应用级文件系统，这有别于系统级的文件系统，因此为 FastDFS 设计的目录结构也有别于系统级文件系统的目录结构。但二者并非毫无联系，在为 FastDFS 设计目录结构时可以参考系统级文件系统的目录结构。

4.2.1 目录文件结构设计

在 Unix/Linux 系统中目录结构也是一种文件，打开目录其实就是打开目录文

件[34-35]。目录文件的结构也非常简单。目录文件是由一条条的目录项构成的列表组成，每一个目录项包含两个部分，分别是文件名和文件对应的 inode 号。inode 记录文件具体的存储信息。

借鉴 Unix/Linux 目录文件的设计思想，可以为 FastDFS 设计类似的目录文件。目录文件同样由一条条的目录项组成，目录项包含四部分，分别是文件名、文件标志、文件标识符和文件大小。目录文件中的目录项结构如表 4-1 所示。

表 4-1 目录项结构

filename	文件名
flag	文件标志
identification	文件标识符
size	文件大小

filename 即为该目录项所代表的文件的文件名，该文件可能是一个目录文件也可能是一个普通文件。flag 是一个标志，表明该目录项所代表的文件是一个普通文件（即用户上传的文件）还是一个目录文件。根据 flag 的不同，identification 的含义也不同。当 flag 表明该目录项所代表的文件是一个普通文件时，identification 记录的是这个文件在 FastDFS 中的文件标识符；当 flag 表明该目录项所代表的文件是一个目录文件时，identification 记录的是这个目录文件在 FastDFS 服务器中的实际存储位置。size 表示文件大小，它只在目录项代表普通文件时有效，即仅当 flag 表明该目录项所代表的文件是一个普通文件时有效。当 flag 表明该目录项所代表的文件是一个目录时，size 无意义。

根据上述设计，用户浏览目录的处理过程如图 4-2 所示。用户首先需要通过用户认证。认证成功之后系统会返回用户的根目录，至此用户可以开始浏览其个人目录，如同日常浏览操作系统的资源管理器一样。当用户选择目录中某一文件点击时，会有两种不同情况出现。第一种就是用户点击的是一个普通文件，即为用户之前上传的文件。这时终端便会从当前目录文件中获取该文件在 FastDFS 中的文件标识符，之后通过该标识符并利用 FastDFS 已经提供的接口下载该文件。第二种情况是用户点击的是之前自己新建的目录。这时终端会从当前目录文件中获取该目录的实际存储位置，之后从该位置获取该目录文件并将其内容呈现在终端上。整个浏览过程直到用户结束终端程序为止。

类似于 Unix/Linux 系统的根目录，每个用户也会有一个根目录文件。这个根目录文件记录用户最初的目录结构。当用户需要在根目录下创建新目录（文件夹）时，系统会创建新的目录文件，并在根目录文件中添加相应的目录项以记录新目

录文件的位置。当用户上传新文件时，系统会在当前目录文件中添加目录项并记录该文件的相关信息，其中最重要的就是文件上传后系统返回的文件标识符。

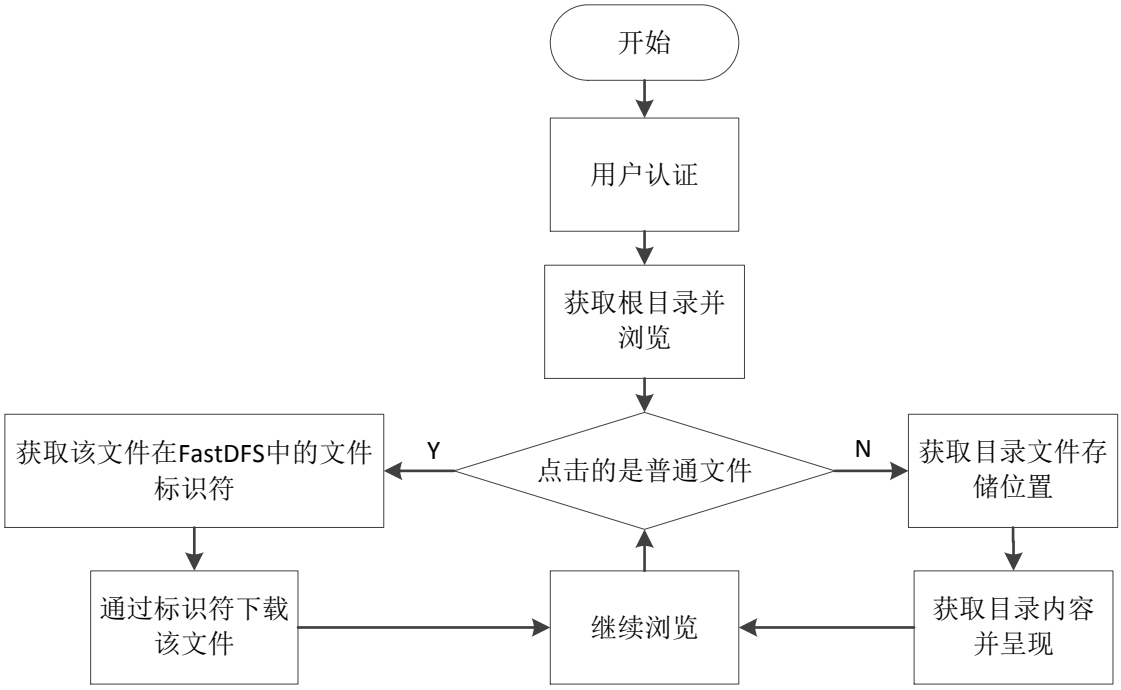

图 4-2 目录浏览过程

4.2.2 目录文件存储设计

除了需要设计目录文件本身的结构外，目录文件如何存储也需要考虑。

首先想到是否可以直接利用 FastDFS 来存储目录文件，即将目录文件也视作一般文件上传至 FastDFS 服务器存储。这样做的好处是最大限度的使用了 FastDFS 本身的各种特性而无需关心其他问题，例如获取目录时的负载均衡问题。

采用这种方案只需重新定义目录项结构中的 identification 字段。即无论目录项代表的文件是普通文件还是目录文件，identification 都表示该文件在 FastDFS 中的文件标识符。这样用户浏览目录的处理过程将被简化，如图 4-3 所示。无论是普通文件还是目录文件都将采用相同的获取方式，之后再根据是否是目录文件进行下一步操作。统一的获取方式将降低程序设计的复杂度。

但这个方案也有缺陷。首先就是效率问题。每一次目录查询都要经历一次完成的下载过程，而用户在不同目录间跳转浏览是比较频繁的操作，这势必会影响用户体验。其次是同步问题，也是最主要的问题。当目录更新时就需要重新上传目录文件，而新的目录文件不会自动替换旧的目录文件。即使先进行删除操作删除原有目录文件再上传新目录文件，但新目录文件将获得新的文件标识符，这就

必须同时更新其父目录文件。这是一个递归的过程，直到更新了根目录的文件标识符为止。每更新一次目录都必须将父目录及祖先目录都更新一遍是十分低效的。综上所述这并不是一个好的方案，需要重新设计。

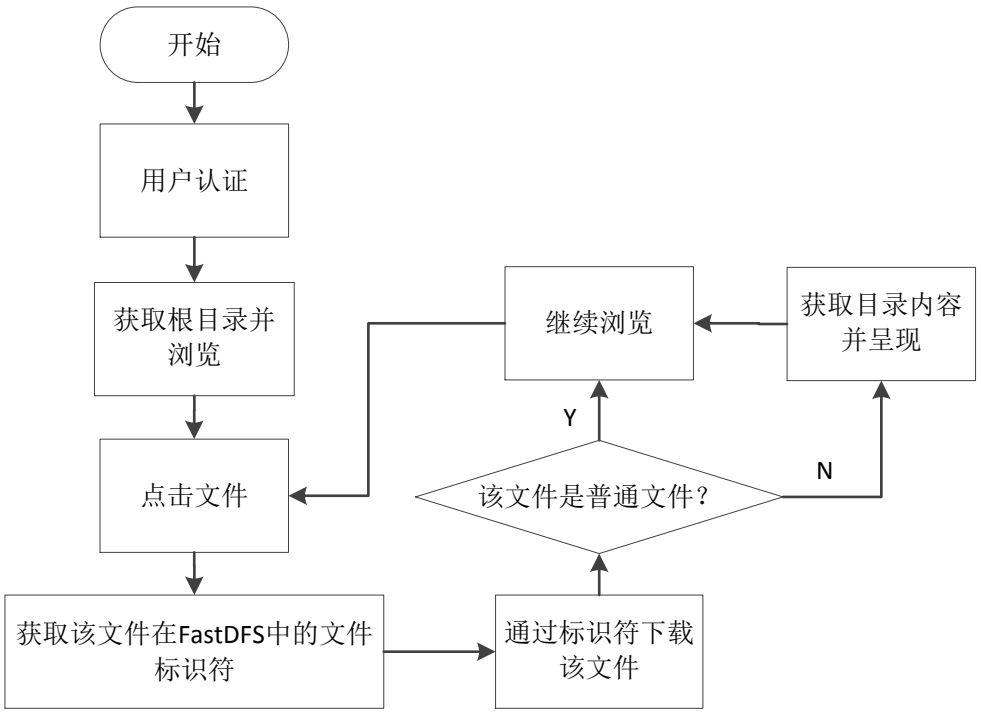

图 4-3 使用 FastDFS 存储目录文件后的目录浏览过程

目录文件在服务器中随意存储显然是不利于管理的，例如目录文件的安全设置，冗余备份等。因此，目录文件采用在相同位置集中存储。但采用集中存储仍然有一些问题需要解决。

首先是根目录在服务器上如何命名的问题。因为所有用户的目录文件都是集中存储在服务器中的同一位置，所以根目录文件必然不可以使用相同名称。解决这个问题比较简单，使用用户名或用户 ID 来命名根目录文件即可，这要求用户名与用户 ID 在整个系统中必须唯一。

其次是用户创建的子目录在服务器上如何命名的问题。最简单最符合逻辑的方法就是直接使用用户创建新目录时使用的目录名。但是不同用户之间有创建同名目录的可能，同时同一用户也有在不同目录下创建同名目录的可能，这将导致在服务器上创建相同文件名的目录文件，而目录文件又是存储在同一位置的，这是不允许的。

结合根目录的命名问题，可以这么设计。在存储位置首先按用户名或用户 ID 创建目录，用于存储每个用户各自的目录文件。在用户的独立目录下存储用户的

根目录文件。当用户创建新的目录时，系统将在其独立目录下的相同位置创建同名目录，并在该目录下创建新目录文件。其结构如图 4-4 所示。

在目录文件的存储位置上，每一个用户都有一个以其用户名或用户 ID 命名的根目录。根目录下包含一个根目录文件和若干用户自定义目录。根目录文件保存了上传至根目录的文件的信息和在根目录下创建的自定义目录的信息。自定义目录的文件名与用户使用的目录文件名相同。同样的，每一个自定义目录下同样包含一个目录文件和若干自定义目录。用户所看到的目录层级关系与实际存储时的层级关系一致。

每一个目录文件记录位于该目录下的子目录信息及文件信息。这样设计不仅简单而且与平常使用相一致，不同用户间的目录互不影响，同级目录下的目录不可同名，不同级目录下的目录可以同名。

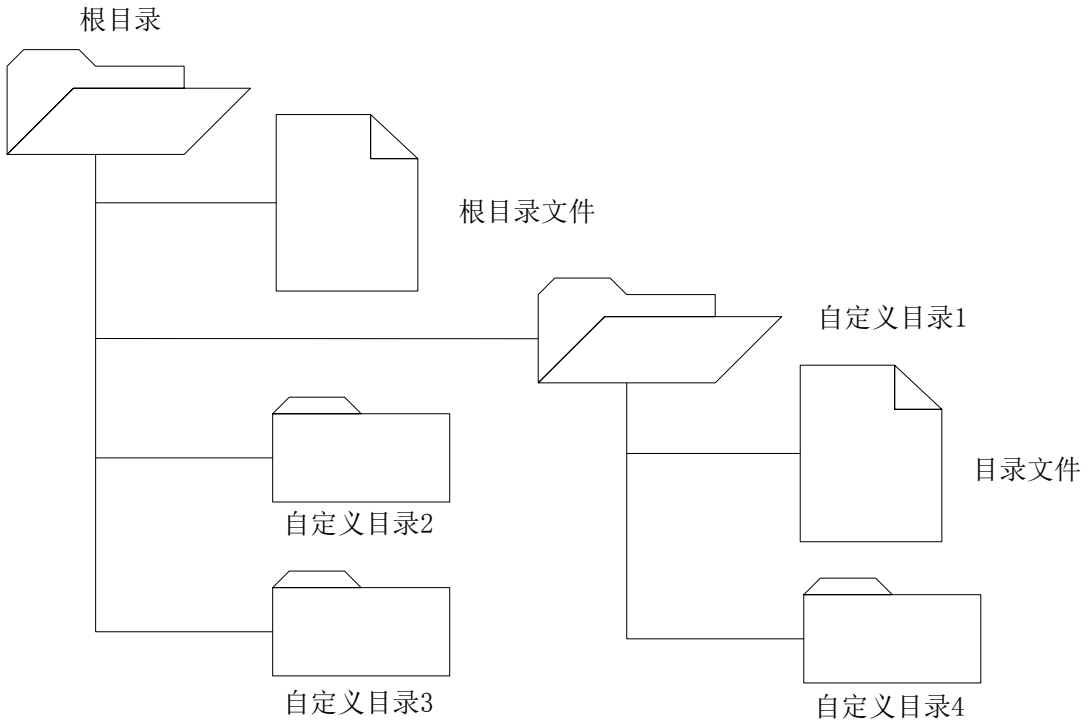

图 4-4 目录文件在服务器中的存储结构

最后是目录文件存储在哪些服务器上的问题。这个问题有三种可选的方案。

第一种是区别 tracker 和 storage 中的服务器，单独设置新的服务器来存储目录文件；第二种是将目录文件存储在 storage 所在的服务器上；第三种是将目录文件存储在 tracker 所在的服务器上。

第一种方案能提供更高的系统可靠性，使目录服务和 FastDFS 其他服务相互

独立，在其中任意一方出现问题时不至于影响另一方。但是这种方案需要额外的服务器，这意味着更高的成本。同时，这种方案提供的高可靠性意义并不大，因为目录服务是依附于 FastDFS 的其他服务的。而第二种方案会增加用户查询目录时的通信次数，用户首先与 tracker 通信要求查询目录，之后 tracker 再与 storage 通信获取目录。增加的这次通信是不必要的。tracker 本就是负责文件访问的调度和负载均衡，在其上存储目录文件是比较合乎逻辑的。终端总是先与 tracker 通信，获得可用 storage 服务器后再与 storage 通信。在终端与 tracker 通信时就能获取目录文件将减少通信开销。综上所述采用第三种方案。

4.2.3 目录文件同步设计

tracker 可能包含多台服务器，客户端可能访问其中任何一台，因此目录文件存储在 tracker 服务器上时就必须考虑多台服务器间目录文件的同步问题。利用 Linux 系统中的现有工具 inotify 和 rsync 可以实现多服务器之间目录文件的实时同步。

inotify 是 Linux2.6 内核之后加入的一个新特性，它能够监控文件系统，能够对诸如文件读取、写入、删除等操作做出反映，并向应用程序报告[36-37]。它既能够监控目录，也可以监控文件。这样当目录文件发生变化时，可以实时触发同步模块进行目录文件同步。例如当目录文件更新时，或是新建目录文件时，或是删除目录文件时应用程序都可以立即捕获这一变化，之后启动同步模块将目录文件的变化同步到其他服务器中。

使用 Linux 系统的 inotify 特性，可以在应用程序中使用相应的 API，也可以直接使用现成的系统工具 inotify-tools。在应用程序中使用 API 更加灵活可控，能够实现更多定制化的内容，直接使用现成工具则简单方便可靠。

rsync 是一款非常优秀的文件同步备份工具，能够很好完成多服务器间文件同步的工作。rsync 能够在中断之后恢复传输；它只传输源文件和目标文件之间不一致的部分，而无需传输整个文件[38-40]。使用 rsync 能够实时完成目录文件在多个服务器间同步，同时不会因为同步而消耗过多资源。

inotify 和 rsync 配合使用，由 inotify 监控目录文件的变化，并在发生变化后启动 rsync 同步目录文件。

4.3 目录结构实现

4.3.1 目录文件结构与存储实现

目录文件将存储在服务器 A 和服务器 B 之上，作为目录节点。

目录文件系统中目录文件采用与 Unix/Linux 系统中目录文件类似的设计，即目录文件由多条目录项组成。

目录项采用结构体实现，其结构体如下。

```
struct diritem{
char filename[256];
char flag;
char identification[4096];
char size[32];
};
```

系统允许的最大文件名长度为 255 个字符，因此成员 filename 设置为 256 个字节（255 个字符加 1 个结束符）。成员 flag 标志只有两种可能，因此设置为 1 个字节。成员 identification 根据 flag 的不同可能存储文件标识符或者目录文件的路径，而系统路径允许的最大长度为 4096 个字符，包含了文件标识符的长度，因此成员 identification 设置为 4096 个字节。成员 size 设置为 32 字节字符串，存储文件大小以字节为单位的十进制表示，同时将成员 size 设置成字符串也将方便网络传输。

目录文件由上述多个结构体以二进制形式顺序存储。

同样的，类似于 Unix/Linux 系统中目录文件的设计，每个目录文件中至少含有两个目录项，即包含当前目录本身的目录项及其父目录的目录项。这将方便终端在实现上传文件、创建目录等功能时与服务端确认当前工作目录是哪个目录。根目录文件中其父目录的目录项仍记录其本身。

4.3.2 目录文件同步实现

用户目录文件集中存储于本地系统/var/userdir/目录下，目录文件同步将同步服务器 A 和 B 系统中这个目录下的所有文件及子目录。下面以服务器 B 同步服务器 A 上的目录文件为例叙述目录文件同步的实现过程。

4.3.2.1 服务器 B 的配置

服务器 B 主要配置 rsync 服务。

首先需要在服务器 B 上安装 rsync 服务，该步只需执行下列命令即可。

```
# apt-get install rsync
```

安装完成后就是对 rsync 服务的配置。对 rsync 服务的配置这主要涉及到两个文件，一个是/etc/rsyncd.conf，一个是/etc/rsyncd.secrets。

/ect/rsyncd.conf 是 rsync 服务的主配置文件，该文件默认不存在需要手动创建。其中主要配置如下。

```
uid=root                          //运行 RSYNC 守护进程的用户
gid= root                         //运行 RSYNC 守护进程的组
use chroot=no                     //不使用 chroot
max connections=0                 //  最大连接数无限制
log file=/var/log/rsyncd.log      //日志记录文件的存放位置
pid file=/var/run/rsyncd.pid        //pid 文件的存放位置
lock file=/var/run/rsyncd.lock      //锁文件的存放位置
[userdir]                   //这里是认证的模块名，在 client 端需要指定
path=/var/userdir           //需要做镜像的目录,不可缺少
comment=rsync from 192.168.2.11   //注释内容
read only=no                      //  非只读
list=no                           //不允许列文件
auth users=rsyncuser              //认证的用户名，此用户与系统无关
secrets file=/etc/rsyncd.secrets  //密码和用户名对照文件
```

“[userdir]”是模块名，用于定义服务端哪个目录需要同步，这个名字也是客户端（即服务器 A）看到的名字。但实际同步的目录是由“path”指定的。

/etc/rsyncd.secrets 是记录 rsync 同步时使用的用户名和密码的文件，该文件默认也不存在需要手动创建。其内容如下所示。

```
rsyncuser:123456
```

创建该文件之后必须使用 chmod 命令修改该文件的权限为 600 才能生效。

至此 rsync 服务配置完成，启动 rsync 服务，执行如下命令。

rsync --daemon

启动成功后需要在服务器 A 上建立密码文件/etc/userdir.pass，该文件记录了同步时使用的用户的密码，内容如下。

123456 //与服务器 B 上/etc/rsyncd.secrets 记录的密码对应

文件只在第一行包含密码即可，系统将忽略其他行的内容。同样的，该文件也须使用使用 chmod 命令修改该文件的权限为 600 才能生效。

4.3.2.2 服务器 A 的配置

服务器 A 主要配置 inotify 服务。

实现 inotify 监控目标目录的操作可使用两种方法实现，一是使用 inotify 提供的系统调用，二是使用 inotify 工具 inotify-tools。

在程序中直接使用 inotify 提供的系统调用方便灵活，但有一个非常致命的缺点，即系统调用不支持目录的递归监控。

一方面，目录文件的存储目录根据用户的使用情况必然包含多级目录，每一级的目录或文件变化都必须及时更新同步至其他服务器。inotify 系统调用 inotify_add_watch 为指定目录或文件添加监控事件，但此监控事件并不能递归地应用于指定目录的所有子目录及文件。若需增加对所有子目录及文件的监控则需多次调用该系统调用为子目录或文件添加指定事件。这将耗费大量时间和资源。

另一方面，目录文件的存储目录中的用户目录数量是不能预先确定的，用户目录的层级也是无法预先确定的，同时用户目录还会随用户的使用而变化的。这是一个动态的过程，使用无法递归监控的系统调用监控这一动态过程需要设计复杂的逻辑或应用更多的技术。这无疑会提升系统的复杂度及日后维护的难度。

综合上面阐述的两个方面局限，在使用 inotify 监控目录文件的存储目录时不再使用 inotify 提供的系统调用，而是直接使用 inotify 工具 inotify-tools。inotify-tools 包含两个工具，一个是 inotifywait，一个是 inotifywatch。inotifywait 使用 inotify 特性等待文件变化，而 inotifywatch 则使用 inotify 特性统计文件变化。这里使用 inotifywait。

结合上面的讨论，首先需要在服务器 A 上安装 inotify-tools，执行下列命令即可。

apt-get install inotify-tools

安装成功之后即可编写 shell 脚本 inotify_rsync.sh 完成对目标目录的监控和同步，脚本内容如下。

```
#!/bin/bash
#function: rsync /var/userdir from 192.168.2.11 to 192.168.2.12
inotifywait -mrq -e create,modify,move,delete /var/userdir | while read DIR EVENT FILE; do
filename="${DIR}${FILE}"        //从输出中获取发生变化的文件名
userfile=${filename##/*var/userdir/}
userid=${userfile%%/*}          //从文件名中获取用户 ID
syncdir="/var/userdir/$userid"  //组合需要同步的用户目录
rsync -avH --delete --progress --password-file=/etc/userdir.pass $syncdir rsyncuser@192.168.2.12::userdir >/dev/null & //同步用户目录
done
```

关于 inotifywait 命令的参数，“-m”表示以守护进程的形式运行，即保持对目标的监控；“-r”表示递归监视目标如果目标是一个目录；“-q”表示精简输出；“-e”表示只监控参数后边列出的事件，这里列出的监控事件有创建（create）、修改（modify）、重命名（move）和删除（delete）；“/var/userdir”为被监控的目标目录。

“while read”不断通过管道获得 inotifywait 的输出，并从输出得到触发事件的目录、事件名及文件名。事件发生后便调用 rsync 进行同步。

关于 rsync 命令的参数，“--delete”表示删除目的端多余的内容，同步可能导致增加内容也可能导致减少内容，这个参数确保在减少内容时删除目的端多余文件以保持源和目的的一致。“rsyncuser@192.168.2.12::userdir”表示使用用户 rsyncuser 启用服务端 192.168.2.12 上的 userdir 模块。

脚本中通过对触发监控事件的文件的文件名进行处理，从而获得该文件所在的用户目录，并通过 rsync 同步该用户目录。脚本中并未在发生触发事件后同步整个/var/userdir 目录主要是因为这样做可能会使其他用户对目录文件的更新丢失。在如图 4-5 所示的访问过程中，用户 A 登录到服务器 A 访问资源，用户 B 登录服务器 B 访问资源。在用户 A 和用户 B 都对目录文件做出修改后，服务器 A 和服务器 B 便需要进行目录文件的同步。此时若先由服务器 A 同步至服务器 B，用户 A 的目录文件将会正确更新至服务器 B，而由于同步涉及/var/userdir 目录下的所有

文件，则服务器 B 上用户 B 的目录文件将被服务器 A 上用户 B 的旧的目录文件所覆盖替换，造成用户 B 对目录文件的更新丢失。这之后服务器 B 的目录文件同步至服务器 A 已无意义。

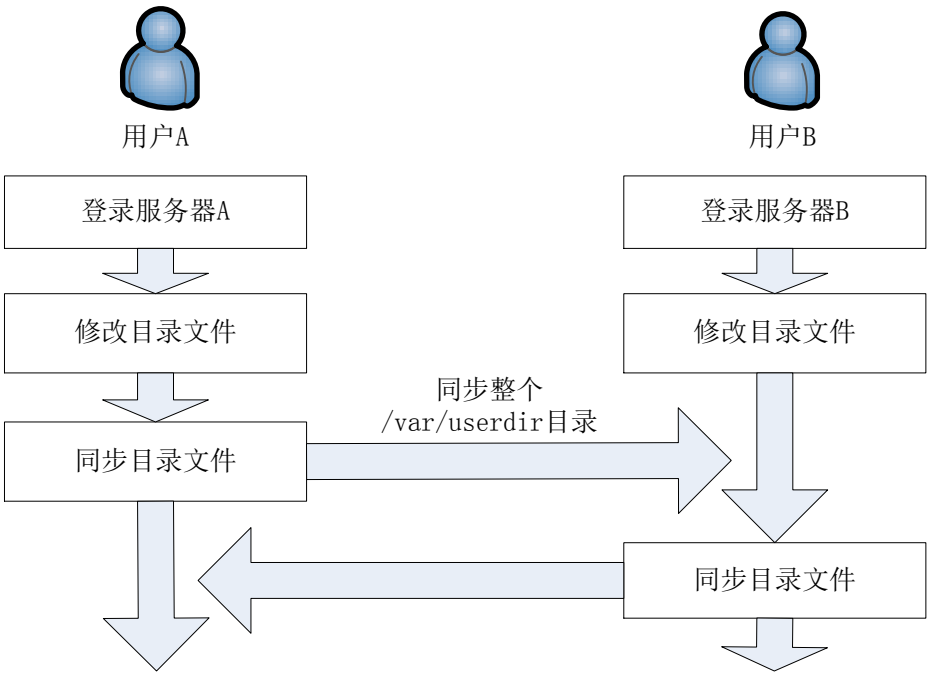

图 4-5 目录文件同步异常

脚本编写完成后添加执行权限并后台执行即可，命令如下。

```
# chmod +x inotify_rsync.sh
# ./inotify_rsync.sh &
```

至此，服务器 A 上用户目录存储位置下的所有变化都会自动同步至服务器 B。按照上述步骤，在服务器 A 上配置 rsync 服务，在服务器 B 上配置 inotify 服务即可实现将服务器 B 上的变化自动同步至服务器 A 上。如此一来服务器 A 和 B 上的目录文件将随时保持一致。

4.3.3 目录文件同步脑裂问题解决方案

所谓目录文件同步脑裂问题即目录服务器间目录文件的不一致问题，这种不一致问题不同于一般的不一致问题。一般的不一致问题是由于其中某台目录服务器上目录文件的更新引起的，这可以通过上一小节中说明的方法避免并使目录服务器间保持一致。而脑裂问题的不一致则主要是由网络异常或系统故障引起的。

举例来说，服务器 B 由于网络异常或系统故障而停止对外服务，在服务器 B 中断服务期间由服务器 A 继续对外提供目录服务。一段时间后服务器 B 重新恢复服务，如果此时有一个服务请求发送至 B 并最终导致服务器 B 上的目录文件被更新，那么将触发监控事件并引起同步。同步将使服务器 A 上的目录文件与服务器 B 保持一致。由此导致的问题就是服务器 A 在服务器 B 中断服务期间对目录文件做的所有修改被清空，从而导致用户上传的所有文件的文件信息以及对目录的修改被清空。这是一个非常严重的问题，也正是脑裂问题的严重之处。

脑裂的最大问题在于脑裂后有可能导致旧的信息覆盖新的信息而使新的信息丢失。因此必须设计相应的逻辑来避免这种情况的发生。

为避免上述情况的发生，除了服务器 A 和 B，还需要引入第三个节点。这第三个节点可以是四台服务器中另外两台中的一台，也可以是网内其他具有独立 IP 地址的网络设备，如网关。这里将网关作为第三个节点，整个结构如图 4-6 所示。

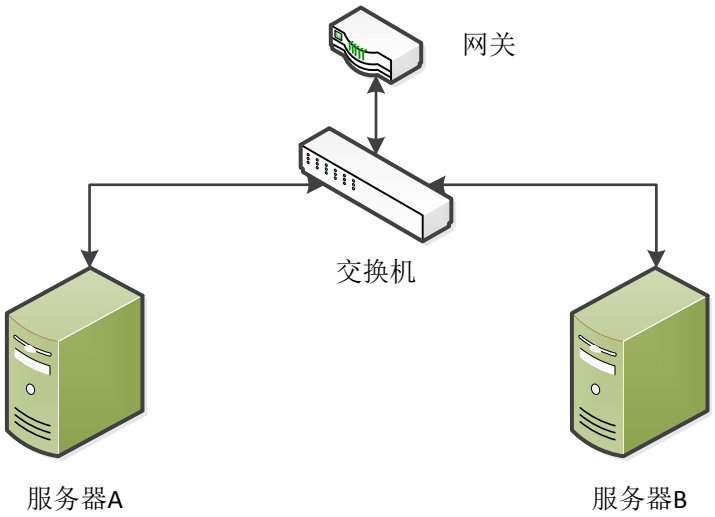

图 4-6 服务器网络连接示意图

服务器 A 和 B 都连接在同一交换机上，同时交换机连接在网关上。事实上，整个结构并不需要特意去实现，因为服务器通常都是这么部署的。

程序 dirHA 将被实现用来完成相应功能，确保在脑裂发生时也能正确处理由此引起的不一致问题。这里需要分两种情况来叙述 dirHA 的工作逻辑，一种是由网络异常引起的脑裂 dirHA 如何处理，另一种是由系统故障宕机引起的脑裂 dirHA 如何处理。

dirHA 将同时在服务器 A 和 B 上运行，这里以服务器 A 为例首先描述 dirHA 处理由网络异常引起的脑裂的工作过程，其工作流程如图 4-7 所示。

第一步。dirHA 首先会不断查询服务器 B 是否在线，确认服务器 B 的状态。如果服务器 B 在线则说明服务器 A 和服务器 B 都处于正常工作状态，间隔一段时间后再次查询。如果服务器 B 不在线则查询网关是否在线。

第二步。若此时网关在线则说明服务器 B 出现问题，这时将由服务器 B 上的 dirHA 完成后续工作，服务器 A 转至第一步继续查询服务器 B 是否在线。若此时网关不在线则说明服务器 A 出现问题，转至第三步。

第三步。在确认出现问题后立即停止目录服务及目录文件的同步服务，之后不断查询服务器 B 是否在线以验证自身网络是否恢复。若服务器 B 不在线则说明服务器 A 的网络并未恢复，间隔一段时间后再次查询。若服务器 B 在线则说明服务器 A 的网络已恢复，转至第四步。

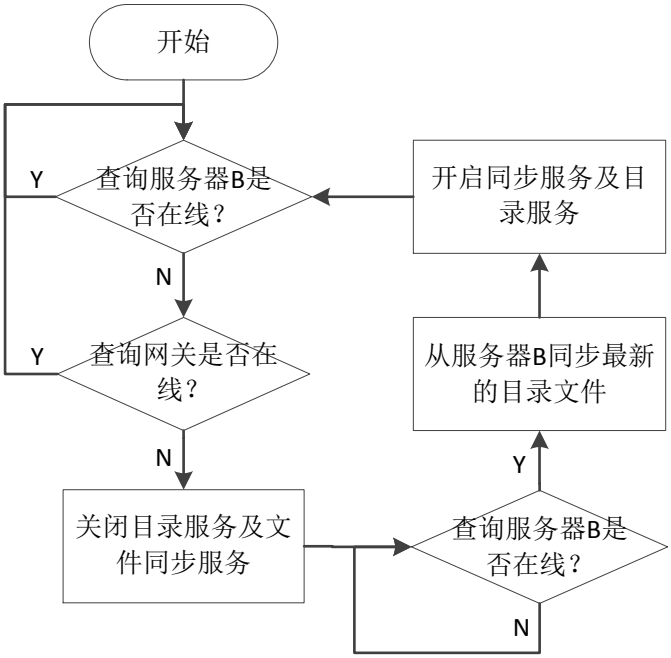

图 4-7 dirHA 处理由网络异常引起的脑裂的工作流程图

第四步。将服务器 B 上的目录文件同步至本地即服务器 A 上。

第五步。同步完成后先恢复目录文件同步服务，再恢复目录服务。至此服务器 A 已恢复完成，转至第一步继续进行自我监控。

上述过程中，第三步关闭目录文件同步服务的同时关闭目录服务是为了避免在网络恢复后，且还未从服务器 B 同步最新目录文件前，由目录请求引起的服务器 A 的目录文件的变化。这个变化会在从服务器 B 同步目录文件之后被覆盖，进而丢失重要信息，因此需要避免。这么做将确保服务器 A 在完全恢复之前不提供

目录服务。第五步先开启目录文件的同步服务再开启目录服务是为了保证由目录请求引起的目录文件变化能及时同步至服务器 B。若先开启目录服务后开启目录文件同步服务，则在目录文件同步服务启动前的目录文件变化将不能同步至服务器 B。

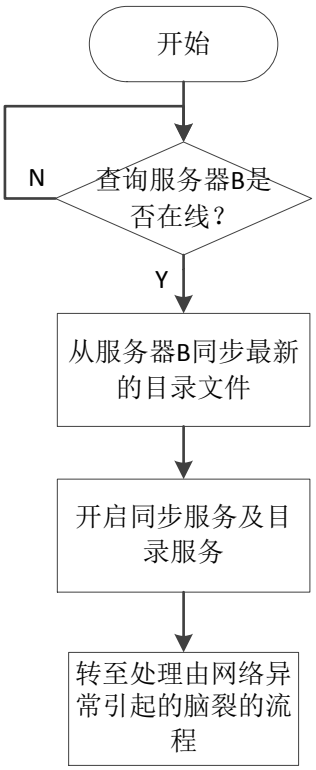

图 4-8 dirHA 处理由系统故障宕机引起的脑裂的工作流程图

dirHA 在处理由系统故障宕机引起的脑裂时与处理有网络异常引起的脑裂略有不同，这里仍以服务器 A 为例描述 dirHA 处理由系统故障宕机引起的脑裂的工作过程，其工作流程如图 4-8 所示。

服务器 A 启动后，dirHA 便不断查询服务器 B 是否在线，以验证自身网络是否畅通。在确认网络畅通后，首先从服务器 B 上同步最新的目录文件。同步完成后先开启目录文件同步服务，再开启目录服务。至此服务器 A 已完全恢复，之后 dirHA 转至处理由网络异常引起的脑裂的流程。

与处理由网络异常引起的脑裂相比，处理由系统故障宕机引起的脑裂时不再需要判断自身是否出了问题，因为这一点是确定的。dirHA 在确认网络畅通后即可进行同步和恢复。

4.4 公共接口详细设计与实现

公共接口是目录文件系统非常重要的组成部分，接口设计好之后终端开发即可通过调用接口来实现相关功能。公共接口主要包括建立连接、用户注册、用户登录、创建目录、打开目录、重命名、删除目录、上传文件、下载文件、删除文件和关闭连接。

在调用公共接口完成相关功能时，各个接口与服务端传递的消息都是使用 JSON 格式。每个接口的消息对应的消息类型不同，服务端根据不同的消息类型区别不同的服务请求。消息类型将在消息中表明。

服务端由一个主进程负责监听服务端口，在建立连接后主进程将生成新线程负责该链接上的所有请求，该线程将根据接收到的消息类型将具体的功能交给对应模块完成。

4.4.1 建立连接

使用网络服务首先需要与服务器建立连接，建立连接是后续操作的基础。接口定义如下。

```
int connectDirServer(void)
```

接口的返回值类型为 int 型，无参数。

接口的返回值有-1 和非负整数。返回值-1 表示连接失败。返回值为非负整数时表示连接成功，此时返回值即为套接字的文件描述符。

接口工作流程如图 4-9 所示。

接口首先会从配置文件中获取服务器 IP。获得服务器 IP 后，为考虑访问的随机性和服务端的负载均衡，接口首先会随机选择其中一个 IP 尝试连接。如果连接成功则返回套接字的文件描述符。如果连接失败，则首先判断是否所有的服务器 IP 都已经尝试。若已经全部尝试，则返回连接失败。若没有全部尝试，则顺序尝试连接剩余的服务器 IP。

在首次尝试连接失败后采用顺序尝试而不继续采用随机尝试主要考虑到以下两点因素。第一，随机选择可能又随机到刚刚尝试连接失败的 IP。尽管有可能是因为网络因素而非服务器本身问题导致连接失败，但再次尝试连接这个 IP 仍然失败的可能性很大。第二，顺序尝试剩余服务器 IP 逻辑简单。与其增加排除随机连接失败 IP 的逻辑，不如直接顺序尝试简单，简单意味着更加可靠。

建立连接接口将建立终端与服务端之间的连接。根据设计，接口在实现时主

要需要考虑尝试连接时对服务器 IP 的选择。

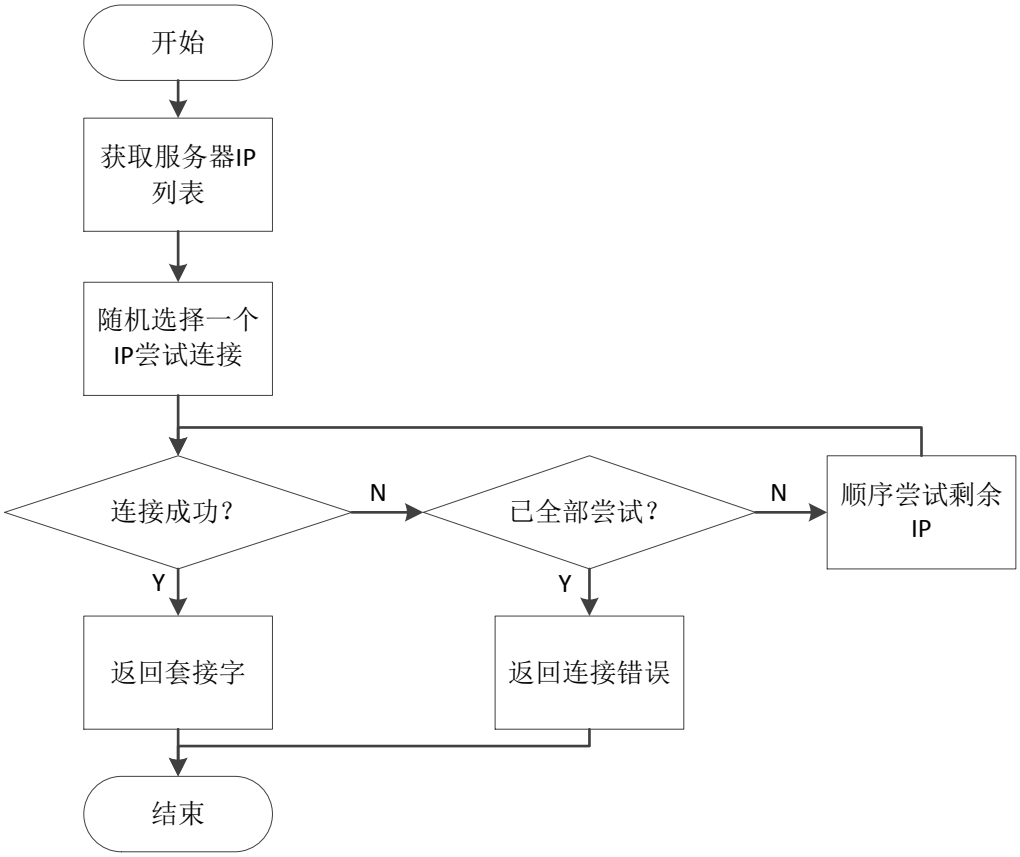

图 4-9 建立连接接口工作流程图

接口中服务器 IP 将使用字符串数组（即字符类型的二维数组）存储。接口将从存储有服务器 IP 的配置文件中读取服务器 IP，并将之存储在这个二维数组内。

接口在首次尝试连接时会生成一个随机数，这个随机数的范围将对应存储有服务器 IP 的二维数组第一维的维数，即服务器 IP 的总个数。之后以该随机数作为下标从该二维数组中获取对应的服务器 IP 作为首次尝试连接的服务器 IP。若连接成功，则返回套接字描述符。若连接失败，则采用循环方式从之前随机下标的下一个开始顺序尝试剩余的服务器 IP。若全部尝试后仍然没有连接成功则返回连接失败。连接过程的伪代码如下。

```
//服务器 IP 存储在字符型二维数组 serverIP 中
srandom((int)time(NULL));
first = random() % serverNum;     //以系统时间作为种子生成随机数
if(socketfd = serverConnect(serverIP[first]) > 0)     //如果连接成功
```

```
{
    return socketfd;
}
for(i = first + 1; i != first; i = (i++) % serNum) //顺序尝试剩余服务器 IP
{
    if(socketfd = serverConnect(serverIP[i]) > 0)
    {
        return socketfd;
    }
}
return -1;    //全部尝试失败后返回错误代码-1
```

4.4.2 用户注册

设计用户注册接口，首先需要设计数据库用户表，它主要有三个作用。第一个作用是用于用户认证，用户表中保存有用户的账户信息，通过验证账户信息认证用户身份。第二个作用是将用户与目录联系起来，形成用户与目录之间的映射关系，从而确认每个用户的个人目录的存储位置。第三个作用是控制用户的使用权限，这里主要指对各用户存储空间的控制。用户表结构如表 4-2 所示。

表 4-2 用户表

字段名	类型	说明
userid	int	用户 ID
username	varchar	用户名
password	varchar	用户密码
totalquota	int	用户总配额
usedquota	int	用户已用配额

username 字段是主键，为用户名。userid 字段是候选键，为用户 ID。每个用户的目录文件存储位置由用户 ID 区分，用户认证使用用户名。当用户使用用户名和密码认证成功后，系统可确认该用户的用户 ID。因为所有用户的目录文件都是在同一位置统一存储并以用户 ID 区分的，因此知道用户 ID 即可确认用户根目录文件的存储位置，进而可以正确的返回用户的根目录内容。password 字段为用户密码使用 SHA-2 散列后的结果。用户认证时对用户输入的密码使用相同算法获得散列值，之后将散列结果与数据库中的值比较以确认用户的合法性。totalquota 字

段为用户云存储空间的总配额，即该用户可使用的最大云存储空间，以字节为单位。usedquota 字段则为用户已使用的配额，也以字节为单位。

建立连接后，用户注册是第一个必须设计和实现的接口，没有用户后面的功能就无从谈起。接口定义如下。

int registerUser(int socket, char *username, char *password)

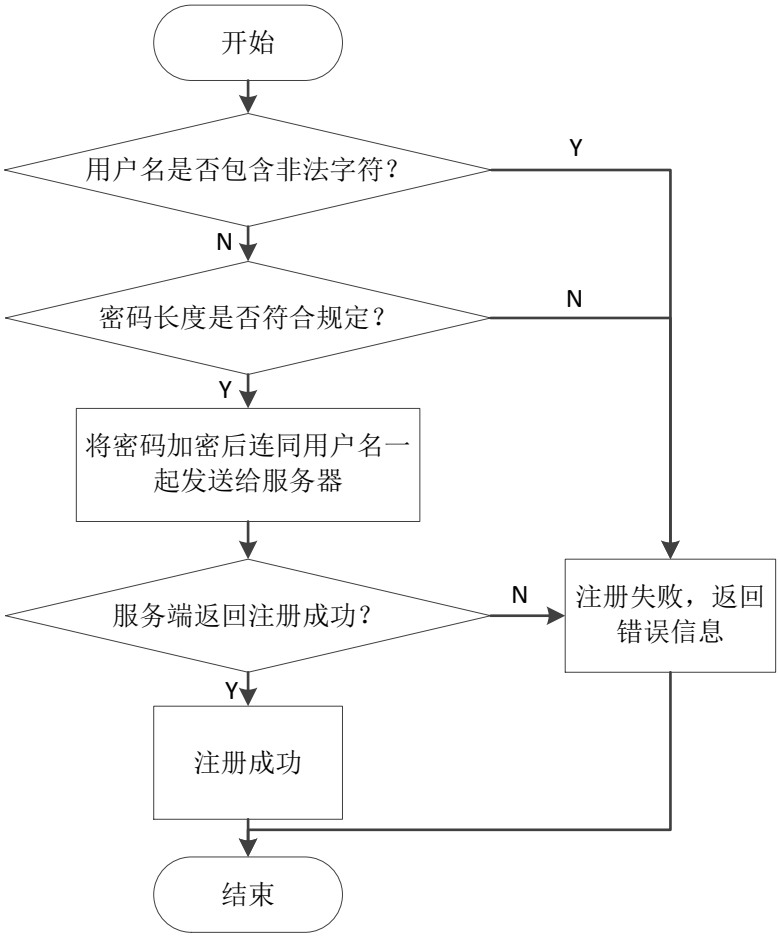

图 4-10 用户注册接口工作流程图

接口的返回值类型为 int 型，参数 socket 为调用 connectDirserver 接口成功后返回的套接字，参数 username 表示欲注册的用户名，参数 password 表示用户欲使用的密码。

接口的返回值有 0、1、2 和 3。返回值 0 表示用户注册成功。返回值 1、2 和 3 都表示用户注册失败。其中，返回值 1 表示用户名中含有非法字符，返回值 2 表示密码长度不符合规定，返回值 3 表示用户名已存在。

接口的工作流程如图 4-10 所示。

用户注册接口主要完成三项工作，分别是用户名合法性检查、用户密码合法性检查及与服务端通信完成注册。

用户名检查旨在规范用户名命名，主要针对用户名中可能出现的非法字符进行检查，通常用户名由英文字母、数字和下划线组成。循环将用户名中的每个字符与非法字符进行比较是不切实际的，因为非法字符是“无限”的。这里利用 C 语言弱类型的特点，对用户名中的字符进行几次数值比较即可判断是否为合法字符。检查过程的伪代码如下：

```
for(i = 0; i < length_username; ++i) //length_username 为输入的用户名的长度
{
    if( (username[i] >= ‘a’ && username[i] <= ‘z’) ||
        (username[i] >= ‘A’ && username[i] <= ‘Z’) ||
        username[i] == ‘_’ )//如果该字符是合法字符，即是大小写字母或下划线
    {
        continue; //继续检查下一个字符
    }
    else
    {
        break; //退出检查
    }
}
if (i != length_username) //如果未检查用户名中的全部字符
{
    return 1; //返回错误代码
}
```

接口在判定用户名合法后接着检查用户输入的密码是否符合长度要求。用户密码合法性的检查主要是针对密码长度的检查，通常密码长度最少要求 6 位，最长一般支持 18 位。用户密码合法性的检查通过判断密码字符串的长度容易实现。

在用户名和密码的合法性检查通过后，接口会将加密后的密码和用户名发送至服务端。服务端接收到用户名和密码后尝试进行注册。若注册成功则返回注册成功，否则返回注册失败。接口与服务端的通信通过传递 JSON 格式的消息实现，

具体内容如下。

{“type”:”1”,”parameter”:{”username”:”xxx”,”password”:”xxx”}}

服务端接收到这条消息后，首先解析“type”为“1”，判断这是条用户注册消息。之后服务端获取“parameter”的值（一个 JSON 串）并将其传递给对应的用户注册模块完成注册。服务端用户注册模块的工作流程如图 4-11 所示。

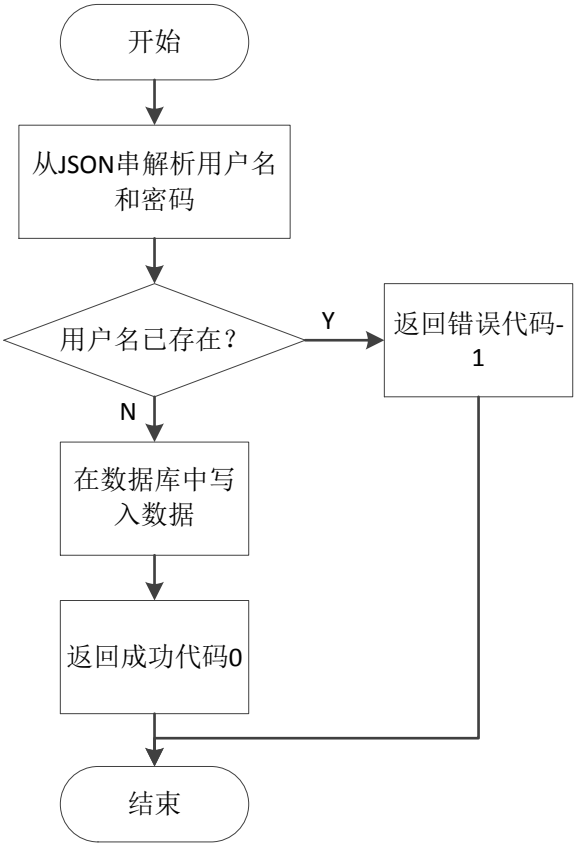

图 4-11 服务端用户注册模块工作流程图

服务端注册模块首先解析作为参数的 JSON 串获取用户名和密码。然后在数据库中查找是否有同名用户。若没有就在数据库中写入用户记录并返回成功，若有则返回错误信息。返回消息的 JSON 消息如下，其中“x”即为返回码。

{“result”:”x”}

用户密码在传递时使用 AES 加密，注册模块接收到密文之后先解密，之后使

用 SHA-2 获得密码的散列值。数据表中用户密码字段存储即该散列值。

4.4.3 用户登录

登录是用户使用云存储服务需要做的第一步。接口定义如下。

int login(char *username, char *password, struct item *dir)

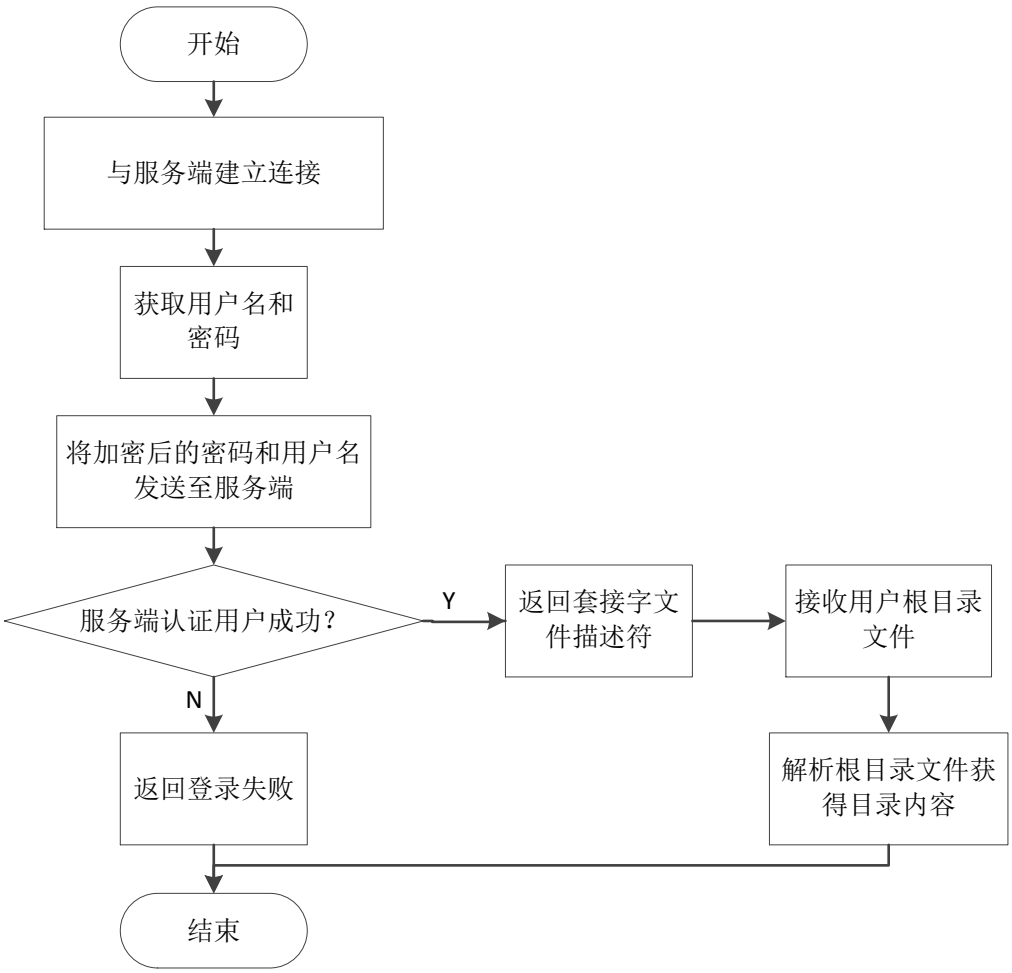

图 4-12 用户登录接口工作流程图

接口的返回值类型为 int 型，参数 username 为登录的用户名，参数 password 为用户密码，参数 dir 是一个指针，指向由用户根目录文件中各目录项组成的数据结构。

接口的返回值有-1 和非负整数。返回值-1 表示登录失败。返回值为非负整数时表示登录成功，此时返回值即为与服务器建立连接的套接字的文件描述符。

接口的工作流程如图 4-12 所示。

终端首先与服务端建立连接，之后从参数中获得用户名和密码，将加密后的密码和用户名一同发往服务端。服务端接收到用户名和密码后对其进行认证。若认证成功则返回套接字的文件描述符，若失败则返回登录失败。用户登录接口在完成用户认证后还有一项工作需要做，那就是拉取用户的根目录内容。用户认证成功后，服务端返回用户的根目录文件给终端。之后接口将解析获得的目录文件，将目录项组成特定的数据结构。开发人员可根据参数 dir 获得该数据结构从而可以将目录内容显示在终端上。

用户登录接口同样会返回建立连接成功后的套接字描述符，但与建立连接接口不同的是，用户登录接口在建立连接后会立即进行用户认证，只有认证通过才能称之为连接成功。在具体实现时，建立连接部分可直接使用建立连接接口，在成功连接之后进行用户认证，认证通过后方可返回套接字描述符。若认证不通过则关闭连接返回连接失败。

接口与服务端通信通传递的 JSON 格式消息具体内容如下：

{“type”:”2”,”parameter”:{”username”:”xxx”,”password”:”xxx”}}

从内容上看，登录用户接口与注册用户接口传递的 JSON 消息几乎一致，只有消息类型不同。“type”为“2”表明这是一个用户登录消息。服务端接收到消息之后将参数用户名和密码传至对应的用户登录模块完成相应功能。服务端用户登录模块的工作流程如图 4-13 所示。

登录模块从 JSON 串中解析出用户名和密码后随即与数据库中保存的信息进行比较。若信息不一致则返回错误代码并退出。若信息一致则返回登录成功，之后通过数据库查询该用户的用户 ID。得到该用户的用户 ID 后即可得知该用户的根目录位置。登录模块将读取该目录文件并发送回终端。

参数中的密码同样采用 AES 加密，登录模块解密之后使用 SHA-2 获得密码的散列值，通过对比数据库中记录的该用户的密码散列值确认信息是否一致。

由于目录文件中每一个目录项所占的字节数是一定的，因此服务端可将目录文件拆分成一个一个目录项进行发送。为方便之后对目录内容的访问和操作，终端接口在接收时需要把所有目录项重新组织成新的数据结构。

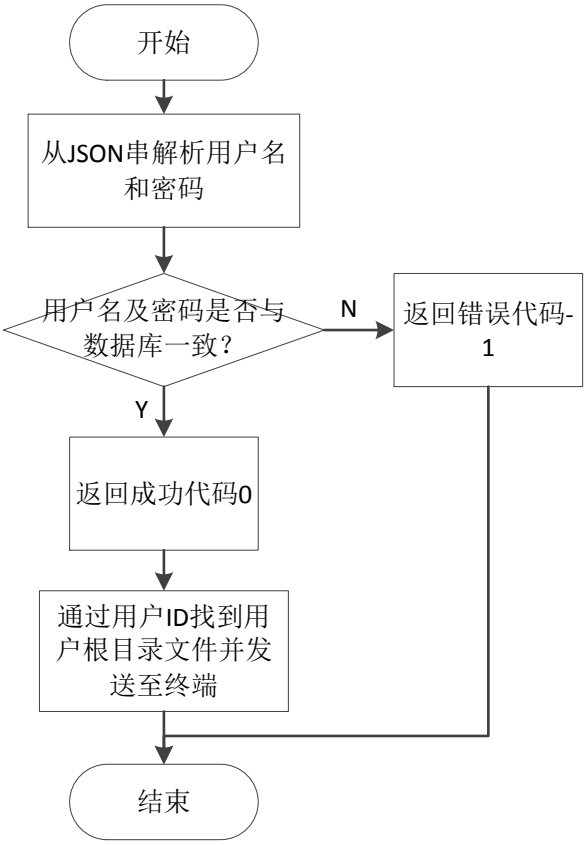

图 4-13 服务端用户登录模块工作流程图

在终端，整个目录文件的内容将由一个单链表来存储（以下称为目录链表），接口定义中传入的参数 dir 即为该链表的头节点指针。链表的每一个节点存储一个目录项，节点结构如下。

```
struct item{
char filename[256];
char flag;
char identification[4096];
char size[32];
struct item *next;
};
```

节点结构中的“filename”、“flag”、“identification”和“size”成员的含义与目录项结构中的对应成员的含义相同。因为是单链表，所以结构中最后一个成员是指向下一节点的指针。

对目录内容的访问和操作主要是指对目录项的查询，无论是打开新目录还是下载文件，都必须通过查询目录项来获得目录及文件的相关信息。为提高用户的使用体验，查询必须能够在尽可能短的时间内完成。考虑到多数用户的使用习惯，同一个目录下并不会创建过多文件或目录，因此即使使用单链表对目录项进行顺序查询，以现有计算机的运行速度也可以在瞬间得到结果。所以为简单起见，目录内容的存储采用单链表实现，并通过参数 dir 返回给调用者。

4.4.4 创建目录

创建目录接口用于用户创建自定义目录，自定义目录可以简单方便的管理个人文件。接口定义如下。

int createDir(int socket, char *DirName, struct item *dir)

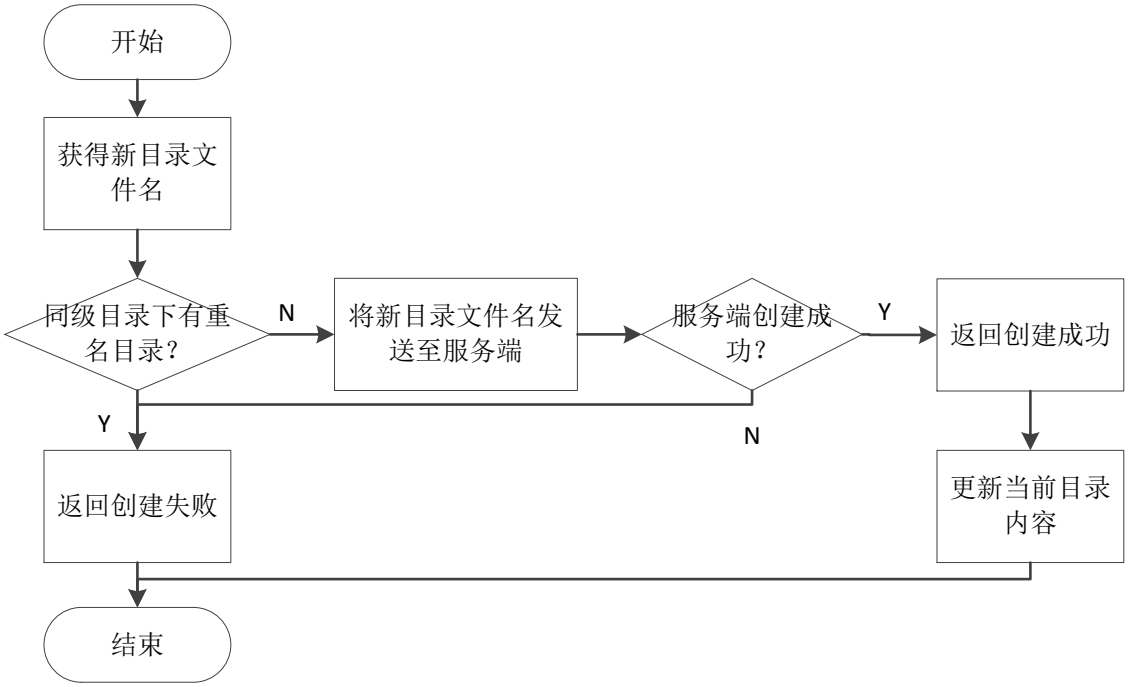

图 4-14 创建目录接口工作流程图

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 DirName 为欲创建的文件夹的文件名，参数 dir 是一个指针，指向由用户当前目录文件中各目录项组成的数据结构。

接口的返回值有 0、1 和 2。返回值 0 表示创建成功，返回值 1 和 2 都表示创建失败。其中返回值 1 表示新创建的目录与同级目录下已有的目录重名，返回值 2

表示其他原因引起的创建失败（例如由于系统 inode 耗尽引起的创建失败等）。

接口的工作流程如图 4-14 所示。

由于终端侧已经拥有当前目录的目录文件，因此在获取到新目录的文件名时首先检查在当前目录下是否有同名的已经存在的目录。若已经存在同名的目录则返回创建失败。若没有则将新目录文件名发送至服务端。服务端接收到新目录文件名之后尝试创建新目录，其中包括创建实际目录及目录文件，同时修改当前目录文件。若创建失败则返回创建失败，若创建成功则返回创建成功。接口得到创建成功的消息后将通过参数 dir 更新当前目录内容。

新目录重名检查通过查询目录链表完成，目录链表通过参数 dir 传入，查询过程的伪代码如下。

```
指针指向目录链表；
while(没有到链表末尾)
{
    if(文件类型是目录且文件名相同)
    {
        返回重名；
    }
    指针指向下一节点；
}
```

重名检查通过后接口将通过 JSON 消息将信息发送至服务端，消息的具体内容如下。

```
{“type”:”3”,”parameter”:{”curpath”:”xxx”,”dirname”:”xxx”}}
```

消息中“type”为“3”表明这是一个创建目录的请求，参数包括当前目录的路径（curpath）及新目录文件名（dirname）。当前目录的路径可以通过目录链表的第一项获得。

服务端创建目录模块获得参数后即可尝试创建，创建目录使用系统调用 mkdir。创建过程的伪代码如下。

```
if(mkdir(path, S_IWRXU) == 0)  //新目录创建成功
```

```
{
    if((fp = fopen(path_dirfile, “w”)) != NULL)     //新目录的目录文件创建成功
    {
        向目录文件中写入当前目录及父目录的目录项；
        fclose(fp);
        更新当前目录的目录文件；
        返回创建成功；
    }
}
else
{
    //返回创建失败
}
```

“path”由传入参数“curpath”和“dirname”组成，“path_dirfile”为目录文件的路径，即在“path”的末尾追加目录文件文件名。只有当新目录及新目录的目录文件都创建成功，并更新了当前目录的目录文件后整个创建目录的过程才能称为成功。

在终端接口获得服务端返回的创建成功消息后，无需重新拉取当前目录的目录文件。接口将在目录链表的第三个节点处插入新目录的目录项节点。目录链表的前两个节点分别是当前目录与父目录的目录项节点，保持其相对位置有利于后续操作，因此新目录的目录项节点将直接插入到第三个节点的位置。同时这样插入节点不仅节省时间（与在表尾处插入相比），而且根据用户使用习惯在创建完新目录后紧接着就是打开新目录，这样也能很快查询到新目录的目录项节点，提高效率。

4.4.5 打开目录

浏览目录的过程实际上就是打开各个目录的过程，打开目录接口将从服务端获取指定目录的目录文件并将目录内容提供给开发者。接口定义如下。

```
int openDir(int socket, char *DirName, struct item *dir)
```

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 DirName 为欲打开目录的文件名，参数 dir 是一个指针，指向由用户当前

目录文件中各目录项组成的数据结构。

接口的返回值有 0 和 1。返回值 0 表示打开成功，返回值 1 表示打开失败。

接口根据欲打开目录的文件名向服务端发送请求，请求对应的目录文件。接口将解析从服务器处获得的目录文件，将目录项组成特定的数据结构。之后开发人员可根据参数 dir 获得该数据机构从而可以将目录内容显示在终端上。

打开目录接口功能比较单一，在实现时没有需要特别注意的地方。

接口首先在目录链表中查询欲打开目录的目录项节点，之后从该节点中获得该目录的存储位置。之后通过 JSON 消息将位置发送至服务端，其消息内容如下。

{“type”:”4”,”parameter”:{”path”:”xxx”}}

消息中“type”为“4”表明这是一个打开目录的请求。服务端接收到消息后将参数交由打开目录模块，模块解析出参数后便从相应位置读取目录文件并传回终端。其中参数“path”即为欲打开目录的路径。同样的，终端接口在接收目录文件时会将新目录文件组织称目录链表，通过 dir 参数返回。

4.4.6 重命名

重命名目录或文件是一个相当普遍的需求，因此提供一个接口方便该操作。接口定义如下。

int modifyDir(int socket, char *oldFileName, char *newFileName, int flag, struct item *dir)

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 oldDirName 是目录或文件的原名称，即欲进行重命名操作的目录或文件的名称，参数 newDirName 是目录或文件的新名称，参数 flag 表明进行重命名操作的是目录还是普通文件，参数 dir 是一个指针，指向由用户当前目录文件中各目录项组成的数据结构。

接口的返回值有 0 和 1。返回值 0 表示修改成功，返回值 1 表示修改失败。

接口的工作流程如图 4-15 所示。

接口获得新旧文件名及文件类型后首先判断新文件名是否与同级目录下同类型文件重名，即是否有重名的普通文件或重名的目录。若存在重名文件则返回重

命名失败。若不存在重名文件则将相关信息一并发送至服务端。服务端接收到相关信息后尝试进行重命名，这其中包括对目录文件的修改和实际目录的修改。如果重命名成功则返回重命名成功，否则返回重命名失败。服务端返回重命名成功后，接口不需要再重新同步目录文件，而是直接通过参数 dir 修改目录内容。

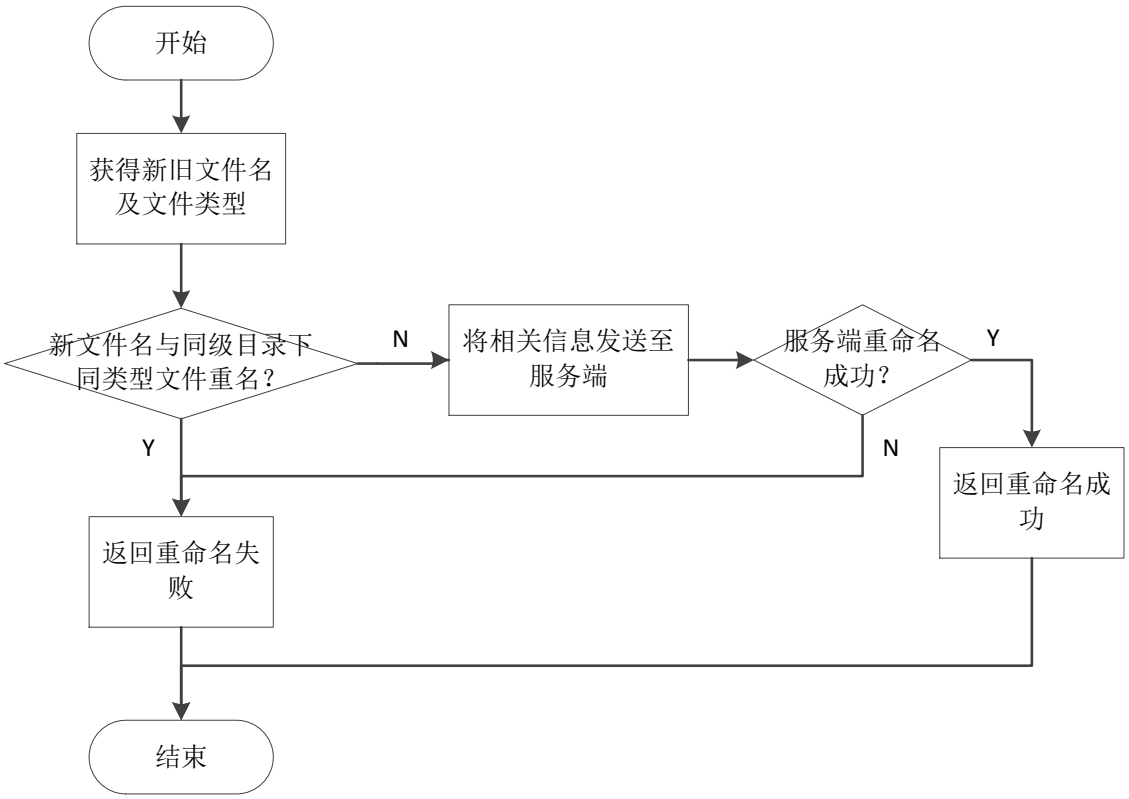

图 4-15 重命名接口工作流程图

重命名接口在实现时首先需要判断新文件名是否与本目录下其他文件重名。重名检查同样是通过查询目录链表实现。同时，重命名接口在实现时须分为两种情况进行处理，即重命名普通文件和重命名目录。这些信息将通过 JSON 消息传递，其具体内容如下。

{“type”:”5”,
”parameter”:{“retype”:”x”,”curpath”:”xxx”,”oldname”:”xxx”,”newname”:”xxx”}}

“type”为“5”表明这是一个重命名请求消息。服务端接收到该消息后会将消息参数传递给重命名模块，重命名模块根据参数内容决定完成何种工作。参数“retype”表示重命名的类型，“retype”为“1”表示重命名普通文件，“retype”为“2”表

示重命名目录。“curpath”为当前目录的目录文件的存储位置。“oldname”和“newname”分别表示旧文件名与新文件名。重命名模块的工作流程如图 4-16 所示。

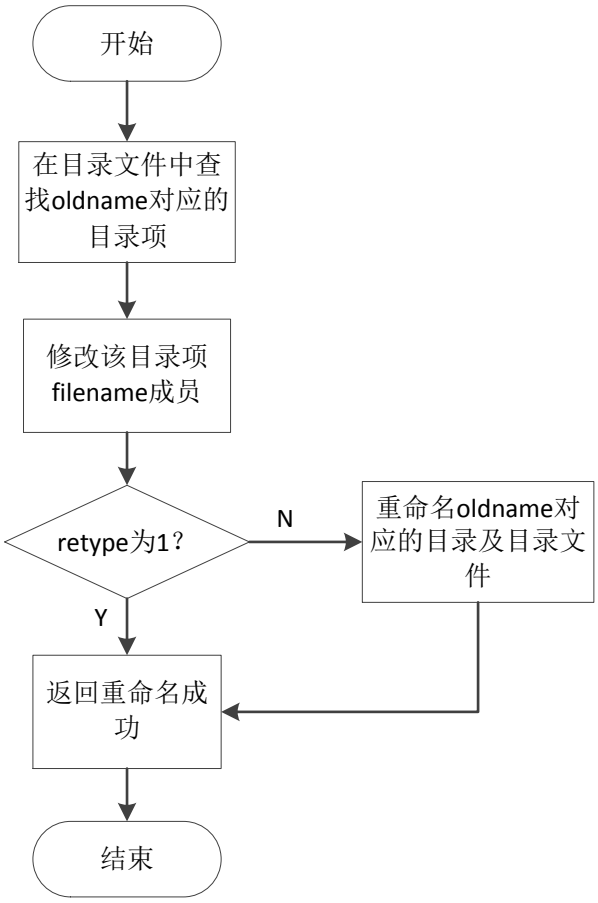

图 4-16 服务端重命名模块工作流程图

无论是重命名文件还是重命名目录，修改当前目录文件中 oldname 对应目录项的 filename 成员都是必须的，这是一致性操作。这个操作涉及到在文件内替换某一部分内容，其伪代码如下。

```
struct diritem buff;
FILE *fp = fopen(path, “rb+”);    //以二进制读写方式打开目录文件
fread((char*)&buff, sizeof(struct diritem), 1, fp);//读取第一个目录项
while((result = strcmp(buff.filename, oldname)) != 0)//对比文件名
{
    if(feof(fp))
```

```
        {
            break;   //读到文件末尾则退出
        {
        fread((char*)&buff, sizeof(struct diritem), 1, fp);//继续读取下一个目录项
    }//while
    if(result == 0)//找到了正确的目录项
    {
        fseek(fp, -sizeof(struct diritem), SEEK_CUR);//文件指针回退
        snprintf(buff.filename, 256, "%s", newname);//替换文件名
        fwrite((char *)&buff, sizeof(struct diritem), 1, fp);//替换旧目录项
    }
    fclose(fp);
```

如果重命名的是普通文件，在修改完目录文件后即可给终端返回结果。如果重命名的是目录，则根据目录文件的存储设计还需修改服务器上对应的目录和目录文件。这可通过系统命令“mv”实现。

4.4.7 删除目录和文件

删除目录接口除了将删除指定目录的目录文件外，还将删除指定目录下的所有文件及所有子目录，这是一个递归的过程。接口定义如下。

```
int deleteDir(int socket, char *DirName, struct item *dir)
```

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 DirName 为欲删除的目录名称，参数 dir 是一个指针，指向由用户当前目录文件中各目录项组成的数据结构。

接口的返回值有 0 和 1。返回值 0 表示删除成功，返回值 1 表示删除失败。

接口将欲删除目录的名称发送至服务端，服务端接收到目录名称后尝试删除，这其中包括删除目录文件，删除实际目录，删除目录下的文件。若删除成功则返回删除成功，否则返回删除失败。返回删除成功后，接口不再重新同步目录文件，而是利用参数 dir 直接修改当前目录内容以保持服务端与终端的一致性。

FastDFS 本身已具有删除文件的接口，与目录结构相结合后，其接口定义如下。

int deleteFile(int socket, char *filename, struct item *dir)

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 filename 为欲删除文件的文件名，参数 dir 是一个指针，指向由用户当前目录文件中各目录项组成的数据结构。

接口的返回值有 0 和 1。返回值 0 表示删除成功，返回值 1 表示删除失败。

在删除前首先通过目录文件查询欲删除文件对应的文件标识符，之后使用 FastDFS 本身具有的删除接口删除文件。删除文件同时需要与服务端通信删除目录文件中该文件对应的目录项，且删除完成后终端需要更新目录内容。

终端删除目录接口与删除文件接口可以通过调用同一终端删除模块实现，该模块首先通过参数获得欲删除目录后，将信息以 JSON 格式发送至服务端，消息具体内容如下。

{“type”:”6”,”parameter”:{”curpath”:”xxx”,”filename”:”xxx”}}

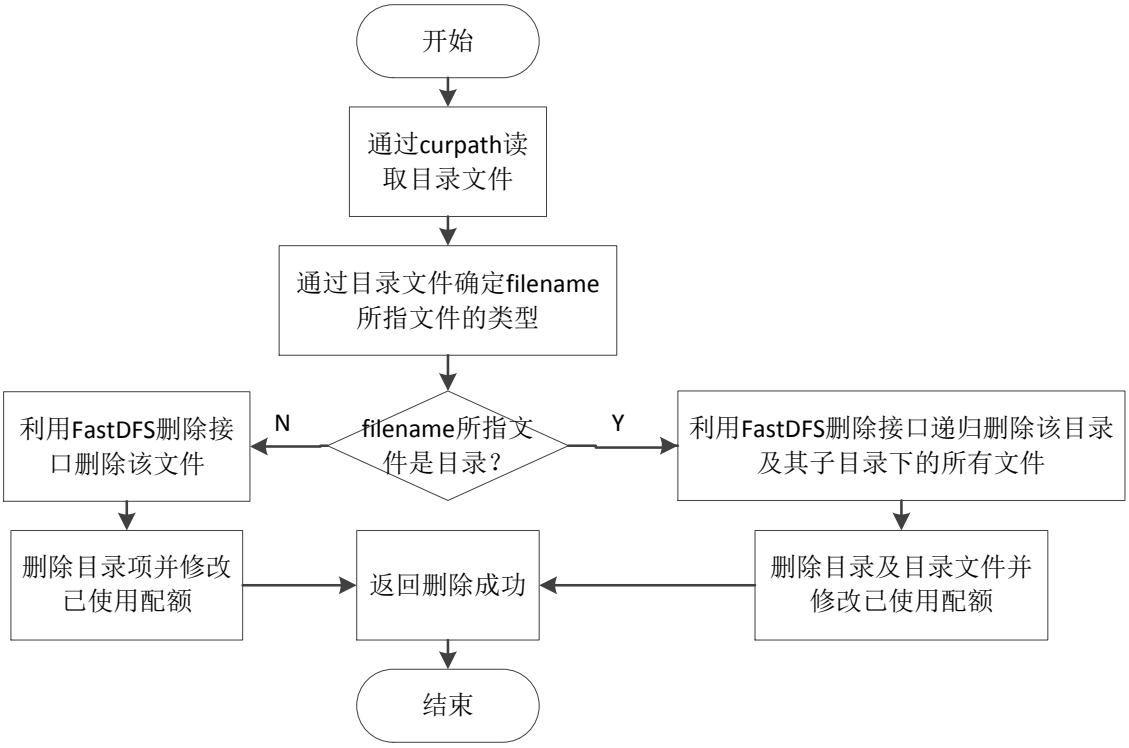

图 4-17 服务端删除模块工作流程

“type”为“6”表明这一条删除请求。服务端接收到该消息后将参数交由服务端删除模块处理。服务端删除模块首先解析参数，“curpath”为当前目录的路径，

“filename”为欲删除文件的文件名。之后服务端删除模块将通过确定 filename 所指文件的类型进行后续操作，其工作流程如图 4-17 所示。

服务端删除模块首先通过参数 curpath 打开当前目录的目录文件，并在其中顺序查找 filename 对应的目录项。根据该目录项保存的文件类型，删除模块将进行不同的操作。

若欲删除文件的文件类型为普通文件，则首先获取该文件的文件标识符并通过 FastDFS 删除接口删除该文件。服务端删除模块在文件删除之后将删除该文件对应的目录项并在数据库中修改用户的已使用配额。因为目录项可能位于目录文件的任何位置，因此为减少对目录文件的读写操作，实现时进行如下处理。

```
//确认 filename 对应的目录项之后
fseek(fp, -sizeof(struct diritem), SEEK_CUR);//文件指针回退到目录项起始位置
location = ftell(fp);//记录欲删目录项的起始位置
fseek(fp, -sizeof(struct diritem), SEEK_END);//定位至最后一个目录项起始位置
newlength = ftell(fp);//该位置将作为新目录文件的长度
fread((char*)&buff, sizeof(struct diritem), 1, fp);//保存最后一个目录项
fseek(fp, location, SEEK_SET);//定位至欲删目录项起始位置
fwrite((char *)&buff, sizeof(struct diritem), 1, fp);//替换欲删除的目录项
fclose(fp);
truncate(curpath, newlength);//截断目录文件，删除最后一项目录项
```

整个删除过程通过将目录文件的最后一个目录项覆盖欲删目录项，并截断目录文件删除最后一个目录项的方法实现。同时通过文件指针定位减少对目录文件的读写次数，从而提高了效率。不仅如此，这样做还最大限度的保持了目录文件的原貌，因此将减少使用 rsync 同步时的数据量。

若欲删除文件的文件类型是目录，则删除模块除了要在当前目录的目录文件中删除该目录的目录项，还需删除该目录下的所有文件，这是一个递归的过程。删除过程如图 4-18 所示。

删除目录下的所有文件在实现时即扫面该目录的目录文件，逐一删除每个目录项代表的文件。若目录项代表的是目录，则进入该目录删除该目录下所有的文件，每删除一个文件都要修改用户的已使用配额。某一个目录下的所有文件都已删除后服务端将该目录及其目录文件一并删除。最后，当欲删目录下的所有子目录及文件都已删除后，在当前目录的目录文件中删除该目录对应的目录项。

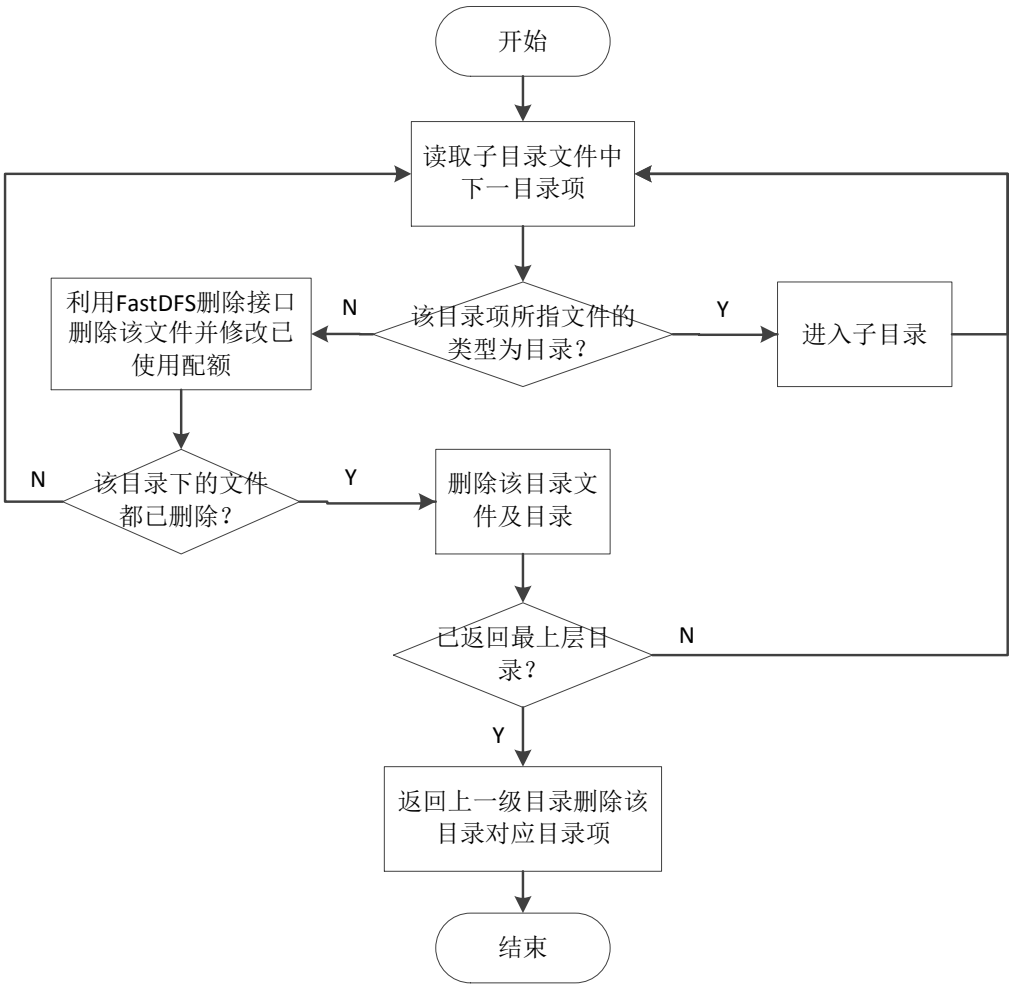

图 4-18 服务端删除模块删除目录工作流程

在服务端返回删除成功后，终端删除模块将欲删目录的目录项节点从目录链表删除。

4.4.8 上传文件

FastDFS 已经具有上传文件的接口，但是要与目录结构相结合则必须在它的基础上做出相应的修改。接口定义如下。

int uploadFile(int socket, char *path, char *DirName, struct item *dir)

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 path 为上传文件的路径，参数 DirName 为文件上传的目录名称，即当前目录，参数 dir 是一个指针，指向由用户当前目录文件中各目录项组成的数据结

构。

接口的返回值有 0 和 1。返回值 0 表示上传成功，返回值 1 表示上传失败。

接口的工作流程如图 4-19 所示。

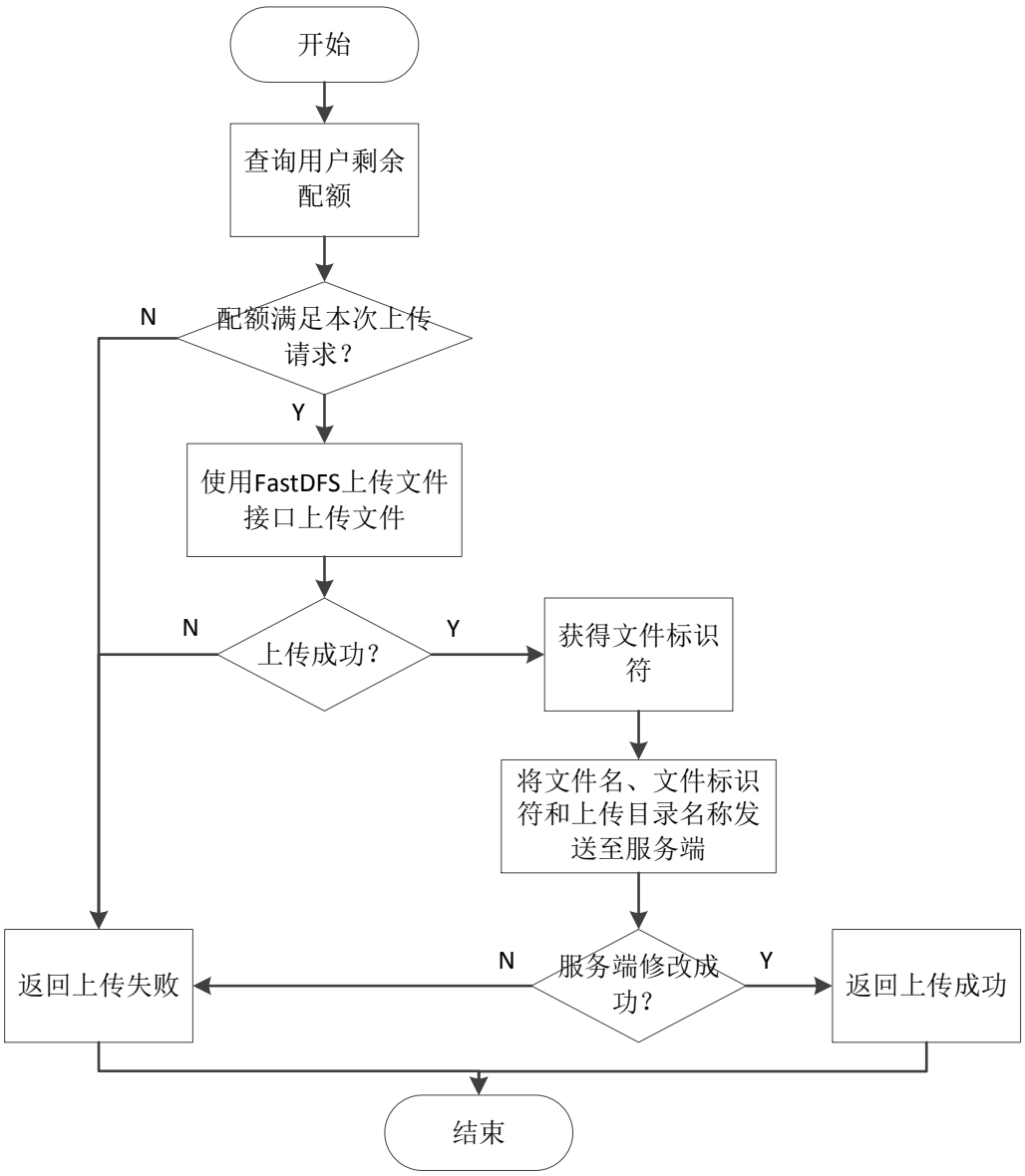

图 4-19 上传文件接口工作流程图

接口首先与服务器通信查询用户的剩余配额。若剩余配额能够容纳当前欲上传文件，则继续进行下一步即上传文件，若不能容纳则返回上传失败。上传文件利用 FastDFS 本身具有的文件上传接口将文件上传是服务器。如果上传失败，则直接返回上传失败。如果上传成功将获得 FastDFS 接口返回的文件标识符。之后接口会将文件名、文件标识符和上传目录名一并发送至服务端。服务端根据接收

到的信息修改目录文件。如果修改成功则返回修改成功。接口收到服务端修改成功的消息后直接通过 dir 添加新的目录项。

实现时，上传文件接口向服务端查询用户剩余配额仍然使用 JSON 消息，其内容如下。

{“type”:”7” }

这条消息只包含消息类型，服务端接收到该消息后会在数据库中查询用户的剩余配额，之后返回查询结果。服务端有相应机制可以知晓本连接的使用用户，因此请求消息中不含相关用户信息。具体相关机制见 4.5 小节。

在确认用户剩余配额满足要求后调用 FastDFS 上传接口上传文件，上传成功后获得文件的标识符。之后通过 JSON 消息将信息传递给服务端，具体内容如下。

{“type”:”8”,
”parameter”:{”curpath”:”x”,”filename”:”xxx”,”identification”:”x”,”size”:”x”}}

“type”为“8”表示上传文件请求。服务端接收到该消息后将参数传递给文件上传模块。该模块实际上并不参与上传文件的实际工作，而负责更新目录文件，即在目录文件中“上传”文件。同时上传模块还将修改用户的已使用配额。上传模块将新上传文件的目录项添加至目录文件末尾后返回上传成功。

终端接口接收到服务端返回的上传成功的消息后在目录链表的第三个节点处插入新文件的目录项节点。

4.4.9 下载文件和关闭连接

同样的，FastDFS 已具有下载文件的接口，但需与目录结构相结合做出相应修改。下载文件接口定义如下。

int downloadFile(int socket, char *filename)

接口的返回值类型为 int 型，参数 socket 为调用 login 接口成功后返回的套接字，参数 filename 为欲下载文件的文件名。

接口的返回值有 0 和 1。返回值 0 表示下载成功，返回值 1 表示下载失败。

在下载前首先通过目录文件查询欲下载文件对应的文件标识符，之后使用FastDFS 本身具有的下载接口下载文件。

关闭连接，切实完成 TCP 断开连接的“4 次挥手”过程可有效节约服务端资源。关闭连接接口定义如下。

int closeDirServer(int socket)

接口的返回值类型为 int 型，参数 socket 为调用 connectDirserver 接口或 login 接口成功后返回的套接字文件描述符。

接口的返回值有 0 和 1。返回值 0 表示关闭成功，返回值 1 表示关闭失败。

接口只完成一项工作，即关闭套接字。

这两个接口并不需要与服务端通信，实现比较简单。

下载文件接口首先在目录链表中查询下载文件的目录项节点，获得该文件的文件标识符。之后调用 FastDFS 提供的下载文件接口完成下载。

关闭连接接口只需执行关闭套接字的操作即可。

4.5 安全管理控制实现

安全管理控制主要包含两部分，一是用户认证，二是接口控制。

用户认证过程采用对称加密，密码的传输过程使用 AES 加密。数据库存储使用 SHA-2 散列后的密码。

接口控制将接口与用户认证相结合，防止用户对接口的非法调用。终端在与服务端建立连接后，服务线程会记录本次连接的认证情况，这通过一个哈希表完成。

哈希表的键为本次连接的套接字描述符，而值为认证后的用户 ID，初始为“-1”。终端与服务端的连接建立，服务线程便会在哈希表中插入这样一条记录。通过用户认证后服务进程将值改为认证用户的用户 ID。关闭连接后删除该条记录或将值改为“-1”。

当终端通过调用不同接口与服务端通信请求不同功能时，服务进程首先会查询哈希表确认该连接是否已经通过用户认证。如果查询的结果是“-1”则表明该连接还未通过用户认证，这时通过该连接只能使用有限的功能（只能进行用户注册或用户登录）。如果查询的结果不为“-1”则表明该连接已通过用户认证，这时通过该连接可以使用全部功能（除了用户注册与用户登录）。

同时，为防止认证用户非法访问其他用户的内容，对于涉及目录文件的操作

在执行前都必须进行检查，检查该用户是否有权操作该目录。由于本次连接认证用户的 ID 可以通过哈希表查询，且用户目录的存储与用户 ID 相关联，因此可以方便的确认该用户是否有权操作请求的目录。

通过以上两个措施可以实现比较完善的数据安全与隐私保护。

4.6 本章小结

本章从系统环境、目录结构、接口和数据安全四个方面描述了整个目录文件系统的详细设计与实现。同时对于设计与实现过程遇到问题予以分析和解决，对于经过认真考虑和优化的过程都有比较详尽的描述。

第五章 系统测试

本章将对基于 FastDFS 的目录文件系统进行测试。由于测试条件的限制，测试的内容以功能测试为主。

5.1 测试方法与指标

由于测试以目录文件系统的功能为主，主要涉及众多公共 API 接口，因此测试将通过编写测试程序进行。测试程序将调用所有目录文件系统设计中的公共接口，测试完整的用户操作。另外对于系统可靠性、数据安全等方面的测试将通过模拟进行。模拟的手段包括中断服务器网络、修改接口调用参数等。

测试通过的指标即符合所有设计预期。

5.2 测试环境

5.2.1 测试环境配置

参照第四章的系统环境设置，测试将使用四台服务器与一台终端进行。测试环境的网络拓扑如图 5-1 所示。

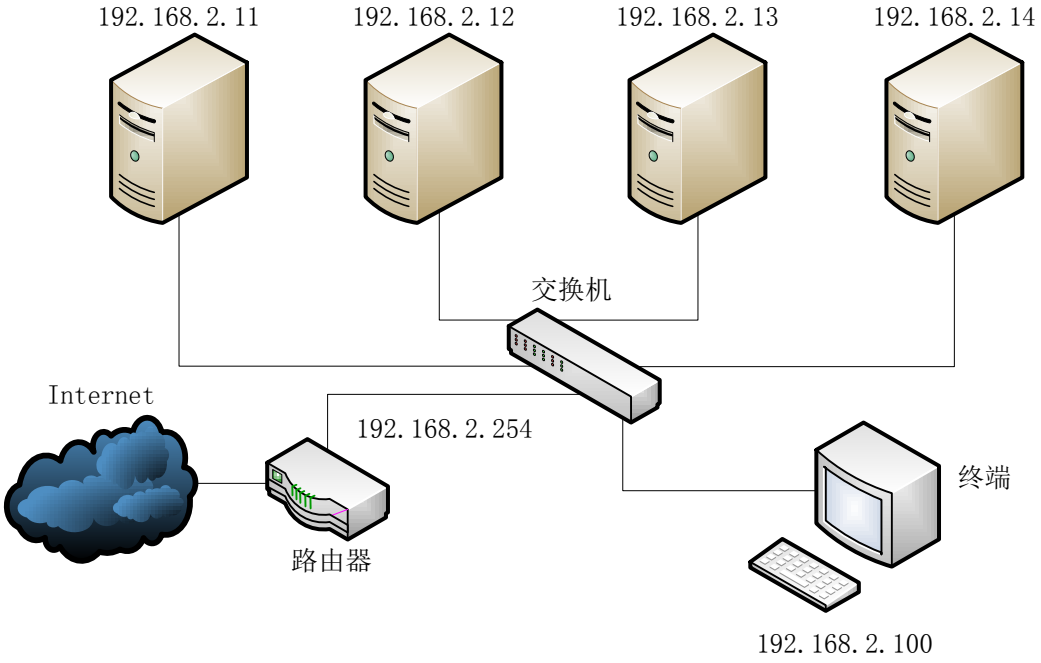

图 5-1 测试环境网络拓扑图

四台服务器、终端及路由器通过交换机相连，组成网段为 192.168.2.0/24 的局域网。服务器的配置如表 5-1 所示。

表 5-1 测试服务器配置

服务器 IP	操作系统	配置	角色
192.168.2.11	ubuntu server 14.04 LTS amd64	CPU 型号：Intel(R) Xeon(R) CPU E5-2403 0 @ 1.80GHz 硬盘大小：3T 内存大小：32G	Tracker Client 目录节点 数据库管理节点 数据库访问节点
192.168.2.12			
192.168.2.13			Storage 数据库存储节点
192.168.2.14			

5.2.2 测试环境搭建

5.2.2.1 安装 FastDFS

四台服务器中都需要安装 FastDFS，通过网站下载 FastDFS 的最新源码包，本次测试使用 FastDFSv5.05 版本。

在开始安装 FastDFS 之前首先需要安装 libfastcommon 库，通过 git 工具下载最新源码包编译安装。

```
# git clone https://github.com/happyfish100/libfastcommon.git
# cd libfastcommon/
# ./make.sh
# ./make.sh install
# cp /usr/lib64/libfastcommon.so /usr/lib
# cp /usr/lib64/libfdfsclient.so /usr/lib
```

libfastcommon 库安装完成后即可开始编译安装 FastDFS，执行以下命令。

```
# tar -xvf FastDFS_v5.05.tar.gz
# cd FastDFS
# ./make.sh
```

```
# ./make.sh install
```

编译安装过程中如未报错则表明 FastDFS 已安装成功。

5.2.2.2 配置启动 FastDFS

四台服务器在 FastDFS 中担任的角色不同，因此配置启动的服务也不同。在服务器 192.168.2.1 和 192.168.2.2 上配置 Tracker 和 Client，在服务器 192.168.2.3 和 192.168.2.4 上配置 Storage。

配置 Tracker 需要修改/etc/fdfs/tracker.conf 配置文件，其中除 base_path 参数外都可保持默认。base_path 指明 Tracker 服务存储数据和日志的路径，这里设置为/var/fastdfs_tracker。设置完之后启动 Tracker 服务。

```
# fdfs_trackerd /etc/fdfs/tracker.conf restart
```

配置 Storage 需要修改/etc/fdfs/storage.conf 配置文件，其中需要配置的参数有 base_path、store_path0 和 tracker_server，其他参数可保持默认。

base_path 的含义与 Tracker 配置文件中的 base_path 含义相同，指明 Storage 服务存储数据和日志的路径，这里设置为/var/fastdfs_storage。store_path0 指明上传文件的存储路径，可以设定多个，之后以 store_path1 等命名。这里设置为/var/fastdfs_storage。tracker_server 指明 Tracker 服务器的 IP 及端口，可包含多个。这里这是为 192.168.2.11:22122 和 192.168.2.12:22122。配置玩之后启动 Storage 服务。

```
# fdfs_storaged /etc/fdfs/storage.conf restart
```

配置 Client 需要修改/etc/fdfs/client.conf 配置文件，其中需要配置的参数有 base_path 和 tracker_server，其他参数可保持默认。base_path 和 tracker_server 的含义与配置 Storage 时雷同，base_path 设置为/var/fastdfs_client，tracker_server 设置为 192.168.2.11:22122 和 192.168.2.12:22122。

至此 FastDFS 以全部设置启动完毕，之后可通过 FastDFS 自带测试程序测试 FastDFS 是否工作良好。以上传文件为例，执行下列命令。

```
# fdfs_test /etc/fdfs/client.conf upload /tmp/test
```

若得到如图 5-2 所示的结果则说明 FastDFS 以正常工作。

```
tracker_query_storage_store_list_without_group:
        server 1. group_name=, ip_addr=192.168.2.13, port=23000
        server 2. group_name=, ip_addr=192.168.2.14, port=23000

group_name=group1, ip_addr=192.168.2.13, port=23000
storage_upload_by_filename
group_name=group1, remote_filename=M00/00/00/wKgCDVUOHx-AY6L5AAAABelE2rQ0417449
source ip address: 192.168.2.13
file timestamp=2015-03-21 18:47:11
file size=5
file crc32=3913603764
example file url: http://192.168.2.13/group1/M00/00/00/wKgCDVUOHx-AY6L5AAAABelE2rQ0417449
storage_upload_slave_by_filename
group_name=group1, remote_filename=M00/00/00/wKgCDVUOHx-AY6L5AAAABelE2rQ0417449_big
source ip address: 192.168.2.13
file timestamp=2015-03-21 18:47:11
file size=5
file crc32=3913603764
example file url: http://192.168.2.13/group1/M00/00/00/wKgCDVUOHx-AY6L5AAAABelE2rQ0417449_big
root@ubuntu:~#
```

图 5-2 FastDFS 上传结果

5.2.2.3 配置 rsync 与 inotify

服务器 192.168.2.11 和 192.168.2.12 都需要配置 rsync 服务与 inotify 服务，配置方法可参考第四章目录文件同步实现，在此不再赘述。

5.2.2.4 配置启动数据库

平台使用的数据库版本为 mysql-cluster-gpl-7.2.7。安装测试环境配置中划分的角色，修改配置文件/var/lib/mysql-cluster/config.ini 和/etc/my.cnf。四台服务器的配置文件须保持一致。之后先在 192.168.2.11 和 192.168.2.12 上启动管理节点，再在 192.168.2.13 和 192.168.2.14 上启动数据节点，最后在 192.168.2.11 和 192.168.2.12 上启动访问节点。

5.3 测试内容与结果分析

5.3.1 功能测试

对公共接口的测试通过如图 5-3 所示的测试程序进行。测试程序包含用户注册、用户登录、新建目录、删除目录、重命名、上传文件、下载文件和删除文件的功能。另外，建立连接并非显式功能，将其包含在用户注册和用户登录中测试。打开目录则由鼠标左键双击事件触发。所有测试程序提供的功能都与各个公共接口提供的功能相对应。

测试的过程将遵照一个新用户使用目录文件系统的过程测试。首先是注册用户。注册用户时即可对设计中考虑的用户名检测和密码检测功能进行测试。注册成功后进行用户登录。用户登录时可以测试用户是否注册成功、非注册用户是否

能够登录以及密码错误是否能够认证通过。

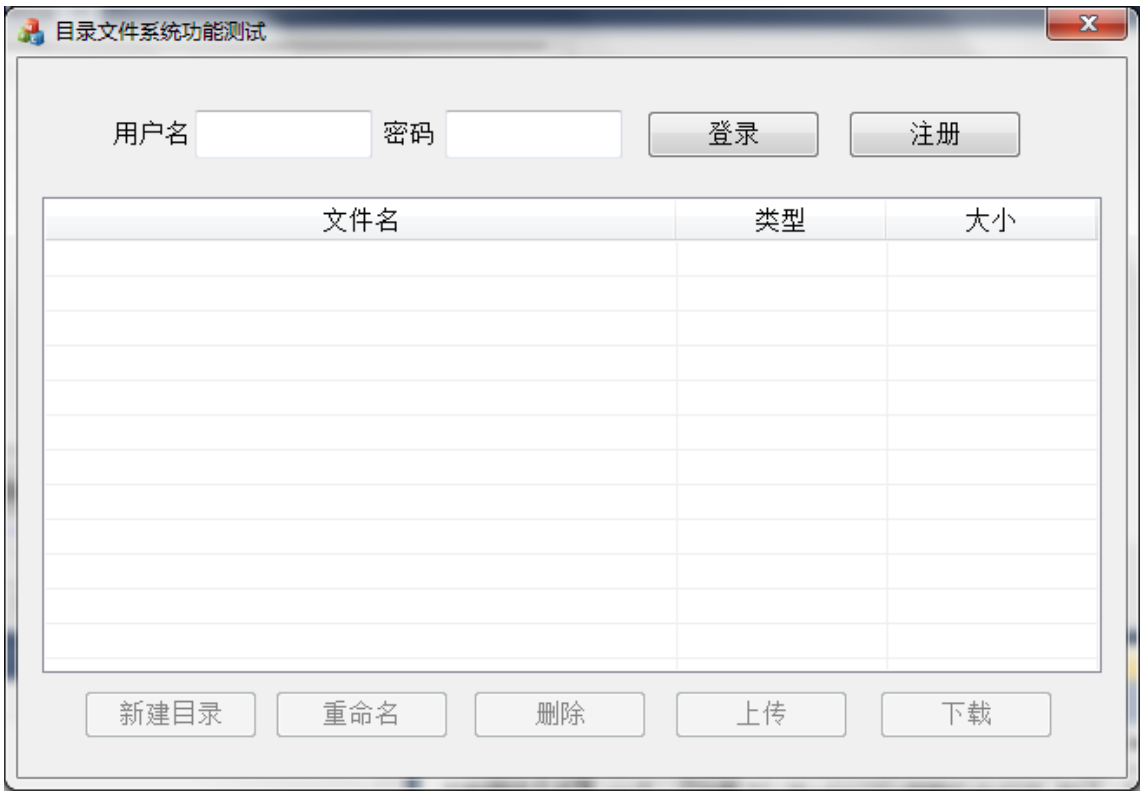

图 5-3 目录文件系统功能测试程序

登录成功之后测试内容无固定顺序。新建目录可测试新建目录及子目录，也可测试在同级下新建同名目录。选中文件或目录后可测试删除或重命名功能。重命名时可测试重命名为与同级下其他文件同名的文件名。选中文件后测试下载。任何时候都可测试上传。

对于用户可接触到的功能均可由上述功能测试程序进行测试，而对于数据安全等对于用户透明的功能则需要重新编写测试程序。

数据安全与隐私保护功能中的用户认证部分已在用户注册与用户登录过程中附带测试，用户注册及用户登录功能成功通过则表明这部分没有问题。测试数据安全与隐私保护功能则只需为接口控制部分重新编写测试代码。测试代码主要分两种情况进行测试，第一种情况是调用完建立连接接口后绕过用户认证接口直接调用创建目录接口等；第二种情况是调用完用户认证接口后，通过伪造目录链表访问其他用户的目录。

通过以上测试的测试结果可以判断，目录文件系统实现的各项的功能均能正确工作。

5.3.2 性能测试

性能测试部分主要针对目录服务的可靠访问进行测试，即针对目录文件的同步备份进行测试。在功能测试结束时进入服务器 192.168.2.11 和 192.168.2.12 比较两台服务器的/var/userdir 目录中的内容是否相同。

```
root@ubuntu:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e6:d2:b5
          inet addr:192.168.2.11  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fee6:d2b5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:816 errors:0 dropped:0 overruns:0 frame:0
          TX packets:731 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:79531 (79.5 KB)  TX bytes:89373 (89.3 KB)

root@ubuntu:~# ls -R /var/userdir/
/var/userdir/:
1  2

/var/userdir/1:
1  test1

/var/userdir/1/test1:
test1  test2

/var/userdir/1/test1/test2:
test2

/var/userdir/2:
2  test3

/var/userdir/2/test3:
test3
root@ubuntu:~#
```

图 5-4 192.168.2.11 上/var/userdir 目录内容

```
root@ubuntu:/tmp# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:5e:eb:d9
          inet addr:192.168.2.12  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe5e:ebd9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:838 errors:0 dropped:0 overruns:0 frame:0
          TX packets:745 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:80313 (80.3 KB)  TX bytes:98553 (98.5 KB)

root@ubuntu:/tmp# ls -R /var/userdir/
/var/userdir/:
1  2

/var/userdir/1:
1  test1

/var/userdir/1/test1:
test1  test2

/var/userdir/1/test1/test2:
test2

/var/userdir/2:
2  test3

/var/userdir/2/test3:
test3
root@ubuntu:/tmp# scp -qr 192.168.2.11:/var/userdir .
root@192.168.2.11's password:
root@ubuntu:/tmp# diff -Nrq userdir/ /var/userdir/
root@ubuntu:/tmp#
```

图 5-5 192.168.2.11 上/var/userdir 目录内容

目录文件内容的对比通过将服务器 192.168.2.11 上的/var/userdir 目录拷贝至

192.168.2.12 上，并通过 diff 命令递归比对两个目录下的文件进行。通过图 5-4 和图 5-5 我们可以很清楚的看到，服务器 192.168.2.11 和 192.168.2.12 上的目录文件没有任何不同。

5.4 本章小结

本章主要描述了对目录文件系统的测试过程。测试先确定了测试方法与指标，之后从测试环境的配置和搭建入手，最后完成整个测试内容并分析结果得出结论。通过以上测试验证了目录文件系统在功能上的正确性与完备性以及目录服务的可靠性。

第六章 全文总结与展望

6.1 全文总结

云存储是在云计算的概念上延伸和发展出来的一个新的概念，是一种新兴的网络存储技术。云存储利用集群应用和分布式文件系统等软件，将网络中大量类型不同、容量不同的存储设备连结在一起，作为一个整体协同工作并以一个统一的方式对外提供数据存储和访问服务。当云计算系统中配置了大量存储设备，并且将系统运算和处理的重点放到数据的存储和管理上而不是数据的计算上，那么该云计算系统就变为一个云存储系统。也可以说云存储是一种新的存储方案，它将存储资源统一集中放至云端，这样无论用户身处何地都可以随时通过网络访问云上的数据资源。

本文在实验室现有云平台的基础上，探索一种新的解决方案以满足平台对云存储的要求。新解决方案将在FastDFS的基础上增加新的功能，设计基于FastDFS的目录文件系统。

文章从目录文件系统的需求分析入手，阐述了目录文件系统的功能需求以及性能需求。网络接入、用户注册与认证、权限控制、文件上传与下载、目录、公共 API 接口和数据安全与隐私保护都是目录文件系统作为云存储解决方案所应具备的基本功。而对于目录文件系统的性能需求则主要集中在网络延时、并发访问和可靠访问上，其中可靠访问更是本次设计的重点。之后根据各种需求阐述了目录文件系统的概要设计、总体架构、模块设计、并发访问与可靠访问设计以及数据安全与隐私保护设计几个部分，对整个目录文件系统进行了总体设计。

在阐述了目录文件系统的总体设计之后，文章紧接着阐述了目录文件系统的详细设计与实现。详细设计主要围绕目录结构、公共接口、数据安全与隐私保护三个部分展开。目录结构的详细设计又分为目录文件的结构设计、存储设计和同步设计三个部分。根据目录文件系统总体设计中的模块设计，将公共接口具体细分为建立连接、用户注册、用户登录、创建目录、打开目录、重命名、删除目录、删除文件、上传文件、下载文件和关闭连接等十一个接口。各个接口的详细设计则与其实现穿插进行。

文章最后对目录文件系统的测试过程进行了阐述。测试先确定了测试方法与指标，之后从测试环境的配置和搭建入手，最后完成整个测试内容并分析结果得出结论。通过测试验证了目录文件系统在功能上的正确性与完备性以及目录服务的可靠性。

6.2 后续工作展望

基于 FastDFS 的目录文件系统尽管已能够满足作为云存储解决方案的基本需求，但仍有可以改进的地方。

首先是功能方面，可以提供更多更丰富的公共接口。只是提供十几个基本功能的公共接口将很难满足用户日益变化的使用需求，同时也将增大终端程序开发者的开发难度。更多更丰富的公共接口可以使目录文件系统对外提供更多更丰富的功能与服务。另外，随着目录文件系统功能的丰富，权限控制也应该更加细化。数据安全与隐私保护被人们重视的程度越来越高，这方面仍有许多工作可以做。

其次是性能方面，可以进一步优化。目前目录文件系统主要应用于私有云中，但发展公共云是趋势，在将目录文件系统应用于公共云时，其应用情景将与私有云时有很大不同。更多的用户数量带来更多的并发数量，各种性能需求指标也会相应增加，只是简单处理显然是不能满足需求的。

致　谢

在攻读硕士学位期间，首先衷心感谢我的导师戴元顺教授，在他的亲切关心和悉心指导下我才能顺利完成本篇学位论文。他严谨的治学精神与精益求精的工作态度一直激励和鼓舞着我。戴元顺教授不仅给予我学习上的教导，也给予我生活上的关心。在此谨向戴元顺教授致以最诚挚的谢意与最崇高的敬意。

本篇学位论文的完成也离不开其他各位老师和同学的关心和帮助。感谢向艳萍教授在论文写作方面的各种指导，感谢余盛季老师在实验室项目中的指导和鼓励，感谢孙鹏博士在开题时的帮助。还要感谢刘明惠、程夏衍等同学在生活及科研项目中的帮助。

最后要特别感谢我的父母，他们在生活上的关心和帮助为我在前进的道路上保驾护航，我获得的每一次成功都有他们背后的支持。

参考文献

[1] T. Dillon, C. Wu, E. Chang. Cloud Computing: Issues and Challenges[J]. Advanced Information Networking and Applications, IEEE International Conference on, 2010, 4(4): 27-33.

[2] 李乔，郑啸．云计算研究现状综述[J]．计算机科学，2011, 38(4): 32-37

[3] 张建勋，古志民，郑超．云计算研究进展综述[J]．计算机应用研究，2010, 27(2): 429-433

[4] 唐箭．云存储系统的分析与应用研究[J]．电脑知识与技术，2009, 5(20)

[5] 周可，王桦，李春花．云存储技术及其应用[J]．中兴通讯技术，2010, 16(4): 24-27

[6] B. Hayes. Cloud Computing [J]. Communications of the ACM, 2008, 51(7): 9-11

[7] G. Lin, G. Dasmalchi, J. Zhu. Cloud computing and IT as a service: opportunities and challenges[C]. Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008: 5-5

[8] J. Namjoshi, A. Gupte. Service Oriented Architecture for Cloud Based Travel Reservation Software as a Service[C]. IEEE Sixth International Conference on Cloud Computing. IEEE, 2009: 147-150

[9] P. A. Laplante, J. Zhang, J. Voas. What's in a Name? Distinguishing between SaaS and SOA[J]. It Professional, 2008, 10(3): 46-50

[10] 陈康，郑纬民．云计算：系统实例与研究现状[J]．软件学报，2009, 20(5): 1337-1348

[11] Sophia．云存储及其分布式系统[EB/OL]．比特网，http://cloud.chinabyte.com/tech/61/12559061.html, 2013

[12] 蒋寅．Linux 下的文件系统发展[J]．福建电脑，2002, (6): 13-14

[13] M. Satyanarayanan. A Survey of Distributed File Systems[J]. Annual Review of Computer Science, 1995, 4(1): 73-104

[14] F. Corbató, V. Vyssotsky. Introduction and Overview of the Multics System[J]. Proc. AFIPS 1965 FJCC, 1965

[15] 高楼居士．分布式文件系统[EB/OL]．百度百科，http://baike.baidu.com/view/771589.htm, 2014

[16] duguguiyu. 分布式基础学习[EB/OL]．博客园，http://www.cnblogs.com/duguguiyu/archive/2009/02/22/1396034.html, 2009

[17] 黄华，杨德志，张建刚．分布式文件系统[J]．信息技术快报，2004

[18] S. Ghemawat, H. Gobioff, S. T. Leung. The Google file system[C]. ACM SIGOPS operating systems review. ACM, 2003, 37(5): 29-43

[19] U. Hölzle, Google. Web Search for a Planet: The Google Cluster Architecture[J]. IEEE Micro, 2003: 22-28

[20] S. Quinlan. The Google file system[J]. Conference on High Speed Computing, 2004, 37: 29-43

[21] S. G. Howard, H. Gobioff, S. Leung. The Google File System[J]. Acm Sigops Operating Systems Review, 2004: 29-43

[22] K. Shvachko, H. Kuang, S. Radia, et al. The hadoop distributed file system[J]. IEEE Symposium on Mass Storage Systems & Technologies, 2010, (11): 1-10

[23] G. Attebury, A. Baranovski, K. Bloom, et al. Hadoop distributed file system for the Grid[C]. IEEE Nuclear Science Symposium Conference Record. IEEE, 2009: 1056-1061

[24] D. Borthakur. The hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, (11): 1-10

[25] 郝树魁．Hadoop HDFS 和 MapReduce 架构浅析[J]．邮电设计技术，2012, (7): 37-42

[26] chuyu. TFS 简介[EB/OL]. TaoCode, http://code.taobao.org/p/tfs/wiki/intro/, 2011

[27] 小緈馥．TFS[EB/OL]．百度百科，http://baike.baidu.com/view/1030880.htm, 2014

[28] E. Thereska, D. S. Gunawardena, J. W. Scott, et al. Distributed File System: US, US20120254116 A1[P]. 2011

[29] T. Nakamura, K. Koyama. Distributed file system: US, US7613786 B2[P]. 2003

[30] A. Whitney, Y. Neeman, S. Koneru, et al. Distributed file system[D]. EP, 1994

[31] Z. H. Ying, H. K. Gao, R. H. Huang. The Distributed File System[J]. Computer Engineering & Science, 1995

[32] 余庆．分布式文件系统 FastDFS 架构剖析[J]．程序员，2010, (11): 63-65

[33] 门维江．树型目录结构在软件开发中的应用[J]．甘肃教育学院学报：自然科学版，2001, 15(2): 24-26

[34] 陈晓玲．浅谈 Linux 目录[J]．湖南科技学院学报，2006, 27(5): 145-147

[35] 王侠，高胜哲．Linux 的文件系统[J]．辽宁税务高等专科学校学报，2005, 17(2): 40-41

[36] R. Love. Kernel korner: intro to inotify[J]. Linux Journal, 2005, 2005(139): 8

[37] 燚杨．inotify -- Linux 2.6 内核中的文件系统变化通知机制[EB/OL]. IBM developerWorks, http://www.ibm.com/developerworks/cn/linux/l-inotifynew/, 2005

[38] A. TRIDGELL, P. MACKERRAS. The rsync Algorithm[J]. Australian National University, 1996

[39] D. Rasch, R. Burns. In-place rsync: file synchronization for mobile and wireless devices[J]. In Proc of the Usenix Technical Conference, 2003

[40] A. Ghobadi, C. Eswaran, C. K. Ho, et al. Automated tools for manipulating files in a distributed environment with RSYNC[C]. International Conference on Advanced Communication Technology. IEEE Press, 2010: 1383-1388

攻读硕士学位期间取得的成果

[1] 周子涵. 参与江苏省镇江市中山路小学教育云系统项目，2012-2013

[2] 周子涵. 参与广东省海事局办公云系统项目，2014-2015