# 哈尔滨理工大学
# 计算机科学与技术学院
## 《云计算及虚拟化技术》
## 课程设计总结报告

| | | |
|---|---|---|
| 题　　目 | ： | 基于 Sringboot 框架平台的 Magic 高可用分布式网盘的设计与实现 |
| 班　　级 | ： | 大数据 22-1 |
| 专　　业 | ： | 数据科学与大数据技术 |
| 姓　　名 | ： | 郭子铭 |
| 学　　号 | ： | 2204050108 |
| 指导教师 | ： | 李金刚 |
| 日　　期 | ： | 2025.1.3 |

# 目录

# 一、需求分析

随着互联网的发展，云存储技术已经成为现代互联网应用中不可或缺的一部分。云盘服务不仅为用户提供了便捷的文件存储方式，还支持跨平台、多终端的数据访问和共享。近年来，随着智能手机、平板电脑等设备的普及，用户对文件存储和管理的需求也日益增加。各大互联网公司纷纷推出了各类云盘服务，如百度云盘、Google Drive、Dropbox 等，它们提供了在线存储、文件分享、同步等基本功能，满足了大多数用户的需求。

然而，尽管市场上已有多个云盘产品，但大部分云盘服务仍然存在着文件上传速度慢、文件管理复杂、文件分享功能不够灵活等问题。特别是在面对大文件上传时，许多用户体验较差，上传过程中容易出现断点、超时等问题。此外，文件在线预览和管理也常常缺乏统一的标准和技术支持，导致用户无法轻松地浏览各种格式的文件。

因此，开发一个功能全面、性能高效的云盘系统，成为了当下的迫切需求。本项目旨在开发一个仿百度云盘的网盘系统，提供用户注册、登录、文件上传、分片上传、秒传、文件分享、文件预览等功能，以期解决当前云盘服务中存在的问题，并提升用户体验。

Minio 是一个基于 Apache License v2.0 开源协议的对象存储服务，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。MinIO 是一个非常轻量的服务,可以很简单的和其他应用的结合，类似 NodeJS, Redis 或者 MySQL。对于中小型企业，Minio 是个不错的选择，麻雀虽小，五脏俱全。当然 Minio 除了直接作为对象存储使用，还可以作为云上对象存储服务的网关。图 1-1 为 Minio 设计模式示意图。

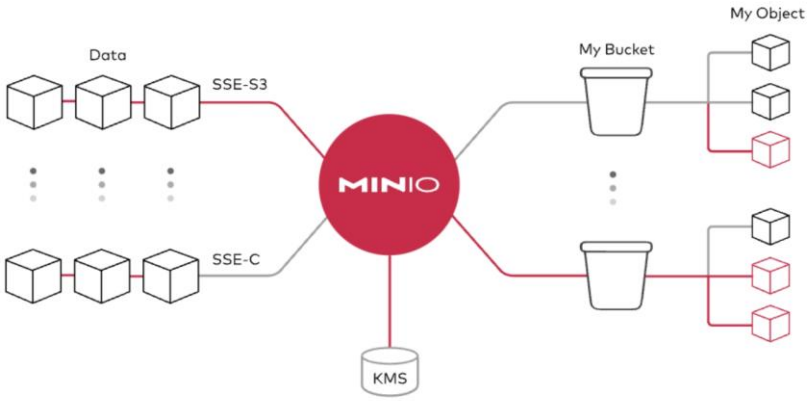


**图 1-1** Minio 模式图

随着互联网的普及，用户对网盘的需求日益多样化和个性化。除了基本的文件存储和分享功能之外，用户还期望网盘具备更高的访问速度、更强的数据安全性以及更便捷的协作功能。此外，伴随着移动互联网的发展，用户更加倾向于通过多终端设备访问和管理自己的数据，这对网盘的兼容性和跨平台支持提出了更高的要求。在这种背景下，基于 Springboot 技术的高可用分布式网盘应运而生，旨在利用分布式存储和处理技术，全面提升网盘的性能、可靠性和用户体验。

基于 Springboot 的高可用分布式网盘需要具备多方面的功能以满足用户需求。通过 MinIO 作为核心分布式对象存储引擎，结合 Spring Boot 框架开发的分布式网盘系统，可以高效地满足用户对大规模数据存储和管理的需求。系统利用 MinIO 的对象存储能力，将用户上传的文件分片并存储在多个节点上，不仅提升了存储效率，还显著提高了数据访问速度。同时，该系统具备文件分类管理、标签标注和元数据管理功能，使用户能够快速查找并高效管理文件，从而显著优化用户体验。

在文件上传和下载方面，通过 MinIO 的分布式架构，文件上传和下载过程中允许采用多节点并行传输机制，显著提高数据传输效率。系统支持断点续传功能，确保用户在网络环境不稳定的情况下能够顺利完成上传或下载任务。还支持批量文件的上传与下载，便于用户高效处理多文件操作。通过 MinIO 内置的哈希校验功能，系统在文件传输完成后自动检查文件完整性，确保数据准确可靠。

综合以上功能，基于 Springboot 的高可用分布式网盘能够为用户提供一个高效、安全、可靠且便捷的云存储和协作平台。基于上述需求分析设计的系统的业务流程图如图 1-2 所示。

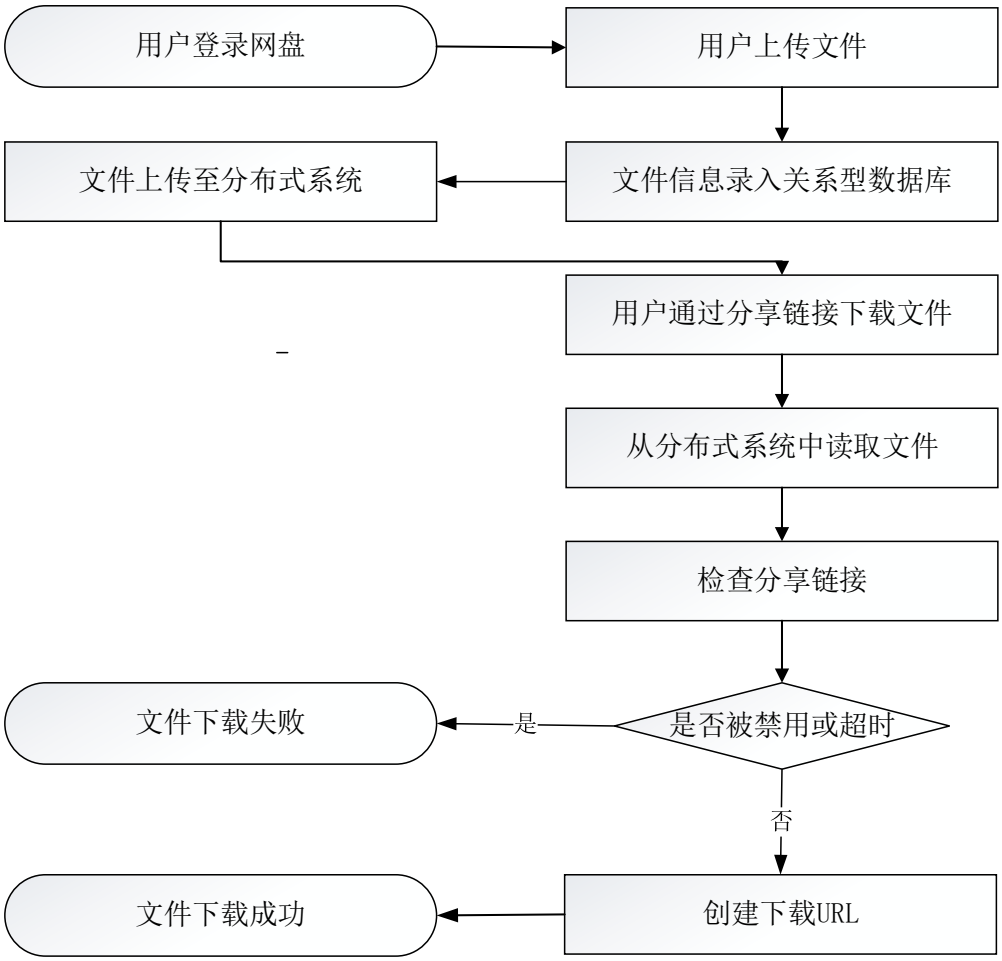


**图 1-2** 系统业务流程图

整个系统的用例模型设计如图 1-3 所示，其展示了所有对象所可能进行的功能和操作。该用例图详细描绘了文件管理系统中管理员和普通用户的功能交互。管理员在系统中拥有广泛的管理权限，能够执行包括但不限于查看用户信息、分配存储空间、封禁用户账号等关键操作，确保系统的健康运行和数据的安全管理。

普通用户则需要享有一系列基础和高级功能。在基础用户功能方面，用户可以注册新账户、登录、注销以及更新个人资料，如头像和密码，保障账户的安全性和个性化。在文件管理方面，用户能够执行上传、浏览、还原、下载、删除和彻底删除文件等操作，以及修改文件信息，满足日常文件处理的需求。

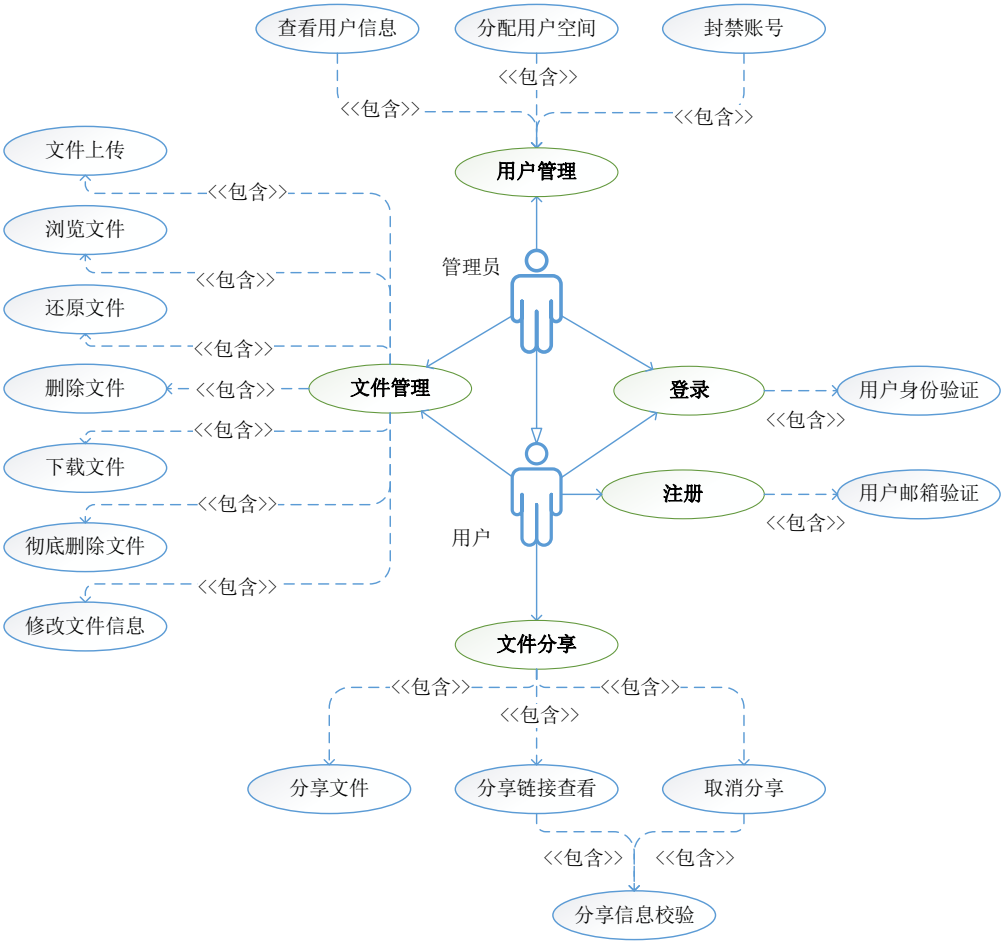


**图 1-3** 系统用例图

此外，系统提供文件分享功能，允许用户通过生成分享链接的方式，将文件共享给其他用户。接收方可以通过查看分享状态、输入正确的密码来访问和下载文件，这增加了文件分享的安全性和可控性。用户还可以管理自己的分享记录，包括取消分享，以控制文件的访问权限。系统还具备回收站功能，用户可以将不再需要的文件暂时移至回收站，之后可以选择恢复或永久删除。为防止重要文件的误删，系统设定了回收站文件的自动清理机制，超过七天未处理的文件将被自动永久删除，以优化存储资源的使用。上述就是整个系统中不同的角色所能够行使的功能。

# 二、系统设计

## （一）系统中的数据定义

在本系统中，整个系统的数据分布在关系数据库 MySQL 与 MinIO 对象存储中。其中关系型数据库中的数据用于支持后端操作。整个关系型数据库中的 ER 图如图 2-1 所示。

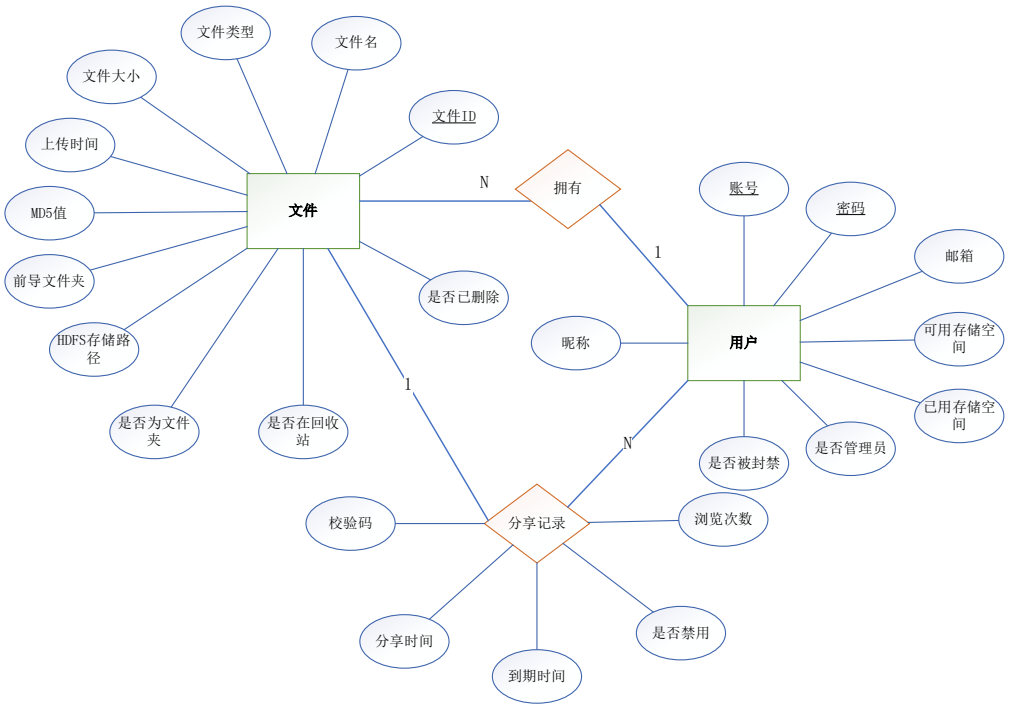


**图 2-1** 关系型数据库中的实体-联系图

整个关系型数据库中包括三个主要实体：文件、用户和分享信息。文件实体主要包括文件 Id、用户 Id、文件 MD5 值、文件父 Id、文件大小、文件名、文件路径、上传时间、修改时间、文件夹类型、文件仓库、文件类型、文件状态、恢复时间、是否删除等属性；分享属性包括分享 Id、文件 Id、用户 Id、分享时间、有效时间、分享码、分享数量等；用户实体包括用户 Id、昵称、QQId、QQ 头像、密码、注册时间、登录时间、状态、已使用空间、总空间。用户与文件通过“拥有”关系能够直接相连，表示用户可以拥有多个文件。用户与文件还通过分享链接相连。分享链接是一种特殊的关系，它表示用户将文件分享给他人。每个用户

可以分享多个信息；而文件与分享信息之间的关系则表明每个分享信息对应一个具体的文件。

更具体的数据表结构如表 2-1、2-2、2-3、2-4 所示。这四张表说明了数据库的主外键关系，以及每个属性的数据类型。

**表 2-1** 文件信息表

| 序号 | 数据项名称 | 数据类型 | 是否关键字 | 是否可以为空 |
|---|---|---|---|---|
| 1 | file_id | varchar(10) | 是 | 否 |
| 2 | user_id | varchar(10) | 是 | 否 |
| 3 | file_md5 | varchar(32) | 否 | 是 |
| 4 | file_pid | varchar(10) | 否 | 是 |
| 5 | file_size | bigint(20) | 否 | 是 |
| 6 | file_name | varchar(200) | 否 | 是 |
| 7 | file_cover | varchar(100) | 否 | 是 |
| 8 | file_path | varchar(100) | 否 | 是 |
| 9 | create_time | datetime | 否 | 是 |
| 10 | last_update_time | datetime | 否 | 是 |
| 11 | folder_type | tinyint(1) | 否 | 是 |
| 12 | file_category | tinyint(1) | 否 | 是 |
| 13 | file_type | tinyint(1) | 否 | 是 |
| 14 | status | tinyint(1) | 否 | 是 |
| 15 | recovery_time | datetime | 否 | 是 |
| 16 | del_flag | tinyint(1) | 否 | 是 |

**表 2-2** 用户信息表

| 序号 | 数据项名称 | 数据类型 | 是否关键字 | 是否可以为空 |
|---|---|---|---|---|
| 1 | user_id | varchar(10) | 是 | 否 |
| 2 | nick_name | varchar(20) | 否 | 是 |
| 3 | email | varchar(150) | 否 | 是 |
| 4 | qq_open_id | varchar(35) | 否 | 是 |

| | | | | |
|---|---|---|---|---|
| 5 | qq_avatar | varchar(150) | 否 | 是 |
| 6 | password | varchar(50) | 否 | 是 |
| 7 | join_time | datetime | 否 | 是 |
| 8 | last_login_time | datetime | 否 | 是 |
| 9 | status | tinyint(4) | 否 | 是 |
| 10 | use_space | bigint(20) | 否 | 是 |
| 11 | total_space | bigint(20) | 否 | 是 |

**表 2-3** 分享链接表

| 序号 | 数据项名称 | 数据类型 | 是否关键字 | 是否可以为空 |
|---|---|---|---|---|
| 1 | share_id | varchar(20) | 是 | 否 |
| 2 | file_id | varchar(10) | 否 | 是 |
| 3 | user_id | varchar(10) | 否 | 是 |
| 4 | valid_type | tinyint(1) | 否 | 是 |
| 5 | expire_time | datetime | 否 | 是 |
| 6 | share_time | datetime | 否 | 是 |
| 7 | code | varchar(5) | 否 | 是 |
| 8 | show_count | int(11) | 否 | 是 |

**表 2-4** 邮箱验证码表

| 序号 | 数据项名称 | 数据类型 | 是否关键字 | 是否可以为空 |
|---|---|---|---|---|
| 1 | email | varchar(150) | 是 | 否 |
| 2 | code | varchar(5) | 是 | 否 |
| 3 | create_time | datetime | 否 | 是 |
| 4 | status | tinyint(1) | 否 | 是 |

## （二）系统的概要设计

整个系统的对象模型设计如图 2-2 所示。这张类与对象图中展示了系统重要的控制类，以及由数据模型映射成的实体类。

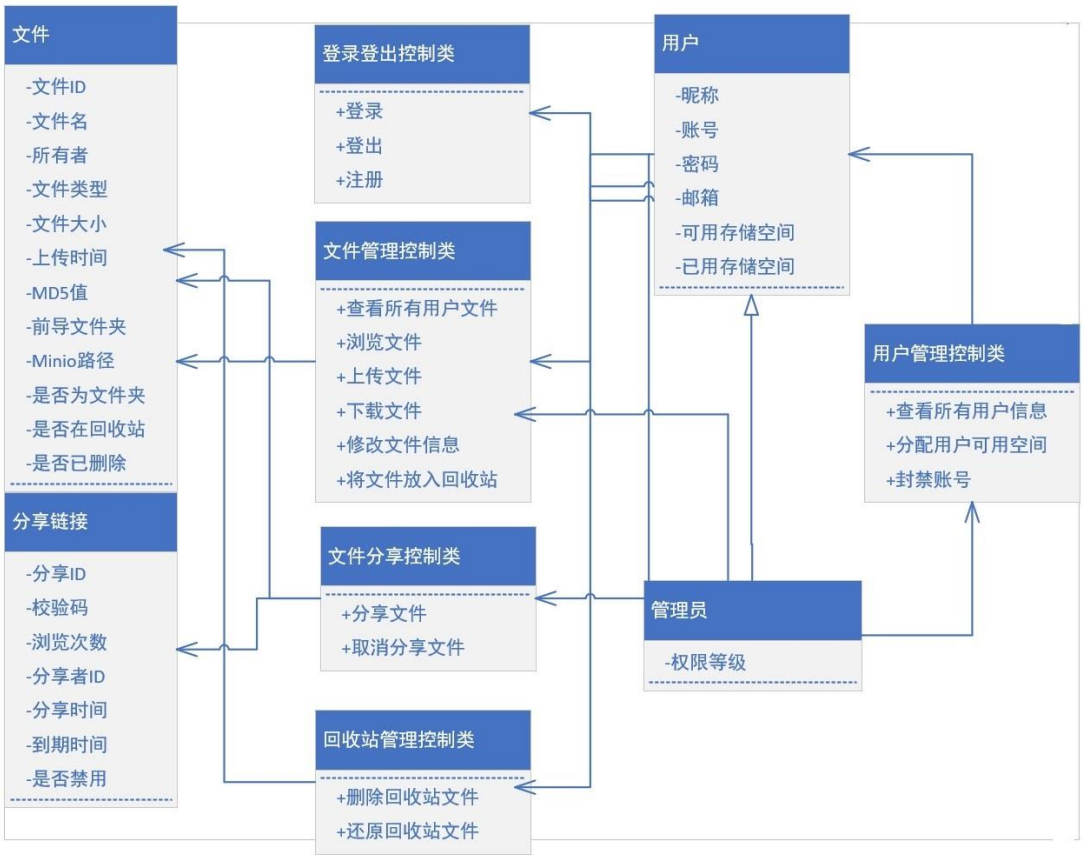


**图 2-2** 系统类与对象图

主整个系统要包括以下几个类：文件类、用户类、管理员类、登录注册控制类、文件管理控制类、文件分享控制类和回收站管理控制类。

文件类包含属性如文件 ID、名称、类型、大小、MD5 值等，是整个系统中最基础的实体类。用户类包含用户名、账号信息及存储空间等属性，管理员类继承了用户类父类，则有权限级别的定义，管理员权限能够执行用户管理控制类中的方法。登录注册控制类负责用户的注册和身份验证。文件管理控制类提供查看、上传、下载和删除等文件基础功能。对于管理员用户，还支持额外的查看用户所有文件的方法。文件分享控制类负责分享操作，主要方法包括分享文件和取消分享文件。回收站管理控制类用于恢复或清除已删除的文件，主要方法包括删除回收站文件和还原回收站文件。这些类之间通过调用关系连接，形成一个完整的系统架构，以支持整个网盘的正常运行。系统还具备回收站功能，用户可以将不再需要的文件暂时移至回收站，之后可以选择恢复或永久删除。为防止重要文件的误删，系统设定了回收站文件的自动清理机制，超过七天未处理的文件将被自动永久删除，以优化存储资源的使用。上述就是整个系统中不同的角色所能够行使

的功能。

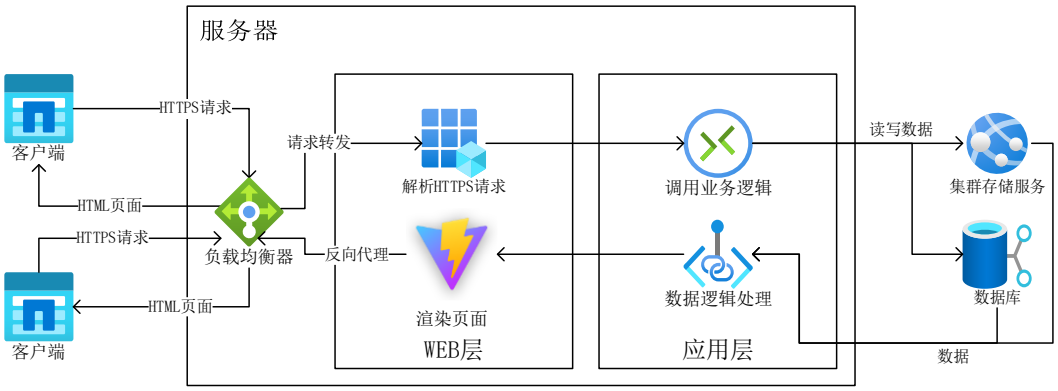


**图 2-3** B/S 体系结构图

对于系统实现，如图 2-3 所示，本系统采用 B/S 架构，客户端通过浏览器向服务器发起 HTTPS 请求，Nginx 作为反向代理服务器接收并转发这些请求到后端的 Web 服务层。服务器中的 Web 服务层使用 Spring Boot 处理请求，应用层调用业务逻辑层的方法，例如将文件存入存储集群。如有必要，服务器与 MySQL 或 Redis 数据库进行数据交换，后端将处理结果返回给 Web 服务层，渲染为 HTML 页面，具体使用 Vite 负责服务端渲染，确保页面的快速加载和动态更新，最终的 HTML 页面通过 Nginx 返回给客户端，完成响应。此架构通过 Nginx 实现负载均衡和静态资源的高效分发，结合 Vite 的快速构建能力，确保系统的高性能和高可用性。

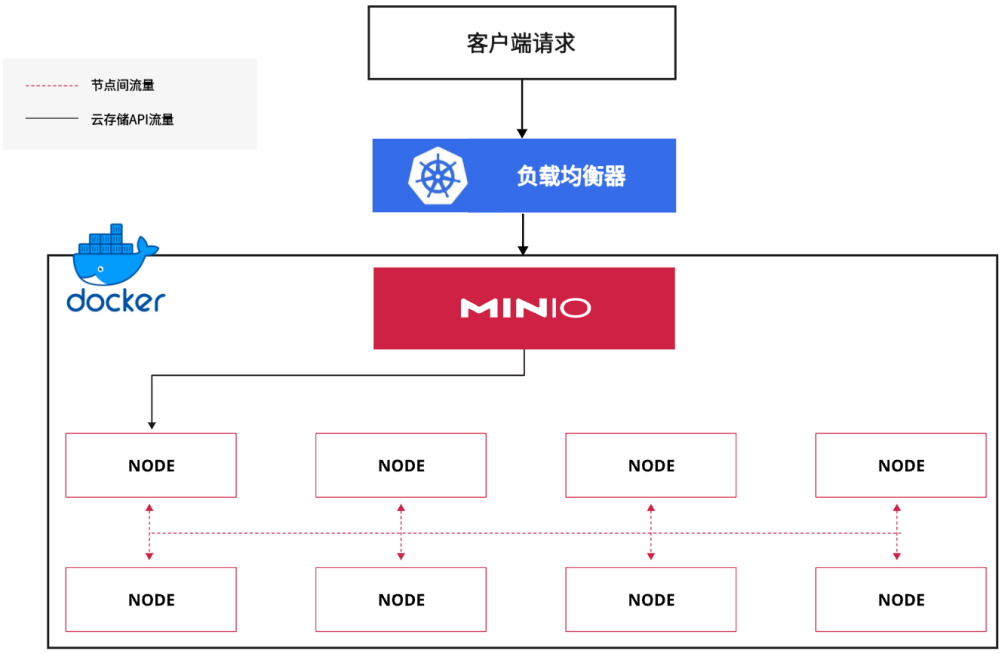


**图 2-6** 本系统结合 kubernetes 的 MinIO 分布式存储架构图

如图 2-6 所示，本系统的分布式存储服务采用 MinIO 作为核心组件。MinIO 是一个高性能的开源对象存储，被广泛应用于云原生应用、数据湖、AI 和机器学习等场景。与类似产品如 HDFS 和 Ceph 相比，MinIO 具备多项优势。首先，它专注于对象存储，优化了读写性能，尤其在处理大文件时表现出色。其次，MinIO 的部署和管理相对简单，支持轻量级的容器化部署，适合快速开发和迭代。它完全兼容 S3 API，便于与现有的云服务和应用集成。此外，MinIO 通过 Kubernetes 的自动化管理，能够根据需求动态扩展存储容量和计算资源。在分布式存储实现方面，MinIO 采用 Reed-Solomon（里德-所罗门纠删码），是一种常见的纠删码，它可以将 n 份原始数据，增加 m 份数据，并能通过 n+m 份中的任意 n 份数据，还原为原始数据。即如果有任意小于等于 m 份的数据失效，仍然能通过剩下的数据还原出来。 本系统通过将数据分成多个数据块和冗余块来提供数据冗余，相比于简单复制（如 HDFS），纠删码能在提供近似恢复能力的情况下减少存储开销，确保数据的安全性和可靠性。

为实现系统的高可用，本系统的 MinIO 部署在 Kubernetes 集群中，利用 Docker 容器化技术来实现灵活的分布式存储。系统架构由多个 MinIO Pods 组成，每个 Pod 运行在一个 Kubernetes 节点上，通过 Kubernetes 的自动化管理功能实现高可用性和弹性扩展。客户端请求首先通过 Nginx Ingress 进行负载均衡，Nginx Ingress 作为入口控制器，将请求分发到使用 Docker 作为容器运行时的不同的 MinIO Pods，确保流量的均匀分布和系统的高效运行。文件将进入 MinIO 并使用纠删码技术实现多 Nodes 分布式存储以提供数据冗余和恢复能力，保证数据的安全性和可靠性。整个架构结合了 Kubernetes 的自动化部署和管理特性，使得系统能够在不同的节点间动态调整资源分配，从而实现高效的存储和处理能力。

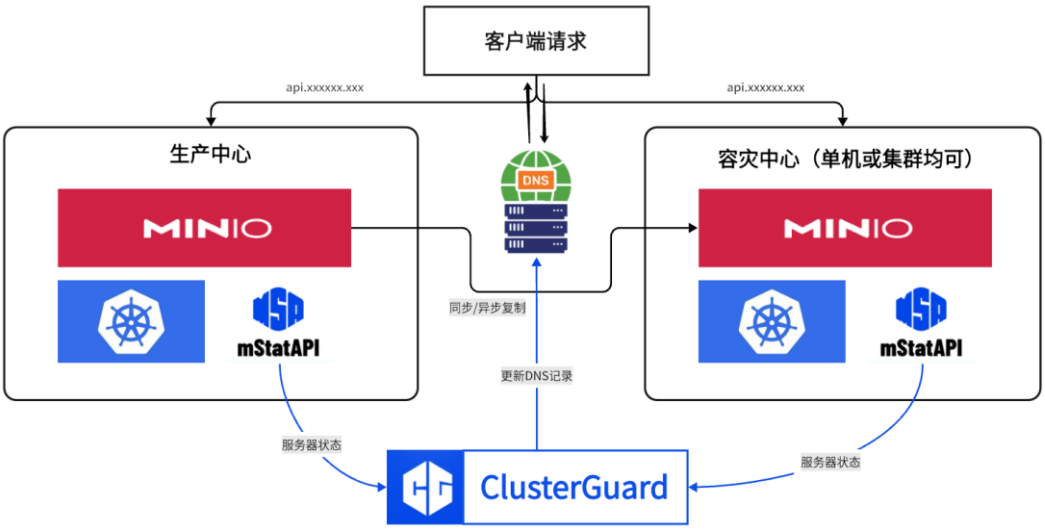


**图 2-7** 本系统基于 DNS 切换的 MinIO 双机（集群）热备的高可用架构图

为保证在集群崩溃或其他极端情况下维持系统稳定并防止数据丢失，如图 2-7 所示，我们基于 Streamlit 和 FastAPI 分别作为前后端开发了 ClusterGuard ，支持自定义规则配置和多次重试机制，提供直观的状态反馈和日志记录。ClusterGuard 通过使用云 DNS 服务 API 修改和切换 DNS 解析并结合节点状态检测信息（我们为此开发了 mStatAPI——一个简单的服务器远程探针获取服务器状态），在节点失效时立即触发预定义的 DNS 重写规则，更新 DNS 记录，将流量切换到容灾中心，确保数据同步和系统的高可用性。

除了文件的上传和下载外，本系统还支持文件分享功能。用户可通过系统生成文件或文件夹的分享链接，并设置访问权限（如只读、可下载、限时访问等）。分享链接支持密码保护与到期时间配置，确保数据在分享过程中的安全性。通过共享文件夹功能，用户可邀请其他用户加入协作，实现团队成员之间的高效文件共享和管理。

## （三）系统的详细设计

通过在系统的概要设计中提出的用例模型、对象模型及数据模型，对整个系统的功能模块进行详细设计。能够得到如图 2-7 所示的软件架构图。

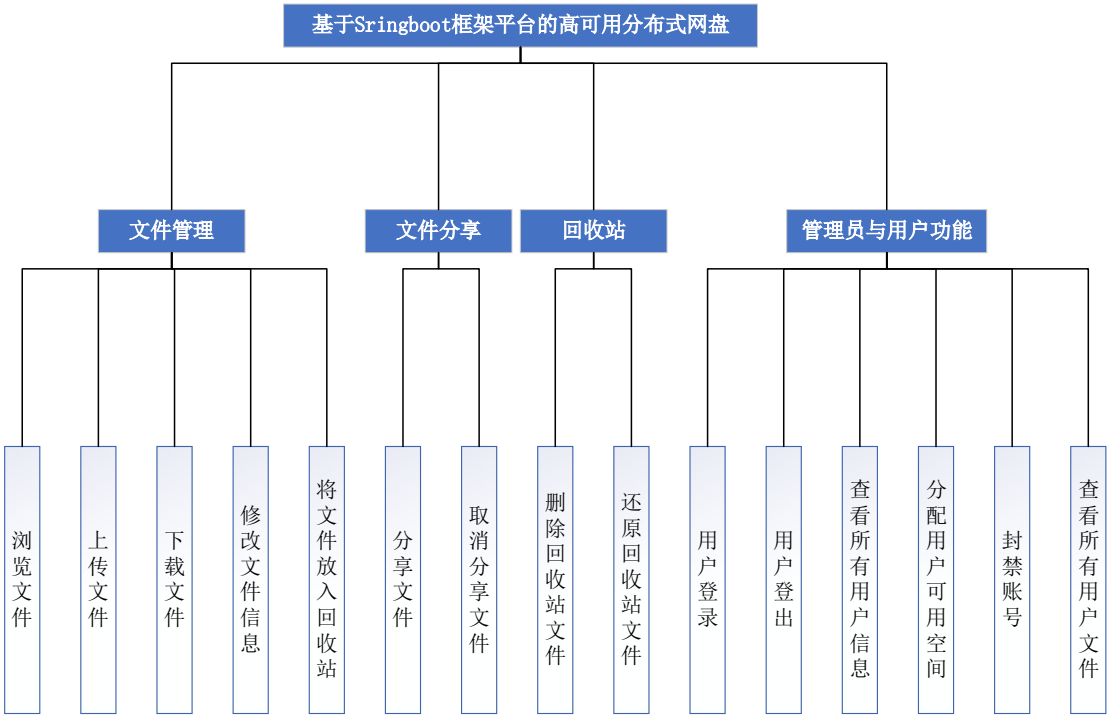


**图 2-8** 系统软件架构图

图 2-8 展示了在详细设计中系统下的四个主要模块，分为文件管理、文件分享、回收站和管理员与用户功能四个模块。下面对于每个功能模块进行详细介绍。文件管理模块支持浏览、上传、下载和修改文件，并提供将文件移入回收站的功能。文件分享模块允许用户分享文件并获取相关信息。回收站模块提供删除和还原文件的功能，确保数据可恢复性。管理员与用户功能模块则涵盖用户登录登出、查看用户信息、分配存储空间及账号管理等操作。这些模块通过 Minio 的分布式能力，提升了系统在大规模数据环境下的效率和可靠性，同时确保了灵活的用户管理和数据共享。

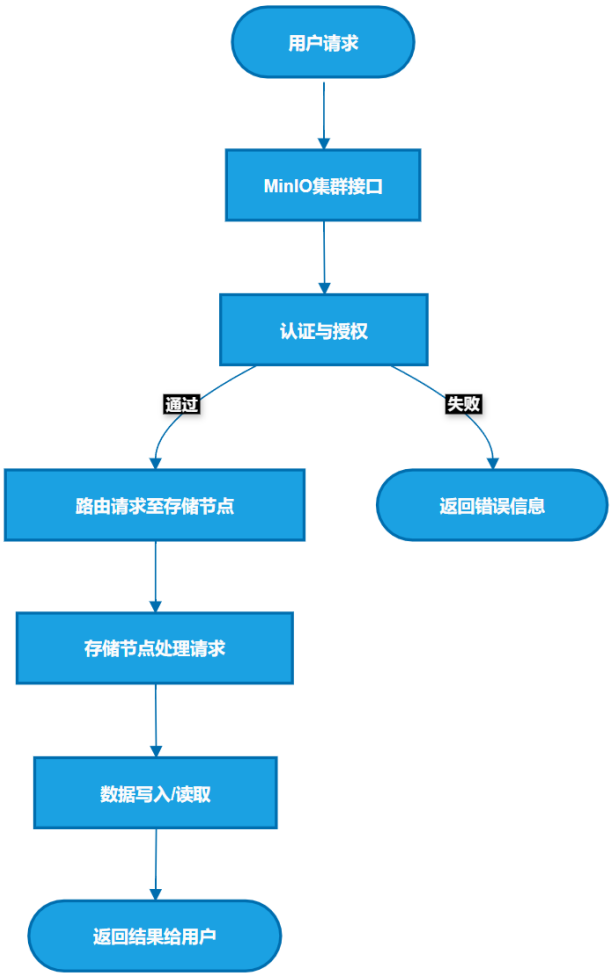


图 2-9 文件集群存储业务流程图

如图 2-9 所示，本系统文件存储业务流程从用户发起的文件存储或检索请求开始，这个请求首先被发送到分布式存储集群的接口，然后集群对用户请求进行认证和授权检查，确保用户有权限进行操作。如果认证失败，集群将返回错误信息给用户；如果认证通过，请求被路由到存储节点，存储节点接收到请求后，开始处理数据的写入或读取操作，操作完成后，存储节点将结果返回给用户，完成整个文件存储业务流程。

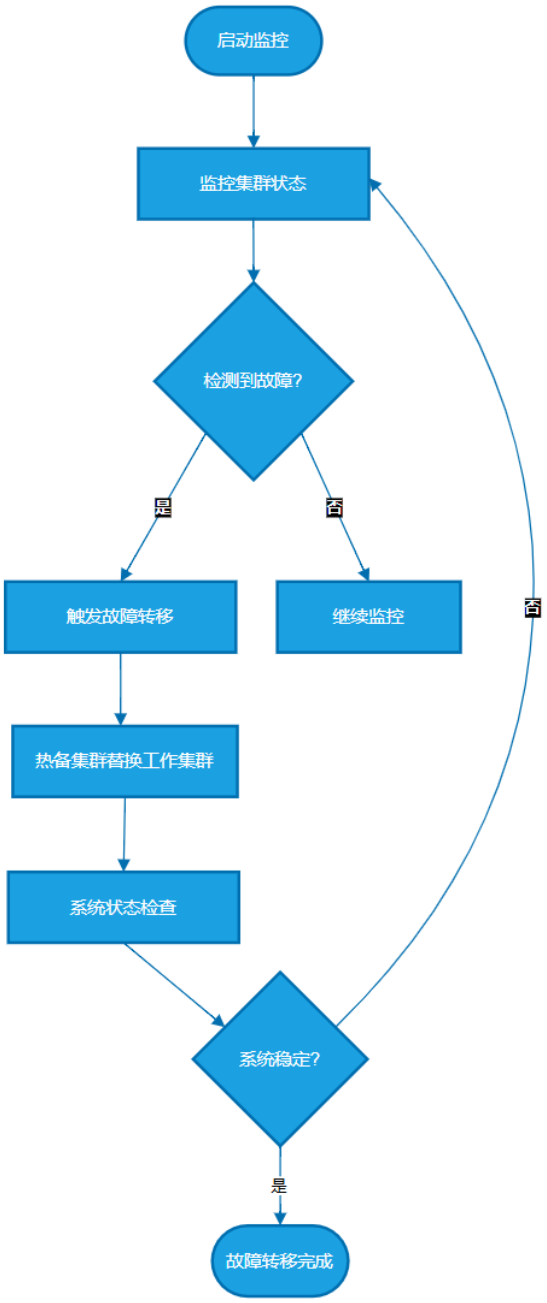


**图 2-10** 自动监控与故障转移业务流程图

如图 2-10 所示，该流程图描述了自动监控和故障转移的完整过程。首先，启动监控系统以实时监控集群的状态。在监控过程中，如果检测到故障，则触发故障转移机制，将热备集群切换为工作集群，并进行系统状态检查。如果系统状态稳定，则故障转移完成，流程结束；如果系统不稳定，则继续监控集群状态，直到问题解决。如果在监控过程中没有检测到故障，则持续进行监控，确保集群的正常运行。

基于前述的软件架构图和对象模型设计，给出系统的动态模型设计，如图

2-11 所示。这张图以文件分享模块的功能为例，展示了人机交互中数据流动的方向，以及各个有关类对于用户请求是如何响应的。

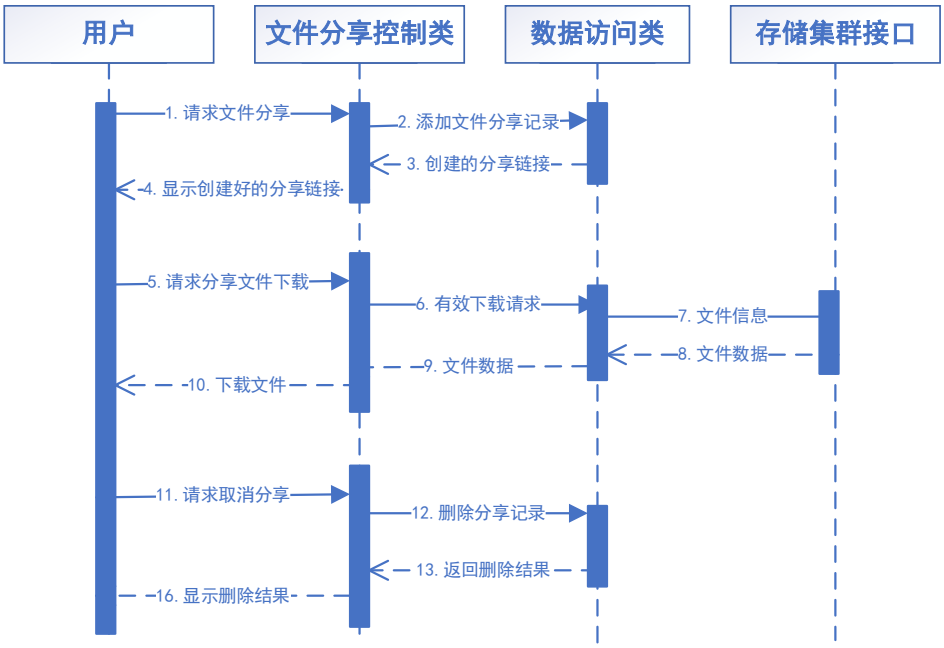


**图 2-11** 文件分享功能顺序图

图 2-11 的顺序图展示了文件分享功能的详细流程：用户首先向文件分享控制类发起文件分享请求，该控制类随后将添加文件分享记录，并创建分享链接返回给用户，用户即可显示并提供该链接供他人使用。当其他用户请求下载分享的文件时，文件分享控制类会验证下载请求的有效性，并向数据访问类请求文件信息，数据访问类通过存储集群接口获取文件数据并返回给文件分享控制类，最终由文件分享控制类将文件数据提供给下载请求者，完成下载过程。若用户决定取消分享，他们可以向文件分享控制类发送取消分享的请求，该控制类将删除分享记录，并将删除结果反馈给用户，同时显示删除结果，确保分享的文件不再对外可见。整个流程确保了文件分享的便捷性、安全性和可控性。

同时，存储集群还具备热备监控与自动故障转移机制，系统启动后，首先开始监控集群状态，持续监控集群的健康状态，包括各个节点的运行情况。在监控过程中，系统会检查是否检测到故障，如果没有检测到故障，系统继续监控；如果检测到故障，系统将触发故障转移，热备集群将接管工作集群的职责，以确保服务的连续性。热备集群接管后，系统会进行状态检查，确保所有服务正常运行，如果系统稳定，故障转移完成，系统继续正常运行；如果系统不稳定，故障转移

流程可能会重新触发，或者采取其他措施以恢复系统稳定性。这种设计确保了MinIO 集群的高可用性和数据的可靠性，即使在部分节点发生故障的情况下，也能通过热备集群的快速接管，最小化服务中断，保障业务连续性。

基于前述的业务流程、架构及静态对象模型设计，给出系统中两个较为复杂的模块即文件分享和文件管理（包括存储与访问）的动态模型设计。

如图 2-12 所示，以文件分享模块的功能为例，展示了人机交互中数据流动的方向，以及各个类对于用户请求是如何相应的。

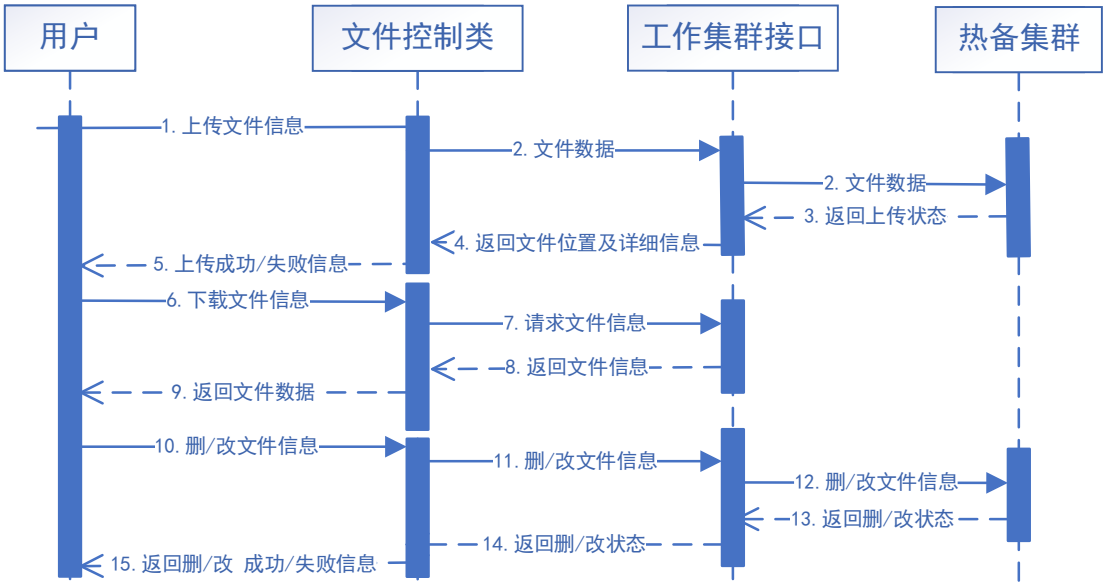


**图 2-12** 文件存储与访问功能顺序图

图 2-12 展示了用户与系统之间进行文件存储与访问操作的完整流程。用户首先通过控制类上传文件信息，控制类随后将文件数据发送到工作集群进行存储。工作集群处理完毕后，会将上传状态反馈给控制类，控制类再将文件位置及详细信息返回给用户。如果上传成功，用户可以请求下载文件信息，控制类会向工作集群请求文件信息，并将文件数据返回给用户。用户还可以发送删除或修改文件的请求，控制类将这些请求转发给工作集群，工作集群处理后将操作状态反馈给控制类，最后由控制类通知用户操作的成功与否。

图 2-10 中，工作集群与热备集群的每次连接是分开的对象，这是由于图中的集群在实际中是以接口类呈现。这种设计意味着每次访问工作集群时，都需要重

新建立连接，以确保在热备切换时能够保持操作的有效性。因此在每次操作时，需要确保访问的是最新的、有效的工作集群实例。这种设计提高了系统的容错性和可靠性，确保在工作集群出现故障时，热备集群可以迅速接管，而用户操作不会受到影响。通过这种方式，系统能够实现高可用性，即使在硬件故障或系统升级的情况下，也能保持服务的连续性。

## （四）系统的核心算法

**纠删码计算 (Reed-Solomon 编码)**

纠删码是一种编码技术，它可以将 n 份原始数据，增加 m 份数据，并能通过 n+m 份中的任意 n 份数据，还原为原始数据。即如果有任意小于等于 m 份的数据失效，仍然能通过剩下的数据还原出来。 纠删码技术在分布式存储 系统中的应用主要有三类，阵列纠删码（Array Code: RAID5、RAID6 等）、RS(Reed-Solomon) 里德-所罗门类纠删码和 LDPC(LowDensity Parity Check Code)低密度奇偶校验纠删码。 LDPC 码目前主要用于通信、视频和音频编码等领域。里德-所罗门码（又称里所码，Reed-solomon codes，简称 RS codes）是一种前向错误更正的信道编码，对由校正过采样数据所产生的有效多项式。编码过程首先在多个点上对这些多项式求冗余，然后将其传输或者存储。对多项式的这种超出必要值得采样使得多项式超定（过限定）。当接收器正确的收到足够的点后，它就可以恢复原来的多项式，即使接收到的多项式上有很多点被噪声干扰有损。

MinIO 使用 Reed-Solomon 编码算法来分割数据并生成校验块。而多副本策略即将数据存储多个副本（一般是三副本，比如 HDFS），当某个副本丢失时，可以通过其他副本复制回来。三副本的磁盘利用率为 1/3。纠删码技术主要是通过纠删码算法将原始的数据进行编码得到冗余，并将数据和冗余一并存储起来，以达到容错的目的。其基本思想是将 n 块原始的数据元素通过一定的计算，得到 m 块冗余元素（校验块）。对于这 n+m 块的元素，当其中任意的 m 块元素出错（包括原始数据和冗余数据）时，均可以通过对应的重构算法恢复出原来的 n 块数据。生成校验的过程被成为编码（encoding），恢复丢失数据块的过程被称为解码（decoding）。磁盘利用率为 n/(n+m)。基于纠删码的方法与多副本方法相比具有冗余度低、磁盘利用率高等优点。文件上传后将使用此编码方式编码并存

储，流程如图 2-13 所示。

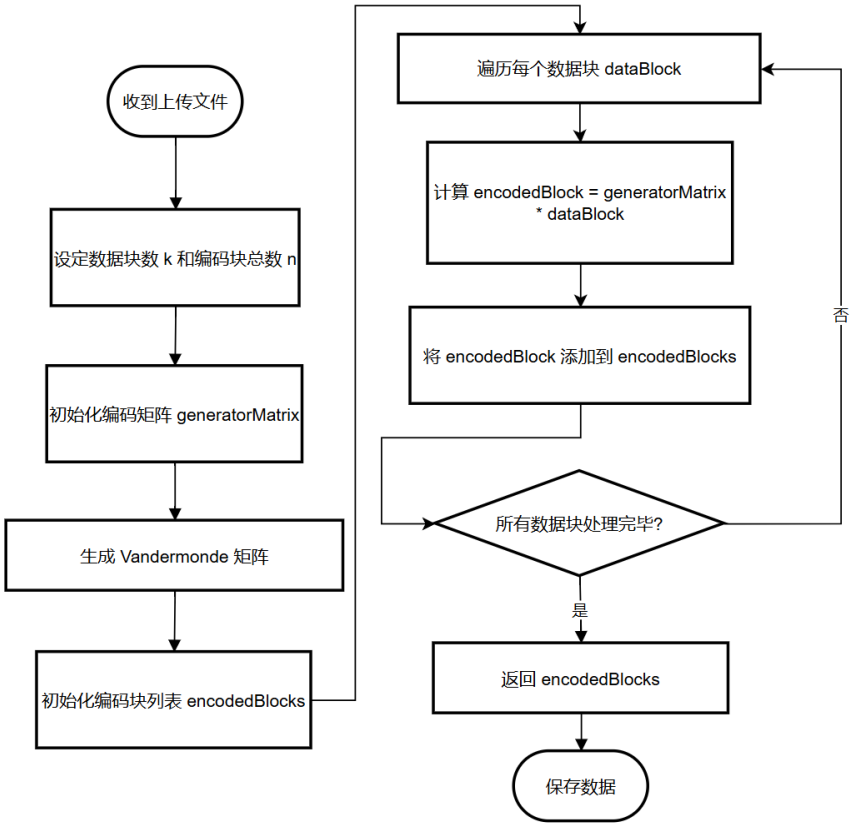


**图 2-13** 数据编码流程图

Reed-Solomon 编码：

函数 reedSolomonEncode(dataBlocks, k, n):

**输入：**原始数据块 dataBlocks，数据块数 k，编码块总数 n

**输出：**编码后的数据块和冗余块

初始化编码矩阵 generatorMatrix = 生成 Vandermonde 矩阵(n, k)

初始化编码块 encodedBlocks = 空列表

对于每个数据块 dataBlock 在 dataBlocks 中:

encodedBlock = 乘法(generatorMatrix, dataBlock)

将 encodedBlock 添加到 encodedBlocks

返回 encodedBlocks

Reed-Solomon 解码：

函数 reedSolomonDecode(encodedBlocks, k, n):

**输入：**编码后的数据块 encodedBlocks，数据块数 k，编码块总数 n

**输出：**恢复的原始数据块

初始化解码矩阵 inverseMatrix = 空

初始化恢复的数据块 recoveredData = 空列表

如果 encodedBlocks 中有缺失:

识别缺失的块并调整矩阵

生成逆矩阵 inverseMatrix = 计算逆矩阵(有效的编码块)

对于每个有效的编码块 encodedBlock 在 encodedBlocks 中:

originalBlock = 乘法(inverseMatrix, encodedBlock)

将 originalBlock 添加到 recoveredData

返回 recoveredData

**分片上传**

分片上传（Chunked Upload）是将大文件分成多个较小的部分（分片）来逐个上传到服务器。 上传完成后，服务器将这些分片重新组装成原始文件。 这个过程通常包括以下几个步骤： 分片：文件被切割成多个小的片段，每个片段的大小通常是预定义的。 上传：每个分片被单独上传到服务器。 上传过程中，通常会附带分片的索引和其他元数据。 组装：服务器接收到所有分片后，将它们按正确的顺序重新组装成完整的文件。下图 2-14 表示了系统在上传文件所使用的分片上传流程。

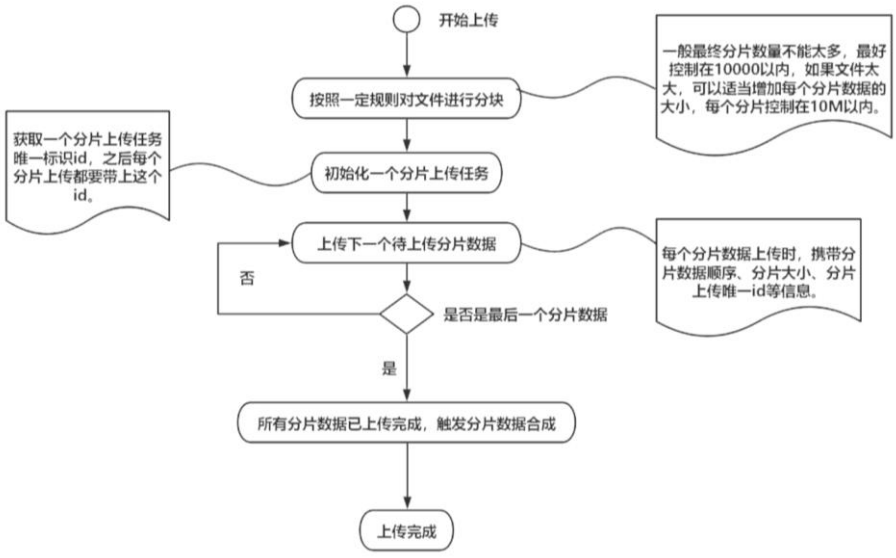


**图 2-14** 分片上传流程图

# 三、系统编码及运行

## （一）系统开发涉及的软件

Microsoft Word 是由微软公司开发的一款功能强大的文字处理软件，作为 Microsoft Office 套件的核心组件之一，广泛应用于文档创建、编辑和格式化。它支持多种文档格式，包括 DOC、DOCX、PDF 等，用户可以轻松编写报告、信件、简历、论文等各种类型的文档。Word 提供了丰富的排版工具，如样式设置、表格插入、图像处理、页眉页脚设计等，极大地提高了文档的美观性和可读性。此外，Word 还集成了强大的校对功能，包括拼写检查、语法纠错和同义词建议，帮助用户提升文本质量。通过实时协作功能，多个用户可以同时编辑同一文档，方便团队合作与信息共享。Word 的模板库涵盖了各类常见文档模板，用户可以根据需要快速生成专业格式的文档。

Microsoft Visio 是微软公司开发的一款专业的图表绘制工具，主要用于创建流程图、组织结构图、网络拓扑图、工程图、BIM 图纸等各种类型的图表。Visio 提供了丰富的模板和形状库，用户可以根据具体需求选择合适的模板，从而快速构建专业级的图表。其直观的拖拽界面和强大的绘图功能，使得即使是没有设计背景的用户也能轻松上手，制作出清晰、美观的图表。此外，Visio 支持与其他 Microsoft Office 应用程序的深度集成，用户可以将 Visio 图表嵌入到 Word 文档、PowerPoint 演示文稿中，提升信息展示的效果。对于需要协作的团队项目，Visio 提供了共享和协同编辑功能，多个用户可以同时对同一图表进行编辑和修改，确保信息的一致性和准确性。Visio 还支持数据连接功能，用户可以将图表与 Excel、SQL Server 等数据源关联，实现数据的动态可视化，帮助用户更好地理解和分析复杂的数据关系。其强大的图表自动布局和智能对齐功能，使得图表的设计更加高效和专业。同时，Visio 支持多种文件格式的导出和导入，方便用户在不同平台和设备间进行数据交换和共享。无论是在企业管理、工程设计还是教育培训领域，Microsoft Visio 都以其强大的功能和灵活的应用场景，成为不可或缺的图表绘制工具。

IntelliJ IDEA 是由 JetBrains 公司开发的一款功能强大的集成开发环境（IDE），主要面向 Java 开发者，但也支持多种编程语言如 Kotlin、Scala、Groovy、JavaScript 等。IntelliJ IDEA 以其智能的代码辅助功能、强大的重构工具和先进的调试能力著称，极大地提高了开发者的生产力和代码质量。其智能代码补全功能不仅能够根据上下文提供准确的代码建议，还能自动检测和提示潜在的代码错误，帮助开发者在编写过程中及时修正问题。IntelliJ IDEA 的重构工具支持多种复杂的代码重构操作，如重命名、提取方法、移动类等，确保在进行代码修改时不会引入新的错误。其内置的版本控制系统（如 Git、SVN、Mercurial 等）集成，使得代码管理和协作更加便捷。IntelliJ IDEA 还提供了丰富的插件生态系统，用户可以根据需求安装各种插件，扩展 IDE 的功能，满足不同项目的特定需求。其强大的调试工具支持断点设置、变量监视、堆栈分析等功能，帮助开发者快速定位和修复代码中的问题。对于 Web 开发，IntelliJ IDEA 提供了对 HTML、CSS、JavaScript 以及主流框架（如 Spring、Hibernate 等）的全面支持，简化了 Web 应用的开发流程。此外，IntelliJ IDEA 还具备优秀的性能优化和资源管理能力，能够在处理大型项目时保持流畅和高效。凭借其全面的功能和用户友好的界面，IntelliJ IDEA 已成为全球众多开发者的首选开发工具，广泛应用于软件开发、系统集成、企业应用等多个领域。

Visual Studio Code 是由微软开发的一款免费的开源代码编辑器，兼具轻量级和强大功能，被广泛用于各种编程语言的开发。VSCode 以其简洁的用户界面和丰富的扩展生态系统著称，用户可以通过插件市场安装各种插件，轻松扩展其功能，满足不同开发需求。其内置的智能代码补全功能能够根据上下文提供准确的代码建议，提高了代码编写效率和准确性。VSCode 支持多种编程语言，包括 JavaScript、Python、Java、C++、Ruby 等，且通过插件可以进一步扩展支持更多语言和框架。此外，VSCode 内置了强大的调试工具，支持断点设置、变量监视、堆栈跟踪等功能，使得调试过程更加便捷和高效。其集成的 Git 支持功能允许开发者在编辑器中直接进行版本控制操作，如提交、推送、合并分支等，简化了代码管理流程。VSCode 还支持远程开发，通过 Remote Development 插件，开发者可以在远程服务器、容器或 WSL（Windows Subsystem for Linux）中进行开发，极大地提高了开发的灵活性和可移植性。其强大的搜索和替换功能、多窗格编辑、

终端集成等特性，使得开发过程更加高效和流畅。由于其跨平台特性，VSCode 可以运行在 Windows、macOS 和 Linux 系统上，满足不同开发环境的需求。凭借其灵活性、可扩展性和强大的功能，Visual Studio Code 已成为全球开发者社区中最受欢迎的代码编辑器之一，广泛应用于软件开发、前端设计、数据分析等多个领域。

Navicat 是一款由 PremiumSoft 公司开发的专业数据库管理工具，支持多种数据库系统如 MySQL、MariaDB、PostgreSQL、SQLite、Oracle、SQL Server 等。Navicat 以其友好的用户界面和强大的功能，帮助数据库管理员、开发者和分析师高效地管理和维护数据库。其主要功能包括数据库设计、数据迁移、数据同步、备份恢复、报表生成等，覆盖了数据库生命周期管理的各个方面。Navicat 提供了可视化的数据库设计工具，用户可以通过拖拽操作轻松创建和修改表结构、设计关系图，提高了数据库设计的效率和准确性。其强大的数据迁移和同步功能支持跨平台和跨数据库的迁移，确保数据在不同环境间的无缝转移和一致性。Navicat 的备份恢复功能允许用户定期备份数据库，防止数据丢失，并在需要时快速恢复，保障数据的安全性。通过内置的查询编辑器，用户可以编写和执行复杂的 SQL 语句，获得快速的查询结果和详细的数据分析。此外，Navicat 还支持自动化任务调度，用户可以设置定时任务，如定期备份、数据同步等，简化了日常数据库维护工作。其报表生成工具能够将数据库中的数据以多种格式（如 PDF、Excel、HTML）生成专业报表，方便数据的展示和分享。Navicat 还提供了便捷的连接管理功能，用户可以轻松管理和组织多个数据库连接，支持 SSH 隧道、HTTP 代理等多种连接方式，确保数据传输的安全性和稳定性。凭借其全面的功能和易用性，Navicat 已成为全球众多企业和开发者的首选数据库管理工具，广泛应用于软件开发、数据分析、系统集成等多个领域。

FinalShell 是一款功能强大的跨平台终端模拟器和远程管理工具，旨在为开发者、系统管理员和 IT 专业人员提供一个高效、安全的远程操作环境。FinalShell 支持多种协议，包括 SSH、Telnet、RDP、VNC 等，用户可以通过它轻松连接到不同类型的服务器和远程主机，进行日常的维护、配置和管理操作。其直观的用户界面和丰富的功能使得远程管理变得更加便捷和高效。FinalShell 内置了图形化的会话管理器，用户可以方便地组织和切换多个会话，支持标签式界面，提升

了多任务操作的效率。其支持多终端会话并发连接，用户可以在同一窗口中同时管理多个远程主机，极大地提高了工作效率。FinalShell 还集成了高级的文件传输功能，支持 SFTP、FTP 等多种文件传输协议，用户可以方便地上传、下载和管理远程服务器上的文件。其内置的终端增强功能，如多窗口分屏、键盘快捷键自定义、脚本自动化等，进一步提升了操作的灵活性和效率。FinalShell 还提供了强大的安全性保障，支持多种加密算法和认证方式，确保远程连接的安全性和数据传输的保密性。此外，FinalShell 支持自定义脚本和插件扩展，用户可以根据自身需求定制功能，满足不同的工作场景。其跨平台特性允许在 Windows、macOS 和 Linux 系统上无缝运行，适应不同用户的操作习惯和工作环境。凭借其强大的功能、灵活的扩展性和优秀的用户体验，FinalShell 已成为众多开发者和系统管理员进行远程管理和操作的理想工具，被广泛应用于软件开发、服务器维护、数据中心管理等多个领域。

Redis 是一种开源的、高性能的内存数据结构存储系统，广泛用于作为数据库、缓存和消息中间件。由 Salvatore Sanfilippo 于 2009 年开发，Redis 以其卓越的性能、丰富的数据类型和灵活的应用场景，成为现代应用程序中不可或缺的组件。Redis 支持多种数据结构，包括字符串（String）、列表（List）、集合（Set）、有序集合（Sorted Set）、哈希（Hash）、位图（Bitmap）、HyperLogLog 和地理空间索引（Geospatial Index）等，用户可以根据具体需求选择最合适的数据类型进行存储和操作。这些丰富的数据结构使得 Redis 在处理复杂的数据模型和业务逻辑时具有显著的优势。Redis 的主要特点之一是其基于内存的存储方式，确保了极高的读写速度，适用于需要快速响应的实时应用场景，如实时分析、缓存加速、排行榜系统等。此外，Redis 还支持持久化机制，通过 RDB 快照和 AOF 日志记录，确保数据在系统重启或故障恢复时的持久性。其内置的复制（Replication）和高可用性（HA）功能，如主从复制（Master-Slave Replication）和哨兵（Sentinel）机制，进一步保证了数据的可靠性和系统的稳定性。Redis 还支持分布式集群（Cluster），允许用户在多台服务器上分布式存储数据，提升系统的扩展性和处理能力，满足大规模应用的需求。其丰富的客户端库支持几乎所有主流编程语言，使得开发者可以轻松地将 Redis 集成到各种应用程序中。除此之外，Redis 还提供了发布/订阅（Pub/Sub）模式，支持消息队列功能，方便实现分布式系统中的

消息传递和事件驱动架构。由于其高性能和多功能特性，Redis 被广泛应用于 Web 开发、游戏开发、金融服务、电子商务等多个行业，成为构建高效、可靠和可扩展应用系统的重要工具。

Jupyter Notebook（此前被称为 IPython notebook）是一个交互式笔记本，支持运行 40 多种编程语言。Jupyter Notebook 的本质是一个 Web 应用程序，便于创建和共享程序文档，支持实时代码，数学方程，可视化和 markdown。 用途包括：数据清理和转换，数值模拟，统计建模，机器学习等等

## （二）系统运行界面及结果

系统运行的界面如下。通过本地的开放式连接访问前端，进入系统登录页面。输入管理员账号和密码后，即可进入系统首页。登录界面如图 3-1 所示。

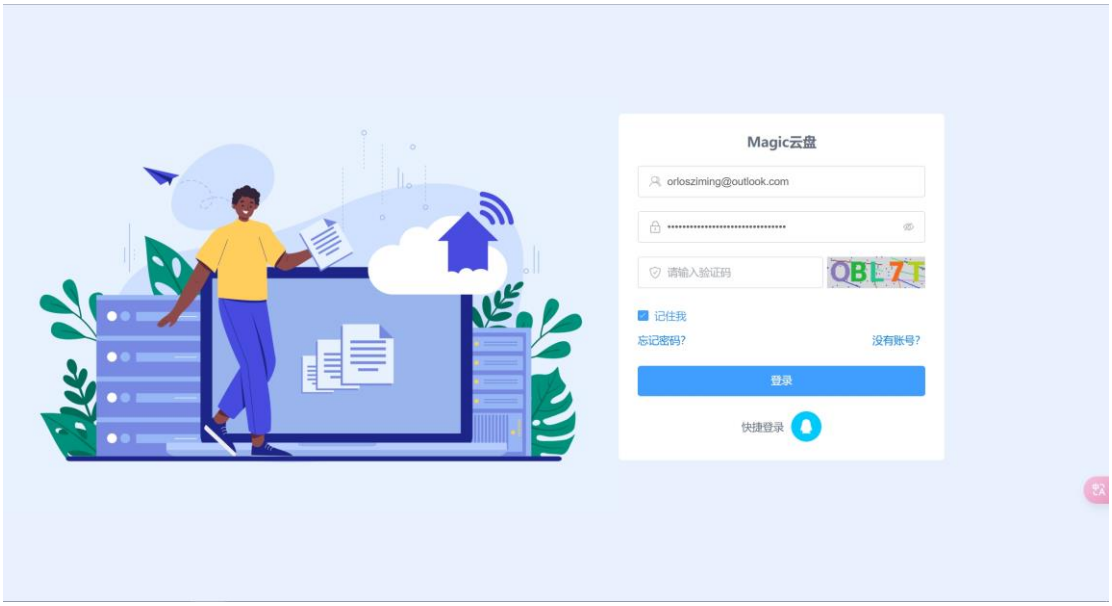


**图 3-1** 登录界面

系统首页如图 3-2 所示，其包含上传文件、下载文件、新建文件夹、将项目移入回收站等功能。登录时的账号昵称和头像会显示在右上角。

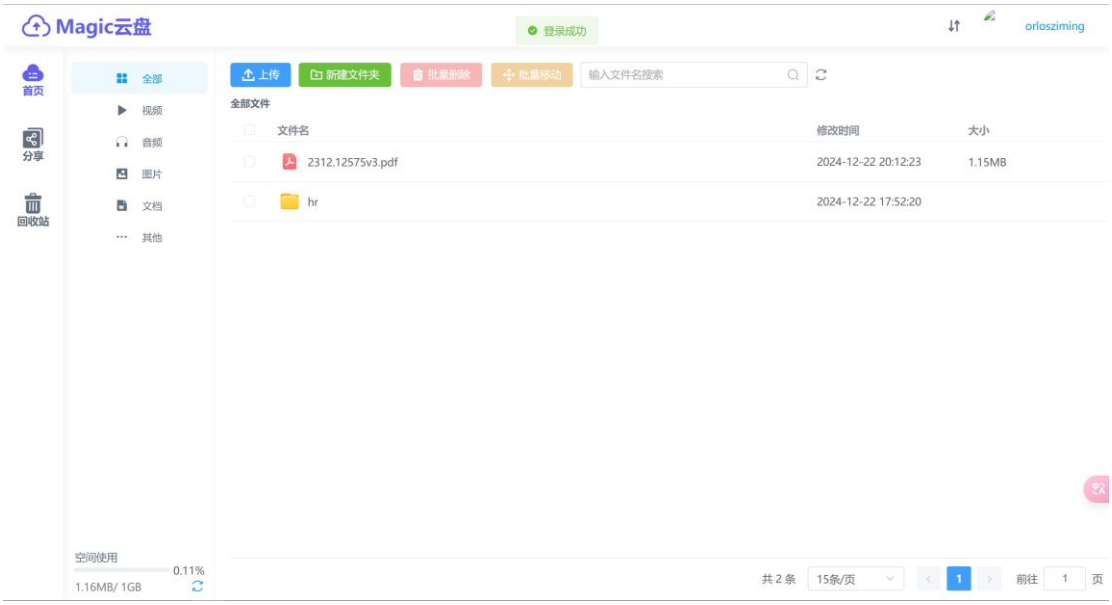


**图 3-2** 系统首页

上传文件时会在右上角显示上传进度，如图 3-3 所示。上传的文件本体会存入 Minio 的 bucket 桶中，MD5 哈希、文件名称等数据则会存放到关系型数据库中。

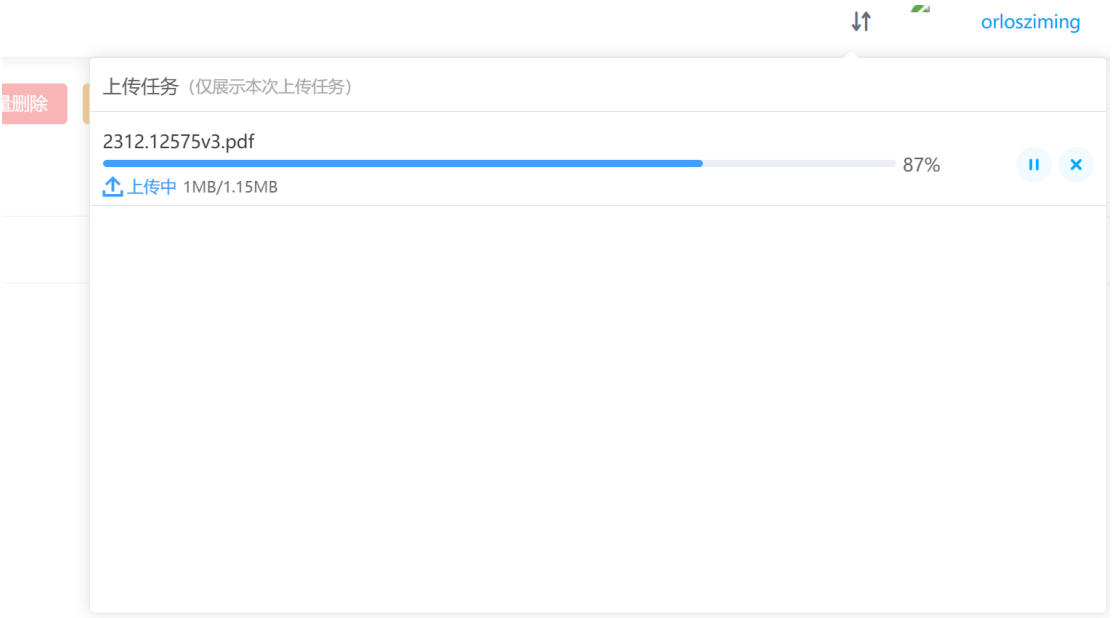


**图 3-3** 分块上传界面

集群自动监控面板自动灾备规则故障转移界面和手动故障转移界面如图 3-4、图 3-5 所示，界面包含对各节点的实时监控并生成实时图表（截取五分钟内的数据），在触发规则时自动执行规则并显示执行日志。


**图 3-4** 集群自动监控面板自动灾备规则故障转移界面


**图 3-5** 集群自动监控面板手动故障转移界面

# 四、系统测试

## （一）功能测试

采用的主要测试方法为黑盒测试。对系统进行测试时，上传了如图 4-1 中的若干文件。文件的所有信息均保存在 MySQL 关系型数据库中，如图 4-1、图 4-2 所示。对于测试用的文件，均能够实现正常的上传下载和文件分享。下载后的文件能够正常打开并使用。所上传的文件，均正常保存在 Minio 中。

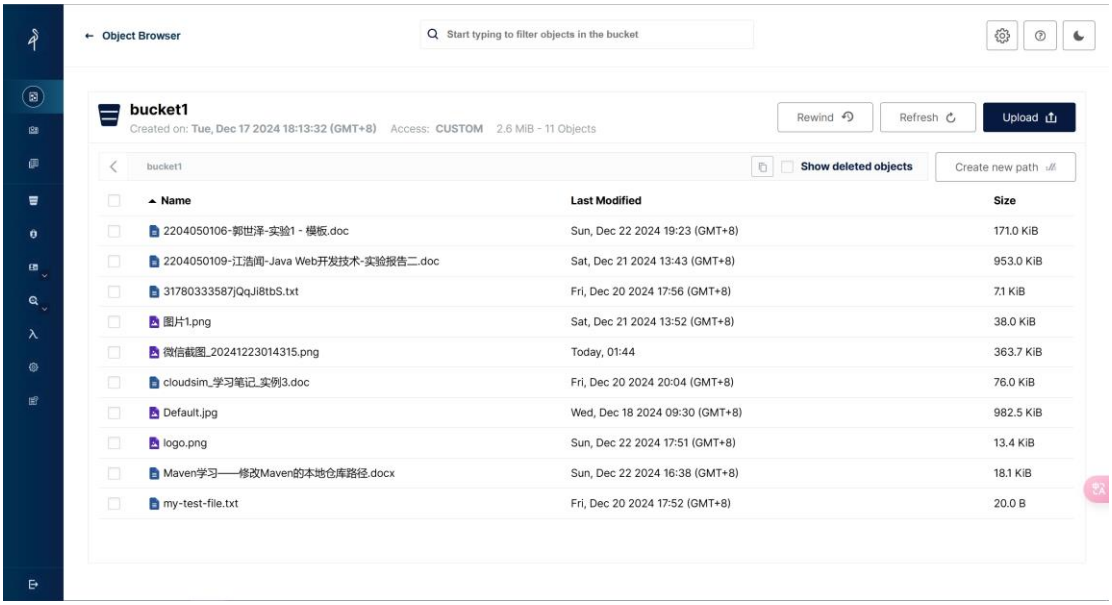


**图 4-1** 保存到 Minio 工作集群结果图

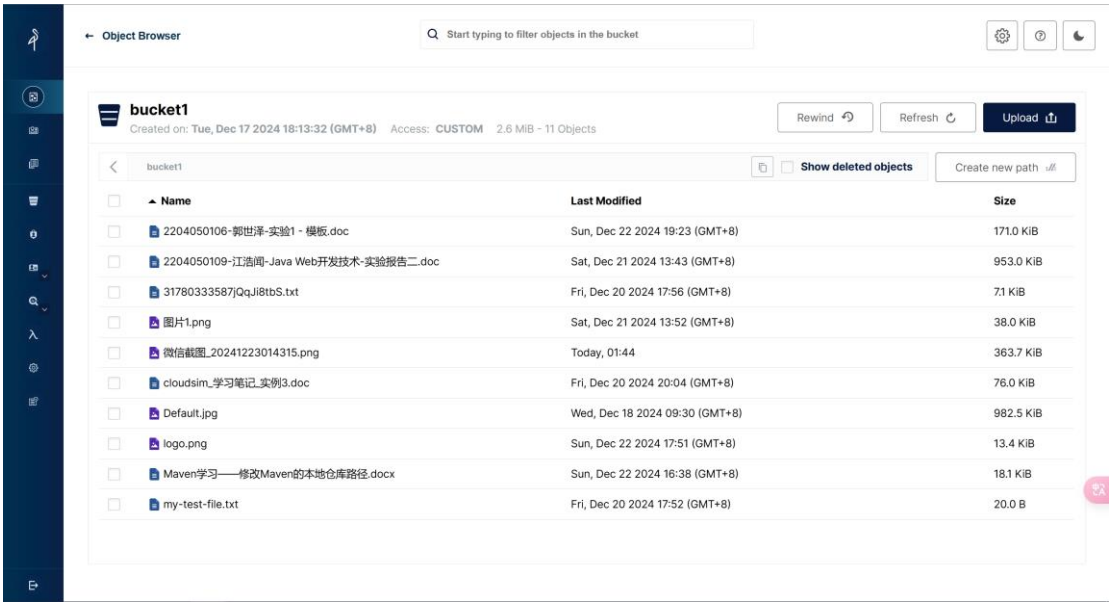

**图 4-2** 保存到 Minio 热备集群结果图

为对功能做进一步的全方位测试，我采用决策表法产生测试用例，决策表法是分析和表达多逻辑条件下执行不同操作的工具，可以全面的测试模块的功能。对文件存储与访问模块的功能测试涉及操作逻辑较少适合采用该方法，可以先将用户的操作、中间状态和结果分析整理并映射到文件管理系统中并据此构建决策表，如表 4-1 所示：

**表 4-1** 文件存储与访问测试决策表

| | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **条件** | **C1: 用户上传文件** | 1 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 |
| | **C2: 用户下载文件** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **C3: 用户修改文件** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| | **C4: 用户删除文件** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **中间状态** | **I1: 文件非该用户所有** | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| | **I2: 文件属于该用户** | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | **I3: 文件处于分享状态** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| **结果** | **E1: 文件下载到本地** | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| | **E2: 文件上传到集群** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **E3: 文件信息修改成功** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | **E4: 文件删除成功** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **E5: 操作失败** | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

其中例如当用户尝试上传文件（C1），如果文件属于该用户（I2），则上传成功（E2）。如果用户尝试下载文件（C2），需要检查文件是否属于该用户（I2）或文件是否处于分享状态（I3）。如果文件属于该用户或处于分享状态，下载成功（E1）。如果文件非该用户所有（I1），则操作失败（E5）。当用户尝试修改文件（C3），只有当文件属于该用户（I2）时，修改才会成功（E3）。如果文件非该用户所有（I1）或处于分享状态（I3），则操作失败。如果用户尝试删除文件（C4），同样需要文件属于该用户（I2）。如果文件属于该用户，删除成功（E4）。如果文件非该用户所有（I1）或处于分享状态（I3），则操作失败。基于决策表法生成的测试用例举例如表 4-3 所示。

**表 4-3** 测试用例部分案例

| 用例号 | 账户 | 密码 | 操作 | 预期结果 |
|---|---|---|---|---|
| 1 | 用户 A | 123456 | 上传 test1,test2,并下载 | 上传、下载成功 |
| 2 | 用户 A | 123456 | 分享 test2 | 分享成功 |
| 3 | 用户 B | 123456 | 下载用户 A 的 test1 | 下载失败 |
| 4 | 用户 B | 123456 | 下载用户 A 的 test2 | 下载成功 |

测试表法产生的所有测试用例已覆盖包括正常操作和异常操作的场景。经过测试可以保证系统对于权限的控制是严格的，确保了文件的安全性和用户数据的隐私。在文件属于用户或处于分享状态时，系统能够正确处理上传、下载、修改和删除操作。当用户尝试进行未授权的操作时，系统能够及时反馈错误信息，防止非法访问。

## （二）性能测试

如图所示，对文件存储与访问模块的性能测试采用分布式全链路自动压测方案。施压机使用 JMeter 分布在北京、杭州、广州、成都四个城市，每个城市承担 25%的流量，配合 Python 测试脚本模拟真实用户请求对当前存储服务进行测试。压力递增策略为每 10-30 秒增加并发用户数，直至探测到系统瓶颈。

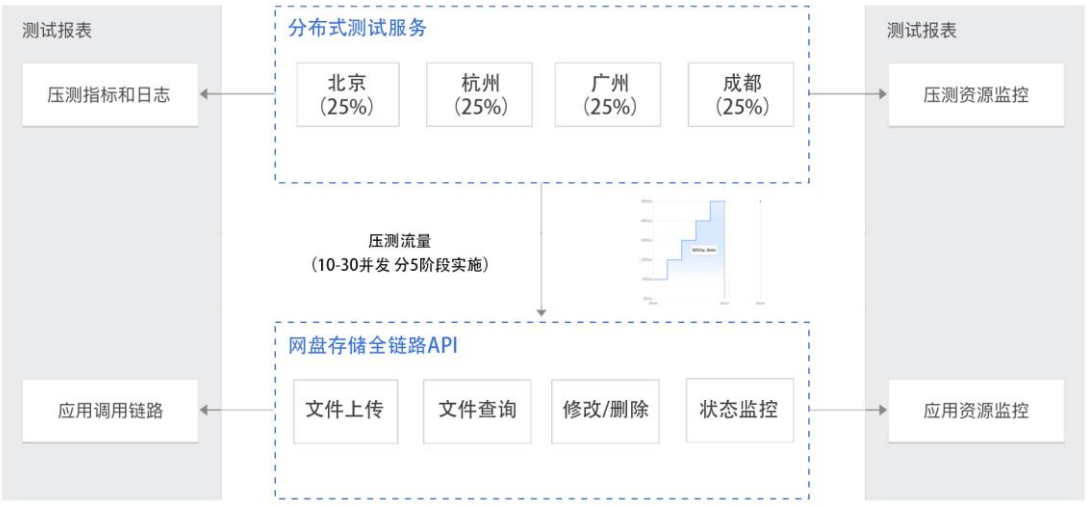


**图 4-3** 分布式全链路压测

图 4-3 展示了压测架构的详细设计，包括存储服务的各项核心 API，如文件上传、文件查询、修改/删除、状态监控。

**表 4-3** 并发压测结果

| *并发用户数* | *请求失败率* | *平均响应时间 (ms)* | *P95 响应时间 (ms)* | *备注* |
|---|---|---|---|---|
| 5 | 0% | 420 | 415 | 初始阶段 |
| 10 | 2.5% | 610 | 600 | 轻微延迟 |
| 15 | 28.2% | 1820 | 1500 | 响应时间增加 |
| 20 | 73.4% | 6000 | 3560 | 系统达到瓶颈 |
| 25 | 100% | 12000 | 12000 | 高延迟高失败率 |

上表列出了不同并发用户数下的系统性能表现，包括请求失败率、平均响应时间、P95 响应时间等关键指标。发现随着并发用户数的增加，系统的响应时间显著增加，尤其在 15 个并发用户后，性能下降明显。在达到 20 个并发用户时，系统出现瓶颈，导致响应时间和失败率大幅上升。针对发现的问题，需要优化系统的资源分配和负载均衡策略，以提升系统的稳定性和性能。据此提出以下解决方案：

1. 优化数据库查询：通过索引优化和查询缓存减少响应时间。
2. 增加服务器资源：扩展 Kubernetes 节点以提高系统处理能力。
3. 改进负载均衡策略：确保流量合理分配，避免单点过载。

# 五、总结

在“基于 Spring Boot 框架平台的 Magic 高可用分布式网盘的设计与实现”项目中，实现了一个功能丰富的网盘系统，专为 C 端用户设计。该系统支持用户注册、QQ 快捷登录、文件上传（包括分片上传、断点续传、秒传）、文件在线预览（支持文本、图片、视频、音频、Excel、Word、PDF 等）、文件分享和管理功能。前端采用了 Vue3、Vite3 和 Streamlit，后端使用 Spring Boot 结合 MyBatis 和 FastAPI，数据库选用 MySQL 和 Redis，存储服务则采用 Minio。整个项目的架构设计为高可用的分布式系统，以确保服务的稳定性和扩展性。实现了自动热备和故障转移机制，通过自定义规则和监控服务器状态的独立守护进程（daemon）来切换 DNS，确保主备服务的快速切换。同时，提供了可视化的监控界面，以便实时了解系统状态。

在项目中，我负责实现文件上传、下载和存储模块，以及存储服务的部署、热备和自动故障转移。通过该项目，我深入掌握了前后端技术的实际应用。前端方面，熟悉了 Vue 的组合式 API、生命周期函数、状态管理、路由等，提升了组件封装和页面布局能力。后端方面，通过 Spring Boot 和 MyBatis 的结合，熟练掌握了接口权限管理、参数校验、异步调用事务处理以及 Redis 缓存的使用。为提高数据的安全性和可靠性，我引入了纠删码技术，以实现数据的冗余存储和快速恢复。此外，使用 Kubernetes 进行容器化部署，确保了服务的高可用性和自动化扩展能力。热备和自动故障转移采用了自定义规则并监控服务器状态的独立守护进程（daemon）来切换 DNS，实现主备服务的快速切换，并提供可视化的监控界面。为了确保系统性能和稳定性，我还进行了基于 JMeter 的全链路云压测，以识别并优化系统中的瓶颈，进一步提升了项目的整体质量。

未来，我将继续深化对分布式系统和微服务架构的理解，尤其是云存储和自动化部署技术。同时，计划进一步研究大数据处理和机器学习在云服务中的应用，提升系统的扩展能力和自动化水平。

# 六、参考文献

[1] Theodoropoulos, T., Makris, A., Tserpes, J., Kardara, K., & Pateraki, M. (2023). Performance Analysis of Storage Systems in Edge Computing Infrastructures. Future Generation Computer Systems, 12(17), 8923. doi: 10.3390/fgcs20208923. Link

[2] Liu, J., Curry, M., Maltzahn, C., & Kufeldt, P. (2020). On construction of a distributed data storage system in cloud. Journal of Cloud Computing, 9(4), 1-22. doi: 10.1007/s11277-014-0399-4. Link

[3] Plank JS. A tutorial on Reed‐Solomon coding for fault‐tolerance in RAID‐like systems. Software: Practice and Experience. 1997 Sep;27(9):995-1012.

[4] 杜振南, 朱崇军. 分布式文件系统综述[J]. 软件工程与应用, 2017, 06(02): 21-27.

[5] 刘晓伟等. 一种基于 P2P 的云存储模型研究[D]. 西安电子科技大学. 现代图书情报技, 2011.

[6] 王义明, 杨利, 高欣. 云技术在网络中的应用及发展前景. 河北科技大学唐山分院. 数字化用户, 2013, 27 期.

[7] 谭霜, 贾焰, 韩伟红. 云存储中的数据完整性证明研究及进展. 国防科学技术大学 计算机学院. 计算机学报, 2015, 01 期. [5] 陈兰香, 许力. 云存储服务中可证明数据持有及恢复技术研究. 计算机研究与发展, 2012.

# 附录

## （一）文件分块上传功能核心代码

```
public UploadResultDto uploadFile(SessionWebUserDto webUserDto, String fileId,
MultipartFile file, String fileName, String filePid, String fileMd5,Integer chunkIndex,
Integer chunks) {
        File tempFileFolder = null;
        Boolean uploadSuccess = true;
        try {
            UploadResultDto resultDto = new UploadResultDto();
            if (StringTools.isEmpty(fileId)) {
                fileId = StringTools.getRandomString(Constants.LENGTH_10);
            }
            resultDto.setFileId(fileId);
            Date curDate = new Date();
            UserSpaceDto spaceDto =
redisComponent.getUserSpaceUse(webUserDto.getUserId());
            if (chunkIndex == 0) {
                FileInfoQuery infoQuery = new FileInfoQuery();
                infoQuery.setFileMd5(fileMd5);
                infoQuery.setSimplePage(new SimplePage(0, 1));
                infoQuery.setStatus(FileStatusEnums.USING.getStatus());
                List<FileInfo> dbFileList =
this.fileInfoMapper.selectList(infoQuery);
                //如果非多块则直接上传
                if (!dbFileList.isEmpty()) {
                    FileInfo dbFile = dbFileList.get(0);
```

```
                        //判断文件状态
                        if (dbFile.getFileSize() + spaceDto.getUseSpace() >
spaceDto.getTotalSpace()) {
                            throw new
BusinessException(ResponseCodeEnum.CODE_904);
                        }
                        dbFile.setFileId(fileId);
                        dbFile.setFilePid(filePid);
                        dbFile.setUserId(webUserDto.getUserId());
                        dbFile.setFileMd5(null);
                        dbFile.setCreateTime(curDate);
                        dbFile.setLastUpdateTime(curDate);
                        dbFile.setStatus(FileStatusEnums.USING.getStatus());
                        dbFile.setDelFlag(FileDelFlagEnums.USING.getFlag());
                        dbFile.setFileMd5(fileMd5);
                        fileName = autoRename(filePid, webUserDto.getUserId(),
fileName);
                        dbFile.setFileName(fileName);
                        this.fileInfoMapper.insert(dbFile);

resultDto.setStatus(UploadStatusEnums.UPLOAD_SECONDS.getCode());
                        //更新用户空间使用
                        updateUserSpace(webUserDto, dbFile.getFileSize());

                        return resultDto;
                    }
                }
                //暂存在临时目录
                String tempFolderName = appConfig.getProjectFolder() +
```

```
Constants.FILE_FOLDER_TEMP;
                String currentUserFolderName = webUserDto.getUserId() + fileId;
                //创建临时目录
                tempFileFolder = new File(tempFolderName +
currentUserFolderName);
                if (!tempFileFolder.exists()) {
                    tempFileFolder.mkdirs();
                }
                //判断磁盘空间
                Long currentTempSize =
redisComponent.getFileTempSize(webUserDto.getUserId(), fileId);
                if (file.getSize() + currentTempSize + spaceDto.getUseSpace() >
spaceDto.getTotalSpace()) {
                    throw new BusinessException(ResponseCodeEnum.CODE_904);
                }
                File newFile = new File(tempFileFolder.getPath() + "/" + chunkIndex);
                file.transferTo(newFile);
                //保存临时大小
                redisComponent.saveFileTempSize(webUserDto.getUserId(), fileId,
file.getSize());
                //不是最后一个分片，直接返回
                if (chunkIndex < chunks - 1) {

resultDto.setStatus(UploadStatusEnums.UPLOADING.getCode());
                    return resultDto;
                }
                //最后一个分片上传完成，记录数据库，异步合并分片
                String month = DateUtil.format(curDate,
DateTimePatternEnum.YYYYMM.getPattern());
```

```
                String fileSuffix = StringTools.getFileSuffix(fileName);
                //真实文件名
                String realFileName = currentUserFolderName + fileSuffix;
                FileTypeEnums fileTypeEnum =
FileTypeEnums.getFileTypeBySuffix(fileSuffix);
                //自动重命名
                fileName = autoRename(filePid, webUserDto.getUserId(), fileName);
                FileInfo fileInfo = new FileInfo();
                fileInfo.setFileId(fileId);
                fileInfo.setUserId(webUserDto.getUserId());
                fileInfo.setFileMd5(fileMd5);
                fileInfo.setFileName(fileName);
                fileInfo.setFilePath(month + "/" + realFileName);
                fileInfo.setFilePid(filePid);
                fileInfo.setCreateTime(curDate);
                fileInfo.setLastUpdateTime(curDate);
                fileInfo.setFileCategory(fileTypeEnum.getCategory().getCategory());
                fileInfo.setFileType(fileTypeEnum.getType());
                fileInfo.setStatus(FileStatusEnums.TRANSFER.getStatus());
                fileInfo.setFolderType(FileFolderTypeEnums.FILE.getType());
                fileInfo.setDelFlag(FileDelFlagEnums.USING.getFlag());
                this.fileInfoMapper.insert(fileInfo);
                Long totalSize =
redisComponent.getFileTempSize(webUserDto.getUserId(), fileId);
                updateUserSpace(webUserDto, totalSize);

resultDto.setStatus(UploadStatusEnums.UPLOAD_FINISH.getCode());
                //事务提交后调用异步方法
                TransactionSynchronizationManager.registerSynchronization(new
```

```
TransactionSynchronization() {
                    @Override
                    public void afterCommit() {
                        fileInfoService.transferFile(fileInfo.getFileId(),
webUserDto);
                    }
                });
                return resultDto;
            } catch (BusinessException e) {
                uploadSuccess = false;
                logger.error("文件上传失败", e);
                throw e;
            } catch (Exception e) {
                uploadSuccess = false;
                logger.error("文件上传失败", e);
                throw new BusinessException("文件上传失败");
            } finally {
                //如果上传失败，清除临时目录
                if (tempFileFolder != null && !uploadSuccess) {
                    try {
                        FileUtils.deleteDirectory(tempFileFolder);
                    } catch (IOException e) {
                        logger.error("删除临时目录失败");
                    }
                }
            }
        }
```

## （二）自动故障转移功能核心代码

```
if "rule" not in st.session_state:
st.session_state.rule = None
    if st.session_state.rule:
        # 更新 inactive 状态持续时间
        status_icon = "🟢" if nodestatus != "inactive" else "🔴"

        current_time = time.time()
        if nodestatus == "inactive":
            if st.session_state.inactive_start_time is None:
                st.session_state.inactive_start_time = current_time
            st.session_state.inactive_duration = int(current_time -
st.session_state.inactive_start_time)
        else:
            st.session_state.inactive_start_time = None
            st.session_state.inactive_duration = 0

        rule = st.session_state.rule
        rule_description = f"正在运行的规则：当 inactive 持续 {rule['duration']}
秒 时，启用 {rule['service']}"
        st.info(f"{rule_description} 当前状态：{status_icon}")
        if nodestatus == "inactive":
            st.info(f"当前 inactive 已持续{st.session_state.inactive_duration} 秒")
        # 检查是否超过阈值
        if st.session_state.inactive_duration >= rule['duration']:
            st.warning("规则触发,开始执行规则...")
            st.warning(f"启用 {rule['service']}...")
            if servicestatus == 'inactive':
```

```
                st.error(f"热备服务故障，启动失败")
            else:
                st.warning(f"执行 DNS 转移 {rule['service']}...")
                switch_dns(rule['service'])
                st.warning(f"转移完成，测试 API cloudapi.huxundao.com...")
                ping_output = ping_domain("cloudapi.huxundao.com")
                max_attempts = 10
                attempt = 0
                success = False

                while attempt < max_attempts:
                    ping_output = ping_domain("cloudapi.huxundao.com")
                    if "成功" in ping_output:
                        st.success("规则执行成功,API 恢复可用")
                        success = True
                        break
                    else:
                        attempt += 1
                        time.sleep(1)  # 等待一秒后重试
                        st.warning(f"测试 API cloudapi.huxundao.com... faild
retry {attempt}/{max_attempts}")
                if not success:
                    st.error("规则执行失败，已达到最大重试次数")
```