|  |  |
| --- | --- |
|  | 学校代码： 10246 |
|  | 学 号： 15212010038 |
|  |  |



|  |
| --- |
| 硕 士 学 位 论 文 |

**（专业学位）**

|  |
| --- |
| **面向云备份的高效中间件优化** |

|  |
| --- |
| **optimization of Efficient Middleware of Backup Cloud** |

院 系： 软件学院

专业学位类别（领域）： 软件工程

姓 名： 刘武

指 导 教 师： 韩伟力 教授

完 成 日 期： 2015年 月日

**指导小组成员名单**

韩伟力 教授

目录

[摘要 1](#_Toc486881487)

[Abstract 1](#_Toc486881488)

[第一章 绪论 2](#_Toc486881489)

[1.1 背景介绍 2](#_Toc486881490)

[1.2 相关研究工作 2](#_Toc486881491)

[1.3 论文组织结构 2](#_Toc486881492)

[第二章 相关技术综述 2](#_Toc486881493)

[2.1 中间件 2](#_Toc486881494)

[2.1.1 中间件简介 2](#_Toc486881495)

[2.1.2 中间件的概念模型 2](#_Toc486881496)

[2.1.3 中间件的特性 3](#_Toc486881497)

[2.2 数据加密算法AES 3](#_Toc486881498)

[2.2.1 AES算法简介 3](#_Toc486881499)

[2.2.2 AES算法的执行过程 4](#_Toc486881500)

[2.3 数据传输技术curl 5](#_Toc486881501)

[2.2.3 AES算法的分析 5](#_Toc486881502)

[2.3.1 curl命令简介 5](#_Toc486881503)

[2.3.2 curl命令使用示例 5](#_Toc486881504)

[2.4 SQLite关系型数据库 6](#_Toc486881505)

[2.5 I/O多路复用技术 6](#_Toc486881506)

[第三章 云备份中间件系统分析 8](#_Toc486881507)

[3.1 系统整体架构 8](#_Toc486881508)

[3.2 功能需求分析 8](#_Toc486881509)

[3.3 核心模块分析 9](#_Toc486881510)

[3.3.1 数据传输模块 9](#_Toc486881511)

[3.3.2 数据压缩模块 9](#_Toc486881512)

[3.3.3 数据加密模块 10](#_Toc486881513)

[3.3.4 安全会话模块 10](#_Toc486881514)

[3.4 数据结构分析 11](#_Toc486881515)

[3.4.1 访问令牌 11](#_Toc486881516)

[3.4.2 文件信息 11](#_Toc486881517)

[第四章 云备份中间件的设计 12](#_Toc486881518)

[4.1 系统总体框架 12](#_Toc486881519)

[4.1.1 系统框架图 12](#_Toc486881520)

[4.2 类图设计 13](#_Toc486881521)

[4.3 系统流程设计 14](#_Toc486881522)

[4.3.1 系统总体流程设计 14](#_Toc486881523)

[4.3.2 安全会话管理流程设计 16](#_Toc486881524)

[4.3.3 数据传输模块流程设计 19](#_Toc486881525)

[4.3.4 数据压缩模块流程设计 23](#_Toc486881526)

[4.3.5 数据加密服务流程设计 24](#_Toc486881527)

[4.4 中间件接口设计 25](#_Toc486881528)

[4.4.1 中间件业务接口 25](#_Toc486881529)

[4.4.2 中间件状态查询接口 27](#_Toc486881530)

[4.4.3 中间件接口流程设计 28](#_Toc486881531)

[第五章 云备份中间件的实现 30](#_Toc486881532)

[5.1 环境配置 30](#_Toc486881533)

[5.1.1 开发环境配置 30](#_Toc486881534)

[5.1.2 部署环境配置 30](#_Toc486881535)

[5.2 运行方式 31](#_Toc486881536)

[5.3 功能实现 31](#_Toc486881537)

[5.3.1通信方式 31](#_Toc486881538)

[5.3.2 会话管理 35](#_Toc486881539)

[5.3.3 解析模块 39](#_Toc486881540)

[5.3.4 数据处理模块 40](#_Toc486881541)

[5.3.5 数据传输模块 42](#_Toc486881542)

[第六章 总结和展望 46](#_Toc486881543)

[6.1 总结 46](#_Toc486881544)

[6.2 展望 46](#_Toc486881545)

[参考文献 48](#_Toc486881546)

[致 谢 51](#_Toc486881547)

# **摘要**

**关键词：**

# Abstract

**Keywords：**

# 第一章 绪论

## 1.1 背景介绍

## 1.2 相关研究工作

## 1.3 论文组织结构

# **第**二**章 核心技术**

## 2.1 中间件

## 2.2 加密算法

## 2.3 Java SDK

## 2.4 WebExtension

## 2.5 大文件分块上传/下载

## 2.6 断点续传

## 2.7 本地缓存

# 第三章 云备份中间件系统分析

## 3.1 系统整体架构

云备份系统主要包括五个部分：云备份基础设施（云备份服务器）、云备份应用程序（即云备份客户端）、身份认证服务器、密钥管理服务器、云备份中间件。这样区分的优势在于：

独立的身份认证系统有助于云备份系统在Linux操作系统生态环境中建立SSO机制，同时身份认证服务器和云备份服务器的独立运行，可以有效的防止用户身份泄露；

1. 云备份桌面客户端的独立开发允许客户端独立更改视图层代码，这样的低耦合性有助于促进客户端持续的进行更新与开发；
2. 云备份基础设施的独立开发将专注于云备份的可靠性、安全性等方面的研究，如果需要升级，或者更换服务器设备时，云备份服务器端只需要设计云端存储的接口层，并不影响上层的应用；
3. 云备份中间件的开发可以有效分离云备份桌面客户端与云备份服务器之间的绑定，使客户端和服务器端独立的进行开发或者升级。

## 3.2 功能需求分析

数据安全存储作为当今关注的焦点，云备份是避免数据丢失的最基本的方法之一。云备份中间件系统需要达到以下目标：

1. 全面支持Linux桌面操作系统；
2. 高效、稳定、安全的数据同步功能；
3. 安全的备份数据和历史记录，并且保证在传输和存储过程中的数据安全；
4. 提供身份验证机制、敏感数据保护、安全会话管理、安全漏洞检测等机制，最大程度上保护用户数据传输、存储的安全；

根据以上要求，云备份中间件主要需要实现的功能如下：

1. 数据传输功能：支持客户端与服务器端的信息交互，实现数据的上传与下载，并且向云备份客户端提供可调用API；
2. 压缩数据功能：为了加快传输速度，增加服务器端空间的利用率，中间件提供数据压缩功能；
3. 数据加密功能：为了保证数据安全，云备份中间件对用户上传的数据进行加密上传，以防数据被非法窃取；
4. 安全会话管理：云备份中间件通过安全会话管理模块管理令牌、并记录相应日志，保证了云备份客户端与服务器端信息交互的安全性。

## 3.3 核心模块分析

云备份中间件是基于Linux系统，用来连接云备份客户端和服务器之间的桥梁，它提供了客户端与存储服务器数据交互，包括文件的传输、加密和解压缩等功能，有效地分离了客户端和服务器之间的绑定。同时，云备份中间件也提供给云备份客户端与云备份服务器，身份认证服务器以及密钥管理服务器之间稳定可靠地数据传输。云备份中间件主要包括四个模块：数据传输模块、数据加密模块、数据压缩模块、安全会话管理模块。

### 3.3.1 数据传输模块

数据传输模块主要负责客户端与云备份服务器之间的数据交互，实现数据的上传与下载，其次数据传输模块需要向云备份客户端提供应用接口，保证客户端可以通过接口调用发送需要执行操作的文件列表，同时云备份服务器也能向客户端返回相关用户在服务器所存储的文件等一系列支撑云备份客户端相关应用功能的底层支持。数据传输模块设计要求API与实现分离，为将来云备份工具的扩展提供支持。

数据上传方式主要有两种：第一种，直接上传，对于较小的文件数据传输模块支持文件直接上传；第二种，分片上传，对于较大的文件，数据传输模块将较大文件切割为一系列编号的特定大小的临时文件，然后分别将这些已经编号的临时文件片上传至云备份服务器端，待所有临时文件片上传完成后再调用文件合并方法将所有的临时文件片按编号合并成为一个完成整的文件。相较于大文件直接上传而言，分片上传可以避免因为单个文件数据量过大而导致阻塞，并且当网络环境较差时，较小的临时文件数据片可以有较高的上传成功率，从而避免无休止的失败重传。

数据下载部分，通过待下载文件队列管理多个待下载文件。云备份中间件接收到来自客户端的下载请求后，调用云备份服务器端提供的下载API完成文件的下载。

### 3.3.2 数据压缩模块

为了提高云备份客户端的传输速度，增加云备份服务器的利用率，云备份中间件提供数据压缩功能，提高云备份的存储效率。并且对用户而言数据压缩过程是透明，即用户下载后的文件仍然是源文件。

文件压缩的具体过程是云备份客户端发出上传文件请求，经解析模块解析后发现请求中含有compress参数，数据压缩模块对指定文件进行压缩，并且返回执行结果，success/failed。

文件解压缩的具体过程是云备份客户端发出下载文件请求，经解析模块解析后发现待下载文件属性含compress参数，待数据传输模块将文件从服务器端下载至中间件后，调用数据压缩模块，将文件解压缩到指定位置，并且返回执行结果success/failed。

### 3.3.3 数据加密模块

由于网络系统的开放性、资源的共享性以及网络恶意攻击等各种原因，为了保证数据的安全性，需要对用户上传备份在云备份服务器端上的数据进行加密，以防数据被非法窃取，泄露用户的隐私。数据加密所需要的密钥由中科院软件中心统一签发和管理，该密钥与用户的唯一标识User\_ID一一对应。

文件加密过程为：当云备份中间件收到客户端发送上传文件请求后，经解析器模块解析后发现含有encrypt参数，则调用封装好的crypto库将文件进行加密后再进行传输，执行完成后返回结果success/failed。

文件解密过程为：当云备份中间件收到客户端发送下载文件请求时，首先调用云备份服务器端提供的获取文件属性API分析文件属性，若文件属性中含有encrypt参数，在云备份中间件下载文件后，调用crypto库对文件进行解密，执行完成后返回结果success/failed。

### 3.3.4 安全会话模块

云备份客户端与服务器端进行数据交互时需要采取安全会话管理机制保证数据交互的安全性，安全会话模块主要包括了会话的创建、传递和失效。

安全会话模块主要通过以下方式实现：首先云备份架构必须以可靠的方式产生用户令牌，并通过系统机制保障用户令牌从产生到废止整个生命周期内的安全管理，确保令牌不会泄露给除用户以外的其他人；其次，会话管理功能可以根据系统记录的日志进行判断，帮助用户识别风险，在适当时间规避风险。

云备份客户端具有用户注册的功能(用户注册时客户端需要提供client\_token)，用户注册并登录后才能使用云备份应用软件。在用户访问存储服务之前，首先需要获取该用户的访问令牌(user\_token)，该令牌可以接入软件中心身份安全管理服务。当客户端需要访问存储服务时，需要携带user\_token通过中间件换取token\_id和tenant\_id与云备份服务器进行交互。此外，每当客户端发出请求后，中间件需要对指令进行解析分类，如果是安全会话管理类的请求，则需要对token的行为进行相关记录，以记录异常行为。

用户在登录之后会在云备份服务器端生成相应的session会话，当客户端进行数据相关操作时会进行用户登录信息以及会话是否超时验证，如果用户登录信息不匹配或者超时，则会要求用户重新登录，验证通过后系统产生新的session来确保客户端使用过程的安全性。

## 3.4 数据结构分析

### 3.4.1 访问令牌

访问令牌包含了安全会话管理中登录需要的安全信息，其组成如图3-1所示：



图3-1 访问令牌

### 3.4.2 文件信息

文件上传和下载首先需要获取文件属性，然后进行操作，文件属性如图3-2所示：



图3-2 文件属性

# 第四章 云备份中间件的设计

## 4.1 系统总体框架

### 4.1.1 系统框架图

云备份中间件系统架构图4-1所示：



图4-1 云备份中间件系统架构图

通过同类技术或者产品的比较分析，云备份中间件设计主要分为三层，包括底层架构、基础功能层、以及接口层：

1. 底层架构主要为云备份中间件提供与网络、内存与本地存储、本地库等相关的硬件软件适配机制，以及为业务逻辑提供基础的算法库和安全机制；
2. 基础功能层主要包括了云备份中间件系统的四个核心模块（即安全会话模块、数据压缩模块、数据加密模块、数据传输模块），提供了数据传输、数据压缩、数据加密以及安全会话功能；为了提高云备份中间件的传出效率，增加了多线程管理模块和数据持久化模块，为云备份工具用户提供高效且稳定的数据交互功能。
3. 接口层是中间件为云备份客户端提供向上接口，连接云备份客户端和服务器，以便云备份客户端和云备份服务器之间的数据交互，客户端与服务器端的数据交互主要包括三类操作：信息处理、文件操作以及回收站操作。

## 4.2 类图设计

云备份中间件系统的类图设计如图4-2所示：



图4-2云备份中间件类图

云备份中间件类图设计如图4-2所示：云备份中间件收到来自客户端请求后，先将请求发送给控制模块，控制模块接着将请求发送给解析模块进行解析，控制模块根据解析结果中的参数判断需要进行何种操作，然后按照逻辑顺序对以下模块进行调用：

1. 安全会话管理模块

安全会话管理模块主要通过记录与令牌操作相关的日志，帮助云备份工具用户在必要时采取防御措施。完成令牌操作后，用户继续根据请求依次调用各个模块完成请求内容，并将请求结果返回给客户端。

1. 数据加密模块

数据加密模块由数据加密类组成。加密类中主要有两个函数：加密函数和解密函数。如果用户请求文件进行加密上传，中间件在接到请求后首先调用加密函数对指定文件进行加密，然后调用数据传输模块将文件上传至云备份服务器。如果用户请求下载的文件为加密文件，中间件调用数据传输模块下载完成后，接着调用数据解密模块对指定文件进行解密。数据加密和解密过程对于用户是透明的，云备份中间件在加密模块中选择了成熟的AES加解密算法保证加解密的安全性。

1. 数据压缩模块

数据压缩模块由压缩交互类和数据压缩类组成。压缩交互类主要作用是识别请求是未压缩请求还是解压缩请求，数据压缩类主要由压缩函数和解压缩函数组成。当中间件收到来自客户端压缩上传文件的请求时，首先调用数据压缩模块对指定文件进行压缩，接着调用数据传输模块将文件上传至云备份服务器；当用户收到来自客户端下载文件请求时，经调用获取文件属性API操作后发现文件为压缩文件时，先调用数据传输模块将指定文件进行下载，然后对文件进行解压缩操作。

1. 数据传输模块

数据传输模块主要由数据上传类，数据下载类以及状态查询类组成。

数据上传类中首先判断上传文件是否为大文件，若是，则根据分片上传规则将大文件分成固定大小的临时文件分片独立上传，上传完成后，在云备份服务器端调用进行合并；若否，直接调用上传函数将文件上传至服务器，请求执行完成后将执行结果返回给客户端。

数据下载类根据客户端指令调用数据传输模块对指定文件进行下载，若文件为加密文件或者压缩文件，下载完成后依次调用数据压缩模块和数据机密模块对文件进行解密和解压缩，请求执行完成后将结果返回给客户端。

中间件为客户端提供了状态查询接口，返回当前正在执行操作的进度。状态查询类主要通过状态查询函数返回当前操作的进度。

1. 数据持久化模块

数据持久化模块主要是为了解决用户在终端设备上使用云备份客户端过程中由于异常出现的中间件被强行关闭的状况，数据持久化类中设计一个持久化线程不断扫描数据库中是否有未返回结果的请求，若有，则执行未返回结果的请求，并将执行结果返回给客户端，接着继续扫描数据库。

## 4.3 系统流程设计

### 4.3.1 系统总体流程设计

云备份中间件总体流程图4-3所示：



图4-3 云备份中间件流程图

云备份中间件系统收到客户端发送的请求后，首先由控制模块将请求发送给解析器模块，经解析器模块解析后将解析结果返回给控制模块。如果解析结果中含有与安全会话相关的指令则调用安全会话模块（若是与身份认证有关的请求则调用身份认证服务器API，并将结果返回给安全会话模块；若是与令牌转换相关，则转换令牌，并将最终结果返回控制模块）；如果解析结果含有与数据压缩有关的命令，调用数据压缩模块进行相应的压缩或解压缩操作，并将结果返回给控制模块；如果解析结果中含有与数据加密的指令，调用数据加密模块，进行相应的加密或者解密操作，并将结果返回给控制模块；如果解析结果含有与数据传输相关的指令，调用数据传输模块（若是查询当前上传或下载状态，调用查询传输状态函数，并将结果返回给控制模块；若是大文件上传则调用分片上传函数，将大文件分成固定大小的临时小文件后分片上传，并将执行结果返回给控制模块；若是普通文件上传或下载则调用传输模块上传或下载函数，并将结果返回给控制模块）；如果解析结果中有其他指令，直接调用服务器交互模块，接着调用云备份服务器API进行相应操作，将执行结果返回给服务器交互模块，再由服务器交互模块将结果发送给控制模块。最后，控制模块将收到的结果发送给客户端交互模块，再由客户端交互模块将结果发送给客户端。

### 4.3.2 安全会话管理流程设计

#### 4.3.2.1 多用户登录模块设计

云备份工具在使用过程中可能会面临同一个终端设备上多个客户端请求接入的状况，当云备份中间件面对多个客户端发送的请求时，如何将请求与用户一一对应并将请求执行结果准确地返回给客户端是中间件实现过程中必须要解决的问题。

由于云备份中间件需要同时处理多个连接状态的socket，为了将数据准确的发送给对应用户，因此云备份中间件通过维护一张userID（云备份工具中用户的唯一标识）与socketID对应的哈希表，达到响应不同用户目的。

多用户登录同一个终端时，在终端上虽然有多个客户端，但是只有一个云备份中间件，这时中间件需要对各个socket连接进行监听，找出已经准备就绪的描述符（sockfd），中间件可以采用多线程或者I/O多路复用技术实现。多线程实现监听主要是通过多个线程与各个socket一一对应的方法，这样的好处是请求响应时间较短，但是多线程的系统开销较大，当终端上有大量用户登陆时严重影响系统的执行效率。另一种解决方式就是I/O多路复用技术，即系统内核缓冲I/O数据，当某个I/O数据准备好后，立刻通知客户端该I/O可读或者可写，这样云云备份客户端可以立刻完成相应的I/O操作，客户端可以不必因等待I/O操作而阻塞。同时相较于多线程技术实现多用户登录I/O多路复用技术具有系统开销小，不用额外创建或维护线程的优点。

云备份中间件利用select模式实现I/O多路复用代码实现如下所示：

|  |
| --- |
| #select函数以轮询的方式获取已经读就绪的socket  read\_sockets,write\_sockets,error\_sockets = select.select(CONNECTION\_LIST,[],[])  for sock in read\_sockets:  #如果socket是server监听到的可读事件队列  if sock == server\_socket:  # 同意建立一个新连接  sockfd, addr = server\_socket.accept()  #将连接加入到select可读事件队列  CONNECTION\_LIST.append(sockfd)  #若不是本机监听的socket而是客户端发送的数据消息  else:  data=’’  try:  #接收客户端发送的数据  recvdata = sock.recv(RECV\_BUFFER) |

云备份中间件通过I/O多路复用技术获得当前活跃的描述符后，中间件执行来自客户端的请求，请求执行结束后，中间件控制模块获得来自服务器的执行结果，这时中间件需要通过socket传输方式将执行结果返回给指定用户。这时通过查询userID，socketID哈希表获得指定用户对应的socketID，接着通过此socket将执行结果返回给用户。

#### 4.3.2.2 数据持久化设计

用户在使用云备份工具过程中可能会出现中间件因异常状况被强行关闭的状况，用户重新登录时，客户端与中间件通过socket重新建立连接，中间件继续执行由于客户端异常关闭未完成的请求。根据此需求可知，云备份中间件需要长期存储客户端发送的未完成请求，以便在用户重新登录时继续执行未完成任务。

云备份中间件为了实现持久化的目标使用数据库存储客户端未完成的请求，考虑到中间件是部署在客户端上的，因此选择稳定且轻量级的数据库SQLite，通过数据库记录客户端未完成请求在目前看来是较好的解决方法。若中间件因异常状况被强行关闭，在下次重启时，中间件自动从数据库中检索是否有未执行完毕的请求，若有，则重新执行该请求。

云备份中间件根据自动执行未完成的纪录需求，SQLite数据库中存储的记录首先应该包括请求的内容，请求执行到的状态（请求还未开始执行，请求执行过程中但是还没有执行完成、请求执行完成返回执行结果），数据库收到该请求的时间（请求执行完毕后保留该记录一个月，方便用户对操作历史进行查询），请求的返回结果（保留返回结果，方便用户查找操作结果）。

根据以上需求，数据库主要存储的每一条记录应该包括：整个请求内容、请求的md5值、请求当前执行的状态（"unfinished","not return","finished"）、收到该事件的时间戳、该请求对应的用户唯一标识userID以及该请求的执行结果。

#### 4.3.2.3 多线程传输设计

在云备份工具使用过程中，用户可以根据个人需求从客户端发出上传或者下载多个文件，以及创建文件夹，合并文件夹等请求，若此时中间件只有一个线程处理这些请求可能会出现由于单个请求耗时过长或者运行过程中出现异常以致阻塞用户操作，降低中间件的效率。因此，中间件为了提高性能设置了多线程处理机制，可以同时处理用户多个请求。多线程处理机制主要是将请求划分为多个独立的任务，多个线程同时处理，使请求的响应速度更快，减少因请求任务占用大量处理时间以致后续任务阻塞状况。

中间件多线程实现可以分为工作线程和轮询线程。工作线程是处理上传、下载、创建文件夹等请求，轮询线程主要是为防止中间件因异常而关闭的状况，轮询线程不断扫描数据库，查看是否有上次未返回结果的请求，若有，则执行上次未完成的请求，请求执行完成后将相应结果返回给客户端，接着继续扫描数据库，查看是否有未返回结果的请求。

Python中提供了两个模块来实现多线程，thread和threading模块，thread模块是比较底层的，而threading模块对thread进行了一些包装，可以更加方便的被利用。在云备份中间件系统中主要使用threading模块来实现多线程处理机制。

在Python中threading模块创建多线程主要分为两类：一种是通过threading.Thread来构造thread对象；另一种是直接从threading.Thread继承，重写\_\_init\_\_方法和run方法，在中间件中通过第二种发发实现多线程功能，threading.Thread的初始化原型函数为：def \_\_init\_\_ (self, group=None, target=None, name=None, args=(), kwargs={})，其中参数group是预留的，用于将来扩展；参数target是一个可调用对象（也称为活动[activity]），在线程启动后执行；参数name是线程的名字；默认值为“Thread-N“，N是一个数字参数args和kwargs分别表示调用target时的参数列表和关键字参数[34]。

Threading模块主要提供以下函数或者功能：

1. Threading.Lock & threading.RLock，threading.RLock允许同一个线程多次获取锁，而threading.Lock只能获取一次锁；
2. Threading.Condition 为用户提供了解决线程同步的方案，通过维护一个锁对象，在使用过程中调用内部锁对象对应的方法。
3. Threading.Event 通过维护内部的标识符来时先线程间的同步问题。
4. Semaphore控制进程同步互斥的量，保证各个线程能够正确、合理的使用公共资源；
5. BoundedSemaphore 与semaphore类似，但不允许超过初始值；
6. Queue：实现了操作系统同中生产者消费者模式的锁队列，能够在多线程模块提供同步支持[35]。

Threading模块中Thread类还提供了下列常用属性与方法[36]：

1. Thread.getName()获取当前线程名称
2. Thread.setName()设置线程名称
3. Thread.ident获取线程的标识符（在调用了start()方法以后）；
4. Thread.is\_alive() 判断线程是否是激活的。例如调用start()方法启动线程，一直到run方法执行完毕或者遇到没有处理的异常中断，整个期间线程处于alive状态；
5. Thread.join([timeout]) 此函数将会阻塞主调用线程，一直到被调用线程执行完成或者超时，timeout参数表示超时的时间，在规定的timeout时间内阻塞主线程 [37]。

云备份工具启动后，中间件以守护进程的方式运行在客户端，通过Python中的threading模块创建六个工作线程和一个轮询线程，工作线程不断轮询用户请求队列，获取请求内容并执行，轮询线程则不断扫描数据库寻找未被执行的请求，然后执行请求并向客户端返回执行结果。通过此种方式提高云备份中间件传输的效率，避免因上传大文件或者其他异常状况阻塞用户操作。

### 4.3.3 数据传输模块流程设计

#### 4.3.3.1 数据上传流程设计

云备份中间件的数据上传流程设计如图4-4所示：



图4-4 数据上传流程图

数据传输模块接收到控制模块的文件上传指令后，首先根据文件大小判断文件需要进行正常上传或者分片上传，若是正常上传，则调用上传函数将文件上传至云备份服务器，若是分片上传，则将文件按照需求分成固定大小的临时文件，然后将分别将临时文件上传至云备份服务器，上传完成后调用云备份服务器端的合并方法将临时文件合并为最终文件，最后服务器交互类将上传结果返回给上传类，然后上传类将上传结果发送给控制模块。

数据传输模块接收到控制模块的查询数据传输状态指令后，直接调用查询状态类，查询状态类将获得的执行状态结果返回给控制模块。

云备份中间件设计的文件分片上传接口表4-1上传临时文件所示，数据上传过程中分为分片上传和正常上传，如果上传大文件则调用文件分片上传接口。

表4-1 上传临时文件接口

|  |  |  |
| --- | --- | --- |
| 接口编号 | 接口名称 | 上传临时文件接口 |
| MW-AC-1 | 接口定义 | 中间件为客户端提供分片上传支持 |
| 接口设计 | curl -k -i -X PUT -T <LOCAL\_FILE> "https://IP:Port/v1/AUTH\_{Tenant\_id}/<segments\_path>?op=CREATE [&overwrite=<true|false>] [&type=<NORMAL|PRIVATE|BACKUP>][&metadata=<STRING>] [&mode=<NORMAL|ENCRYPT| COMPRESS|ENCRYPT\_COMPRESS>][&storetype=<USER|CORRESPONDING>]"-H "X-Auth-Token: { Token\_id}" --cacert capath |
| 传入参数 | LOCAL\_FILE:上传的文件  segments\_path：上传至segments容器下的路径  Overwrite：是否覆盖相同的文件  Tenant\_id：租户id  Token\_id：租户对应的token的id  metadata：用户数据标签  type: 数据空间类型  mode: 上传模式  storetype: 存储模式  capath:ssl验证证书路径 |
| 返回结果 | path string 该文件的用户空间相对路径。  size uint64 文件字节大小。  ctime uint64 文件创建时间。  mtime uint64 文件修改时间。  md5 string 被上传文件的md5签名。 |
| 传入参数示例 | curl -k -i -X PUT -T /home/ler/fruits "https://10.3.3.236:443/v1/AUTH\_Jimmy2163com/segments/fruits?op=CREATE&overwrite=false&type=NORMAL&metadata=forsegment&mode=COMPRESS&storetype=USER" -H "X-Auth-Token: uYoPztVKoLPS99pBnBUqoXjiWrJThKobIE2BeVkt" |
| 返回结果示例 | 示例：  201Created. {"path": "/segments/fruits", "size": "13", "mtime":"1435075275.92182","ctime":"1435075275.92182", "md5": "0617e4990505c02f6f39e9a351823536"} |

文件合并接口设计如表4-2所示，将文件分片上传完成后，需要对文件进行合并，调用文件合并接口将各个临时文件按照编号合并为完整大文件。

采用分片上传的方式可以有效的解决因单个文件数据量过大阻塞用户操作的状况并且在网络较差的状况下增加上传成功的几率。云备份中间件通过将大文件分为带有编号的固定大小的临时文件，在上传完成后，通过调用文件合并借口将文件进行合并，这样在云备份服务器端的文件仍是完整的大文件。

表4-2 合并临时文件接口

|  |  |  |
| --- | --- | --- |
| 接口编号 | 接口名称 | 合并临时文件接口 |
| MW-AC-2 | 接口定义 | 中间件合并客户端上传的临时文件 |
| 接口设计 | curl -k -i -X PUT -d  '[  {  "path": "/segments/seg\_test/1",  "etag":"9951ef01bc03745e6ebf3b50e990bc67",  "size\_bytes":7572  },  {  "path":"/segments/seg\_test/2",  "etag":"8dd16a3d50854caae6a23917d41688f3",  "size\_bytes": 27312  }  ]'  "https://IP:Port/v1/AUTH\_{Tenant\_id}/<PATH>createsuperfile?multipart-manifest=putMERGE[&overwrite=<true|false>]" -H "X-Auth-Token: { Token\_id}" -H "X-Static-Large-Object: <True|false>" --cacert capath |
| 传入参数 | path:原文件路径  etag：文件分片md5值  size\_bytes:文件分片大小  Overwrite：是否覆盖相同的文件  Tenant\_id：租户id  Token\_id：租户对应的token的id  X-Static-Large-Object:参数未置说明  capath:ssl验证证书路径 |
| 返回结果 | path string 该文件的用户空间相对路径。  size uint64 文件字节大小。  ctime uint64 文件创建时间。  mtime uint64 文件修改时间。  md5 string 被上传文件的md5签名。 |
| 传入参数示例 | curl -k -i -X PUT -d '[{"path": "/normal/apple","etag":"eb0880af743dbc4a2528ef8a8c977f6e","size\_bytes":14},{"path": "/segments/fruits","etag":"0617e4990505c02f6f39e9a351823536","size\_bytes":13}]' "https://10.3.3.236:443/v1/AUTH\_Jimmy2163com/normal/Merfile?multipart-manifest=put&overwrite=true" -H "X-Auth-Token: uYoPztVKoLPS99pBnBUqoXjiWrJThKobIE2BeVkt" -H "X-Static-Large-Object:false" --cacert /etc/ssl/certs/ca-certificates.crt |
| 返回结果示例 | 示例：  {"path": "/normal/Merfile", "size": "171", "mtime": "1435076689.24329", "ctime": "1435076689.24329", "md5": "bbf93cfecccf9f06b9854e5e31d84616"} |

#### 4.3.3.2数据下载流程设计

云备份数据下载流程设计如图4-5所示：



图4-5 数据下载流程图

数据传输模块接收到文件下载指令后，首先调用获取文件属性接口，将获取的文件属性返回给控制模块。控制模块根据文件属性判断文件是否加密，是否压缩，按照逻辑顺序依次调用数据传输模块、数据压缩模块和数据加密模块。数据传输模块进行数据下载时，由下载类将传入的下载参数发送给服务器交互模块进行下载，服务器交互模块将下载结果返回给下载类，控制模块接着调用解密或者解压缩模块对文件进行解密或者解压缩操作，并将执行结果返回给控制模块。

### 4.3.4 数据压缩模块流程设计

#### 4.3.4.1 数据压缩流程设计

云备份中间件系统数据压缩流程设计如图4-6所示：



图4-6数据压缩流程图

控制模块调用数据压缩模块执行压缩文件操作，数据压缩模块收到压缩参数（待压缩文件路径、压缩后文件路径）后，将压缩参数发送给压缩文件类，压缩文件类执行压缩操作并将执行结果返回给压缩交互类，并最终压缩结果发送给控制模块。

#### 4.3.4.2 数据解压缩流程设计

云备份中间件系统数据解压缩流程如图4-7所示：



图4-7数据解压缩流程图

控制模块将客户端发送的解压缩请求发送给数据压缩模块，数据压缩模块收到解压缩参数（待解压缩文件路径，解压缩后文件路径）后，通过解压缩交互类将解压缩参数发送给压缩文件类，压缩文件类接着执行解压缩操作并将执行结果返回给解压缩交互类，最终将解压缩操作执行结果发送给控制模块。

### 4.3.5 数据加密服务流程设计

#### 4.3.5.1 数据加密流程设计

云备份中间件加密流程设计如图4-8所示：



图4-8 数据加密流程图

控制模块得到数据加密指令后，调用数据加密模块，数据加密类调用crypto库加密数据，并将结果返回给控制模块。

#### 4.3.5.2数据解密流程设计

云备份中间件数据解密流程设计如图4-9所示：



图4-9数据解密流程图

控制模块得到数据解密指令后，调用数据加密模块，数据加密类调用crypto库解密数据，并将结果返回给控制模块。

## 4.4 中间件接口设计

云备份中间件部署在客户端，并且以守护进程的模式工作，在指定端口监听客户端发送的请求，以完成客户端与服务器端的通信。

中间件主要为客户端提供两份接口：首先是为客户端提供通用接口，其次是为获取传输状态提供的状态查询接口。

### 4.4.1 中间件业务接口

云备份客户端请求中间件服务的过程如下：1、云备份客户端将用户的操作转化为相应的指令；2、应用程序将指令添加用户的唯一标识Tenant\_ID，User\_ID和timestamp封装为相应报文发送给中间件；3、中间件收到来自于应用程序的报文后，首先交由解析模块进行解析，再由控制模块按照逻辑顺序执行请求内容；4、请求执行完成后将结果以json数据格式返回给应用程序。中间件业务接口的示例如表4-3所示:

表4-3 中间件业务接口表

|  |  |  |
| --- | --- | --- |
| 接口编号 | 接口名称 | 中间件接口 |
| MW-IF-1 | 接口定义 | 中间件在设定端口提供服务 |
| 传入参数 | 请求报文，报文格式如下所示：  client\_request = 'userid = ' + Tanent\_id + '\n' + 'time = ' + Timestamp + '\n' + 'request = ' + Request + '\r\n'  其中：Tanent\_id为该请求对应租户的Tanent\_id值；Timestamp为请求发送时的时间戳，以字符串形式发送且精度要求小数点后四位. |
| 返回结果 | 返回结果是字符串格式的json数据，返回格式为：  client\_result = ‘uuid = ’+ UUID + '\n' + 'mwresult = ' + Result + '\r\n'  其中：UUID为所对应请求的md5值，Result为对应请求的执行结果，执行结果是返回以json格式封装的返回结果。 |
| 通信 | 网络连接（socket） |
| 传入参数示例  （获取文件属性） | "https://IP:Port/v1/AUTH\_{Tenant\_id}/<PATH>?op=GETFILEATTR[&type=<NORMAL|PRIVATE|BACKUP>][&version=<LATEST|指定版本>]" -H "X-Auth-Token: { Token\_id}" --cacert capath  传入参数的定义是：  path：文件路径  Tenant\_id：租户id  Token\_id：租户对应的token的id  capath:ssl验证证书路径 |
| 返回结果示例  （获取文件属性） | 示例：  返回字符串格式的json数据：  {  "Content-Length": "116007",  "X-File-Type": "f",  "ETag": "d34e3d609811b44e56e07e04c1fe7ec3",  "X-Timestamp": "1432653661.66864",  "X-Object-Permisson": "777"  "metadata": "xxxx"  } |
| 说明 | 该API会将当时请求时的req字符串中的url作为返回结果的一部分给客户端，需要客户端在请求时自行记录相应请求的req字符串中的url，以便标记返回结果。对于传入参数定义中的’token\_id’和’tenant\_id’，客户端不需要传入具体值，只需要把’token\_id’和’tenant\_id’当字符串传给中间件即可，便于中间件根据客户端传入的user\_token填入’token\_id’和’tenant\_id’具体值，进而跟服务器进行通信。 |

云备份中间件在于客户端进行通信的过程中选择socket进行数据交互，涉及到的函数如表4-4所示：

表4-4 socket接口函数

|  |  |  |
| --- | --- | --- |
| 功能 | 函数 | 说明 |
| 创建socket对象 | socket.socket(family,type) | Family的取值可以是AF\_UNIX (Unix域，用于同一台机器上的进程间通讯)或者AF\_INET（适用与IPV4协议的TCP和UDP），socket参数可以选取OCK\_STREAM（流套接字），或者 SOCK\_DGRAM（数据报文套接字）, SOCK\_RAW（raw套接字） |
| socket与地址进行绑定 | socket.bind(address) | 这里的address（host，post）包括了主机地址和端口号，若主机地址错误或者端口号被占用，则会引发socket.error异常； |
| 准备socket  监听端口 | socket.listen(connect\_num) | 这里的connect\_num表示最大连接数，至少为1，服务器接到链接请求后，把请求放入socket队列，如果队列已满，服务器拒绝请求； |
| 服务器等待客户端请求一个连接 | connection,address=  socket.accept() | 调用accept方法后，socket会进入阻塞状态。云备份客户端向中间件请求连接时，accept方法返回（connection ，address），其中connection是新的socket对象，服务器通过它与客户端通信，address是客户端的internet地址 |
| 数据发送与接收 | socket.send()  socket.recv() | Send函数返回已发送的字符个数，云备份中间件使用recv函数从客户端接受信息，调用recv函数时必须设置一个缓冲区，防止由于字符串长度过长被删除； |
| 关闭socket连接 | close() | 数据交互完成后，调用close方法关闭连接。 |

### 4.4.2 中间件状态查询接口

云备份中间件执行上传与下载请求过程中，用户希望能够看到实时的传输状态，包括传输速度、剩余传出时间等内容，中间件为应用程序提供了状态查询接口，以便用户查看当前作业完成状况，中间件状态查询接口设计如表4-5所示：

表4-5 中间件状态查询接口表

|  |  |  |
| --- | --- | --- |
| 接口编号 | 接口名称 | 中间件状态查询接口 |
| MW-SW-1 | 接口定义 | 中间件为应用程序提供状态查询接口 |
| 传入参数 | logpath = uuid + '.log'  其中 uuid = hashlib.md5(client\_request).hexdigest()；  其中 client\_request即为客户端发给中间件的请求：  client\_request = 'userid = ' + Tanent\_id + '\n' + 'time = ' + Timestamp + '\n' + 'request = ' + Request + '\r\n'； |
| 返回结果 | 返回信息主要包括八个并且以json的形式给出：  TotalPercent //当前下载文件的百分比  Current Speed //当前下载文件的速度  Total //下载文件的大小  Received //接收到文件的大小  AverageDload //平均下载速度  SpeedUpload //平均上传速度  Time Spent //总的时间花费  Time Left //下载剩余时间 |
| 通信 | 网络连接（socket） |
| 传入参数示例  （获取下载状态） | 'userid = ' + Tanent\_id + '\n' + 'time = ' + Timestamp + '\n' + 'request = ' + Request + '\r\n' |
| 返回结果示例  （获取下载状态） | 示例：  返回示例：（下载一个69.4M完成时的进度信息反馈）  {"TotalPercent":"100","Current Speed":"11.2M", "Total": "69.4M", "Received": "69.4M", "AverageDload": "11.0M", "SpeedUpload" : "0","Time Spent":"0:00:06 ","Time Left":"--:--:--"} |

### 4.4.3 中间件接口流程设计

云备份系统中间件的中间件接口设计流程图如图4-10所示：



图4-10 中间件接口流程图

云备份客户端启动时，中间件首先开启守护进程，然后创建socket对象、建立连接并调用监听函数实时监听客户端请求报文，接着等待用户发出请求，当客户端关闭或者用户退出时，云备份中间件自动关闭socket。

云备份客户端发送请求时，根据请求的种类调用不同中间件业务接口或者中间件状态查询接口，中间件调用接收函数接收来自客户端的请求时；若将消息返回给客户端时，调用发送消息函数将结果返回给云备份客户端。

# 第五章 云备份中间件的实现

## 5.1 环境配置

### 5.1.1 开发环境配置

为了开发云备份中间件，配置开发环境如表5-1所示：

表5-1 开发环境配置

|  |  |
| --- | --- |
| 类别 | 配置 |
| 云备份系统运行环境 | Linux系统 |
| 开发环境 | Linux 14.04 |
| 开发工具 | Sublime |
| 开发语言 | Python 2.7.6 |

### 5.1.2 部署环境配置

云备份系统的部署图如图5-1所示：



图5-1云备份中间件部署图

云备份工具主设计为五个部分：云备份应用程序（客户端），云备份中间件，云备份服务器，身份认证服务器，密钥管理服务器。云备份客户端和云备份中间件绑定在一起并且部署在终端设备，客户端通过中间件与云备份服务器进行数据交互。

## 5.2 运行方式

云备份中间件以运行守护进程为客户端和服务器提供稳定的服务，终端设备运行云备份客户端后，除非强行关机或者杀死中间件进程，否则中间件会一直在后台运行。

中间件的守护进程在实现过程中的主要代码如下所示：

|  |
| --- |
| #通过fork()创建子进程，exit()方法退出子进程  pid = os.fork()  if pid > 0:  sys.exit(0)  #setid()方法创建新会话，使中间件进程完全独立  os.setsid()  #chidir(“/”)设置工作目录  os.chdir("/")  #unmask(0)重设文件权限掩码  os.umask(0)  #close()关闭文件描述符，保证与父进程断开  pf.close() |

## 5.3 功能实现

### 5.3.1通信方式

#### 5.3.1.1中间件与客户端的交互

云备份中间件部署在云备份客户端。由于在终端上允许多个用户同时登录，但是一个终端上只有一个中间件，因此云备份中间件与客户端之间的交互需要通过网络通信来实现，这里我们选择socket进行通信，传输数据。云备份中间件为客户端提供一个发送请求的接口，客户端可以通过socket将请求发送给中间件，中间件将得到的执行结果通过socket发送给客户端。

socket为特定网络协议（比如TCP/IP,ICMP/IP,UDP/IP）建立通信通道，允许不同进程之间进行连接，发送和接受数据，socket作为BSD UNIX系统核心的一部分，也被其他系统所支持。Python提供socket模块，利用socket模块所包含的函数和定义，可以实现客户端和中间件之间的交互，在实现过程中主要包含六个步骤：1、创建socket对象；2、socket与地址进行绑定；3、监听socket端口；4、服务器等待客户端请求一个连接；5、数据发送与接受；6、关闭socket。

由于Socket网络通信具有双向性，云备份中间件与客户端的交互涉及到两个接口：云备份客户端接口和中间件接口。

1）云备份中间件客户端接口实现流程为：用户启动云备份应用程序后，客户端创建socket对象，并建立连接。当用户进行操作转化成相应指令时，socket将相应报文发送给中间件，当云备份中间件发现token失效或者应用程序关闭等状况，将会自行关闭socket。

根据已经设计的业务接口和状态查询接口的报文格式，请求报文的长度小于2096个字符，所以不用额外设置缓冲区域；但是中间件将请求结果返回给客户端时，字符串长度可能会超过2096个字符，因此设置了消息缓冲区，以便接受任意长度的返回结果。

2）云备份中间件接口实现流程为：socket是双向连通的（即云备份客户端和云备份中间件），客户端启动时，中间件首先开启守护进程，然后创建socket对象、建立连接并调用监听函数实时监听客户端请求报文，并且调用accept函数等待用户发出请求，当客户端关闭或者用户退出时，云备份中间件自动关闭socket。

云备份中间件接收来自客户端的请求时，调用recv函数，由于请求报文不超过2096个字符，因此不用创建缓冲区，但是从云备份服务器端接受的返回结果可能超过2096个字符，因此需要创建额外的缓冲区存放返回结果，接着调用send()函数将返回结果发送给云备份客户端。

#### 5.3.1.2 中间件与服务器的交互

云备份系统中间件接收到来自客户端的请求后，需要对请求进行处理并且与云备份服务器进行数据交互，云备份服务器提供以curl命令为基础的数据传输接口，在数据传输过程中，中间件与服务器（密钥管理服务器、身份认证服务器、云备份服务器）间的通信主要是通过http协议实现。

云备份中间件与各个服务器之间的数据交互方式如下所示：

1）身份认证服务器：身份认证服务器负责用户令牌签发，令牌时效管理等任务。密钥管理服务器也需要与其进行交互。通过与客户端、服务器开发人员的调研和讨论，与身份认证服务器的交互任务由客户端直接进行，不需要通过中间件进行转接。

2）密钥管理服务器：为保证用户数据的安全性，为每个用户分配一个唯一密钥是必要的。密钥管理服务器提供的功能主要有：第一，中间件在数据加密时携带用户的访问令牌与密钥管理服务器通信，密钥管理服务器会将数据加密完毕并返回给中间件，为了保证用户密钥的绝对安全，用户的唯一密钥中间件与客户端无法获取。由此可知，为了提高数据保障安全性和可靠性，只需要在数据加密阶段才会涉及到中间件与密钥管理服务器间的交互；第二，当客户端携带用户令牌发给中间件请求时，中间件需要携带此令牌与身份认证服务器进行交互，以换取之后用于和云备份服务器进行交互的用户存储令牌。中间件实现的与密钥管理服务器交互进行令牌转换的接口函数如表5-2所示：

表5-2 令牌转换接口函数

|  |  |
| --- | --- |
| 功能 | 传入access\_token:user\_token完整信息, 获取云备份tenant\_id及access\_token:user\_token |
| 函数名 | Bool convert\_ukey2\_cloudtoken(char\* input, char\* output); |
| 输入参数说明 | 参数input请给出access\_token:user\_token完整信息（原始串需要中间件从客户端获取）:  {  "access\_token":"access\_token:user\_token",  "token\_type":"token类型",  "expires":失效时间,  "expires\_in":有效时间,  "refresh\_token":"令牌刷新口令"  } |
| 返回结果 | tenant\_id和access\_token:user\_token（仅为token\_id） |

3）云备份服务器：客户端所发送的请求，如上传文件、删除文件和读取文件等，中间件在进行处理后，需要将处理后的请求与云备份服务器进行通信，服务器响应之后将返回结果给中间件，中间件处理后返回给客户端。

由于云备份服务器提供的接口是curl数据传输接口，中间件实现时，需要生成符合服务器所提供的接口规则命令，然后执行该curl命令，对于返回的结果处理方式是：如果是不需要经过中间件处理的命令，如查询配额信息，删除文件和目录，重命名文件和目录等，中间件可以通过python调用os.popen()方法得到结果，然后通过socket返回给客户端；如果是需要中间件进行处理的命令，中间件可以重定向至文件中进行后续处理，如数据上传下载时所需要的实时状态信息，中间件会通过在生成的curl命令后增加' 2>&1 | tee + logpath'字段开启输出流，然后与云备份服务器通信，并将数据的传输状态结果实时地重定向到文件中，客户端可以在传输过程的任何时候调用中间件提供的查询接口来查询状态。

云备份服务器提供的curl接口示例如表5-3所示（以数据上传为例）：

表5-3 服务器上传文件接口

|  |  |
| --- | --- |
| 接口名称 | 上传文件 |
| 接口定义 | curl -i -X PUT -T <LOCAL\_FILE> "http://IP:Port/v1/AUTH\_{Tenant\_id}/<PATH>?op=CREATE [&overwrite=<true|false>] [&md5=原文件的MD5值] [&type=<NORMAL|PRIVATESECRET|BACKUP>][&metadata=<STRING>][&mode=<NORMAL|ENCRYPT|COMPRESS|ENCRYPT\_COMPRESS>][&storetype=<USER|CORRESPONDING>]"-H "X-Auth-Token: { Token\_id}" |
| 传入参数定义 | LOCAL\_FILE: 上传的文件  PATH： 上传至路径  Overwrite： 是否覆盖相同的文件  Tenant\_id： 租户id  Token\_id： 租户对应的token的id |
| 返回结果 | path string 该文件的用户空间相对路径。  size uint64 文件字节大小。  ctime uint64 文件创建时间。  mtime uint64 文件修改时间。  md5 string 被上传文件的md5签名。 |
| 协议 | http/https |
| 示例 | curl -i -X PUT -T ufw.log "http://IP:Port/v1/AUTH\_e8b18580e5b44cb79b10bd0f7a03bbdc/temp/ufw.log?op=CREATE&overwrite=true&type=NORMAL" -H "X-Auth-Token: 22ec6a37ed6b4b23a435e6b9050f21a4" |

中间件通过处理客户端发来的请求，以上述类似的方法生成对应云备份服务器所提供的34个接口对应的curl命令，然后与服务器进行交互，均得到了预想的结果，将结果进行处理后，通过与客户端进行交互传给客户端，完成了一次完整的云备份系统通信过程。

### 5.3.2 会话管理

#### 5.3.2.1 多用户登录功能实现

在云备份工具使用过程中，多用户可能同时登录统一终端，这时会启动多个客户端但是只会启动一个云备份中间件，中间件需要对多个用户建立的链接进行监听，找到其中活跃的链接，进行非阻塞的I/O操作。

首先，用户在启动客户端后，中间件也以守护进程的模式运行在客户端，中间件成功运行后将用户的userID与对应socketID添加到哈希表中，在将添加之前，首先要遍历整张用户\_socket表，查看表中是否已经存在相应的userID与socketID键值对，若有，则忽略此操作，若无则将userID和socketID以键值对的形式添加至哈希表中。

将userID与socketID对应表添加至哈希表后，云备份中间件需要监听来已经建立的socket连接，找出已经准备好的sockfd。本文中选择select模式来实现I/O复用功能，即使用select函数监听监听多个描述符事件，然后找出已经就绪的I/O操作。在Python中的select模块以列表的形式接收四个参数，select.select（rlist, wlist, xlist[timeout]）共有四个参数，第一个参数rlist是为等待读取的对象，第二个参数wlist是等待写入的对象，第三个参数xlist是等待异常的对象，最后一个参数是可选对象，指定最长等待时间，表示以秒为单位的超时时间。select函数的返回一个准备好的对象三元组，在指定时间内没有准备好对象，则返回一个空的列表。

综上所述：select函数实现I/O多路复用的步骤如下：

1. 首先清空描述符集合；
2. 其次建立坚实的描述符与描述符集合的关系；
3. 系统调用select函数；
4. 不断检查select函数监视的描述符是否已经准备好；
5. 挑选可读或者可写的描述符进行I/O操作。

云备份中间件通过select模式轮询找到活跃的socket连接后，如果监听到客户端传来请求，中间件根据请求的解析结果依次执行加密、压缩或者传输等操作，操作执行完成后，中间件将获得返回结果通过socket通信发送给相应用户。在确定使用哪一个socket时需要通过遍历之前保存的用户\_socket哈希表，查找指定用户对应的socket，最后将返回结果发送给指定用户，从而保证了数据传输的准确性。云备份中间代码实现的主要过程主要如下所示：

|  |
| --- |
| #云备份中间件监听到客户端发送请求  if recvdata:  data = data + recvdata  ##检测请求是否已经传输完成  dataend = data.find('\r\n')  if(dataend != -1):  clientrequest = data[:dataend+2]  data = data.replace(clientrequest,'')  #通过用户发来的request获得用户唯一标识userid  userid = getUserId(clientrequest)  #记录用户userid对应的socket  if (soc\_uuid\_dict.get(userid)==None):  soc\_uuid\_dict[userid]=sock |

综上所述，多用户登录模块的实现过程主要是：首先在用户登录时将用户唯一标识userid和socketID添加到一一对应的哈希表中；然后，云备份中间运行过程中通过使用select模式对当前的read\_sockets进行轮询，找到一个或多个准备好的描述符（sockfd），接着进行I/O操作；接着，云备份中间件执行来自客户端的请求，执行完成后得到来自服务器端的返回结果；最后，通过userid\_socketID哈希表找到返回结果对应的socket，将返回结果发送给指定用户。

#### 5.3.2.2 持久化的实现

云备份中间件在使用过程中可能因外部异常而突然关闭，关闭时中间件中可能有正在执行的请求或等待执行的请求，用户希望在下次登录中间件时，继续执行未返回结果的请求。因此云备份中间件需要提供数据持久化功能，即保存用户未返回记过的请求，以便用户重新登录时自动执行未完成请求。

云备份主要采用SQLite3将云备份与云备份工具客户端发送的每一条请求记录在数据库中，存储目录为/var/log/mwlog.db。每条请求记录的属性为request, uuid, state, time, userid, result ,分别对应整个请求内容，请求的md5值（这里uuid主要是将用户的请求进行hash，这样用户可以判断收到的返回结果对应的是那一条请求），该请求当前执行状态（"unfinished", "not return" ,"finished"），收到该请求时的时间戳，该请求对应的Tenant\_id值以及该请求的执行结果。

云备份中间件数据库中记录的各个执行状态：unfinished对应队列中等待执行的客户端请求，not return状态指的是正在执行但是还未返回结果的客户端请求，finished指的是执行完成返回结果的请求。云备份中间件通过轮询线程不断轮询数据库中各个记录的状态，若发现当前用户对应的请求状态为unfinished，重新执行该请求，执行完成后将结果返回给客户端。

中间件持久化功能实现的具体代码如下：

|  |
| --- |
| #云备份中间件建立与sql数据的连接  sql\_connection = sqlite3.connect('/var/log/mwlog.db',isolation\_level=None)  ##创建数据库表  sql\_connection.execute('''create table if not exists Log(request text not null, uuid char[33] not null,state char[20] not null,time char[20] not null,userid varchar[100] not null,result text not null);''')  #定义持久化函数，传入当前客户端用户的唯一标识userid  def Recovery(userid):  #寻找当前状态为unfinished且userid为传入的客户端用户userid  result=sql\_connection.execute("select request from Log where state = 'unfinished' and userid='"+userid+"';")  #如果记录中有状态为unfinished且userid为制定用户的记录  for row in result:  #执行该请求  handle(data)  #执行过程中将记录状态更改为not return  result=sql\_connection.execute("select result,uuid from Log where state = 'not return' and userid='"+userid+"';")  for row in result:  #通过便利userID\_socketID表寻找该用户对应的socket  sock=soc\_uuid\_dict.get(userid)  sock.sendall(data)  while (1):  #将返回结果的请求状态更改为finished  sql\_connection.execute("update Log set state='finished' where uuid='"+uuid+"';") |

上述代码描述了数据持久化操作的主要步骤，通过以上操作，当云备份中间件因异常关闭是，用户重新启动云备份客户端时，中间件可以自动执行上次未完成操作，为用户提供了良好的体验。

#### 5.3.2.3 多线程的实现

云备份中间件为了提高数据传输的效率和操作的稳定性，采用多线程模式进行数据的交互。云备份中间件中的线程主要分为两类：一类是工作线程，负责执行用户请求队列中的客户端请求；另一类是持久化轮询线程，负责不断轮询数据库中当前用户处于unfinished状态的记录，找到unfinished请求后，执行该请求并向客户端返回请求结果。

云备份中间件多线程的实现具体过程如下所示：云备份客户端启动后，中间件以守护进程模式运行在客户端，中间件立刻新建六个工作线程与一个轮询线程。此时用户可以通过操作云备份客户端将请求发送给中间件，中间件接收到客户端发送的请求后将请求报文放在请求队列中，如果当前有工作线程处于空闲状态，则其中一个工作线程在轮询请求队列后发现有任务，立刻执行，如果当前工作线程都处于工作状态，则请求暂时存放在队列中，直到有工作线程执行完任务。而轮询线程在中间件启动时，就开始扫描数据库，根据当前用户的唯一标识userid查看数据库中是否该用户请求处于unfinished状态，如果有，则立刻执行，并将结果返回至客户端，如果没有则不断轮询数据库。这样操作的好处是通过多线程操作处理用户请求，可以防止因单个操作异常造成的阻塞。

在云备份中间多线程实现的过程中对工作线程又做了更细致的划分：六个工作线程包括了五个上传下载传输服务线程和一个执行其他请求操作的工作线程。这样做的优点是：如果不对工作线程进行划分，那么这六个工作线程按照用户请求队列依次执行来自客户端的请求，如果同时上传六个大文件，接着创建一个新的上传文件夹，那么创建文件夹操作就会被阻塞较长时间。为了防止该状况的发生，本文中把工作线程进一步细化，分为了上传下载线程和其他操作线程，有效的提高了请求的响应速度。在实现过程中，主要是通过解析模块对用户发送的请求进行解析，确定该请求是上传下载类请求亦或是其他操作请求，根据请求的类型调用相应的线程，从而实现了高效率的云备份中间件数据交互。

云备份中间件多线程的代码实现主要如下所示：

|  |
| --- |
| # 创建六个线程，五个工作线程和一个数据库轮询线程  if (threading.activeCount()<7):  for i in range(6):  mwhandlethread = Mwhandlethread()  mwhandlethread.start()  #定义线程的运行过程：  def run(self):  #首先建立与数据库之间的链接  connect=sqlite3.connect('/var/log/mwlog.db',isolation\_level=None)  while True:  for i in range(1):  #如果请求队列的大小为零则不做任何操作  if mwhandlequeue.qsize() < 1:  pass  else:  #如果队列中有请求按照队列顺序获取队列前端的请求  clientreq = mwhandlequeue.get()  #根据请求的内容,hash算法获得请求的唯一标识uuid  uuid = hashlib.md5(clientreq).hexdigest()  #创建控制模块对象，将请求交给控制模块中的线程执行  controller = mwController.Controller(mwreq,uuid)  mwresult = controller.req(mwreq)  #工作线程执行过程中同时更新数据库中请求的状态 connect.execute("update Log set state='not return',result = '" + mwresult +"' where uuid='"+uuid+"';") |

### 5.3.3 解析模块

解析模块主要是讲从云备份客户端发来的指令进行解析，并将解析结果发送至控制模块，解析过程如图5-2所示：

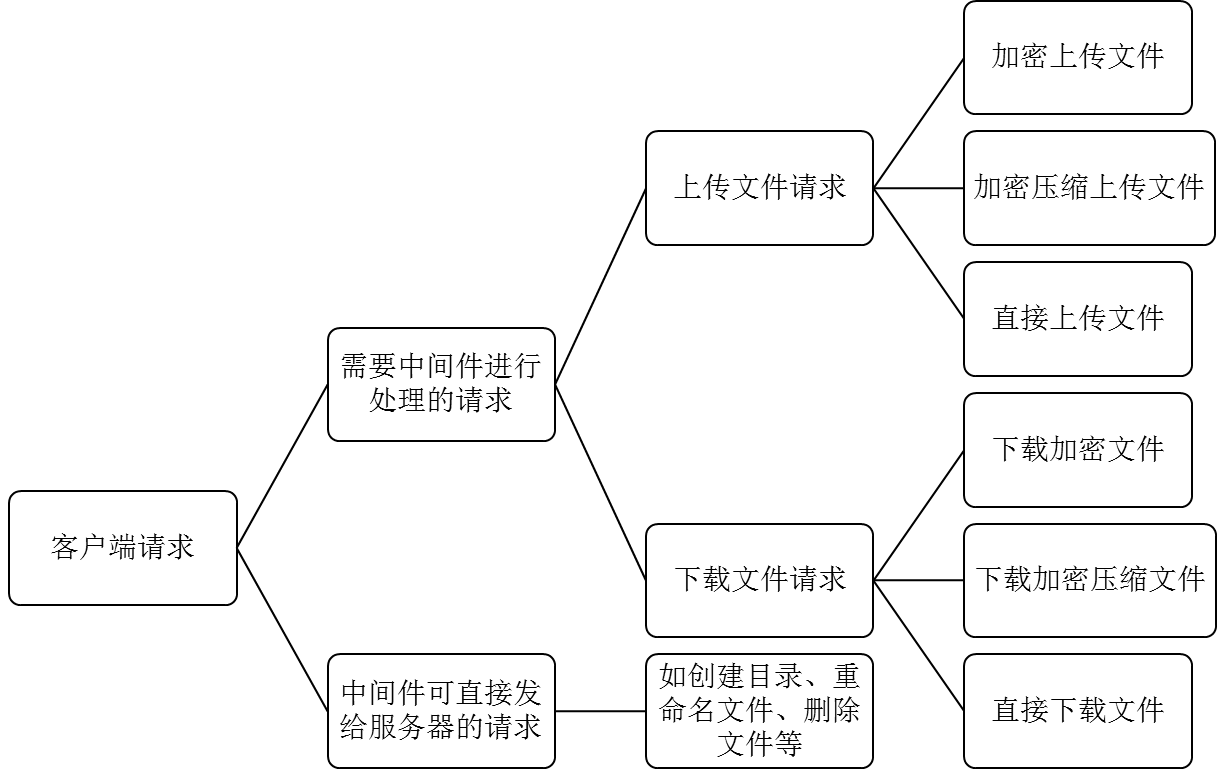


图5-2 解析请求分类图

当客户端将请求发送给中间件时，中间件需要解析出客户端的指令类别，根据指令的类别，将指令进行分类处理。

提供的34个功能接口的参数，用python的find()函数寻找关键字出现与否或出现位置。为保证安全性，所有上传文件必须加密。若请求中含有“op=CREATE”和“mode=ENCRYPT”参数则为加密上传请求；若含有“op=CREATE”和“mode=ENCRYPT\_COMPRESS”参数的为加密压缩上传文件请求；含有“op=OPEN”和“mode=ENCRYPT”关键字的为解密下载文件请求；“op=OPEN”和“mode=ENCRYPT\_COMPRESS”参数的位解密解压下载文件请求；其余的请求为其他请求。

### 5.3.4 数据处理模块

#### 5.3.4.1 数据加密模块

数据的加解密基于libsecrypto库的接口进行开发，是基于密钥服务器实现的。为了保证安全性，不泄露用户的唯一加密密钥，加解密具体过程对中间件与客户端是透明的，中间件只能调用相应接口进行数据加密功能的实现。libsecrypto库基于标准EVP接口开发，兼容PadLock硬件加解密。提供的两个接口如下所示：

1）数据加密接口：

void sAES\_encrypt(char \*in, char \*out, char\* access\_token)

参数in需给出待加密的文件路径，out为加密后传出的文件路径，access\_token请给出access\_token:user\_token完整信息，需要中间件从客户端获取:

{

"access\_token":"access\_token:user\_token",

"token\_type":"token类型",

"expires":失效时间,

"expires\_in":有效时间,

"refresh\_token":"令牌刷新口令"

}

2）数据解密接口：

void sAES\_decrypt(char \*in, char \*out, char\* access\_token)

参数in请给出待解密的文件路径，out为解密后传出的文件路径，access\_token的形式与数据加密接口中相同。

数据加密模块实现的流程图如图5-3所示：



图5-3 数据加密实现流程图

为保证安全性，加密是云备份系统用户不可自定义的必须功能。在解析到客户端的上传文件指令时，需首先调用数据加密接口，将加密后的文件保存至本地的“/tmp”临时路径下，并将客户端请求的md5值作为加密文件的文件名，便于客户端做出传输状态查询。需要注意的是，数据加密对于客户端透明，即加密文件后中间件需要修改客户端发来的指令，将该加密文件上传至服务器。如果用户发送了加密压缩上传请求，则需要先压缩再进行加密。

在解析到客户端指令是下载文件且需要解密时，中间件需要将客户端请求的md5值作为下载后的目的地址文件名，将从服务器下载的加密文件放至临时文件夹“/tmp”目录下，解密后将还原的文件移至下载目的路径，完成下载操作。用户发送的下载请求中mode参数为ENCRYPT或ENCRYPT\_COMPRESS，需要客户端首先发起获取文件属性的请求，得到上传该文件时的mode参数，才可发起下载请求，否则下载将可能会出错。

实现该模块时的部分关键步骤如下所示：

1）获取加密文件路径：python通过引用hash模块获得md5值。

encrypt\_path = '/tmp/' + hashlib.md5(request).hexdigest() + '.en'；

2）对文件进行加密：myEncrypter.sAES\_encrypt(path,encrypt\_path,dll,acces\_token)；

3）将加密文件的临时路径替换原路径：rq.replace(path,encrypt\_path)，其中rq是生成的curl命令。

#### 5.3.4.2 数据压缩模块

数据压缩采用zip压缩方法，采用python的zipfile模块进行压缩或者解压缩文件。默认在上传文件时，若是需要加密压缩的指令，则要先压缩，然后加密。该文件下载后也需要先解密，然后通过解压缩进行还原。由于压缩是可选的，客户端需要先发起获取待下载文件属性请求，得到上传该文件时的mode参数，才能发起下载该文件的请求。加密和压缩模块对于客户端均是透明的。需要加密压缩上传文件时，压缩文件保存在“/tmp”目录下，命名为客户端请求的md5值；需要解密解压缩下载文件时，待解压缩的文件保存在“/tmp”目录下，命名为客户端请求的md5值。

实现数据压缩模块的部分关键步骤如下所示：

1）压缩文件需要用到的接口函数：

将工作目录切换到当前文件目录：os.chdir(os.path.dirname(name))；

进行文件压缩：f = zipfile.ZipFile(name,'w',zipfile.ZIP\_DEFLATED)；

写压缩文件：f.write(filename)；

关闭压缩文件：f.close()；

2）解压缩文件需要用到的接口函数：

进行文件解压缩：zfile = zipfile.ZipFile(filename,'r')；

读取解压缩文件流：data = zfile.read(filename)；

以读写方式打开文件：file = open(tofilename, 'w+b')；

写文件：file.write(data)；

关闭文件：file.close()；

### 5.3.5 数据传输模块

#### 5.3.5.1 数据上传

对于客户端发来的请求，中间件解析为上传文件请求时，通过加密或者压缩加密，之后将交给数据传输模块进行数据的上传。通过curl命令进行数据上传操作，上传的文件是加密文件，并且需要将传输状态信息写入log文件，将客户端请求的md5值作为log文件名，保存在临时文件夹“/tmp”下，客户端需要文件传输状态时，可实时调用中间件的获取传输状态接口。

实现数据上传模块的部分关键步骤如下所示：

1）获取数据上传过程的log文件名：

logpath = '/tmp/' + hashlib.md5(client\_request).hexdigest() + '.log'；

2）生成curl命令：

upload\_req = 'curl ' + request + ' 2>&1 | tee ' + logpath;

其中' 2>&1 | tee '字段用于开启实时状态流，request已经是中间件将客户端请求经过转换令牌、替换加密路径等操作之后生成的新请求；

3）获取服务器返回结果：

result = commands.getoutput(upload\_req)；

4）中间件提取结果中的json字符串

result\_start = result.find('{')；

result\_to\_client = result[result\_start:]；

5）将result\_to\_client返回给会话管理模块。

#### 5.3.5.2 数据下载

对客户端的下载文件请求，mode参数是客户端前一步已经通过获取文件属性得到的。通过curl命令进行数据下载操作，下载到的加密文件首先需要解密，根据mode参数决定是否需要解压缩。为了客户端方便查询文件状态，加密文件名、压缩文件名和log文件名均为客户端请求的md5值，保存在临时文件夹“/tmp”下，供获取传输状态信息接口调用。

实现数据下载模块的关键步骤如下所示：

1）获取数据下载过程的log文件名：

logpath = hashlib.md5(client\_request).hexdigest() ;

logpath = '/tmp/' + logpath + '.log';

client\_request是客户端请求，由于服务器对下载文件后得到的重定向流没有提供接口，所以中间件为客户端提供了下载文件保存路径的接口。客户端可以通过'-o + 本地下载路径'来实现下载文件到指定路径。

2）生成curl命令，获得可实时读取的传输状态文件：

download\_req = 'curl ' + request + ' 2>&1 | tee ' + logpath;

request是中间件经过对客户端请求进行令牌转换、下载路径替换等操作之后生成的新请求。

3）中间件执行curl命令：

result = os.system(download\_req);

4）将返回结果result发送给会话管理模块。

#### 5.2.5.3 获取传输状态信息

对于中间件提供的查询传输状态功能，如果中间件实时通过socket将状态返回给客户端，那么对socket的占用太频繁，可能会导致请求处理效率下降，此外，对于中间件实时返回的状态信息，客户端可能也不会及时处理或并不需要。由于中间件部署在客户端，且传输过程中实时更新的传输状态信息的log文件客户端也有权限读取，因此，客户端自行通过中间件提供的接口读取该log文件，以获得JSON格式的当前传输状态信息，这种方式是可取的。

需要注意的是，该接口是中间件提供给客户端的独立接口，并不需要经过socket来传输数据，因此，可以最大程度地保证状态信息的实时性与可靠性。

客户端需要调用中间件的get\_status(logpath)接口来获取json格式的文件传输状态。此接口函数直接给客户端使用。客户端需要对其发送的request请求取md5值，获取传输状态信息的log文件名，即接口的传入参数。

1. 传入参数logpath的生成方法：

import hashlib;

logpath = '/tmp/' + hashlib.md5(request).hexdigest() + '.log';

1. 每次调用该接口时，读取文件最末一行即为当前最新状态信息。

读取文件：lines=f.readlines()

得到所有状态信息：curr=lines[-1]

将所有状态信息分行：arr = curr.split('\r')

获取状态信息的最后一行即为实时传输状态：lastline=arr[-1]

将实时传输状态分开字段：lastlinejson=lastline.split()

1. 生成返回结果：

|  |
| --- |
| speedresult= '{"TotalPercent": "' + lastlinejson[0] + ' ","Total":"' + lastlinejson[1] + '","Received":"' + lastlinejson[3] + ' ","Average Upload":"' + lastlinejson[6] + '","Speed Upload":"' + lastlinejson[7] + '","Time Spent":"' + lastlinejson[9]+' ","Time Left":"' + lastlinejson[10] + ' ","Current Speed":"' + lastlinejson[11] +'"}' |

4）返回信息主要包括八个并且以json的形式给出：

TotalPercent //当前下载文件的百分比

Current Speed //当前下载文件的速度

Total //下载文件的大小

Received //接收到文件的大小

AverageDload //平均下载速度

SpeedUpload //平均上传速度

Time Spent //总的时间花费

Time Left //下载剩余时间

返回示例：（下载一个69.4M完成时的进度信息反馈）

{"TotalPercent": "100","Current Speed":"11.2M", "Total":"69.4M", "Received":"69.4M ","AverageDload":"11.0M","SpeedUpload":"0","Time Spent":"0:00:06 ","Time Left":"--:--:--"}

客户端可以通过引用python的JSON模块自行解析，获取所需信息。

# 第六章 总结和展望

## 6.1 总结

随着互联网的发展，大数据时代数据量的增加，以及人们对数据安全关注度的增加，现今社会对于数据安全存储越来越重视。经充分调研国内外数据安全存储的现状，发现云备份已经成为解决数据安全存储的基本方法之一。云备份主要通过集群应用将不同类型的存储设备集合起来协同工作，为用户提供数据存储备份功能服务。云备份具有以下优势：首先，可扩展节约成本，相交于购置存储设备，依靠第三方软件提供的云备份服务价格低廉；其次，高效可靠，第三方软件可为用户提供高效可靠的数据传输，并利用加密算法对存储在云端的数据进行保护，防止用户隐私泄露；第三，第三方软件可以为用户提供实时监控，保证存储在云端的数据安全。

本文大量调研了现今市场上常用的备份工具，比较分析各个备份工具的优缺点，设计并实现了基于Linux系统的高效云备份中间件，云备份工具具有以下优点：首先，提供完整高效的数据服务，即用户可以高效的管理个人的备份空间；其次，采用相关的数据保护方案，保证用户存储在云备份服务器端数据的安全性；再次，全面支持国产操作系统。

云备份中间件以守护进程的模式运行在客户端，解除客户端与云备份服务器绑定的同时也为两者提供了交互的桥梁。云备份中间件为客户端提供了业务接口和状态查询接口，实现了数据传输、数据加解密、数据压缩以及安全会话功能，为用户提供了高效稳定且安全的用户体验。

在云备份工具实现的过程中，为了提升云备份中间件传输效率，加强中间件工作的稳定性，中间件采用多线程工作模式，支持数据持久化，且支持多用户登录。通过以上努力，本文最终实现了高效云备份中间件开发，经测试确认可为用户提供高效稳定的数据备份功能。

## 6.2 展望

本文设计并实现的基于Linux系统的高效云备份系统中间件具有实际应用价值，可以为个人和企业提供高效安全的数据存储、备份等功能。然而，本文所实现的高效云备份中间件仍然存在挑战，这里列举出作者认为在未来仍然需要进一步研究探讨的几点内容：

1. 云备份中间件尚未提供自动备份功能，在以后的优化开发中应提供此功能，允许使用者按照一定时间间隔定期进行存储备份，以便在使用者数据出现问题的时候，及时进行相应恢复；
2. 在以后的优化中还将制定相应的备份管理策略，如增量备份与差量备份，实现每次执行备份请求时，对使用者上一次备份后增加的和修改过的目录进行备份；
3. 云备份中间件在以后的优化中可以提供个性化的分享功能，用户可以通过资源链接等方式实现资源共享。

# 参考文献

1. 郭瑞. 数据泄露风险的趋势分析[J]. 信息安全与技术,2014,10:18-21.
2. Donald Kossmann, Tim Kraska, Simon Loesing, et al. Cloudy: A Modular Cloud Storage System[J]//Proceedings of the VLDB Endowment, 2010 : 1533-1536.
3. Rahumed A, Chen H C H, Tang Y, et al. A Secure Cloud Backup System with Assured Deletion and Version Control.[J]. International Workshop on Security in Cloud Computing, 2011:160-167.
4. Milojicic D. Middleware's Role, Today and Tomorrow[J]. IEEE Concurrency, April-June, 1999, 7(2):70-80.
5. Hadim S. Middleware: middleware challenges and approaches for wireless sensor networks[J]. IEEE Distributed Systems Online, 2006, 7(3):1.
6. Darren Quick,Kim-Kwang Raymond Choo. Dropbox analysis: Data remnants on user machines[J]. Digital Investigation,2013,101:.
7. Anonymous. OneDrive[J]. Computer Act!ve,2015,445:.
8. Kurt Oestreicher. A forensically robust method for acquisition of iCloud data[J]. Digital Investigation,2014,11:.
9. 李海宝,李海燕,任常愚,任敦亮,丁红伟. 基于360云盘的大学物理网络教学平台构建[J]. 实验室研究与探索,2014,09:175-179..
10. 李心语. 在线网盘系统的设计与实现[D].吉林大学,2014.
11. maomaobear. 百度云OS宣告ROM丛林时代终结[J]. IT时代周刊,2015,05:61.
12. 引火虫. 将百度云资源分享给好友[J]. 电脑迷,2014,10:85.
13. Milojicic D. Middleware's Role, Today and Tomorrow[J]. IEEE Concurrency, April-June, 1999, 7(2):70-80.
14. Bernstein P A. Middleware: A Model for Distributed System Services.[J]. Communications of the Acm Cacm Homepage, 1996, 39(2):86-98.
15. Hadim S. Middleware: middleware challenges and approaches for wireless sensor networks[J]. IEEE Distributed Systems Online, 2006, 7(3):1.
16. 何思敏,康慕宁,张晓,孙展望. 云存储性能评测技术与方法研究[J]. 计算机与现代化,2011,12:1-4+8.
17. 李琪林, 刘强, 周明天. 论中间件技术及其分类[J]. 四川师范大学学报：自然科学版, 2001, 24(6):657-660. DOI:10.3969/j.issn.1001-8395.2001.06.031.
18. Cecchet E, Candea G, Ailamaki A. Middleware-based Database Replication: The Gaps Between Theory and Practice[J]. Eprint Arxiv, 2008:739-752.
19. Anonymous. Kingston Technology Company, Inc: Kingston Technology Launches DataTraveler BlackBox USB Flash Drive; 256-bit, Hardware-Based AES Encrypted and FIPS Validated Drive Ideal to Safeguard Critical Data[J]. M2 Presswire,2008,.
20. Rabin T. A Simplified Approach to Threshold and Proactive RSA.[J]. Proceedings of Crypto, 1998:89--104..
21. Ran C, Halevi S, Katz J. Chosen-Ciphertext Security from Identity-Based Encryption[J]. Siam Journal on Computing, 2006, 36(5):1301-1328.
22. 何明星, 林昊. AES算法原理及其实现[J]. 计算机应用研究, 2002, 19(12):61-63. DOI:10.3969/j.issn.1001-3695.2002.12.018.
23. Gentry C. Practical Identity-Based Encryption Without Random Oracles[J]. Lecture Notes in Computer Science, 2006:445-464.
24. 刘洪志. PHP多线程抓取多个网页及获取数据的通用方法[J]. 电脑编程技巧与维护,2013,01:60-62+64.
25. Sangjun Jeon,Jewan Bang,Keunduck Byun,Sangjin Lee. A recovery method of deleted record for SQLite database[J]. Personal and Ubiquitous Computing,2012,166:.
26. Yue-Shan Chang,Ruey-Kai Sheu,Shyan-Ming Yuan,Jyn-Jie Hsu. Scaling database performance on GPUs[J]. Information Systems Frontiers,2012,144:.
27. Anonymous. Research and Markets: Inside Symbian SQL Offers a Unique View into the Internals of the Implementation and a Wealth of Practical Advice[J]. M2 Presswire,2010,:.
28. Y李蔚,陈亚峰. 嵌入式数据库SQLite及其应用研究[J]. 沿海企业与科技,2010,10:45-47.
29. A.A. Vinokurov,V.B. Il’in,V.G. Farafonov. ScattPy: A new Python package for light scattering computations[J]. Journal of Quantitative Spectroscopy and Radiative Transfer,2011,11211:.
30. Anonymous. FotoTime, Inc.; FotoTime Announces FotoAlbum Version 6.0[J]. Computer Business Week,2008,:.
31. Yasuhiro Tsujimura,Mitsuo Gen,Syunsuke Ishizaki. Optimal routing in multiple I O data network using neural network with perturbed energy function[J]. Computers & Industrial Engineering,1997,333:.
32. Agoes A. Moelja,Gjerrit Meinsma. H 2 -optimal control of systems with multiple i/o delays: Time domain approach[J]. Automatica,2005,417:.
33. Wai-Kei Mak. I/O placement for FPGAs with multiple I/O standards.[J]. IEEE Trans. on CAD of Integrated Circuits and Systems,2004,23:.
34. Matteo Boschini. Il Nuovo Sistema di Data Transfer per l'esperimento AMS-02[J]. Bollettino del CILEA,2007,0108:.
35. Arnim Eijkhoudt,Tristan Suerink. Uforia: Universal forensic indexer and analyzer[J]. Journal of Computer Virology and Hacking Techniques,2013,92:.
36. J. Borgdorff,M. Mamonski,B. Bosak,K. Kurowski,M. Ben Belgacem,B. Chopard,D. Groen,P.V. Coveney,A.G. Hoekstra. Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment[J]. Journal of Computational Science,2014,:.
37. Anonymous. Si2 to Hold 'OpenAccess Scripting Language Workshop' Co-Located Event at DAC 2011[J]. Manufacturing Close - Up,2011,:.
38. 李新宇. 网络云盘介绍——以360云盘和百度云为例[J]. 无线互联科技,2014,01:38.

# 致 谢

在论文完成之际，首先感谢我的研究生导师韩伟力老师，从项目的设计到开发以及测试整个过程，韩老师不断给予技术上的支持；在论文创作的过程中，韩老师严肃的科学态度，精益求精的工作作风，严谨的治学精神一直督促这我，促使我不断进步。在韩老师的悉心指导下，不断优化论文结构和论文内容。这两年多来，韩老师不仅在学业上给我以精心指导，在生活上还给与我无微不至的关怀，再次谨向韩老师表示衷心的感谢！

同时，我还要感谢陪我度过了研究生生涯的实验室同学，在我遇到困难时给予我帮助，心情低落时给予我孤立，特别要感谢梁蛟同学，她在项目开发过程中，不断提出创新性意见并寻找解决方案，给予我不少的帮助。

在此还要感谢复旦大学的各位老师，为我提供了一个良好的学习平台，并且给了我丰富的资源去实现我的研究，感谢我的家人，没有你们的支持，就没有今天的。

最后，衷心感谢评阅论文的各位专家和老师!

**复旦大学**

**学位论文独创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。论文中除特别标注的内容外，不包含任何其他个人或机构已经发表或撰写过的研究成果。对本研究做出重要贡献的个人和集体，均已在论文中作了明确的声明并表示了谢意。本声明的法律结果由本人承担。

作者签名： 卜瑞琪 日期： 2015.10.29

**复旦大学**

**学位论文使用授权声明**

本人完全了解复旦大学有关收藏和利用博士、硕士学位论文的规定，即：学校有权收藏、使用并向国家有关部门或机构送交论文的印刷本和电子版本；允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。涉密学位论文在解密后遵守此规定。

作者签名： 卜瑞琪 导师签名： 韩伟力 日期： 2015.10.29