学校代码： 10246
学　　号： 15212010038

硕 士 学 位 论 文

（专业学位）

云备份中间件关键技术优化设计与实现

Design and Implementation of Key Technologies in Backup-Cloud Oriented Optimized Middleware

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：　2017 年 9 月 8 日

指导小组成员名单

韩伟力 教授

目录

摘要

近年来，云备份服务因其海量的存储空间、便捷的存取方式，越来越受人们的青睐，云备份服务产业的市场份额不断增长。然而，云备份服务在近几年面临着诸多的挑战，衍生出了如安全性、稳定性、可扩展性等问题，而用户对云备份服务也提出了更高的要求。目前，一款自主研发的云备份服务系统以中间件作为云备份服务器与客户端之间的数据交互枢纽，实现了数据传输、数据压缩、数据加密等基本功能。为提供安全可靠的云备份服务，亟需对其关键技术进行优化。

本论文基于对当前云备份服务系统的深入调研，厘清关键技术的发展趋势，同时基于前期的研究成果，对现有的云备份中间件系统进行优化设计与实现。整个优化设计分为四个部分：（1）通过物理隔离和数据加密的方式，保证缓存在中间件中的用户信息和会话记录的安全性；（2）借助大文件分块传输和断点续传技术，解决大文件上传过程中遇到网络中断或服务器死机等故障而引发的数据重传问题；（3）利用本地缓存和服务器提供的高级接口，减少数据在上传下载过程中产生的冗余数据，并提供快速上传功能，以提升数据传输的稳定性；（4）优化并完善平台框架和 API 支持机制，解决不同应用和云备份服务之间请求适配的问题。

本文实现了上述优化设计，经测试表明：优化后的中间件系统具有更好的安全性、稳定性，能确保数据在传输、存储以及缓存在中间件时的安全性，且能够在网络不稳定的环境下保证数据的传输效率。同时，通过对源码的重构和优化减少了系统自身的漏洞，新增的异常处理机制提高了系统的鲁棒性。此外，在系统的可扩展性上，本文开发了两款客户端以支持更多的应用使用云备份服务。

关键词： 云备份，中间件，数据加密，断点续传，本地缓存，扩展性

Abstract

In recent years, cloud backup service becomes more and more popular because of its massive storage space and convenient access. However, cloud backup service is now facing many derivative challenges such as issues about security, stability and scalability, while users also put forward higher requirements for the services. Currently, an independently developed cloud backup system uses middleware as the data exchange hub between client and server, which achieves some basic functions of data transmission, data compression and data encryption. It is imperative to optimize the key technologies of cloud backup service in order to provide a secure and reliable service.

Based on a deep research of the current cloud backup system and previous research results, and after clarifying the key technical trends, this paper designs and implements the optimization of the existing cloud backup middleware system. The entire optimization design consists of four parts: (1) Ensuring the security of cached user information and session records in the middleware by adopting physical isolation and data encryption. (2) Solving the problem of data retransmission caused by the network interruption or server crash by applying the technologies of block transfer of large files and breakpoint resume. (3) Utilizing local cache and high-level interfaces provided by the server to reduce redundant data, and providing fast upload function to improve the stability during data transmission. (4) Optimizing and improving platform framework and API mechanisms to address request adaptation problem between different applications and cloud backup server.

The test results show that the optimized middleware system performs better in security and stability, which ensures the data security in transmission, storage and cache and can guarantee the efficiency of data transmission in an unstable network environment. At the same time, this work reduces the code loopholes through the reconstruction of the source code and improves the system robustness by adding exception handle mechanism. In addition, to increase the scalability of the system, this paper develops two clients, which can support more customers to use cloud backup service.

Keywords： Cloud Backup, Middleware, Data Encryption, Breakpoint Resume, Local Cache, Scalability

第一章 引言

1.1 研究背景

随着互联网技术的飞速发展，网络数据呈现出井喷式的增长，传统的数据备份方式存在太多的局限性，已经无法满足人们的需求。在这样的背景下，云备份服务逐渐成为主流的数据备份方式。因其海量的存储空间、便携的存取方式、可靠的安全特性，越来越受大众的青睐，大型的机构如政府、医院、企业等也逐步将数据转移到云端。数据显示，2016 年，中国云服务市场规模超过 500 亿元，达到 516.6 亿。预计 2017 年中国云服务市场份额将达到 690 亿以上。此外近三年来，年复合增长率超过 32%，而且 2016 年增长率更高[1]。

然而，大范围普及的云备份服务在近几年迎来了巨大的挑战，除了提供云端数据备份这项基本的功能之外，人们对云备份服务的安全性、稳定性以及可扩展性投入了更多的关注[2][3][4]。在传统的数据备份方式中，数据保存在本地，对用户来说是可管控的，而将数据备份在云端则可能产生如个人数据泄露、商业信息遭到窃取、科研成果外泄等问题[1]，这可能对用户造成不可逆的损失。此外，云备份服务以网络作为基础设施，而网络又时常受限于硬件、软件和地域等多方面的因素，一旦出现网络波动或中断，将会影响数据在网络层的传输，尤其对较大的文件来说，遇到网络问题只能进行重传，这将极大地降低用户的体验度。同时，随着云端数据不断地积累，产生了大量冗余的数据。一方面这些冗余的数据会占用服务器大量的备份空间，另一方面用户备份在服务器的文件只是一个映射的链接，如果重复上传服务器中已存在的文件，将会占用大量的网络带宽。更进一步，介于移动互联网和智能手机的热度持续走高[2]，人们的娱乐、办公环境逐渐趋向于平台化，针对单一应用的云备份服务已不适应当前平台多元化的格局，实现云端数据在多平台的共享，成为了云备份服务行业的共识。

表 1.1 国内外知名云备份服务平台技术特性分析

	加密算法	数据去冗余	大文件上传	断点续传	平台数
百度网盘	128 位 SSL	全局	20G / 4M	支持	6 种
360 云盘	128 位 SSL	全局	5G / 4M	支持	5 种
Google Drive	128 位 AES SSL/TLS	无	5TB / 字节流	支持	5 种
Dropbox	256 位 SSL/TLS	单一用户	不限 / 8M	支持	7 种

目前，国内外成熟的云备份服务供应商为应对这一系列新的挑战，纷纷推出了新的技术或对现有技术进行了优化[5][6]，以最大化程度的满足用户需求。如表

[1] http://www.freebuf.com/news/97925.html

[2] http://mobile.zol.com.cn/597/5974727_all.html

1.1，本文选取了四个国内外较为知名的云备份服务供应商进行了调研，国内的百度网盘、360 云盘，国外的 Google Drive、Dropbox[7]，分别对他们的技术特性进行了全面的分析。

保证云端数据的安全性是云备份服务的基本要求，对数据进行加密是一种强有力的安全措施来保护个人信息不被窃取或篡改。调研结果显示，所有的云备份服务供应商都对数据在传输过程中及存储在云端时两种状态进行了加密。在传输过程中，百度网盘和 360 云盘使用的是 128 位的 SSL 加密技术，Google Drive 和 Dropbox 分别采用了的 128 位/256 位的 SSL/TLS[8]加密技术。数据在云端时，百度网盘和 360 云盘未强制对数据进行加密，而是由用户在上传之前决定是否要将数据以加密的形式备份。反观 Google Drive 和 Dropbox，它们选择了对上传至云端的数据进行强制加密，Dropbox 使用的是 256 位的 AES 加密[9]方式，而 Google Drive 采取了规模相对较小的 128 位 AES 加密技术。

此外，考虑到网络的不稳定因素，亟需一种有效的方式确保数据传输的稳定性，尤其是针对大文件的传输。各大供应商都对大文件传输的关键技术进行了优化，以避免因网络故障而导致的大文件重传问题[10]。百度云盘支持单个文件上传的大小上限为 20G，360 云盘为 5G，Google Drive 为 5TB，而 Dropbox 不限制单个文件的大小。在大文件上传的方式上，它们都将文件分割成固定大小的文件流形式，通过设定偏移量进行分块传输，这样做的好处是遇到网络故障时，只需记录已上传数据流的偏移量，重传时可以根据偏移量续传而无需从头开始[11][12]。

随着云端数据的积累，服务端存储了大量冗余的数据[13][14]。减少数据冗余以节省服务器的存储空间是十分有必要的，除此之外还能为用户提供一项新的功能，这种功能称之为“秒传”[15]，指的是当服务器存在当前用户待上传的文件时（通过文件的 MD5 值进行校验[16]），不会真实地上传该文件，而是会将该文件的链接发送到用户的网盘中，这样能有效地解决冗余数据重复上传的问题。百度网盘和 360 云盘都支持全局的文件去重校验，为用户提供了“秒传”功能。Dropbox 采取的是一种相对折中的方式，它不会在全局文件系统中校验该文件是否存在，只针对单一用户进行去重处理，这样能在很大程度上地减少由去重而导致的数据安全问题，如侧信道攻击[17]。Google Drive 对待“秒传技术”极为谨慎，尽管这样做能够大幅度地提升数据上传的效率，但出于安全方面的考虑，Google Drive 未在云盘中使用该技术。

同时，这些成熟的云备份服务供应商都会针对主流的操作系统和平台，提供不同类型的客户端，这样使得用户可以跨平台地使用云备份服务。常见的平台有 Windows、Mac、Android、iPhone、Web，以上四个云备份服务供应商都提供了这些平台的客户端，其中百度云盘还支持 Windows Phone，Dropbox 是唯一支持

BlackBerry 智能手机的平台[18]。

尽管这些云备份服务产品在市场上取得了成功，但仍存在一些不足。在功能上，都没有提供本地缓存的功能，也就是说它们只做到了服务器端的数据去重，但重复地从服务器端下载同一文件也是一种常见的用户行为，通过利用本地缓存技术，节省重复下载文件所需的网络带宽，从而实现快速下载功能。此外，Web 扩展应用形式的客户端相较于其他类型的客户端使用更加方便，更加轻量级，同时还支持用户将网页上浏览到的任何形式的资源实时上传到服务器，以上供应商都没有提供该类型的客户端。在安全性上，成熟的云备份服务平台也存在一些漏洞，有些漏洞被黑客利用后对用户利益造成了严重的损伤，如百度云 SDK 虫洞漏洞[3]，安装了百度云的 Android 手机遭到攻击后能被黑客任意操作。再如 Dropbox CVE-2014-8889 漏洞[4]，只要用户安装了含有 Dropbox SDK 漏洞的应用程序，攻击者便可在未经用户同意的情况下直接访问用户的 Dropbox 资源。

1.2 研究内容及意义

由国家“核高基”科技重大专项自主研发的云备份服务平台以中间件作为服务器与客户端之间通信的桥梁，实现了数据传输、数据压缩、数据加密以及安全会话等基本功能，有效地分离了客户端云备份工具与基础设施之间的绑定[19]。然而，这些基础的功能已无法满足当前云备份服务行业的需求。云备份关键技术作为产品的核心竞争力，已成为了各大供应商研究的重点，而这些技术都是不对外公开的，因此只能通过对现有的中间件系统进行全面分析，同时借鉴业内成熟产品的技术特点，对中间件的关键技术进行优化，以符合当前云备份服务行业的发展趋势。

同时，成熟的云备份产品在功能上和和安全性上都存在一定的不足，原有的中间件系统也存在类似的问题，因此需要对现有的中间件系统进行全面的分析和优化，在功能上进行扩展，丰富中间件系统的功能特性，以更好地满足用户的需求。另外，通过对现有中间件系统源码进行重构，提高系统的安全性和鲁棒性。

本文的主要研究工作内容如下：

- 持久会话管理：云备份中间件在客户端和服务器之间工作，多个客户端的命令通过中间件提交给服务器。用户的命令会以会话的形式保存在中间件，当中间件系统出现异常时（如网络中断、中间件死机），会在下次重启中间件时自动恢复执行未完成的会话。此外，对会话信息进行了物理隔离[20]，

[3] http://anquan.baidu.com/bbs/thread-395277-1-1.html

[4] https://www.slideshare.net/ibmsecurity/remote-exploitation-of-the-dropbox-sdk-for-android

并且以加密的形式存储在数据库中，以提高用户信息的安全性。

- 大文件上传：利用服务器提供的合并文件接口，实现了大文件分块上传功能，在受到网络状况影响的情况下，若文件上传中断，会自动启用断点续传功能，避免数据重传。
- 本地缓存：利用现代计算机普遍较大的存储空间以及计算能力，实现了本地缓存管理，减少系统 I/O 操作，提高了中间件系统数据传输性能。同时，利用服务器提供的云端数据信息，实现了针对单一用户的“秒传”功能。
- 多应用适配：现有的中间件系统只支持单一的 CLI（Command Line Interface）客户端，为实现对不同客户端请求的适配，支持更多的客户端类型，对系统的跨平台性做了优化。本研究工作开发了一款基于Firefox浏览器的扩展应用，用户可以利用扩展应用使用云备份服务，并提供了 Java 平台的软件开发包（SDK）[5]，Java 开发者可以利用该开发包提供的应用接口来集成云备份服务。

本文实现以上研究工作内容，经测试表明，优化后的中间件系统具有更好的安全性、稳定性、鲁棒性并具备一定的可扩展性。相较于国内外成熟的云备份服务产品，在核心功能特性上基本保持一致。此外，提供了下载本地缓存功能以及浏览器扩展应用形式的客户端类型。

1.3 论文的组织

本文分七个章节进行阐述。第一章介绍本文研究内容的背景以及相关的研究工作。第二章介绍本文在研究和实现过程中使用到的核心技术。第三章将分析现有云备份中间件系统的功能特性和不足，指出本文需研究和实现的内容。第四章详细阐述云备份中间件系统优化的整体设计，包括系统框架、类图设计、流程设计以及接口设计。第五章介绍云备份中间件系统优化的实现细节。第六章将以实际的测试结果对优化进行评估，并与原有的中间件系统和成熟的云备份服务产品进行比较。第七章对全文进行总结，并对未来进行展望。

[5] https://en.wikipedia.org/wiki/Java_Development_Kit

第二章 核心技术

2.1 云备份与中间件

云备份指的是将个人数据通过云存储的方式备份在云端服务器上，相较于传统的数据备份方式，云备份技术具备的优势为数据信息备份更加安全、支持平台数据共享、提供大量的存储空间。

2.1.1 云备份特点

利用云备份技术可以突破传统数据备份方式在时间和空间上的局限性，让数据传输更加便捷、安全、可靠。云备份的主要技术特性如下[21]：

- 可节约成本：个人或企业使用云备份后，可以依靠第三方云备份服务提供商海量的存储能力对大数据进行存储，而不需担心数据备份过程中产生的采购和实施投资的问题，可以降低运作成本、提升生产效率。
- 高效可靠：在现行的计算机云备份数据库中，主要是利用较为先进的存储技术，其中主要包括磁盘的备份、压缩、加密、存储虚拟化等重要数据保护功能。利用计算机云备份技术可以有效提升数据信息的安全性，同时还能提升其效率与可靠性。
- 高容灾性：传统的数据恢复方式主要是从磁盘开始将数据恢复，因此，如果发生数据丢失的情况，技术管理人员需要找到磁盘，将其在计算机上加载，根据加载所显示的数据位置便可对信息数据进行恢复。一旦磁盘出现不可逆的故障，则会造成数据永久丢失。而从云中恢复数据要可靠得多，云端服务器存有数据的备份，它不需要从磁带存放点运送磁带，而是通过网络进行数据恢复，节省时间并无需建设本地磁带设备。

2.1.2 云备份中间件

云备份中间件是一种独立于云服务器与客户端的系统组件，位于客户机与服务器之间，管理计算机资源与网络通信[22]。通过中间件技术，即便不同的应用之间存在差异，如接口差异、操作系统差异等，仍可实现资源共享，解决系统的跨平台、跨网络、跨硬件特性等问题。在云备份服务平台中，中间件扮演着重要的角色，能够显著地降低系统开发和维护成本，同时提升系统的稳定性。

云备份中间件的系统模型如图 2-1 所示。中间件处于云备份服务器与客户端应用之间，充当两者之间的“桥梁”作用，通过适配来自不同应用的请求，转化为云备份服务器可识别的标准形式，同时调用服务器提供的应用层接口，实现多

种应用与服务器之间的通信。

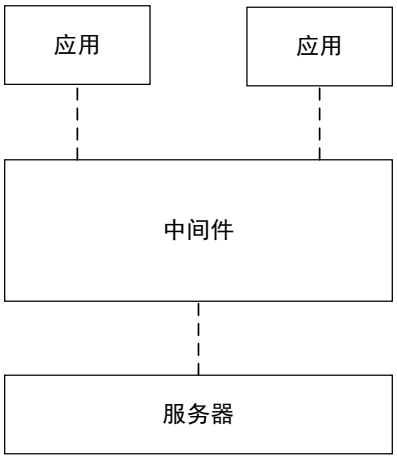


图 2-1 云备份中间件系统模型

2.3 SQLite

SQLite 是 D.Richard Hipp 用 C 语言编写的开源嵌入式数据库引擎。它是一款轻量级的关系型数据库，绝大多数主流的操作系统上都能够运行 SQLite，同时它为大多数编程语言，如 C#、PHP、Java 等提供了编程接口。此外，它还支持大多数的 SQL92 标准，且源代码不受版权限制[23]。目前，由于其占用的内存低、性能较好以及零成本管理等特点，被广泛地应用于嵌入式应用的开发中，像 Android、iOS 等平台都为开发人员提供了内置 SQLite 数据库[24]。

在存储容量方面，SQLite 虽然是一款轻量级的数据库，但是最高可支持高达 2TB 大小的数据备份，并且每个数据库都是以单个文件的形式存在的。

在事务处理方面，SQLite 支持多个进程可以同时读取同一个数据，但只有一个进程可以执行写操作。通过数据库级上的独占性和共享锁机制，使得某个进程或线程想要往数据库写入数据之前，必须获得独占锁，而其他进程则无法执行写操作。

在数据类型方面，SQLite 支持 NULL、INTEGER、REAL、TEXT 和 BLOB 数据类型，分别代表空值、整型值、浮点值、字符串文本、二进制对象。同时，SQLite 采用动态数据类型的机制，当某个值插入到数据库时，首先会对它的类型进行检验，如果该类型与关联的列不匹配，SQLite 则会尝试将其自动转换为对应列的类型，如果不能转换，则将该值将作为本身的类型。

2.4 断点续传

断点续传技术是一种结合本地存储和网络备份的技术，主要用来解决网络失效时数据丢失的问题[11]。具体来说是在上传文件时，将文件以固定的大小，划分为若干部分，或以固定长度字节流的形式上，对每部分文件块都单独采用一个线

程进行上传。服务器端监听数据传输请求并接收数据，记录数据传输进程。如果遇到网络故障，则可通过捕获网络异常的形式，将此时的断点信息记录到数据库中。等待网络恢复正常后，根据断点所记录的信息续传。通过此项技术，可以帮助用户在遇到网络故障时，节省因重传而浪费的大量时间，提升数据传输的效率。

断点续传是 HTTP 1.1 协议的一部分，只要利用了 HTTP 协议[6]中的 Range、Accept-Ranges 以及 Content-Ranges 字段来和服务器端交互，就可以实现文件下载的断点续传。

Range 字段用于客户端到服务器的请求报文中，通过该字段可以指定传输文件的某一段，典型格式如下：

Range : bytes=0-1024 //第 0-1024 个字节范围的内容

Accept-Ranges 用于服务器端给客户端的应答，通过该字段可以获取服务器是否支持断点续传的信息，典型格式如下：

Accept-Ranges: bytes //支持以 bytes 为单位进行传输。

Accept-Ranges: none //不支持以 bytes 为单位进行传输。

Content-Ranges 用于服务器到客户端的应答，通过该字段可以获取传输的资源文件的具体范围以及资源的总大小，典型格式如下：

Content-Ranges: bytes 0-1024/5645 //大小为 5645 字节的前 1024 字节。

请求报文如下所示。

```
GET /test.docx HTTP/1.1
Connection: close
Host: 10.131.1.63
Range: bytes=0-1024
```

响应报文如下所示。

```
HTTP/1.1 200 OK
Content-Length: 256823
Content-Type: text/html
Content-Range: bytes 0-1024/256823
```

在断点续传过程中，首先需要将上传的文件拆分为固定大小的部分，随后逐一发送 PUT 请求到服务器。请求报文中需要包含 Range 字段，表示当前请求上传数据的字节范围。一旦发生网络中断，下次连接时首先发送一个 GET 请求到服务器，在请求的 header 中放一个 Content-Range 字段，通过服务器返回的信息，从 Content-Range 字段中获取已上传的字节数，之后再将剩余字节的上传至服务器。

[6] http://www.ietf.org/rfc/rfc2616.txt

2.5 WebSocket

WebSocket 主要用于 Web 浏览器和服务器之间进行双向数据传输[25]，相较于 Ajax 技术需要客户端发起请求，WebSocket 服务器和客户端可以彼此相互推送信息，建立持续的连接。其协议基于 TCP 协议实现，包含初始的握手过程，以及后续的多次数据帧双向传输过程。其目的是在 Web 应用和后台服务器进行双向通信时，可以使服务器避免打开多个 HTTP 连接进行工作来节约资源，提高工作效率和资源利用率[26]。

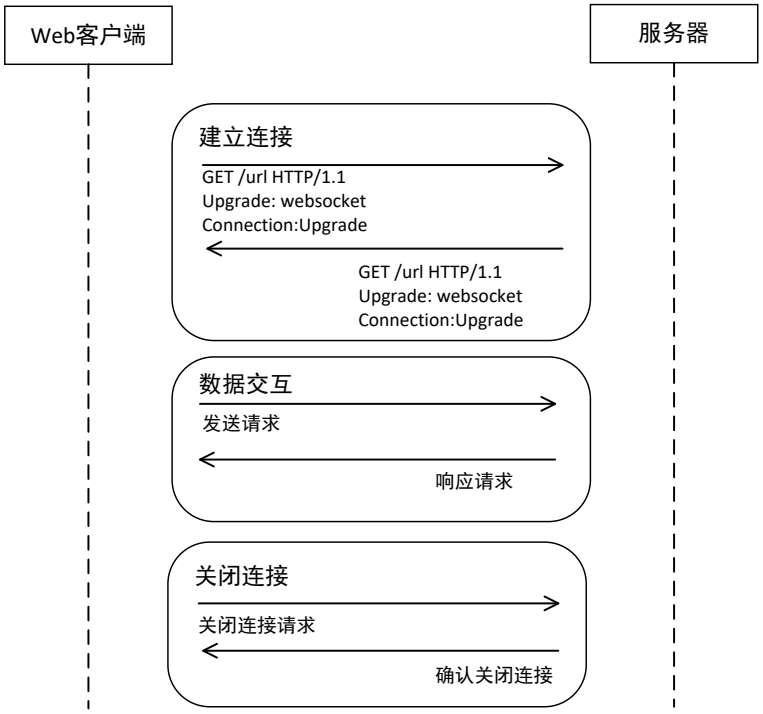


图 2-1 WebSocket 通信示意图

WebSocket 是一种类似 TCP/IP 的 socket 技术，其协议为应用层协议，定义在 TCP/IP 协议栈之上。WebSocket 连接服务器的 URI 以"ws"或者"wss"开头。ws 开头的默认 TCP 端口为 80，wss 开头的默认端口为 443。具体来说，共分为握手阶段和数据通信两个阶段[27]。

如图 2-1 所示，在握手阶段，户端和服务器建立 TCP 连接之后，客户端向服务器发送握手请求，随后服务器向客户端发送握手响应，随后连接建立，双方可以互通信息。

在数据通信阶段，数据被组织为依次序的一串数据帧(data frame)，然后进行传送。帧的类型分为两类：数据帧(data frame)和控制帧(control frame)。数据帧可以携带文本数据或者二进制数据；控制帧包含关闭帧和 ping/pong 帧。如要关闭连接，只需任何一端发送关闭数据帧给对方即可，关闭连接以状态码进行标识。

2.6 WebExtension

基于Firefox浏览器的附加组件(Add-ons)扩展是一种具有新功能的加载项。它们使用标准的JavaScript、Html、CSS等网页技术编写[28]，再加上一些专用的javascript API进行开发。附加组件的主要功能是可以为浏览器增加新的特性或者改变某些网站的外观。

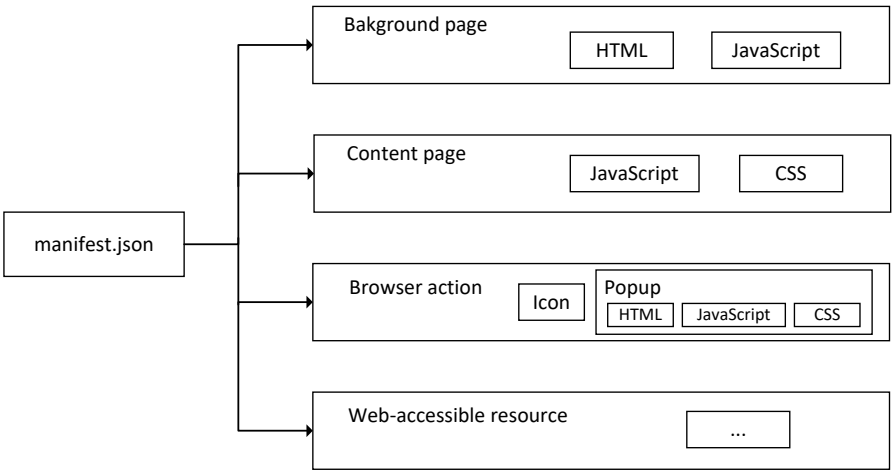


图 2-3 WebExtension 框架图

WebExtensions（扩展）是跨浏览器的用于开发附加组件的工具[7]，是目前浏览器开发扩展应用的通用工具。在很大程度上兼容谷歌浏览器Chrome和欧朋浏览器Opera所支持的扩展API ，这些浏览器上所写的扩展只需少量修改的便可在火狐浏览器FireFox和Microsoft Edge浏览器上运行，此外这些API与多线程Firefox完全兼容，因此本文选择了WebExtensions作为Firefox扩展应用的开发工具。图2-3所示的是本文中使用到配置信息以及整体框架图。

WebExtensions 的配置信息保存在一个名为 manifest.json 的文档中，以json的数据格式保存各项配置信息。这是惟一一个在每个 WebExtension 里面必须存在的文件。包含了关于这个扩展插件基本的元数据（metadata），比如它的名字、版本和所需权限，并对 WebExtension 中其他文件进行了链接。

Background page：WebExtension常常需要在浏览器窗口或特定网页的存活期中独立的维持一种长期的状态或者执行长期的操作，比如与服务器建立持久的连接，这个时候就需要background page中的JavaScript脚本来实现。运行在后台的文件不会因为当前界面关闭也停止，它的生命周期直到浏览器关闭才结束。

[7] https://developer.mozilla.org/zh-CN/Add-ons/WebExtensions

Content page：该文件夹下保存的文件称之为 content scripts，被用来获取、操作页面。这些脚本会被加载到页面中并运行在页面的特定环境下，可以像普通脚本一样获取、操作页面的 DOM。

Browser action：浏览器动作是指通过在工具栏上添加扩展的图标，使得用户可以点击并与之交互。该文件加下保存了扩展的图标以及初始界面的 html 文件等资源。

Web-accessible resources: 是指像图片、HTML、CSS 和 JavaScript 之类的外部资源，引入插件并且想要获得访问权限的内容脚本和页面脚本。成为 web-accessible 的资源可以在页面脚本和内容脚本中通过使用特定的 URL 方案来引用。

第三章 云备份中间件优化分析

本章主要通过对现有的中间件系统进行全面的分析，详细阐述其系统框架以及功能特性，在此基础上结合对国内外云备份服务平台的调研结果，指出当前中间件系统的不足之处，由此分析出优化的具体内容。

3.1 现有中间件系统分析

现有的云备份中间件系统是基于 Linux 系统的，它提供了客户端与备份服务器之间的数据交互，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的数据传输[19]。

3.1.1 系统框架与功能分析

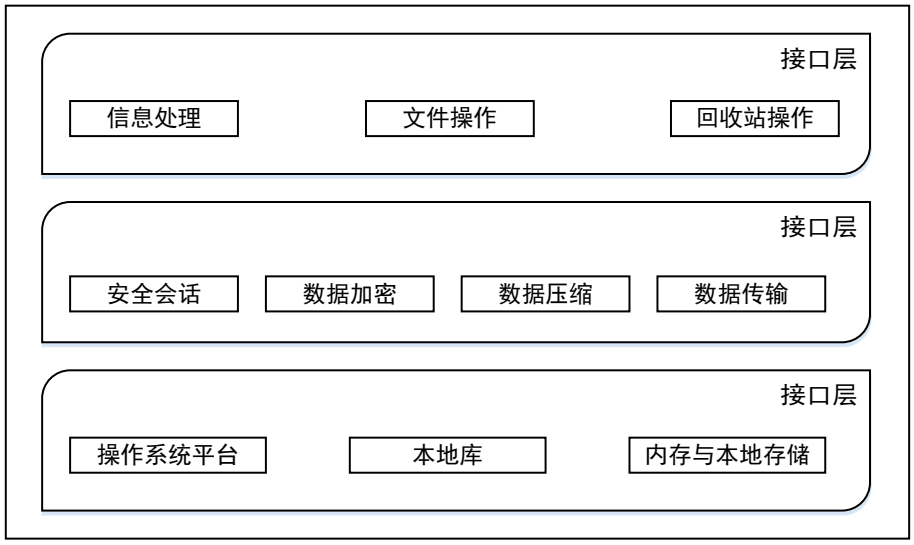


图 3-1 现有云备份中间件系统架构图

图 3-1 所示为现有的云备份中间件系统框架图。中间件作为客户端与服务器之间的交互枢纽，负责对用户从客户端发送的请求和数据进行处理，然后转发给服务器。中间件系统一共分为三层，底层结构包括操作系统平台、本地库、内存与本地备份三部分，它提供了中间件可以正常运作的基础软硬件；逻辑层由四大核心模块组成，分别是安全会话模块、数据传输模块、数据加密模块，以及数据压缩模块；接口层包括了为客户端提供的所有服务，这些服务可以分成三大类，分别是信息处理类、文件操作类，和回收站操作类。

目前，云备份中间件在功能上主要包括四个模块：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，各模块的说明简要如下：

- 数据传输模块：实现了服务器与客户端之间的数据交互，主要提供数据的上传与下载功能，这也是云备份中间件系统的基本功能。
- 数据加密模块：为了保证中间件系统能够抵御各种恶意网络攻击，中间件系会将数据以加密的形式进行网络传输。使用的是 AES 对称加密算法。
- 数据压缩模块：支持对文件的压缩功能。当上传文件时，用户可以通过 COMPRESS 参数指定对文件进行压缩，以减少数据传输量，降低所需的网络带宽，提高传输的效率。
- 安全会话模块：中间件需要保存用户的登录信息，以方便客户端与服务器之间建立持续的连接。现有的中间件系统将用户的登录信息保存在 SQLite 数据库中。

3.1.2 现有云备份中间件的不足

现有的中间件系统实现数据传输、加密、压缩和安全会话等功能，能够有效地将用户的数据传输到云端，并且具备一定的安全机制。然而，根据目前行业的发展趋势，现有的中间件系统存在很多的不足之处，具体来说有以下几个方面。

首先，现有的中间件系统没有考虑到数据传输模块是以网络作为基础的，默认网络在数据传输过程中是绝对稳定的。但是，在实际生活中，网络状况常常受限于硬件、软件以及地域等多方面的因素，网络波动或网络中断是常见的问题。由于网络带宽是有限的，因此上传文件，尤其是较大的文件，通常需要花费一定的时间，在此期间存在因网络失效而导致文件传输失败的问题。在现有的中间件系统里，用户只能重传该文件，倘若多次遇到网络故障，则可能导致多次的重传，这对用户来说是极不友好的体验。

其次，为保证用户数据的安全性，现有的中间件系统在文件传输过程中，系统采用了 128 位的 SSL 加密技术，在文件的存储形式上，让用户主动选择是否加密，使用的是 128 位 AES 加密算法，这样的加密策略与主流的云备份服务平台使用的方式基本一致。然而中间件需要保存一定的用户信息，以作为交互的保障。用户的登录信息、会话信息等敏感数据在现有中间系统中是以缓存的形式保存的，这些信息以明文的形式保存在 SQLite 数据库中，同时也未设置数据库的访问权限[29]。因此，用户的私密信息存在被窃取和篡改的可能。此外，由于会话等信息是以缓存的形式备份在中间件系统中的[30]，若中间件系统出现故障，则有可能造成用户会话信息的丢失。

再者，现有的中间件系统未对冗余的数据进行处理。当用户上传一个已有的文件到服务器时，服务器不会对文件进行去重检验，而是会重命名该文件，并以文件的原始形式保存。当冗余的文件过多时，会占用服务器大量的备份空间。去

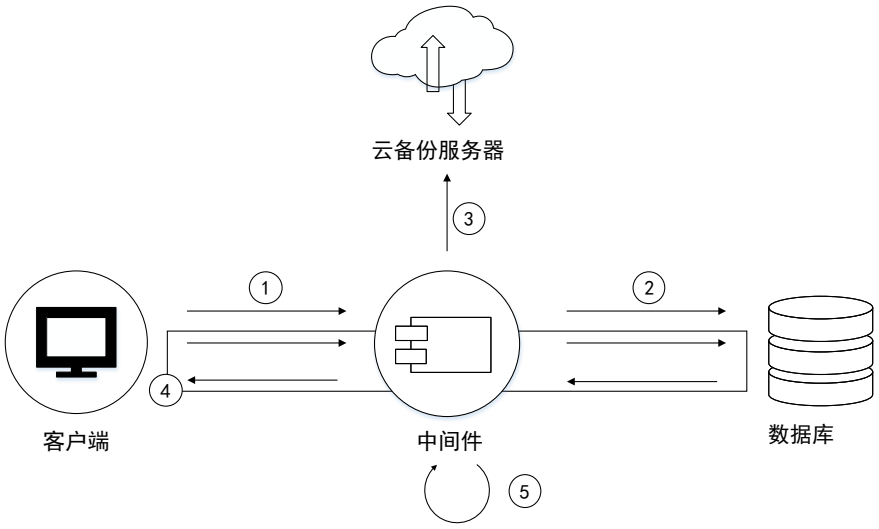
重技术虽然有一定的安全隐患，但是可以借鉴 Dropbox 针对单一用户的去重策略，这样既能缓解服务器的备份压力，又能为在一定程度上保证用户数据的安全。

最后，现有的中间件系统只支持 Linux 平台下的 CLI 客户端。考虑到主流的云备份服务供应商都提供了多平台数据共享的服务，因此需要对现有的中间件系统进行重构，解决来自多平台的不同客户端与云备份服务器之间的适配问题，使其具备可扩展性以支撑更多的客户端类型。

3.2 中间件系统优化需求分析

通过上文对现有中间件系统不足之处的分析，并结合业内同类产品关键技术的调研结果，本文将从持久会话管理、大文件上传、本地缓存、多应用适配四个方面进行优化，具体分析如下。

3.2.1 持久会话管理



①用户发送请求至中间件；

②中间件将用户的会话信息加密后保存至数据库；

③中间件将用户的请求发送至云服务器；

④用户发送查询、撤销会话请求，中间件从数据库中获取会话信息解密后发送给客户端；

⑤中间件系统遇到故障重启后，恢复执行未完成的会话。

图 3-1 持久会话管理过程分析图

云备份中间件在客户端和服务器之间工作，多个客户端的命令通过中间件提交给服务器，用户在访问备份服务器上的资源时，需要提供身份和授权信息，因此中间件也需要保存用户的私密信息。由于存在多用户同时将命令提交给中间件的可能，因此用户的命令不会马上被执行，而是以缓存的形式备份在数据库中，只有被处理请求的进程轮询到时才会被执行。在现有的系统中，用户的会话信息

是以明文的形式保存的，因此需要设计一种行之有效的策略来保证用户私密信息的安全。具体来说，可以从以下几个方面进行优化：

- 不同用户的数据，按照会话进行物理隔离，一个用户只能访问自己所属会话的数据。
- 明文形式的秘密信息，只能作为会话数据在内存中存在。
- 用户的持久私密信息以加密的形式存储，通过会话管理模块只能获取到加密形式的数据。

此外，客户端可以批量提交任务给中间件，即使用户注销后，中间件仍要在后台继续执行任务。更进一步，在中间件系统重启后，中间件也需要自动执行未完成的任务。因此，中间件需要持久化保存多个客户端的请求，以可靠地执行客户端提交的任务。同时，本文需要设计一套基于会话信息的接口，便于用户查询已提交的请求，如果请求没有执行，用户可以撤销该任务。图 3-1 详细描述了持久会话管理的过程。

3.2.2 大文件上传

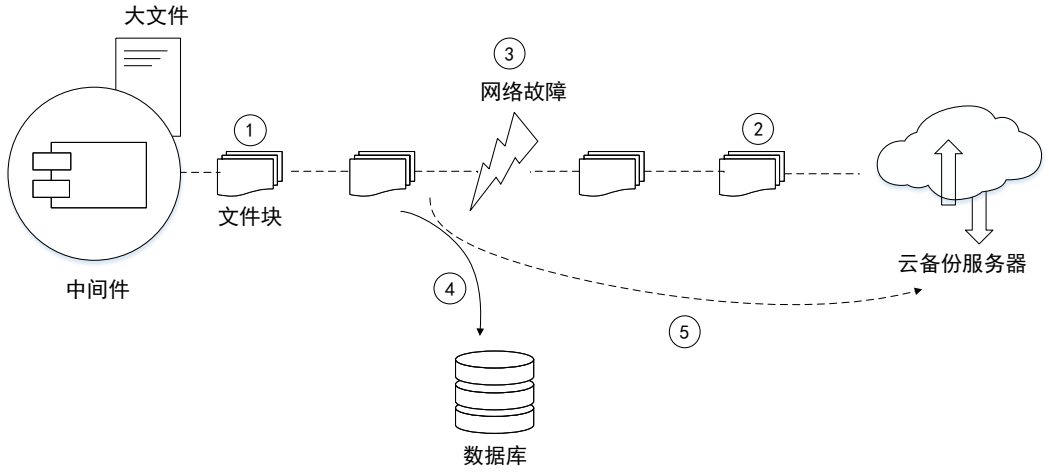
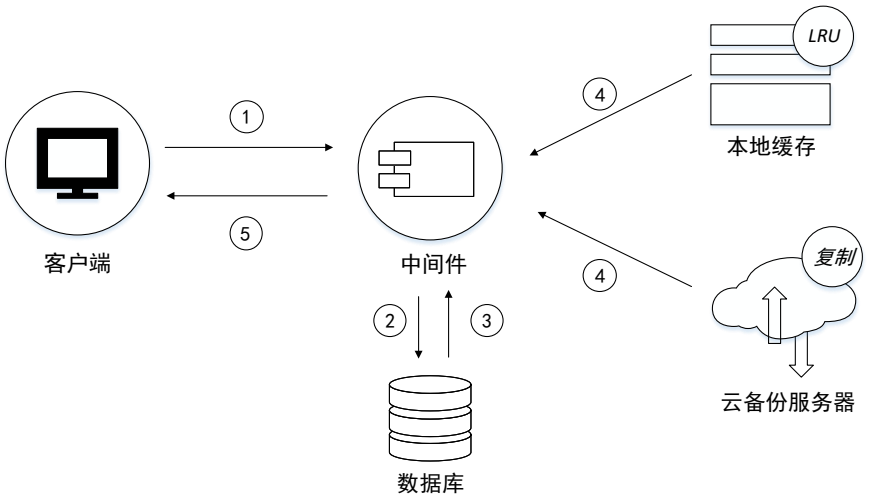


图 3-2 大文件上传及断点续传过程分析图

在很多情况下，客户端和云备份服务器分布在不同的地域，网络传输速度受各种条件制约。在现有的中间件系统中，一个文件如果传输不成功，下一次只能重传，这浪费了大量的网络带宽和时间。事实上，服务器支持大文件分块传输的功能，也就是说客户端可以传输指定大小的文件数据。因此，中间件可以根据网络数据传输的质量，确定文件分块传输的大小，并在每次上传时记录文件以上传

的块数，即已上传文件的偏移量。如果遇到网络故障，可以根据偏移量实现断点续传的功能。这样做能够很大程度上地减少因网络因素导致的数据重传问题，提高大文件传输的效率。图 3-2 详细描述了大文件上传的过程。

3.2.3 本地缓存



1 用户发送上传/下载请求至中间件；

2 中间件查询服务器/本地缓存中是否存在用户待上传/下载的文件；

3 返回数据库查询结果；

4 上传文件上，若服务器存在该文件，则将服务器中的文件复制到用户的上传路径；
下载文件时，若本地缓存存在该文件，则将缓存中的文件复制到用户的下载路径；
若不存在缓存文件，则执行正常的上传/下载流程；

5 将请求结果返回给用户。

图 3-3 本地缓存示意图

云备份是网络 I/O 密集的服务，大量的 I/O 会引起机器性能下降，也可能会干扰其它的应用。现代计算机通常有很大的本地备份空间，并且计算能力普遍过剩。因此，可以通过利用本地存储和计算能力，如本地缓存的利用，减少 I/O 操作，提高系统性能。

现有的中间件系统在接受到用户文件上传和下载时，首先将文件存储在临时区域，该区域只能被中间件访问。由于客户端有远程访问文件的功能，所以用户可能会反复访问同一个文件，如果每次都从远程下载该文件，则是对网络带宽资源的浪费。因此，需要高效的临时备份区域，以降低下载文件时所需要的 I/O 请求。考虑到本地缓存的具有一定的容量限制，当缓存容量达到上限时，可以采用 LRU 替换策略以保证缓存文件的时效性。

此外， 同一个用户备份数据时会有相同的文件。在上传文件时，同一用户的相同文件是可以复用的，可以利用服务器提供的复制文件接口，直接在服务器操作，以减少上传文件所需的网络带宽，从而实现“秒传”功能。图 3-2 详细描述

了本地缓存的过程。

3.2.4 多应用适配

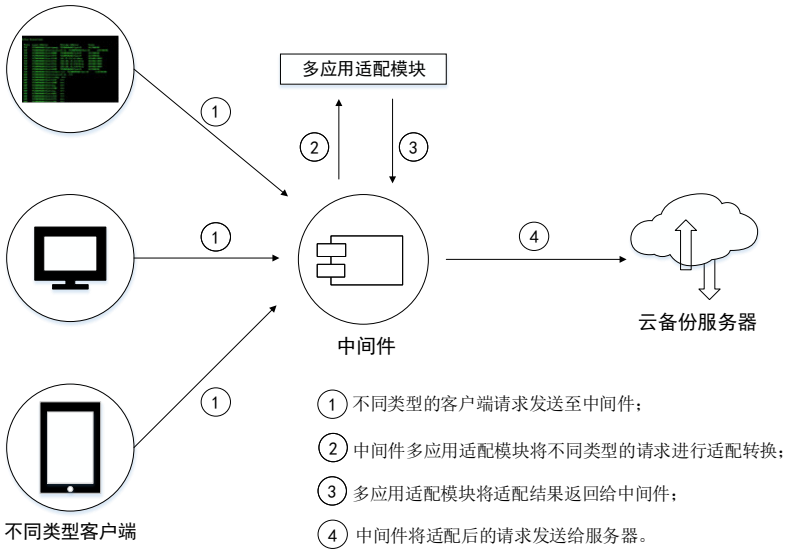


图 3-4 多应用适配示意图

作为中间件，其主要的目标之一是完成不同客户端和服务器之间的适配，以使平台具备一定的可扩展性。由于平台和操作系统之间的差异性，加上不同客户端在设计时考虑的情形和目标不一样，客户端和服务端通常会定义不同的接口，导致服务器和客户端之间请求不匹配。

为了减少客户端和服务端之间的差异，可以通过中间件来对请求的形式进行转换，适配分为两种情况：

- 提供一个标准的接口，客户端调用中间件给出的接口；
- 在中间件中引入新的适配模块，以较小的代价将不同的客户端和服务端整合在一起。

第一种情况适合于规范新开发的客户端，从用户使用的角度来统一定义一个访问接口。第二种情况，实际上是第一种情况的强化形式，需要中间件来设计更灵活的机制来支持更一般化的适配。在这里，本文选择了第二种方式作为中间件多应用适配的方案。

为了适配客户端发出的请求（输出形式）和服务端的接收请求（输入形式）之间的差异，中间件需要对来自不同客户端的请求进行适配，以服务器的标准接口规范作为基准，转化为统一的、服务器可识别的形式。同时，由于目前版本的中间件只有 CLI 一个客户端，因此还需要开发其他类型的客户端来支撑多应用适配，本文选择了开发基于 Firefox 浏览器的扩展应用和 Java SDK 开发工具。图 3-4 详细描述了多应用适配的过程。

第四章 云备份中间件系统优化设计

4.1 中间件优化系统框架设计

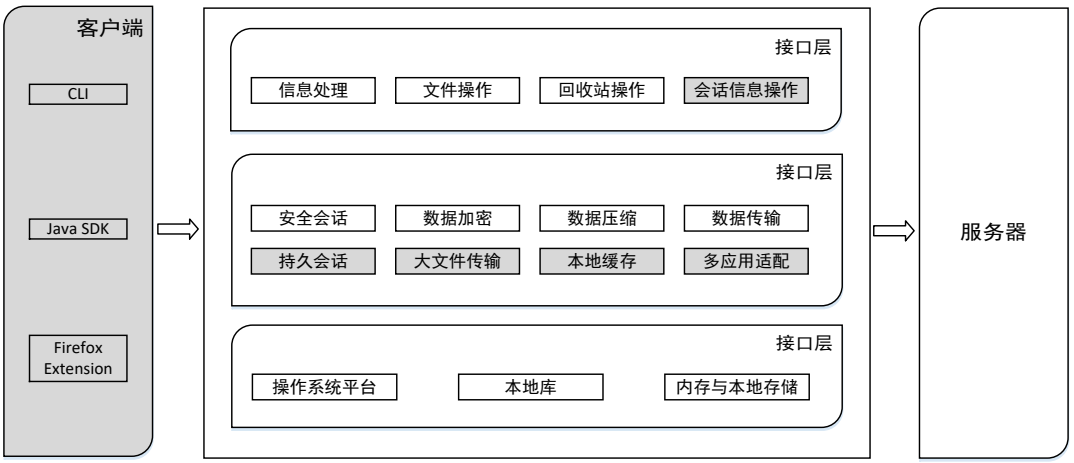


图 4-1 云备份中间件优化系统结构图

图 4-1 所示为优化后的中间件系统框架设计图，灰色区域为新增部分，在现有中间件系统基础上，主要对第二层架构进行扩展，新增了持久会话模块、大文件上传模块、本地缓存模块以及多应用适配模块。同时为支持优化的中间件系统的可扩展性，新增两款客户端，分别是 Firefox 扩展应用和 Java SDK。

4.2 类图设计

本文对现有的中间件系统源代码进行了优化重构[31]，在类图的设计上也做了较大的改变。由于现有的中间件系统只实现了数据上传、加密、压缩这三项基本功能，因此类图设计相对比较简单，由 mwServer 类接收用户请求，该类的主要作用是启动系统的配置文件，并在接收到用户的请求后为其分配一个线程，该线程通过调用 mwController 类中的方法将用户请求发送给服务器，并将服务器返回的结果返回给 mwServer 类，最终由 mwServer 类将结果返回给客户端。此外，现有的中间件系统提供对数据的加密和压缩功能，这两个功能分别由 mwEncrypter 类和 mwCompresser 类实现。

依据类图设计的标准[33]，本文重新设计了中间件系统的类图，并对类图进行的分类。如图 4-2 所示，优化后的中间件系统的 UML 类图设计分为四个模块，分别是 Core 模块、Features 模块、Utils 模块以及 Database 模块，每个模块具有特定的功能。相较于现有的中间件系统的实体类，优化后的中间件系统增加了许多实体类，主要用于实现各优化项的功能以及用于源码进行重构，优化代码的组织结构和提高系统的可维护性。

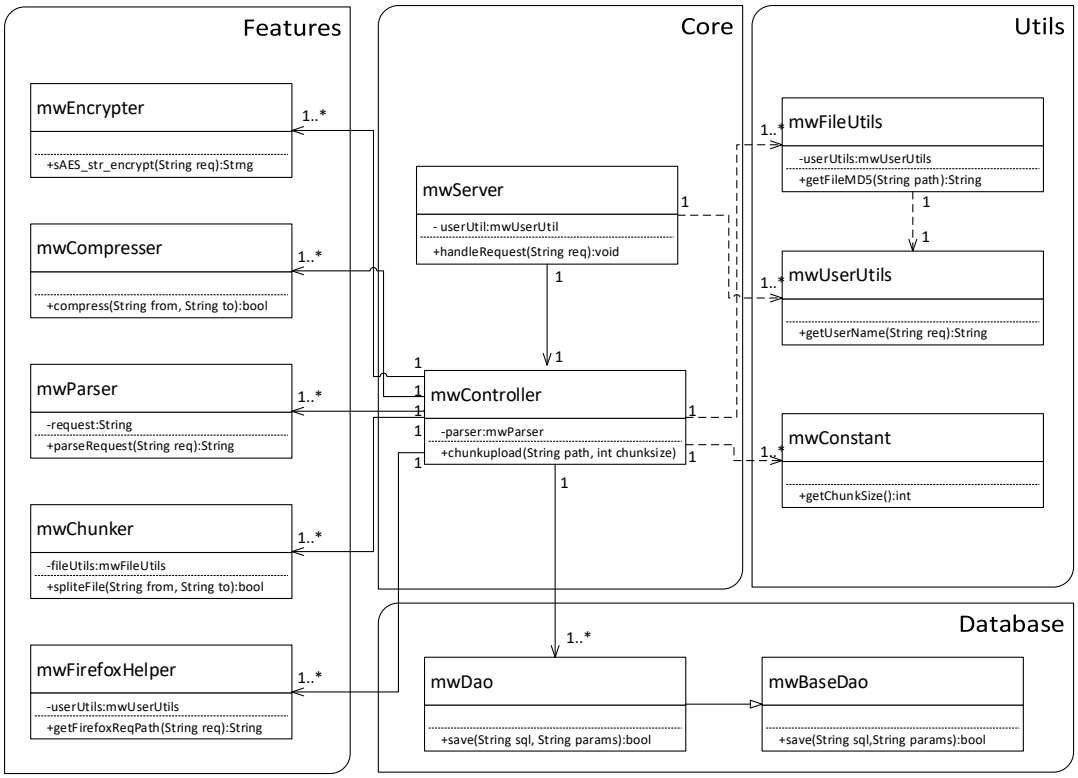


图 4-2 云备份中间件优化系统 UML 类图

Core 模块的类是中间件系统的核心，负责接收和处理用户的请求。通过调用其他模块的类方法，将用户的请求做格式化处理，并将预处理后的数据传输至服务器。接收到来自服务器的返回结果后，将结果以 Json 的数据格式返回给客户端；Features 模块的类主要负责优化功能的实现，包括优化的几大主要的功能模块，如大文件分块传输、本地缓存以及多应用适配的核心功能都封装在该模块的类中；Utils 模块的类提供开发所需的常用方法和常量信息，用来同一管理和维护系统变量；Database 模块的类负责与数据库的交互，提供基本的增、删、改、查接口以及其他高级数据库操作。具体类的功能如下：

mwChunker：该类主要用于大文件传输，将大文件拆分为固定大小的文件块，并按当前用户的信息以及时间戳信息自动命名，保存在缓存中。此外，该类还需记录上传文件的状态，如断点信息，为断点续传技术的实现提供支持。

mwConstant：该类的主要作用是为系统提供全局常量，如字符串常量、整型的数字常量（主要用来标识状态或异常信息），并对这些常量进行统一管理。在现有的中间件系统中，常量都是以独立的形式存在于各类中，这样存在大量冗余的常量信息，且在对源码进行维护时，难以找全所有的常量进行统一修改，这给开发和维护工作带来了很大的困扰。

mwDao：该类主要负责数据库层接口的定义与实现，实现业务逻辑与数据库操作的分离。在现有的中间件系统中，数据库连接在需要的时候创建，并在执行完毕后关闭。这样的做法使得对数据库的操作难于管理，因此在优化的过程中，

本文将数据库层的操作与业务层的操作进行了分离。在 mwDao 类中，提供了大量与数据库操作相关的接口，如 add，delete，update，query 等，这些接口使得系统的实现层次更加清晰，并且便于修改和管理。

mwFileUtil: 该类的作用是处理与文件相关的请求，如获取文件的大小、MD5 值、复制文件等。这些操作对于云备份服务来说十分常见的，因此将这些方法统一封装在 mwFileUtil 类中，以方便在其他模块中使用。

mwUserUtil: 该类的主要作用是获取用户的信息，如用户的 id、用户的 token、查询用户登录状态是否过期。这些在 service 层中是经常使用到的信息，因此对这些功能进行了封装。

mwFirefoxHelper: 该类的主要作用是处理来自 Firefox 扩展的请求。由于 Firefox 扩展请求流程的特殊性，使用到了 WebSocket 作为服务器与客户端之间的通信方式，与 Socket 通信方式不同的是，它需要进行一次握手协议，因此需要对来 Firefox 的请求进行单独处理。

mwParser: 该类的主要目的是适配来自不同客户端的请求，并将请求转化为服务器可识别的标准形式。此外该类预留了接口，使得中间件系统具备一定的可扩展性，为以后开发更多种类的云备份客户端奠定基础。

4.3 关键流程设计

在系统总流程上，优化后的中间件系统并未改变原有的中间件系统的整体流程，作为云备份服务客户端与服务器交互的枢纽，中间件主要负责对用的请求和数据进行预处理操作。客户端向中间件发送请求后，在原有的中间件系统中，只需将用户的请求通过解析类进行格式化操作，即可传递给服务器。而在优化后的中间件系统中，将进行持久会话管理操作，当用户的请求到达中间件后，首先会将用户的登录信息 id、token 等以及请求内容保存在 SQLite 数据库中，同时由于这些信息中包含用户的隐私，因此会首先对这些信息加密，然后再存入数据库中。此外，用户的请求可能来自不同的客户端，它们可能使用不同的通信协议，不同的请求信息，因此需要对来自不同客户端的用户请求进行适配。用户的请求类型可分为三类，若是与数据传输相关的请求，需要首先判断本地缓存中或者服务器端有没有当前文件，如果存在，那么执行文件复制操作，若不存在，则执行正常的文件长传下载操作。再者，需要判断上传文件的大小，如果超过规定的文件块大小，那么需要执行大文件分块传输；若是与会话相关的请求，则根据请求的内容，通过数据库操作，删除或是返回相应的会话信息；其他类型的请求则与原有中间件系统的流程一致，经过简单适配之后发送到服务器，再将服务器返回的结果发送到客户端。中间件系统优化后总流程如图 4-3 所示。

4.3.1 系统总流程设计

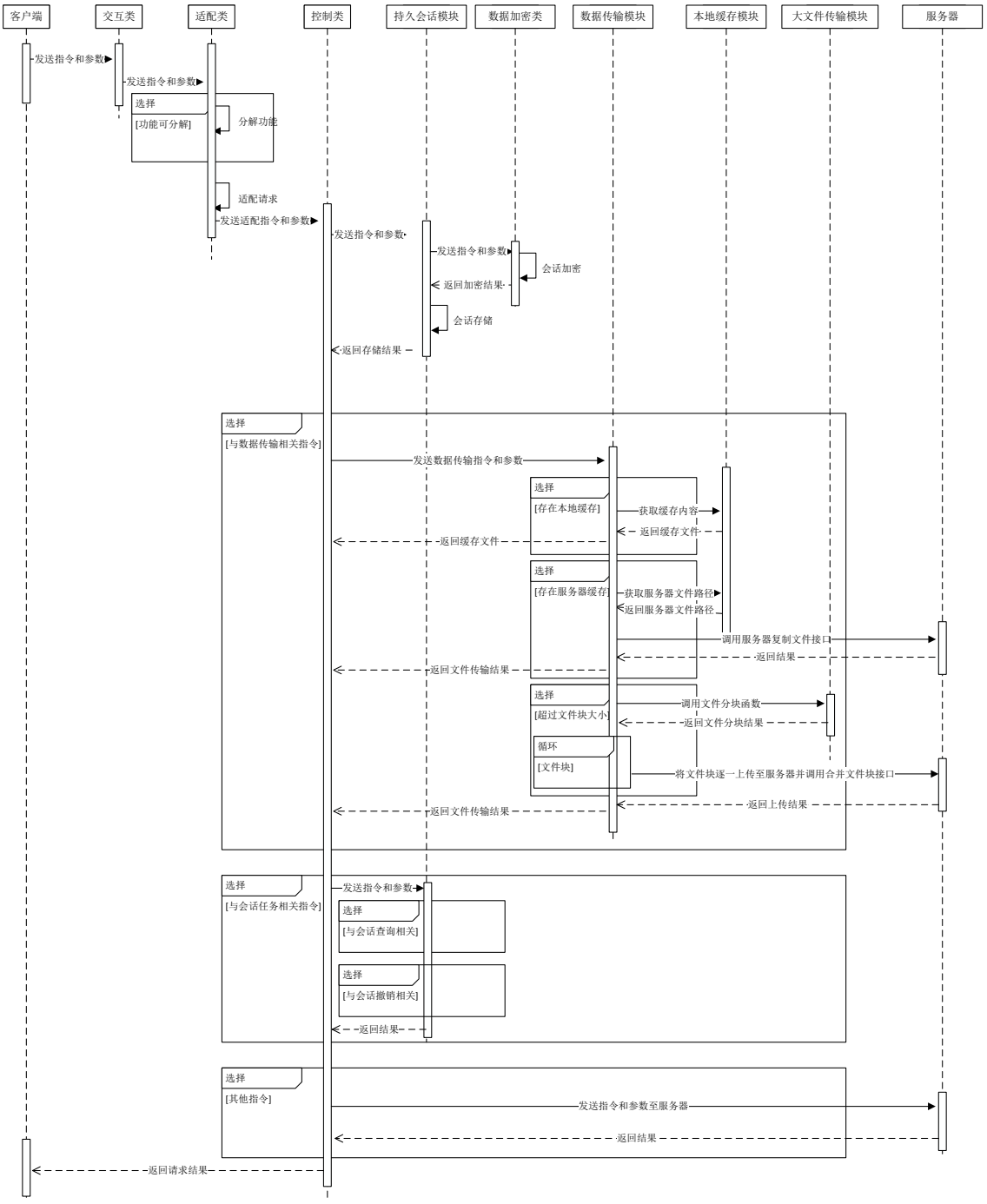


图 4-3 云备份中间件系统总体流程图

4.3.2 持久会话管理流程设计

云备份中间件系统在客户端和服务器之间工作，多个客户端的命令通过中间件提交给服务器。当用户注销或者在机器重启的情况下，中间件系统仍然需要执行未完成的任务。因此，中间件需要持久化保存多个客户端的请求，以可靠地执

行客户端提交的任务。同时，用户在访问备份服务器上的资源时，需要提供身份和授权信息，因此中间件也需要保存用户相关的私密信息。会话持久化管理流程设计包括会话持久化与会话加密两部分内容。

1）会话持久化流程设计

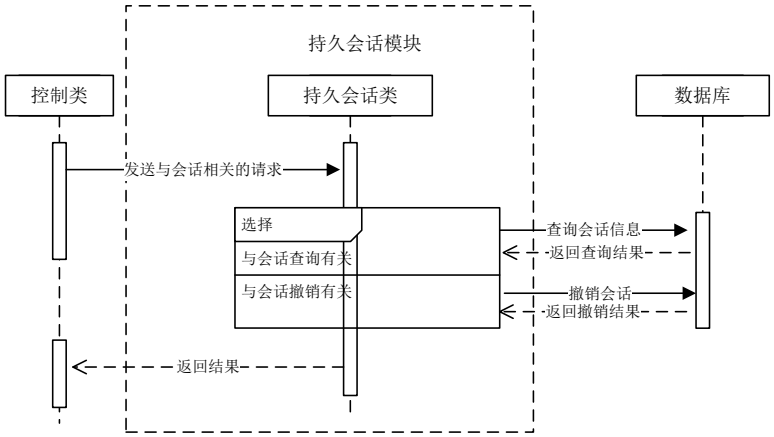


图 4-4 云备份中间件系统会话持久化流程图

如图 4-4 所示，中间件接收到客户端的请求后，首先将请求发送给持久会话模块进行会话持久化管理。若请求中含有与会话查询有关的指令，持久会话模块会从数据库中查询该条指令，并调用数据加密模块进行相应的解密操作，然后将结果返回给客户端；若请求中含有与会话撤销有关的指令，则直接在数据库中将相应的会话删除，并将删除结果返回给客端。

2） 会话加密流程设计

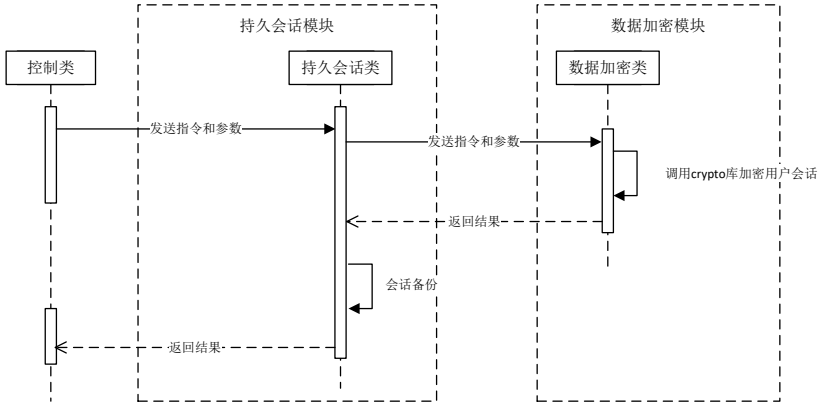


图 4-5 云备份中间件系统会话加密流程图

如图 4-5 所示，中间件接收到客户端的请求后，需要对会话进行持久化管理。若将会话以明文的形式保存在数据库中，则可能导致用户信息泄露，因此在对会话进行备份之前，首先调用数据加密模块对用户会话进行加密，然后再存入数据库中，并将结果返回给客户端。

4.3.3 大文件传输流程设计

大文件传输利用本地缓存和服务器的高级功能，解决在网络状况不佳的情况下，文件传输的稳定性问题。以下展示的是大文件传输的流程设计，包括大文件上传和下载两部分。

1） 大文件上传流程设计

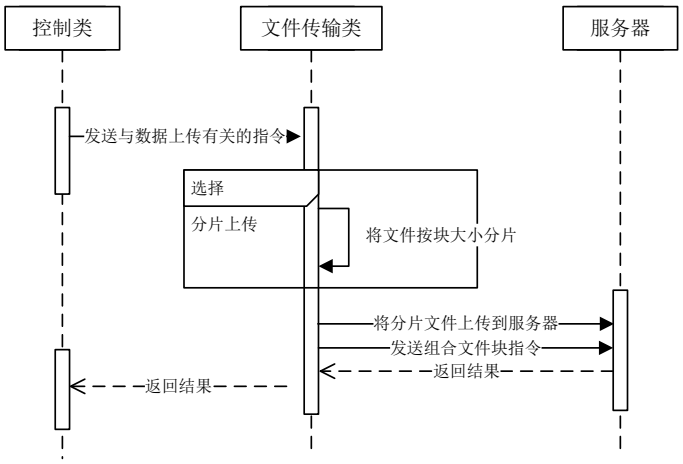


图 4-6 云备份中间件系统大文件上传流程图

如图 4-6 所示，中间件接收到数据上传的指令后，首先会对文件大小进行判断。若是大文件则调用分块上传函数，将大文件分成固定大小的临时小文件后逐一上传，之后再向服务器发送文件整合的命令，将分块上传的文件进行整合，并将执行结果返回给控制类。

2） 大文件下载流程设计

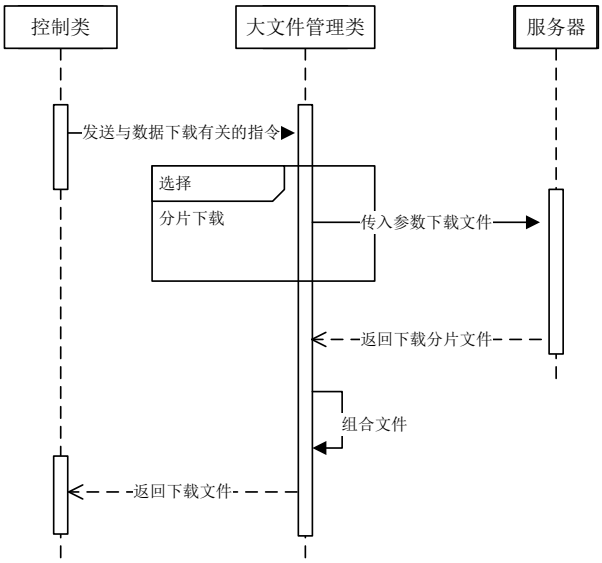


图 4-7 云备份中间件系统大文件下载流程图

如图 4-7 所示，中间件接收到数据下载的指令后，首先会对文件大小进行判断。若是大文件则调用分块下载函数，每次请求包含文件块的起止索引和大小，通过多次下载请求任务完成大文件的下载。将所有分片文件下载完成后，对文件块进行组合并将结果返回给客户端。

4.3.4 本地缓存流程设计

用户在执行上传和下载操作时，可能会反复访问同一文件，这时可用充分利用本地临时存储区域以及服务器端的文件，将反复被访问的文件做缓存处理，这样做可以在客户端和服务器端上降低系统 I/O，提高系统的传输效率。以下展示的是本地缓存管理模块的流程设计，包括数据上传和下载缓存管理流程设计两个部分的内容。

1） 数据上传本地缓存流程设计

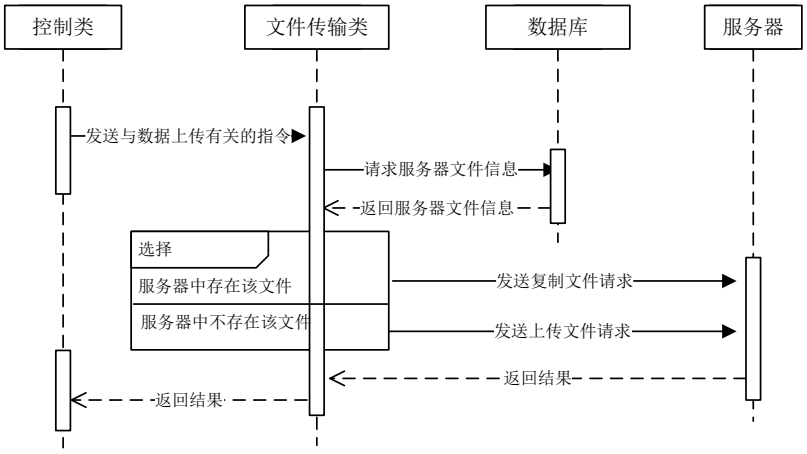


图 4-8 云备份中间件系统数据上传缓存管理流程图

如图 4-8 所示，中间件接收到用户上传文件的指令时，首先获取待上传文件的信息，包括文件大小和 MD5 值等，根据数据库中记录的服务器端的文件信息，通过 MD5 值校验，判断服务器中是否存在该文件。若服务器中存在该文件， 则调用服务器提供的复制文件接口，将目标文件复制到上传路径；如服务器不存在该文件则执行正常的上传任务。

2） 数据下载缓存流程设计

如图 4-9 所示，中间件接收到用户下载文件的请求后，首先会查询数据库中的本地缓存文件信息，根据 MD5 值校验是否存在待下载的文件。若存在该文件，则直接从本地缓存中将该文件复制到下载路径；若不存在该文件，则从服务器端下载该文件。

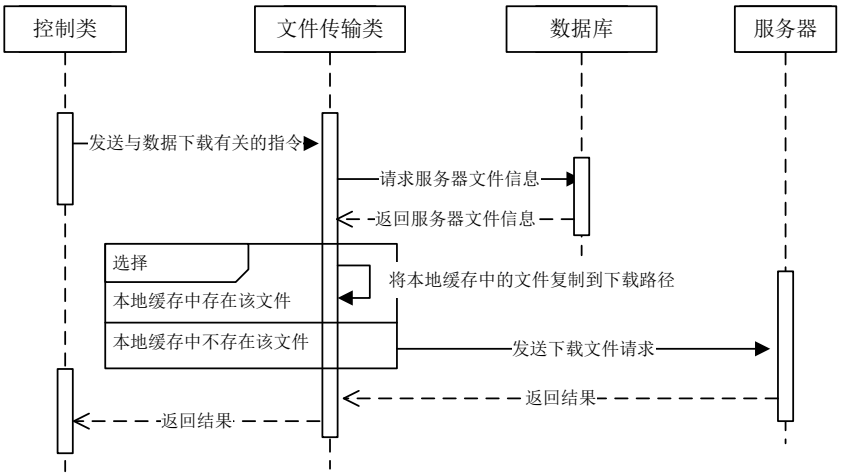


图 4-9 云备份中间件系统大文件下载流程图

4.3.5 多应用适配流程设计

由于操作系统与编程语言的差异性，且设计时考虑的情形和目标不一样，客户端和服务端之间通常会定义不同的接口，导致请求不匹配。交互适配模块的主要的目标是完成不同客户端和服务之间请求的适配。以下展示的是交互适配模块进行命令接口转换时的流程设计。

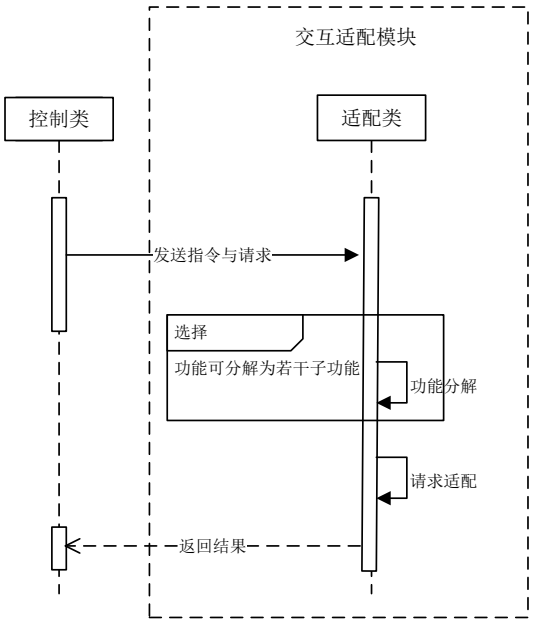


图 4-10 云备份中间件系统交互适配模块流程图

如图 4-10 所示，交互适配模块接收到客户端交互模块发送的指令后，首先判断该指令中的请求是否可以分解为若干个子功能，若能分解为若干个子功能，如删除会话信息请求需要分解成为查询会话信息和删除选定会话两个子功能，大文件上传需要拆分成文件分块和文件上传两个子功能，则先将指令分解然后逐一

适配，并将适配后的指令交给控制模块处理，最后将处理后的结果进行整合返回给客户端。

4.4 数据库设计

中间件系统优化运行过程中需要存储数据信息，如对用户信息加密备份、记录大文件上传断点信息、记录本地和服务器文件的缓存信息，这些信息都需要保存在数据库中。本文使用的是 SQLite 关系型数据库，之所以选择 SQLite，是因为它是一款轻量级的数据库，且可以在几乎所有的主流操作系统上运行，并支持大多数的 SQL92 标准，源代码不受版权限制。此外，最大支持 2TB 的备份空间也足够备份缓存信息。同时在处理事务的能力上，也具备一定的优势。

本文一共设计了三张表，分别用来保存用户的会话信息、缓存信息以及断点信息。详细信息如下：

表 4-1 用户会话信息数据库表

```
create table if not exists log(
    request text not null,              //请求内容
    uuid char[33] not null,             //请求标识
    state char[20] not null,            //请求状态
    time char[20] not null,             //请求时间
    userid varchar[100] not null,       //请求用户 id
    result text not null);              //请求执行结果
```

表 4-1 所示为用户会话信息数据库表，request 属性保存用户的请求内容；uuid 表示是请求的唯一标识；state 属性为请求的执行状态，一共有三个值，unfinish 表示请求尚未执行，not return 表示请求已被执行但是未将结果返回给客户端，finished 表示请求已完成；time 表示请求提交到中间件的时间；userid 为用户的 ID 信息；result 记录请求的执行结果。

表 4-2 缓存信息数据库表

```
create table if not exists fileCache(
    userid varchar[100] not null,              //用户 id
    file_name varchar[100] not null,           //文件名
    length bigint not null,                    //文件长度
    MD5 varchar[100] not null,                 //文件 MD5 值
    use_time int not null,                     //使用次数
    modified_time varchar[100] not null);      //最后一次修改时间
```

表 4-2 所示为缓存信息表，记录存储在本地缓存空间的文件信息。Userid 为用户的 ID 信息；file_name 为文件名；length 表示文件的字节长度；MD5 存储文件的 MD5 值；use_time 记录该文件的使用次数；modified_time 记录文件最后一次使用的时间。

表 4-3 断点信息数据库表

```
create table if not exists breakpoint(
    userid varchar[100] not null,          //用户 id
    MD5 varchar[100] not null,             //文件 MD5 值
    server_path varchar[100] not null,     //文件在服务器的位置
    point int not null default 0,          //断点偏移量
    flag bit not null default 0);          //标识是否需要断点
```

表 4-3 所示为断点信息表，记录文件上传时遇到网络故障的断点信息。Userid 为用户的 ID 信息；MD5 存储文件的 MD5 值；server_path 记录文件上传至服务器的路径；point 表示断点的偏移量；flag 标识当前文件是否需要断点续传。

4.5 核心接口设计

程序接口是操作系统为用户提供的两类接口之一，编程人员在程序中通过程序接口来请求操作系统提供服务，同时也是应用程序提供给用户使用程序的入口[34]。现有的中间件系统为用户提供了大量的接口，满足了用户使用云备份服务的基本功能。优化后的中间件系统为用户提供了更多的高级功能，这些功能都是以接口的形式呈现给用户和开发人员，下面将对中间件系统优化的核心接口进行详细的阐述。

4.5.1 持久会话管理接口设计

表 4-4 新增会话

接口编号	接口名称	新增会话
Cloud-Opt-01	接口原型	bool add_session(String request, String userid)
	接口参数	request：用户请求 userid：用户 id
	返回结果	True：新增会话成功 False：新增会话失败

表 4-4 所示为新增会话接口，由控制类在用户请求到达中间件时调用，以用户请求作为第一个参数，以用户的 id 作为第二个参数。存储在 SQLite 中的用户

会话信息除了请求和用户 id 两个属性之外，还有请求标识 uuid、请求时间 time、请求状态 status、请求返回结果 result 等属性，这些信息在接口的实现过程中，根据当前的系统环境和服务器返回的结果生成。返回的信息为新增会话信息的结果。

表 4-5 删除会话

接口编号	接口名称	删除会话
Cloud-Opt-02	接口原型	bool delete_session(String sessionid, String userid)
	接口参数	sessionid：会话 id userid：用户 id
	返回结果	True：删除会话成功 False：删除会话失败

表 4-5 所示为删除会话接口，在用户发送删除会话请求到中间件时调用，该接口以会话 id 作为第一个参数，用户 id 作为第二个参数。在接口的实现中，会校验当前用户 id 与传入的参数 id 是否一致，如果一致则根据会话 id 删除用户指定的会话；若 id 不一致则返回删除该会话失败。此外，通常情况下，用户是不知道会话 id 具体信息的，因此在执行删除会话操作之前，需要结合查询会话接口使用，获取到会话的 id 信息后，再执行删除会话操作。返回给用户的信息为删除会话的结果。

表 4-6 查询会话

接口编号	接口名称	查询会话
Cloud-Opt-03	接口原型	String query_session(String userid, int recent)
	接口参数	userid：用户 id recent：最近会话次数

	返回结果	{
		"status": "0",
		"request": "curl -k -X POST -d '{"password": "123456", "email":"zhu__feng006@163.com"}'
		https://localhost:443/oauth/access_token",
		"state": "finished",
		"time": "1432653661.66864",
		"userid": "kaeyika@163.com"
		}

表 4-6 所示为会话查询接口，在用户发送查询会话信息请求到中间件时调用，该接口以用户 id 作为第一个参数，以最近会话次数 recent 作为第二个参数。返回的结果根据 recent 参数指定的会话个数，以 json 的数据格式，按会话的时间顺序返回给用户。返回的会话信息包括状态码 status、请求内容 request、请求状态 state、请求时间 time 以及用户 id 信息 userid。

表 4-7 会话加密

接口编号	接口名称	会话加密
Cloud-Opt-04	接口原型	String encrypt_session(String request)
	接口参数	request：用户请求
	返回结果	encrypt_req：加密后的请求信息

表 4-7 所示为会话加密接口，当用户的请求到达中间件时调用，该方法为 mwEncrypter 类的成员方法，用于对用户的会话信息加密，使用的是 128 位的 AES 加密算法。该接口以用户的请求作为唯一的参数，返回加密后的请求信息。

表 4-8 查询登录状态

接口编号	接口名称	查询登录状态
Cloud-Opt-05	接口原型	String query_session(String userid)
	接口参数	userid：用户 id

	返回结果	{
		"status": "0",
		"expired": "false",
		"time": "1432653661.66864",
		"userid": "kaeyika@163.com"
		}

表 4-8 所示为查询登录状态接口，在中间件提交用户请求到服务器之前调用，以检验用户的登录信息是否过期，如果没有过期则将请求发送给服务器；反之则由中间件授权代理登录，并将重新获取到的 token 替换请求中已过期的 token。确保不会因为登录状态过期而导致请求失败。

4.5.2 大文件上传接口设计

表 4-9 获取文件信息

接口编号	接口名称	获取文件信息
Cloud-Opt-06	接口原型	String get_file_info(String file_path)

	接口参数	file_path：服务器文件路径

	返回结果	{
		"status": "0",
		"Content-Length": "200",
		"X-File-Type": "f",
		"ETag":
		"7add118c851212736b362105e6235b84",
		"X-Timestamp": "1438679161.39779",
		"X-Object-Permisson": "500",
		"mode": "COMPRESS"
		}

表 4-9 所示为获取文件信息接口，在判断是否存在本地缓存时使用，以服务器文件路径作为参数，获取在服务器的文件详细信息，包括文件长度 Content-Length、ETag 即文件的 MD5 值以及其他信息。中间件系统可以根据文件的 MD5 判断本地缓存中是否存在该文件。

表 4-10 大文件分块

接口编号	接口名称	大文件分块
Cloud-Opt-07	接口原型	bool split_file(String from, String to)
	接口参数	from 目标文件路径 to：存储文件块的路径
	返回结果	True：文件分割成功 False：文件分割失败

表 4-10 为大文件分块接口，当传输的文件大小超过中间件系统规定的文件块大小时，会调用大文件分块接口将文件分割成固定大小的文件块。from 参数表示大文件的路径，to 参数表示存储文件块的路径，该接口返回文件分块的结果。

表 4-11 大文件分块上传

接口编号	接口名称	大文件分块上传
Cloud-Opt-08	接口原型	String chunk_upload(String file_path, String server_path, String user_id)
	接口参数	file_path：文件路径 server_path：服务器路径 userid：用户 id

	返回结果	{
		"status": "0",
		"X-Object-Permisson": "500",
		"path":" “/temp”,
		"size":" 200",
		"ctime":" 1438679161.39779",
		"md5":"
		7add118c851212736b362105e6235b84",
		}

表 4-11 为大文件分块上传接口，在待传输的文件大小超过中间件规定的文件块大小时调用，file_path 参数表示待传输文件的路径，server_path 表示传输到服务器的路径，userid 表示用户的 id 信息。该接口在实现的过程中会调用大文件分块接口，然后将文件分块后逐一上传到服务器。返回文件上传的结果，包括上传状态 status、文件的上传路径 path，文件大小 size、上传时间 ctime 以及 MD5 值。

4.5.3 本地缓存接口设计

表 4-12 是否有本地缓存

接口编号	接口名称	是否有本地缓存
Cloud-Opt-09	接口原型	bool has_local_cache(String server_path, String userid)
	接口参数	local_pat：服务器文件路径 userid：用户 ID

	返回结果	{
		"status": "0",
		"cached": "true",
		"path":" “/temp/test.docxs”,
		"size":" 200",
		"md5":"
		7add118c851212736b362105e6235b84",
		}

表 4-12 为是否有本地缓存接口,在用户发送下载文件请求至中间件时调用，server_path 表示文件在服务器的路径，userid 为用户的 id 信息。该接口在实

现的过程中调用了获取文件信息接口，以获取服务器文件的 MD5 中，再通过数据库查询，通过文件 MD5 的比较判断是否存在本地缓存，并将结果返回。

表 4-13 是否有服务器缓存

接口编号	接口名称	是否有服务器缓存
Cloud-Opt-09	接口原型	bool has_local_cache(String local_path, String userid)
	接口参数	file_pat：本地文件路径 userid：用户 ID

	返回结果	{
		"status": "0",
		"cached": "true",
		"path":" “server/test.docxs”,
		"size":" 200",
		"md5":"
		7add118c851212736b362105e6235b84",
		}

表 4-13 是否有服务器缓存接口，在用户发送上传文件请求时调用，local_path 表示待上传的文件路径，userid 表示用户 id 信息，通过查询本地数据库中保存的服务器文件信息，比较 MD5 值判断服务器是否存在待上传的文件，并将结果返回。

4.5.4 多应用适配接口设计

表 4-14 请求适配

接口编号	接口名称	是否有服务器缓存
Cloud-Opt-09	接口原型	bool adapt_request(String request)
	接口参数	request：原始请求
	返回结果	String：adapted_request //适配后的请求

表 4-14 为请求适配接口，在用户请求到达中间件时调用，根据请求的特征，判断请求的类型，并将请求转换为服务器可识别的标准形式，将适配后的请求返回。

第五章 云备份中间件系统优化实现

5.1 开发环境

表 5-1 中间件优化开发环境表

类别	具体参数
操作系统	Ubuntu 14.04
开发语言	Python、 Java、JavaScript、HTML
开发工具	Atom、WebExtension
数据库	SQLite

如表 5-1 所示，云备份中间件系统优化是在 Ubuntu14.04 下进行的，使用的编程语言有 Python、Java、JavaScript 以及 HTML，使用了 Atom 作为代码编辑和调试的工具，采用了 WebExtension 框架开发 Firefox 扩展程序，使用的数据库为 SQLite。

5.2 关键技术实现

5.2.1 加密方式

缓存在中间件的用户登录信息是基于******.so 库的接口进行加密的，该接口由密钥服务器提供，使用的是 128 位的 AES 加密算法。由于该接口只提供文件级别的数据加密接口，不适应用户信息这类字符串类型的数据加密，因此需要对现有的接口进行修改，本文新增了对字符串的加解密接口，具体实现如下文所示。

```
def sAES_str_encrypt(req, access_token):
    result = " "
    dll = CDLL("/usr/lib/******.so")
    user_token = '{"access_token": "#"}'
    user_token = user_token.replace("#", access_token)
    offset = 0
    lens = len(req)
    tag = True
    while tag:
        buff = create_string_buffer(1024)
        if lens < 1024:
            chunk = req[offset:]
```

```
                chunk = "%s%s" % (chunk,'\001' * (1024 - lens))
                tag = False
        else:
                chunk = req[offset:offset+1024]
                offset = offset + 1024
                lens = lens - 1024
                inputs = c_char_p(chunk)
                token = c_char_p(user_token)
                dll.sAES_encrypt(inputs,buff,token)
                result = result + buff.raw
        return result
```

该接口的是需要传入两个参数，req 表示用户请求，access_token 表示用户登录 token。通过引入/usr/lib/******.so 模块，对字符串进行加密。该模块提供的接口每次只能加密 1024 个字节的数据，因此需要对字符串的长度进行判断，如果超过了 1024 个字节，那么需要对字符串按 1024 个字节长度逐一进行加密后再整合形成最后的加密结果。数据的解密算法是加密算法的逆过程，从数据库获取到加密后的会话信息后，需要提供用户的 token 来进行解密。同理，若是请求的长度超过了 1024 个字节，那么也需要逐一进行解密，最后组合得到解密信息。加密和解密接口的使用方式如下：

```
//加密算法
enreq = mwEncrypter.sAES_str_encrypt(clientrequest, access_token)
//解密算法
dereq = mwEncrypter.sAES_str_decrypt(enreq, access_token)
```

5.2.2 中间件授权代理

在现有系统中，客户端提交任务如果长时间不活跃，则会导致大量任务的失败和重新提交，浪费了资源，用户体验也不够好。结合会话信息管理，可以进一步支持用户对中间件的授权，从而解决身份认证和授权信息过期所引发的问题。

当中间件系统将用户的请求发送到服务器时，如果服务器返回给用户的结果是用户登录信息过期，那么需要在中间件进行自动授权代理，获取新的 token 后再重新发送用户的请求。

```
//生成用户 token 认证请求
req = __VERIFY_TOKEN__.replace("#token#", token);
```

```
//判断用户请求是否过期
data = os.popen(req)
data = data.read()
if(data.find('"status": "0"')!=-1):
    return True
else:
    return False
```

上述代码实现的是用户登录信息认证，若服务器返回的状态码不为“0”，则表示用户的登录信息过期，需要通过中间件代理重新登录。

```
//生成登录请求
req = __LOGIN__.replace("#email#", userid).replace("#password#", password)
//更新用户 token
data = os.popen(req)
data = data.read()
rstjson = json.loads(data)
newToken = rstjson["access_token"]
```

中间件授权代理即通过用户的用户名和密码重新登录，登录成功后，更新系统中保存的该用户的 token。对于用户的其他请求，也需要替换其现有的过期 token。

5.2.3 大文件上传

当文件的小于超过 4M 时，执行正常的上传文件操作。当文件的大小超过 4M 时，需要对大文件进行分块上传，每个文件块的大小均为 4M，系统为每个文件块的上传任务分配一个进程。当文件块在上传的过程中遇到网络故障时，将断点信息保存在数据库中，再次上传该文件时，若存在断点，则从断点的位置续传。

```
//判断文件大小是否超过 4M
filesize = os.path.getsize(path)
if(filesize > __CHUNKSIZE__):
    result = self.chunkupload(rq, path, chunksize)
else:
    result = self.upload(rq)
//将文件分割成文件块
fileUtil.mwSplitFile(path, tempfile, chunksize)
```

```
//获取断点信息
if(breakpoint>0):
    breakpoint = breakpoint - 1
//将文件块逐一上传
for filepart in files:
    rq = rq.replace(tempfilename + '?op',filepart + '?op')
    result = self.upload(rq)
//在服务器端合并文件
for filepart in files:
    rq = rq.replace(tempfilename + '?op',filepart + '?op')
    result = self.upload(rq)
```

5.2.4 本地缓存

本地缓存的是指将客户机本地的物理内存划分出一部分空间用来备份用户频繁使用的文件，当用户反复下载同一个文件，如果每次都从远程下载该文件，则是对网络带宽资源的浪费。因此，需要高效的临时备份区域，以降低下载文件时所需要的 I/0 请求。此外， 同一个用户备份数据时会有相同的文件。在上传文件时，同一用户的相同文件是可以复用的，可以利用服务器提供的复制文件接口，直接在服务器操作，以减少上传文件所需的网络带宽，从而实现“秒传”功能。本地缓存实现的核心代码如下：

1）下载文件本地缓存实现

```
//判断是否存在本地缓存
result = self.doCache(rq, n)
//查询本地缓存数据库
cx = sqlite3.connect('/var/log/mwcache.db')
fname_result = cx.execute("select file_name from localCache where MD5 = '"
+ MD5 + "';")
//若存在本地缓存，则将缓存中文件拷贝到下载路径
shutil.copy(cache_path, destination)
```

2）上传文件实现“秒传功能”

```
//判断是否存在本地缓存
result = self.doCache(rq, n)
//查询服务器端是否存在待上传文件
```

```
cx = sqlite3.connect('/var/log/mwcache.db')
spath_result = cx.execute("select server_path from serverCache where user_id
= '" + user_id + "' and MD5 = '" + MD5 + "';")
//若服务器存在该文件，调用服务器端复制文件接口
cp_req = __COPY__.replace("#src_path#", src_path)
cp_req = cp_req.replace("#des_path#", des_path)
data = os.popen(cp_req)
```

上述代码中，使用到了两张表分别保存本地缓存数据信息和备份在服务器端的数据信息，数据库的详细设计参见 4.4 数据库设计。表 localCache 用来保存本地缓存文件信息，用文件 MD5 进行唯一标识；表 serverCache 用来保存备份在服务器端的数据信息。由于服务器端不提供去重服务，因为需要组合服务器提供的复制文件接口，以实现“秒传”功能，通过获取文件属性的接口获取服务器端文件的 MD5 值，并通过该值与 serverCache 中 MD5 值进行匹配，若存在相同的 MD5 值，则说明服务器端存在该文件。此时，调用服务器提供的复制文件接口，提升上传文件的效率。

5.2.5 LRU 替换策略

LRU 是 Least Recently Used 的缩写，即“最近最少使用”，也就是说，LRU 缓存把最近最少使用的数据移除，让给最新读取的数据。往往最常读取的，也是读取次数最多的，所以利用 LRU 缓存策略能够提保证本地缓存文件的时效性。

要实现 LRU 缓存，需要用一个数据库记录每个文件的使用频率，每当用户上传或者下载一个文件时，将会更新数据中关于该文件使用次数的字段，自动加一。当本地缓存达到存储空间的上限时，LRU 缓存策略会将使用次数最少的文件替换掉，如果同时存在两个使用次数相同的文件，那么会根据最近使用的时间来判断替换哪个文件。LRU 替换策略的核心代码如下：

```
//计算当前缓存文件的总大小
for fsize in file_size_result:
    cache_size += fsize[0]
cache_size += os.path.getsize(path)
//如果超过缓存空间的最大容量，执行 LRU 替换策略
if(cache_size > __CACHESIZE__):
    self.doLRU(rq, cx, path, MD5)
//获取使用次数最少的文件
```

```
fcache_result = cx.execute("select * from fileCache where use_time <= %d"%use_time + " order by use_time, modified_time;")
//删除使用次数最少的文件
cx.execute("delete from fileCache where MD5 = '" + fcache[3] + "';")
os.remove(sys.path[0] + "/mwcache/cache/" + fcache[1])
//将当前文件拷贝至缓存文件夹
shutil.copy(path, sys.path[0] + "/mwcache/cache/" + file_name)
```

5.2.6 Firefox 扩展

浏览器扩展应用是一种用来修改 web 浏览器功能的工具。使用 JavaScript、Html、CSS 等标准的 web 技术，再加上一些专用的 javascrip API 进行编写。火狐扩展(Extensions for Firefox)由 WebExtensions API 构建而成。 在很大程度上，与谷歌浏览器 Chrome 和欧朋浏览器 Opera 所支持的扩展 API 兼容。通过浏览器扩展可以为浏览器增加新的特性或者改变某些网站的外观和内容。本文为支持优化后的中间件系统具备多应用适配的特性，因此开发了一款基于 Firefox 的扩展应用，使得用户可以通过 web 扩展应用来使用云备份服务。

1）组织结构

WebExtension 中包含的组件如下：

- background pages：执行一个长时间运行的逻辑
- content scripts：与网页进行交互
- browser action files：在工具栏中添加按钮
- page action files：在地址栏添加按钮
- options pages：为用户定义一个可浏览的 UI 界面，可以改变曾经的设置
- web-accessible resources：是打包好的内容可用于网页与目录脚本

manifest.json 是唯一一个在每个 WebExtension 中必须存在的文件。包含了关于这个扩展应用基本的元数据。比如扩展的名字，版本和所需权限。以及扩张需要的版本信息与权限。并且，也对 WebExtension 中其他文件进行了链接。本文中 manifest.json 中的关键内容如下：

```
{
  //版本信息
  "manifest_version": 2,
  "name": "Middleware",
  "version": "1.0",
  //所需的权限信息
```

```
  "permissions": [
    "activeTab",
    "tabs",
    "cookies",
    "<all_urls>"
  ],
  //图标信息与初始界面
  "browser_action": {
    "default_icon": "icons/beasts-32.png",
    "default_title": "Middleware",
    "default_popup": "popup/middleware_login.html"
  },
}
```

上述代码描述的是 manifest.json 中一些较为关键的内容，从配置文件中可以清晰地看到扩展的名称、版本信息以及所需的权限信息。作为扩展的入口，需要在浏览器的导航栏里定义一个图标，以及一个初始界面，browser_action 参数中的 default_icon 属性指定扩展的图标，而 default_popup 指定了初始界面。

2）通信方式

WebExtension 使用的是网页开发的标准技术，客户端与服务器通过 Websocket 进行通信。Websocket 需要经过一次握手协议才能建立客户端与服务器之间的连接，其实现的核心代码如下：

Firefox 扩展客户端核心代码：

```
//设置服务器地址和端口号，并新建 WebSocket 对象
var host = "ws://127.0.0.1:****/"
socket = new WebSocket(host);
//连接建立时
socket.onopen = function (msg) {
    send(request);
};
//收到服务器返回的消息时
socket.onmessage = function (msg) {
    pyresult = pyresult + msg.data
    var JResult = JSON.parse(pyresult);
    portFromCS.postMessage({"JResult": JResult });
```

```
};
//连接关闭后
socket.onclose = function (msg) {
    portFromCS.postMessage({"mstatus": "connection closed"});
    login = false;
};
```

Python 服务器端核心代码：

```
//判断是否为来自 Firefox 的请求
if(recvdata.find("websocket")!=-1):
    clientrequest = handle_WebRequest(sock, recvdata)
//返回握手协议信息
def handle_WebRequest(sock, request):
    sock.send('\
    HTTP/1.1 101 WebSocket Protocol Hybi-10\r\n\
    Upgrade: WebSocket\r\n\
    Connection: Upgrade\r\n\
    Sec-WebSocket-Accept: %s\r\n\r\n' % token)
//处理来自 Firefox 客户端的请求
data = sock.recv(RECV_BUFFER)
```

在 Firefox 扩展客户端中，首先需要通过服务器的地址和端口号建立连接，在创建 WebSocket 对象时，会主动向服务器发送握手协议，报文由 WebSocket 自动生成。建立连接之后，WebSocket 通过三个回调函数 onopen、onmessage、onclose 分别来处理连接建立时、收到服务器返回的消息时、连接关闭后的请求和数据。

在 Python 服务器端，接收用户的请求之前需要完成握手协议。WebSocket 有标准的格式来规定返回服务器的信息，通过此报文以完成握手协议。随后，通过 recv 方法接收并处理用户的请求。

3）部署与实现

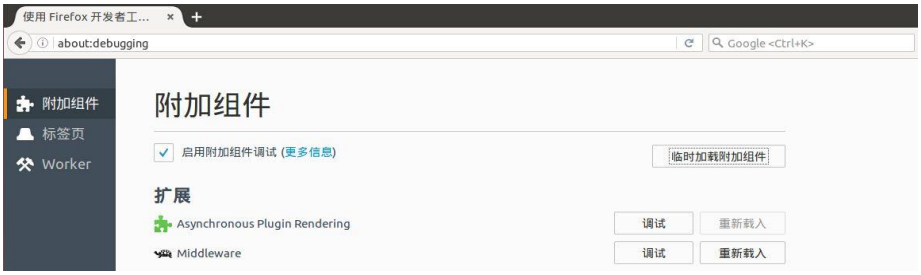


图 5-2 Firefox 扩展应用部署

在 Firefox 浏览器中输入“about:debugging” 页面，点击 ″临时加载附加组件按钮″ 并选择附加组件目录，即选择 manifest.json 文件，如图 5-3 所示。

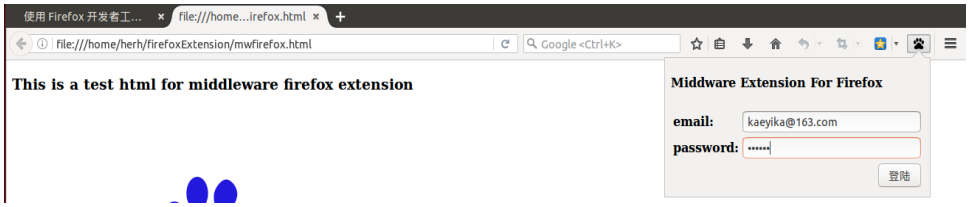


图 5-3 Firefox 扩展登录后查询备份空间效果图

图 5-2 所示为将开发好的 Firefox 扩展导入到 Firefox 浏览器中，从上图中可以看到，名为 Middleware 的扩展已经被部署到了浏览器中。5-3 所示为使用浏览器扩展登录云备份服务器的界面，也是扩展应用的初始界面。

5.2.7 Java SDK

优化后的中间件系统具备一定的可扩展性，即支持多种类型的客户端使用云备份服务。除了提供基于 Firefox 浏览器的中间件扩展应用以外，本文还开发了一款基于 Java 平台的 SDK。Java SDK 是提供给 Java 开发人员建立应用时使用的开发工具集合， 开发人员通过调用 SDK 提供的接口，便可以与云备份服务器进行交互，从而使用云备份服务。Java 语言能够适用于当前很多开发场景，如 Web 开发、Android 开发等，因此通过提供 SDK 的形式，让现有的中间件系统支持更多的应用。

1）通信方式

服务器端的代码是由 Python 语言编写的，Java 与 Python 可以直接通过 Socket 进行通信，因此本文使用了 Java 自带的 java.net.Socket 工具包。相较于 WebSocket 来说，Socket 与服务器进行连接无需通过握手协议，因此，可以在建立连接后直接向服务器发送请求。用户在使用 Java SDK 时，每调用一个接口便会开启一个线程来处理用户的请求，由于需要处理服务器的返回信息，因此我们选用了 java.util.concurrent.Callable 类作为线程类的接口。核心代码如下：

```
//通过服务器地址和端口号简历 socket 连接
socket= new Socket(SOCKET_HOST, SOCKET_PORT);
//向服务器发送请求
PrintWriter mwWriter = new PrintWriter(socket.getOutputStream());
mwWriter.write(this.request);
mwWriter.flush();
//接收来自服务器的响应
```

```
InputStream mwInputStream=mwSocket.getInputStream();
InputStreamReader isr = new InputStreamReader(mwInputStream);
BufferedReader mwBufferedReader = new BufferedReader();
String info=mwBufferedReader.readLine();
```

2）接口设计与实现

中间件 Java SDK 的设计目是使 Java 开发人员集成云备份服务器到 Java 应用中去，从而使应用能够使用云备份服务。在设计 Java SDK 的接口时，参照现有的中间件 CLI 客户端接口，确保了 Java SDK 开发工具提供的接口与其保持一致，除了现有的中间件系统提供的基本接口外，还加入了优化后实现的一些高级接口，接口的定义和实现如下所示。

```
//接口定义
public interface IMiddlewareAPI {
    //用户登录接口
     String getAccessToken(String userID, String pwd);
    ……
}
```

```
//接口实现
public class MiddlewareAPI implements IMiddlewareAPI{
   @Override
     public String getAccessToken(String userID, String pwd) {
        //构建登录请求
        String request = Constant.GET_ACCESS_TOKEN.replace("#email#",
        userID).replace("#password#", pwd);
        //发送请求
         String result = this.sendRequest(request)
        //处理请求
          JSONObject jsonObject = JSONObject.fromObject(result);
          token = (String) jsonObject.get("access_token");
          return token;
     }
}
```

在 IMiddlewareAPI 类中定义了 Java SDK 提供的所有接口，包含了 CLI 客户端所有的功能以及优化后扩展的功能。为保证与 CLI 客户端的特性一致，相同功

能的接口参数和返回值与其保持一致。接口的实现一共分为三个步骤，首先构建请求，通过引用 Constant 常量类中预定义的请求模式，替换相应的参数值；然后新建一个线程，发送用户请求；最后处理服务器返回的信息，将其转化为标准的 Json 格式。

3）部署与使用

Java SDK 的使用一共分为五个步骤：

1. 新建一个 Java 项目，并创建 lib 文件夹；
2. 将 Java SDK MiddlewareSDK.jar 拷贝纸 lib 目录下；
3. 将 lib 目录下的 MiddlewareSDK.jar 添加到 Build Path 中；
4. 实例化 MiddlewareAPI 对象

```
// MiddlewareAPI mwAPI = new MiddlewareAPI("userid@**.", "12356");
```

5. 调用 API 接口,如获取配额信息接口

```
// String result = mwAPI.getQuota();
```

、

第六章 云备份中间件优化系统测试

本章主要介绍中间件系统优化实现后，进行的系统测试。测试分为两部分内容，一个是功能测试，即保证系统分析中所提到的优化功能能够正常运行。另一个是优化后的系统性能测试，主要测试的是系统的稳定性和鲁棒性，同时与现有的中间件系统进行纵向比较，体现优化后系统的特性。此外，与同类型的产品进行横向比较，描述优化后中间件系统的特点和优势。

6.1 测试环境部署

表 6-1 所示为中间件优化测试环境参数，使用的测试机器内存大小为 10.0G RAM，磁盘大小为 750GB。测试的环境与开发环境一致，在 Ubuntu14.04 上进行，测试的中间件版本号为 0.94.

图 6-1 测试环境表

硬件环境	具体参数
计算机型号	Dell Inc. OptiPlex 990
处理器型号	Intel(R) Core(TM) i5-2500S CPU @ 2.7GHz
内存大小	10.0 GB RAM
硬盘总大小	750 GB
操作系统	Ubuntu 14.04
测试软件名称	Backup-Cloud Oriented Middleware
测试软件版本号	0.94
测试服务器	IP：192.168.7.62

6.2 测试目标

本文对中间件优化的内容为在 Linux 平台上，针对需要改善的问题，综合运用前期的技术积累，优化并完善平台框架和 API 支持机制，以支持更多的桌面 OS 组件使用云备份服务。为了确保优化的功能需求与性能需要[35]，对优化后的云备份中间件进行了测试。

测试目标如下：

- 测试优化后的中间件系统是否完成系统分析设计提出的功能要求。
- 测试系统的安全性、稳定性、鲁棒性。
- 与原有的中间件系统和国内外成熟的云备份服务产品进行对比分析。

6.3 功能测试

中间件系统的功能测试就是对各优化项进行验证，设计一套完备的测试用例，逐项测试，检查是否达到设计要求的功能。由于优化的功能复杂且测试用例较多，所以本文选取了每个优化项中具有代表性的核心功能进行分析。

6.3.1 持久会话管理功能测试

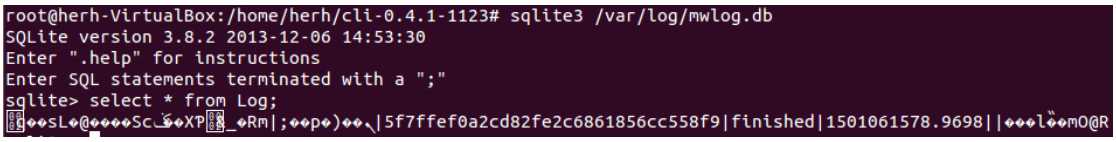
持久会话管理的主要功能是将用户的会话信息以加密的形式缓存在中间件中，当中间件系统出现异常时（如网络中断、中间件死机），会在下次重启中间件时自动恢复执行未完成的会话。此外，对会话信息进行了物理隔离，并且以加密的形式备份在数据库中，以提高用户信息的安全性，表 6-2 所示为会话备份及加密测试用例。

表 6-2 会话备份及加密测试

测试编号	Cloud-Opt-Test-001
测试内容	用户的请求是否以加密的形式备份在数据库中
测试用例	request = '-k -X POST -d \'{"password": "123456", "email":"kaeyika@163.com"}\' https://192.168.7.62:443/oauth/access_token'
测试结果	用户的请求以加密的形式备份在 SQLite 数据库中

本测试主要对持久会话管理模块中会话备份及加密进行了测试，测试用例来自用户的请求，表 6-2 所示为用户名为 kaeyika163com，密码为 123456 的用户请求登录。优化后的中间件系统首先会调用请求加密函数，对会话进行加密，然后调用数据库层的服务，将加密后的请求存入到数据库，测试结果如图 6-1 所示。

图 6-1 会话信息加密测试图



从图 6-1 中可以看出，请求已经被存在 SQLite 数据中，数据库的路径为 /va/log/mwlog.db。用户的请求是以加密的形式存储在数据库中的，这很好地验证了对会话信息进行加密备份的需求。

6.3.2 大文件上传功能测试

大文件上传利用服务器提供的高级功能，实现了大文件分块上传功能，在受到网络状况受到影响的情况下，若文件上传中断，会自动启用断点续传功能，避

免数据重传。测试中，本文分别对网络状况正常和出现异常两种情况进行测试。在正常的网络状态下，超过规定大小的文件将被分割成固定大小的文件块，然后逐一上传至服务器，等所有文件块上传完之后，调用服务器提供的合并文件接口，完成大文件在服务器端的合并。在网络异常的状态下，中间件需要记录文件上传的断点信息，然后在下次网络连接正常后，根据断点信息实现续传。表 6-3 所示为大文件分块上传测试用例。

表 6-3 大文件分块上传测试

测试编号	Cloud-Opt-Test-002
测试内容	大文件分块上传
测试用例	request = -k -X PUT -T /home/herh/test.docx "https://192.168.7.62:443/v1/AUTH_kaeyika163com/end1/bigtest.docx?op=CREATE&overwrite=&metadata=&mode=ENCRYPT" -H "X-Auth-Token:G6i7w90RCH6DoL0ZNv85rORwNBsThSF6wyouYq8s"
测试结果	大文件分块逐一上传至服务器，并在服务器端合并

本测试主要针对大文件分块上传功能，请求表示用户名为 kaeyika163com 的用户上传一个名为 test.py 的大文件至服务器，上传至服务器的路径为 /biye/test.py，测试的结果如图 6-2 所示。

```
no cache and go on......
-k -X PUT -T /home/herh/middleware0.94/mwcache/uploadtemp/temp-kaeyika163com/part1181142f80d6070e1f4fc9be04f99b9e6.py "https://192.168.7.62:443/v1/AUTH_kaeyika163com/segments/part1181142f80d6070e1f4fc9be04f99b9e6.py?op=CREATE&overwrite=true&metadata=&mode=" -H "X-Auth-Token:0wfSxNgoNc5U2b6dfAW83cWAIUTSTEXEPbnAx0Rf"
-k -X PUT -T /home/herh/middleware0.94/mwcache/uploadtemp/temp-kaeyika163com/part2181142f80d6070e1f4fc9be04f99b9e6.py "https://192.168.7.62:443/v1/AUTH_kaeyika163com/segments/part2181142f80d6070e1f4fc9be04f99b9e6.py?op=CREATE&overwrite=true&metadata=&mode=" -H "X-Auth-Token:0wfSxNgoNc5U2b6dfAW83cWAIUTSTEXEPbnAx0Rf"
-k -X PUT -T /home/herh/middleware0.94/mwcache/uploadtemp/temp-kaeyika163com/part3181142f80d6070e1f4fc9be04f99b9e6.py "https://192.168.7.62:443/v1/AUTH_kaeyika163com/segments/part3181142f80d6070e1f4fc9be04f99b9e6.py?op=CREATE&overwrite=true&metadata=&mode=" -H "X-Auth-Token:0wfSxNgoNc5U2b6dfAW83cWAIUTSTEXEPbnAx0Rf"
-k -X PUT -T /home/herh/middleware0.94/mwcache/uploadtemp/temp-kaeyika163com/part4181142f80d6070e1f4fc9be04f99b9e6.py "https://192.168.7.62:443/v1/AUTH_kaeyika163com/segments/part4181142f80d6070e1f4fc9be04f99b9e6.py?op=CREATE&overwrite=true&metadata=&mode=" -H "X-Auth-Token:0wfSxNgoNc5U2b6dfAW83cWAIUTSTEXEPbnAx0Rf"
```

图 6-2 大文件分块上传测试图

从图 6-2 中输出的日志文件可以看出，test.py 文件按用户 id 信息及时间信息被分成大小相等的 4 个文件块，最后一个文件块的大小不足 4M。每个文件块都由一个上传文件的进程单独上传到了服务器，并且最后调用了服务器端的合并文件接口。通过查询服务器端文件可以看到，文件块合并成了一个完整的文件。

6.3.3 本地缓存功能测试

本地缓存主要利用现代计算机普遍较大的存储空间以及高性能的计算能力，实现本地缓存管理，减少系统 I/O 操作，提高了中间件系统文件传输性能。同时，利用服务器提供的个人备份在云端的数据信息，实现了针对单一用户的“秒传”

功能。测试中，本文主要针对存在本地或是服务器缓存的情况下，是否实现了文件复用。表 6-4 所示为本地缓存测试用例。

表 6-4 本地缓存测试

测试编号	Cloud-Opt-Test-003
测试内容	本地缓存测试
测试用例	request = -k -L "https://192.168.7.62:443/v1/AUTH_kaeyika163com/end/test.docx?op=OPEN&offset=&length=&version=&mode=" -H "X-Auth-Token:G6i7w90RCH6DoL0ZNv85rORwNBsThSF6wyouYq8s" -o /home/herh/end.docx request = -k -L "https://192.168.7.62:443/v1/AUTH_kaeyika163com/end/test.docx?op=OPEN&offset=&length=&version=&mode=" -H "X-Auth-Token:G6i7w90RCH6DoL0ZNv85rORwNBsThSF6wyouYq8s" -o /home/herh/mend.docx
测试结果	第一次下载文件时正常下传，第二次从本地复制

本测试针对本地缓存设计，测试用例由两个相同的下载文件请求构成。该请求表示用户 kaeyika163com 下载服务器端的/end/test.docx 文件至本地的/home/herh/end.docx，测试的结果如图 6-3 所示。

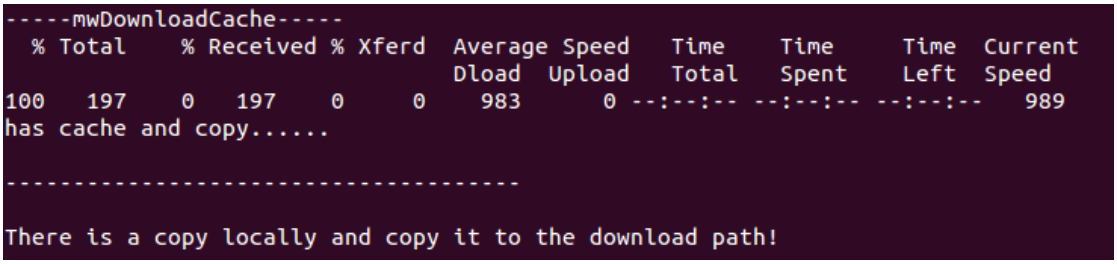


图 6-3 本地缓存测试图

从图 6-3 中输出的日志文件可以看出，第一次下载文件时，正常调用了下载文件接口，并将文件保存到了下载路径。再次下载该文件时，并没有调用下载文件接口，而是直接从本地缓存中将该文件拷贝到了下载路径，下载所需的时间极短，大大地提升了文件下载的效率。表 6-5 所示为服务器去重测试用例。

表 6-5 服务器去重测试

测试编号	Cloud-Opt-Test-004
测试内容	本地缓存测试
测试用例	request = -k -X PUT -T /home/herh/test.docx "https://192.168.7.62:443/v1/AUTH_kaeyika163com/e

	nd/test.docx?op=CREATE&overwrite=&metadata=&mode=" -H "X-Auth-Token:G6i7w90RCH6DoL0ZNv85rORwNBsThSF6wyouYq8s" request = -k -X PUT -T /home/herh/test.docx "https://192.168.7.62:443/v1/AUTH_kaeyika163com/end1/test.docx?op=CREATE&overwrite=&metadata=&mode=" -H "X-Auth-Token:G6i7w90RCH6DoL0ZNv85rORwNBsThSF6wyouYq8s"
测试结果	第一次上传文件时正常上传，第二次从服务器复制

本测试针对服务器端的去重测试，测试用例由两个相同的上传文件请求构成。该请求表示用户 kaeyika163com 上传本地文件/home/herh/test.py 至服务器端/biye/test.py，测试结果如图 6-4 所示。

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   114  100   114    0     0    910      0 --:--:-- --:--:-- --:--:--   919
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    60  100    60    0     0    299      0 --:--:-- --:--:-- --:--:--   303
has cache and copy......

--------------------------------------

{"status": "0", "msg": "txe5abf1f629cc457da111ea028a75272c"}
```

图 6-4 服务器端去重测试图

从图 6-4 中输出的日志文件可以看出，第一次上传文件时，正常调用了上传文件接口，将文件上传至了服务器端的指定路径。再次上传该文件时，并没有调用上传文件接口，通过服务器端去重校验，发现服务器存在该文件，所以调用的是复制文件的接口。该接口的执行时间很短，实现了“秒传”功能。

6.3.4 多应用适配功能测试

多应用适配为实现中间件系统对不同客户端请求的适配，支持更多的客户端类型，实现系统的跨平台性。本文开发了一款基于 Firefox 浏览器的扩展工具，用户可以利用扩展使用云备份服务，并提供了 Java 平台的软件开发包（SDK），Java 开发者可以利用该开发包提供的应用接口 API 来集成本文优化的云备份服务。测试中，本文对通过 Firefox 扩展和 Java 开发包用户的功能测试，验证优化后的中间件系统能够适配各种类型的客户端。表 6-6 为多应用适配测试用例。

表 6-6 多应用适配测试

测试编号	Cloud-Opt-Test-005

测试内容	本地缓存测试

测试用例	//Firefox 登录测试
	firefox_request = '-k -X POST -d \'{"password": "123456", "email":"kaeyika@163.com"}\' https://192.168.7.62:443/oauth/access_token'
	//Java SDK 登录测试
	MiddlewareAPI middlewareAPI = new MiddlewareAPI("kaeyika@163.com", "123456");

测试结果	Firefox 浏览器与 Java 开发包登录成功

本测试同时测试了 Firefox 扩展以及 Java SDK 的登录功能。Firefox 通过 WebSocket 向服务器发送请求，Firefox 扩展的请求由“firefox_”标识。Java SDK 通过 Socket 向服务器发送请求，并有“java_”标识。通过身份认证之后，便可使用云备份平台提供的接口，测试结果如图 6-5 以及 6-6 所示。

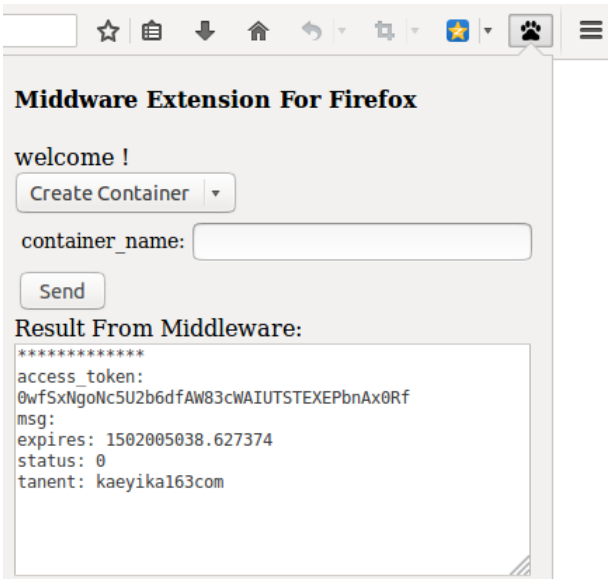


图 6-5 Firefox 扩展登录测试图

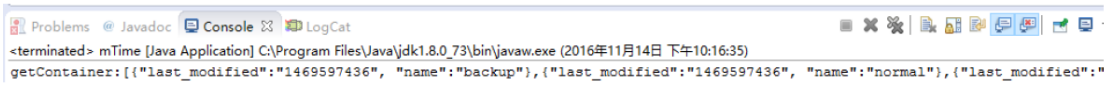


图 6-6 Java SDK 测试图

图 6-5 所示的是 Firefox 扩展登陆功能，用户点击扩展应用图标后，会弹出登陆界面，输入正确的登录信息后会将服务器返回的登陆成功信息展示在扩展界面上。图 6-6 所示的是 Java SDK 的测试结果，开发人员将 Middleware.jar 导入到工程中后，调用登陆接口创建中间件接口实例对象，然后调用获取备份空间接口，返回的是 Json 格式的数据[36]。Firefox 扩展以及 Java SDK 实现了与现有的 CLI 相同的功能，且具有一定稳定性、安全性和鲁棒性。

6.4 性能分析

6.4.1 安全性分析

1）会话信息安全性

为确保中间件系统的安全性，现有的中间件系统在文件传输过程中，采用了128 位的 SSL 加密技术，在文件的存储形式上，让用户主动选择是否加密，使用的是 128 位 AES 加密算法。然而现有中间件系统是以明文的形式保存用户的登录信息、会话信息等敏感数据的。这很有可能造成用户隐私的泄露。

优化后的中间件系统对安全性进行了更为全面的考虑。为了保证缓存在数据库中用户会话信息不被窃取，对会话信息进行了加密。具体实现如下：

```
//对用户的请求进行加密处理
enreq = mwEncrypter.sAES_str_encrypt(clientrequest, uuid)
//构建插入数据的 SQL 语句
sqlString = "insert into Log values(?,?,?,?,?,?);"
//调用 Dao 的 save 方法，将加密后的会话信息存入数据库
dao.save('/var/log/******.db', sqlString, (buffer(enreq), uuid, "unfinished", t, userid, ""))
```

2）本地缓存安全性

优化后的中间件系统新增了本地缓存的功能，旨在解决同一用户多次从服务器下载同一文件而导致的网络带宽浪费的问题。在现有的计算机系统中，存在一台终端被多人使用的情况，因此需要对不同用户的缓存数据进行物理隔离，防止某一用户的缓存信息被其他用户窃取[37]。具体实现如下：

```
//根据用户名创建缓存文件夹名称
cache_path = sys.path[0] + "/mwcache/cache" + "/" + userid
if not os.path.exists(cache_path):
//文件夹不存在时创建
    os.makedirs(cache_path)
//修改文件权限，S_IRWXU 代表该文件所有者具有可读、可写及可执行的权限
    os.chmod(cache_path, stat.S_IRWXU)
```

3）数据库安全性

现有的中间件系统在执行数据库操作时，采取的方式是建立临时的数据库连

接，并编写相应的 SQL 语句，最后执行数据库操作。

```
//建立数据库连接
sql_connection = sqlite3.connect('/var/log/mwlog.db')
//执行数据库操作
result = sql_conn.execute("select request from Log where state = 'unfinished'
and userid='" + userid + "' and time<'" + t + "';")
```

这样的操作存在一定的安全隐患，如 SQL 注入攻击[38]。SQL 注入攻击是黑客对数据库进行攻击的常用手段之一，程序员在编写代码的时候，没有对用户输入数据的合法性进行判断，使应用程序存在安全隐患。用户可以提交一段数据库查询代码，根据程序返回的结果，获得某些他想得知的数据，这就是所谓的 SQL Injection，即 SQL 注入。优化后的中间件系统对数据库的操作进行的全面优化，不仅分离了数据库层操作，更是改变了执行数据库操作的方式。优化后的中间件系统使用了占位符的方式来抵御 SQL 注入攻击。

```
//使用占位符的方式构建 SQL 语句
sqlString = "insert into Log values(?,?,?,?,?,?);"
//调用 Dao 的 save 方法，执行 SQL 语句
dao.save('/var/log/mwlog.db', sqlString, (buffer(enreq), uuid, "unfinished", t,
userid, ""))
```

6.4.2 稳定性分析

1）系统稳定性

中间件系统可以同时处理来自不同用户的请求，同时用户也能批量地发送请求给中间件。因此用户的请求不会立马被执行，而是以缓存的形式保存的。当用户注销或是中间件系统遇到故障之后，优化后的中间件系统会在中间件系统重启后，自动执行未完成的任务。保证用户提供的请求不会丢失。此外，新增了异常处理机制，当用户执行错误的操作时，会抛出相应的异常，而不是强制退出。

2）传输稳定性

网络环境受到很多因素的制约，云备份服务是以网络作为基础的，尤其是上传和下载操作。优化后的中间件系统充分地考虑到了网络的不稳定因素，当出现网络故障时，会记录断点信息，再次上传文件时，会根据断点信息进行续传，这大大地提升了数据传输的稳定性。

6.5 对比分析

中间件优化的主要目的是基于前期的中间件系统，同时研究云储存服务中的关键技术，对其功能和性能两方面进行优化。本文将从纵向和横向两个方面，对优化后的中间件系统与现有的中间件系统和市场上同类产品进行比较分析。

6.5.1 原中间系统对比分析

表 6-7 与原中间件系统对比分析表

	传输加密	存储加密	缓存加密	防注入攻击
原中间件系统	√	√		
优化后中间件系统	√	√	√	√

安全性上，表 6-7 展示了原有的中间件系统与优化后的中间件系统在安全性上的对比结果。原有的中间件系统对数据在传输过程中和备份在服务器时两种状态进行了加密，具有一定的安全性。优化后的中间件系统在此基础上做了进一步的改进，对保存在中间件系统中的用户信息进行了加密。此外，对用户备份在本地的缓存文件通过权限设置进行了物理隔离，确保了缓存数据的安全性。同时，规范了数据库层的操作。

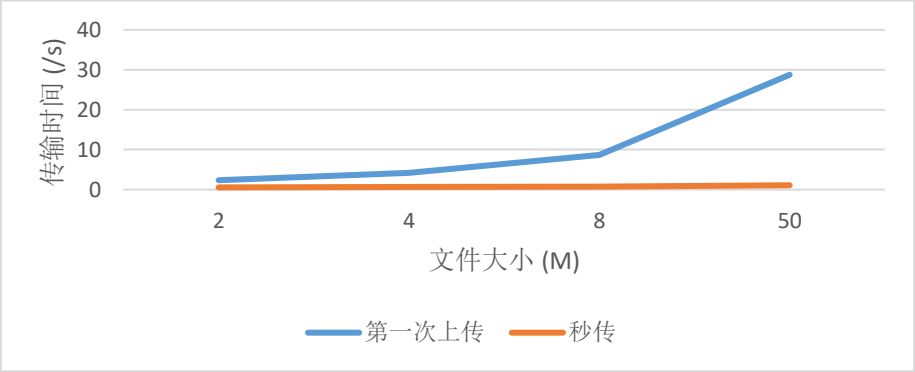


图 6-7 “秒传”功能传输速率图

在稳定性上，现有的中间件系统只做了简单的异常处理。优化后的中间件系统对网络状况、代码的稳定性、异常处理三个方面进行了改进，能够在遇到网络异常后断点续传，图 6-7 展示了“秒传”功能的传输效率，此外中间件系统出现异常后用户请求不会丢失，并提了供更多的异常处理机制。

表 6-7 与原中间件系统功能模块对比分析表

	功能模块
原中间件系统	数据传输模块、数据加密模块、数据压缩模块
优化后中间件系统	数据传输模块、数据加密模块、数据压缩模块、持久会话管理

	模块、大文件上传模块、本地缓存模块、多应用适配模块

在功能上，现有的中间件系统主要提供基础的数据传输功能，一共分为三个模块，分别为数据传输模块、数据加密模块、数据压缩模块。优化后的中间件系统新增了持久会话管理、大文件上传、本地缓存以及多应用适配四个模块，使当前的中间件系统的功能特性更加完备。

6.5.2 同类产品对比分析

中间件系统关键技术的优化主要基于对同类产品的调研，本文对国内外的一些云备份服务平台进行了详细的分析，从而得出了现有的中间件系统需要优化的内容。经过设计与开发，现有的中间件系统在一些重要的功能上已经达到了成熟产品的要求，此外，一些新的功能特性具备一定的优势。

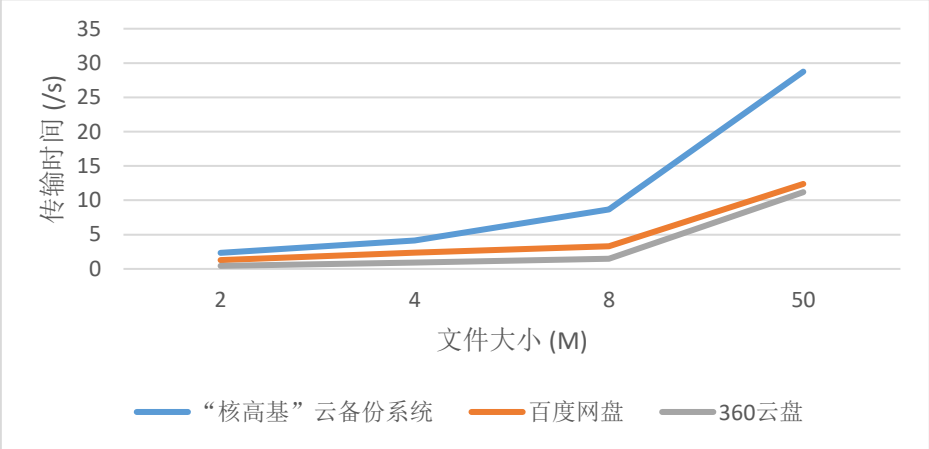


图 6-8 云备份服务系统与同类产品文件传输效率对比图

在数据传输效率上，由于传输效率是由网络状况和服务器处理文件的性能决定的，而中间件只负责服务器与客户端之间的信息交互，因此无法改变原有系统的文件传输效率。通过测试，相较于其他成熟的云产品，云备份中间件系统在数据传输效率上存在一定的劣势，本文在测试中分别采用了 2M、4M、8M、50M 四个大小的文件对云备份服务平台、百度网盘、360 云盘的传输效率进行了测试，由于只能通过网络代理的方式访问 Dropbox 和 Google Drive，而代理的网络状况较差，测试出的数据不具备参考性，因此没有展示其测试结果。图 6-8 所示的是云备份服务与同类产品文件传输效率对比图。

在系统的安全性上，优化后的系统对缓存在中间件的用户登录信息、会话信息进行了加密，使用的是与其他同类产品类似的加密方式，采取了 128 位的 AES 加密算法，相较于 258 位的 AES 加密算法，128 位的加密算法能够节省大约 40% 的加密时间，同时能够保证数据的安全性，因为它足够复杂无法被暴力破解。

表 6-8 同类产品去重分析表

	云备份服务器	百度网盘	360 云盘	Dropbox	Google Drive
服务器去重	√	√	√	√	
本地缓存	√				

在数据去重上，如表 6-8 所示，百度网盘和 360 云盘都采取了全局去重的方式，Dropbox 使用了单一用户去重，而 Google Drive 未进行系统去重处理。优化后的中间件借鉴了 Dropbox 的去重方式，系统只支持单一用户的去重，也就是说即使服务器中的其他用户存在当前用户待上传的文件，也不会将其他用户的文件拷贝至当前用户的上传路径下。这样能避免由服务器端全局去重而导致的侧信道攻击文集，保证了系统在一定程度上能够缓解服务器备份压力，又确保了系统的安全性。此外，调研的结果显示，目前百度网盘、360 云盘已经 Dropbox 和 Google Drive 都不支持本地缓存。也就是说当用户反复下载同一文件时，需要多次从服务器端下载同一文件。优化后的中间件系统在客户端本地建立了一个缓存文件，记录文件的下载信息和数据，以解决多次重复下载同一文件的问题。

优化后的中间件系统支持大文件的断点续传功能，当遇到网络故障的时候，会在网络恢复正常后，根据断点信息续传，以上云备份服务平台都支持此项功能。

表 6-9 同类产品支持客户端类型表

	Windows	iOS	Linux	Android	Web	Java SDK	Extension
云备份服务器			√			√	√
百度网盘	√	√	√	√	√	√	
360 云盘	√	√	√	√	√	√	
Dropbox	√	√	√	√	√	√	
Google Drive	√	√	√	√	√	√	

在平台的可扩展性上如表 6-9 所示，其他同类产品大多支持主流的操作系统和平台，如 Windows、iOS、Linux、Android、Web 等。为适应当前平台多元化的格局，本文开发了两款云备份客户端，分别是 Java SDK 和 Firefox 扩展工具。Java SDK 可用于与 Java 相关的应用，如 Web 开发和 Android 开发。Firefox 扩展工具相较于其他客户端来说，更加轻量级，访问更加便利，且支持将浏览器中的文件实时上传到服务器中。

第七章 总结和展望

7.1 总结

云备份服务在近几年来发展十分迅速，越来越受人们的青睐，而云备份服务的安全性、稳定性、可扩展性等问题日益突显，对云备份服务的关键技术进行优化改进显得尤为重要。本文针对一款自主研发的云备份中间件系统，对其关键技术进行了优化。通过对原有中间件系统全面的分析，结合对国内外成熟的云备份服务产品充分的调研，设计并实现了对中间件系统关键技术的优化，解决了以下四个问题：

- 持久会话管理通过 128 位的 AES 加密技术，解决了缓存在中间件的用户信息如登陆信息、会话信息的安全性问题，同时新增了撤销、查询会话的功能。
- 大文件上传利用断点续传技术缓解了云备份数据传输服务受网络波动的影响，避免了因网络故障而导致的数据重传问题，保证了在网络状况不佳的情况下，数据传输的稳定性。
- 本地缓存利用本地计算机的存储能力和服务器提供的高级接口，解决对同一文件重复上传/下载所导致的网络带宽浪费问题，实现了针对同一文件重复上传/下载的“秒传”功能。
- 多应用适配通过开发两款客户端，并对来自不同客户端请求的适配，解决了现有中间件系统缺乏可扩展性的问题。Java SDK 可用于 Java 相关的应用开发，如 Web、Android 等。Firefox 扩展应用是其他同类产品不具有的客户端类型，支持用户在浏览网页时，实时地将各种资源上传到云备份服务器。

测试表明，优化后的中间件系统具有更好的安全性、稳定性、鲁棒性以及具备一定可扩展性。用户的数据在传输过程中、缓存在中间件时以及存储在服务器不会被窃取和泄露；同时，遇到网络状况异常时，通过断点续传保证数据传输的稳定性；此外，通过对源码的优化和重构，减少了代码自身的漏洞，提高了系统自身的鲁棒性；Java SDK 和基于 Firefox 的扩展应用两款客户端提升了系统的跨平台性，同时通过请求适配使中间件系统具备一定的可扩展性，支持更多的客户端类型。

7.2 展望

本文针对现有的云备份中间件系统关键技术的优化，总体符合了当前云备份

行业发展的趋势，与同类成熟的云备份服务产品在功能上基本保持了一致性。然而，仍存在一些挑战，需要在后续的工作中不断的完善。

- 该款云备份服务平台尚没有投入商用，只适用于内部小范围的使用。与其他同类产品相比，没有经历过市场和用户的检验，而测试受硬件、人员条件的限制，虽然对基本功能进行了较为完备的测试，然而在压力测试、边界测试上具有一定的局限性，因此需要在后续的工作中通过搜集用户使用时的数据以及反馈信息，对中间件系统进行持续优化，提高系统的稳定性。
- 优化的后的中间件系统虽然具备了一定的可扩展性和跨平台性，并提供了CLI、Java SDK 和 Firefox 扩展三种客户端，分别支持在 Linux 平台下、移动端和 Web 端使用中间件，但是相较于其他同类产品，支持的客户端类型较少，在未来的工作中，可以考虑实现 Android 和 iOS 两大当今主流的客户端类型，以支持更多的用户使用。
- 本文只对云备份服务中部分关键技术进行了优化，随着云备份技术不断的发展，以及人们日益增长的需求，云备份服务平台的高级功能越来越多，如在线预览、视频播放、自动备份等，这要求云备份中间件系统做持续的更新和优化。在未来的工作中，我们会持续调研云备份服务关键技术的发展趋势，使这款自主研发的云备份服务系统能最大程度地满足用户的需求。

参考文献

[1] 运营商世界网. 2016 年中国云服务及云备份市场分析报告[R]. 2017. http://www.qianjia.com/html/2017-03/15_267452.html.

[2] Naresh vurukonda, B. Thirumala Rao. A Study on Data Storage Security Issues in Cloud Computing[J]. Procedia Computer Science, 2016, 92.

[3] 刘建毅,王枞,薛向东. 云备份安全分析[J]. 中兴通讯技术, 2012, 18(06):30-33.

[4] Jian-zong Wang, Peter Varman, Chang-sheng Xie. Optimizing storage performance in public cloud platforms[J]. Journal of Zhejiang University SCIENCE C, 2011, 12 (12):951-964.

[5] 刘若冰. 面向大数据云存储系统的关键技术研究[J]. 现代电子技术, 2016, 39(06): 21-24.

[6] Zhong Lin He, Yu Hua He, Li Yu Chen. A Study on the Key Issues of Cloud Storage Technology[J]. Applied Mechanics and Materials, 2010, 979(29).

[7] 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.

[8] Anonymous. Research and Markets; Implementing SSL/TLS Using Cryptography and PKI - Guide to Implementing Protocols for Internet Security [J]. Computers, Networks & Communications, 2011.

[9] 刘艳萍, 李秋慧. AES 算法的研究与其密钥扩展算法改进[J]. 现代电子技术, 2016, 39(10): 5-8.

[10] 刘召富. 面向网络应用的大文件传输服务的设计与实现[D]. 北京邮电大学, 2014.

[11] 郑营营. 基于 HTTP/FTP 协议的断点续传多线程下载组件[D]. 济南大学, 2012.

[12] 鲁孙林. 基于 HTTP 协议的断点续传[J]. 济南职业学院学报, 2006, (01): 43-46.

[13] K. Akhila, Amal Ganesh, C. Sunitha. A Study on Deduplication Techniques over Encrypted Data[J]. Procedia Computer Science, 2016.

[14] Xuan Li, Jin Li, Faliang Huang. A secure cloud storage system supporting privacy-preserving fuzzy deduplication. Soft Computing, 2016, 20 (4): 1437-1448.

[15] 胡渝苹. 文件秒传系统在云存储环境下的设计与实现[J]. 计算机应用与软件, 2016, 33(04): 329-333.

[16] Lei Wang, Yan Yan Yu, Qian Huang, Jun Yang, Zheng Peng Zhao. The Design and Implementation of the MD5 Algorithm Based on FPGA[J]. Applied Mechanics and Materials, 2013, 2748(427).

[17] Fei Liu, Lanfang Ren, Hongtao Bai. Mitigating Cross-VM Side Channel Attack on Multiple Tenants Cloud Platform [J]. Journal of Computers, 2014, 9(4).

[18] Darren Quick, Kim-Kwang Raymond Choo. Dropbox analysis: Data remnants on user machines[J]. Digital Investigation, 2013.

[19] 卜瑞琪. 面向云备份的高效中间件设计与实现[D]. 复旦大学, 2015.

[20] 王济意. 基于物理隔离技术的安全信息交换系统[D]. 西安电子科技大学, 2015.

[21] 朱思怡. 计算机云备份技术的特征初探[J]. 电脑迷, 2017, (03): 96.

[22] Amirhossein Farahzadi, Pooyan Shams, Javad Rezazadeh, Reza Farahbakhsh. Middleware technologies for cloud of things-a survey[J]. Digital Communications and Networks, 2017.

[23] 张广斌, 宫金林, 陈爽. SQLite 嵌入式数据库系统的研究与实现[J]. 单片机与嵌入式系统应用, 2008, (06): 11-13.

[24] 朱清华. 基于 Android 手机 SQLite 的取证系统设计实现[D].大连理工大学, 2015.

[25] Wen Tao Liu. Research on the Development of WebSocket Server[J]. Advanced Materials Research, 2014, 2986(886).

[26] 杜明远. 基于 WebSocket 的即时通信系统设计与实现[D]. 大连理工大学, 2016.

[27] 陆晨, 冯向阳, 苏厚勤. HTML5 WebSocket 握手协议的研究与实现[J]. 计算机应用与软件, 2015, 32(01): 128-131.

[28] Anonymous. Research and Markets: Beginning HTML, XHTML, CSS, and JavaScript: Essential Update to the Key Web Authoring Standards[J]. M2 Presswire, 2010.

[29] Yi Huang, Xin Qiang Ma, Dan Ning Li, Rong Wu. Research and Applications of Access Control Based on Logic SQL Database System[J]. Advanced Materials Research, 2011, 1043(143).

[30] 谢杰涛, 吴敏, 吴娟, 史睿冰. Web 系统高性能本地数据缓存实现机制[J]. 计算机应用研究, 2014, 31(07): 2074-2077.

[31] 韩菲. 设计模式和重构的研究与应用[D]. 北京邮电大学, 2006.

[32] Sargon Hasso, Carl Robert Carlson. Software Composition Using Behavioral Models of Design Patterns[J]. Journal of Software Engineering and Applications, 2014, 07(02).

[33] Oksana Nikiforova, Janis Sejans, Antons Cernickins. Role of UML Class Diagram in Object-Oriented Software Development[J]. Scientific Journal of Riga Technical University. Computer Sciences, 2011, 44(01).

[34] Jian Zhang, Peihua Gu, Qingjin Peng, S. Jack Hu. Open interface design for product personalization[J]. CIRP Annals - Manufacturing Technology, 2017, 66(1).

[35] 李阿妮, 张晓, 张伯阳, 柳春懿, 赵晓南. 公有云存储系统性能评测方法研究[J]. 计算机应用, 2017, 37(05): 1229-1235.

[36] 张沪寅, 屈乾松, 胡瑞芸. 基于 JSON 的数据交换模型[J]. 计算机工程与设计, 2015, 36(12):3380-3384.

[37] 李伟. Linux 系统中文件权限管理及应用[J]. 无线互联科技, 2014, (04):70.

[38] Harish Dehariya, Piyush Kumar Shukla, Manish Ahirwar. A Survey on Detection and

Prevention Techniques for SQL Injection Attacks[J]. International Journal of Wireless and Microwave Technologies(IJWMT), 2016, 6(6).

在读期间发表论文

[1] 梁蛟，刘武，韩伟力，王晓阳，甘似禹，沈烁. 安卓云备份模块的代码安全问题分析[A]. 网络与信息安全学报, 2017, 3(1).

致　谢

能够顺利完成毕业论文，首先想要感谢的是我的研究生导师韩伟力老师。从论文的选题到项目的实施，以及最后毕业论文的撰写，韩老师都给予了我最大的支持和帮助。在项目实施的过程中，韩老师悉心地对我进行了指导，每当我遇到问题时，都会为我指明方向，提供技术支持。在论文的撰写过程中，韩老师严谨的学术作风，一丝不苟的工作态度一直敦促着我不断修正、完善论文。除了在学习上对我的指导以外，韩老师在生活上也给予了我细致的关怀，在这里，我想再次向韩老师表示由衷的感谢。

其次，我还要感谢实验室的小伙伴们，在两年半的学习生活中，你们给了我太多的支持和感动。生活中我们一起谈天说地，学术上我们一起共同探讨，特别是在我撰写论文期间，无私地给予我帮助，让我感受到了实验室的友爱。能够与你们同处一个实验室，是一件很幸运的事。

此外，我想要感谢复旦大学以及学校的各位老师，给了我一个很好的平台，让我有机会参与一些重要的项目和会议，这对我的影响十分深远。此外，我还能够与其他优秀的同学一起学习，一起进步。

最后，感谢各位评阅老师对本文给出的宝贵意见。

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：刘武　　日期：2017.07.02

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：刘武　导师签名：韩伟力　日期：2017.07.02