

# 云备份机制（二期）（三）

## 中间件优化

刘武

2016/11/23



# 目录

1

## 中间件简介

简单地介绍云备份中间件的背景、意义以及一期任务的完成情况和实现的功能。

2

## 中间件优化

根据云备份机制二期任务书，分析和介绍中间件优化的内容以及实施方案。

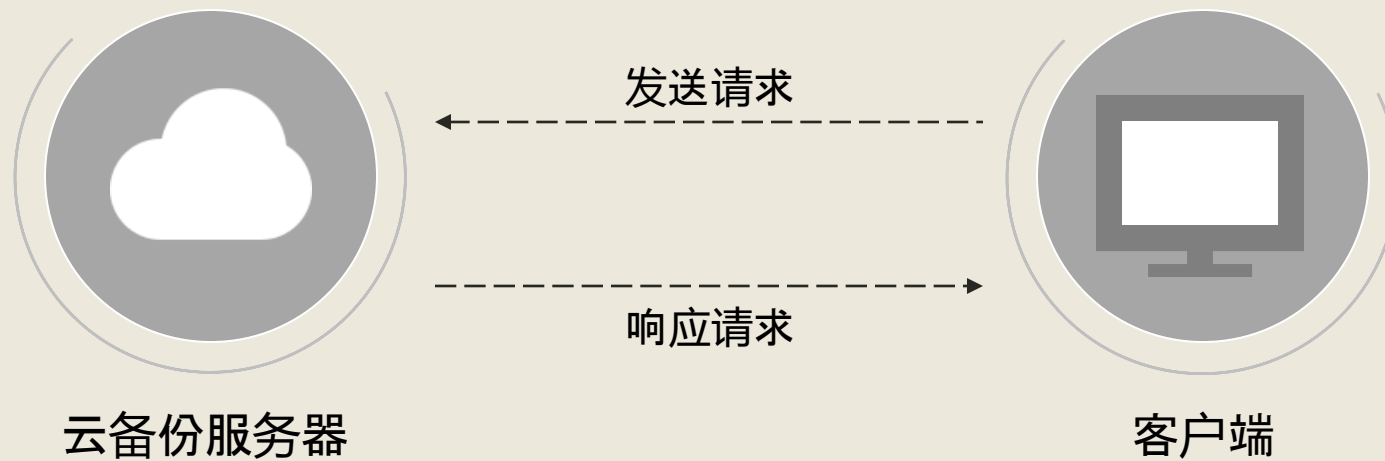
3

## 优化完成情况

介绍中间件优化的完成情况，包括优化的进度、方法以及后续工作的介绍。

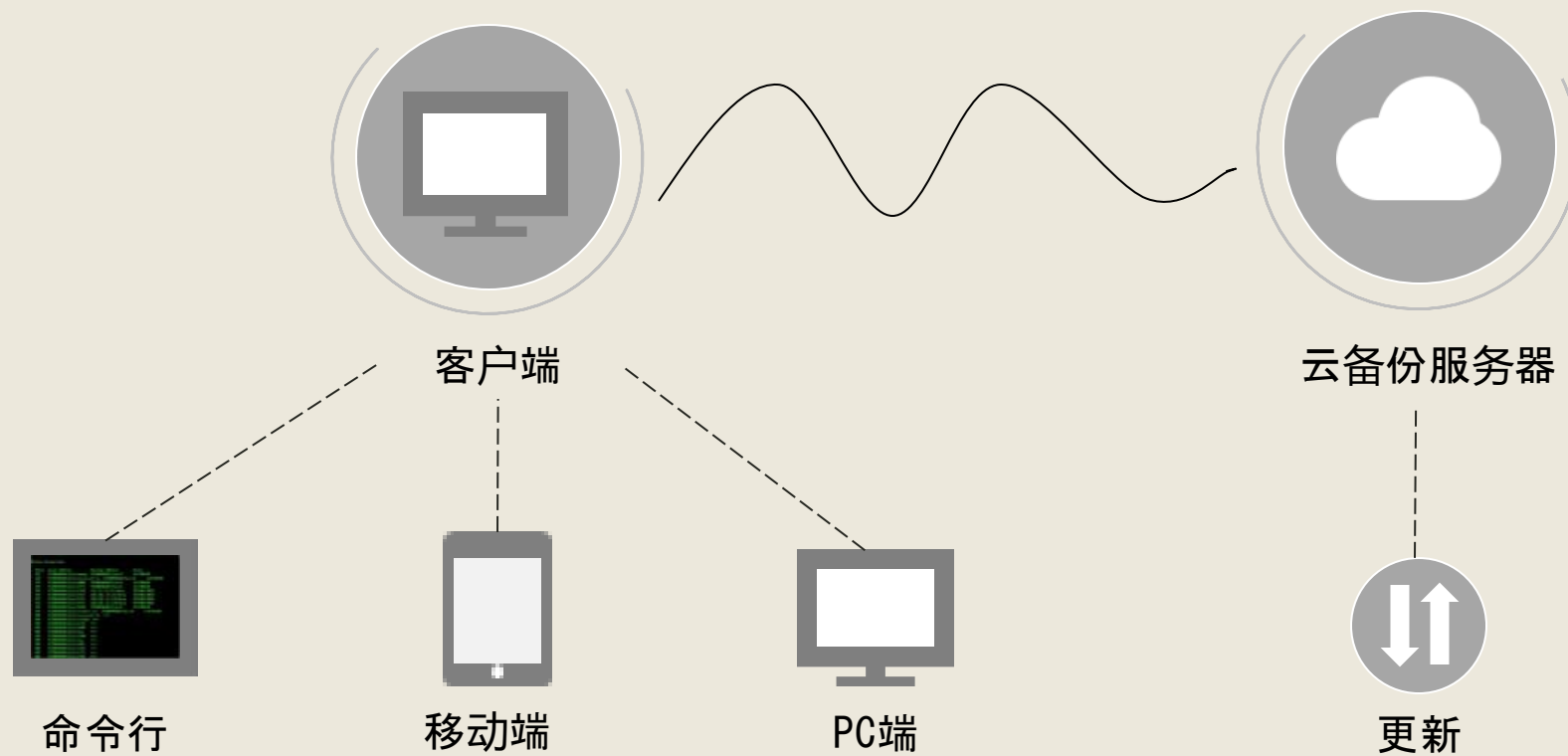


## 中间件简介





## 中间件简介

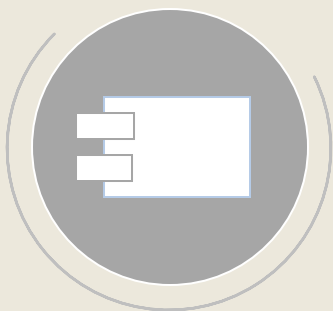




## 中间件简介



云备份服务器



中间件



客户端



## 中间件简介

### 01 数据传输

负责与云存储服务器交互，实现数据的上传与下载。

### 02 数据加密

对用户上传备份的数据进行加密存储，以防数据被非法窃取，泄露用户隐私。

### 03 数据压缩

对用户的备份数据进行压缩，减少数据量，提高云存储的存储效率。

### 04 安全会话管理

以可靠的方式获取用户令牌，通过系统机制支持用户令牌从产生到废止整个生命周期内的安全管理。



## 中间件优化



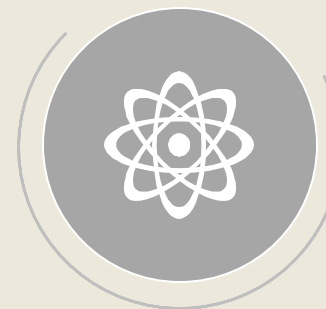
多应用适配



会话持久化管理



本地缓存管理



系统优化



## 中间件优化



多应用适配

问题：客户端的多样性，存在多种编程语言和操作系统，此外客户端和服务端可能会定义不同的接口，导致接口不匹配。

目标：适配不同的命令接口，以支撑更多桌面OS组件使用云备份系统。



会话持久化管理

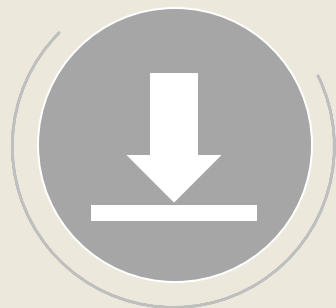
问题：中间件服务器异常或崩溃会导致会话数据丢失。此外，会话以明文形式存储，存在用户信息泄露的风险。

目标：服务器出现异常，在机器重启之后自动执行未完成任务；机密存储用户信息，并隔离不同用户的数据。





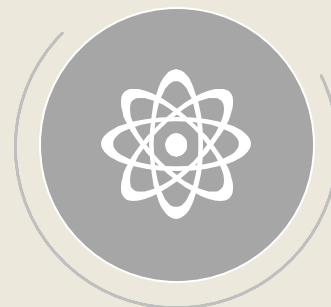
## 中间件优化



本地缓存管理

问题：用户可能反复访问同一文件，如果每次都从远程服务器获取，对时间和资源都是一种浪费。

目标：利用本地存储和计算能力，缓存上传和下载中频繁访问的文件，以减少客户端的I/O操作。



系统优化

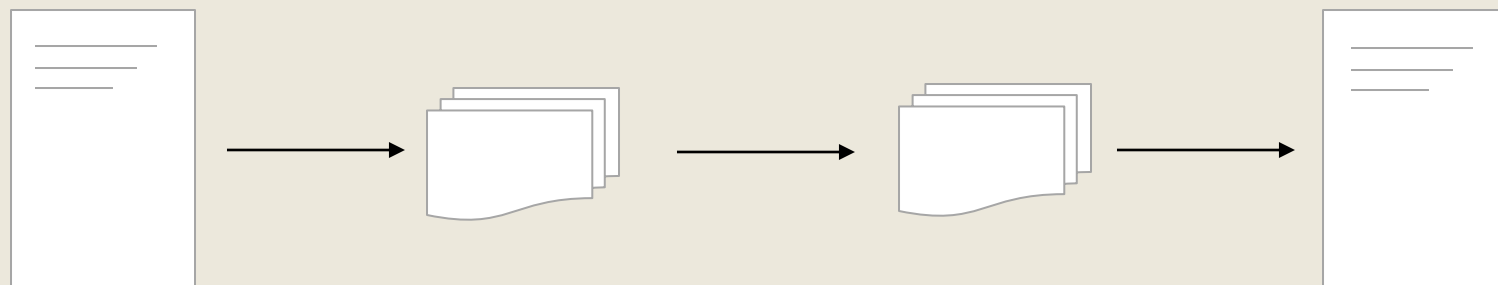
问题：中间件可以利用服务端的一些高级功能以及现代计算机的高性能提供更好的用户体验。

目标：利用服务器支持分块传输的功能传输大文件；利用计算机的高性能实现并发I/O。



## 优化完成情况

|系统优化>大文件传输

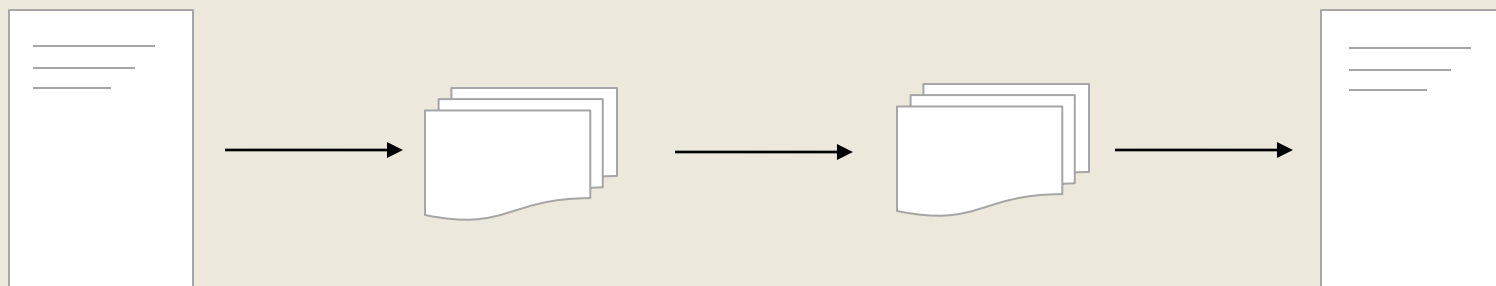


```
filesize = os.path.getsize(path)
chunksize = __CHUNKSIZE__
if(filesize > chunksize):
    result = self.chunkupload(rq, path, chunksize)
else:
    rq = rq.replace(path, '' + path + '')
    result = self.upload(rq)
```



## 优化完成情况

|系统优化>大文件传输

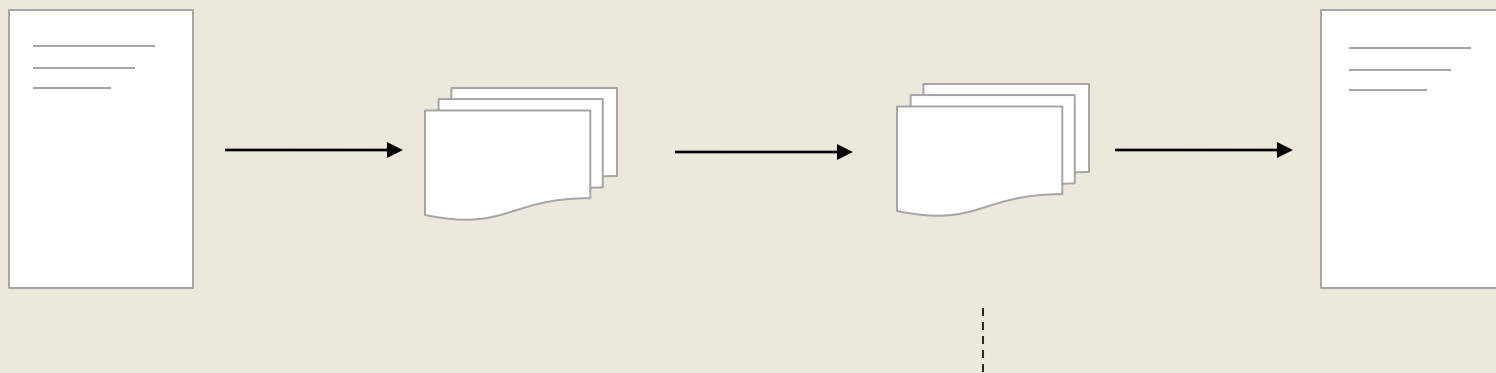


```
def chunkupload(self, rq, path, chunksize):  
    tempfile = "/home/herh/lw/uploadtemp/temp-" + userid  
    if not os.path.exists(tempfile):  
        os.mkdir(tempfile)  
    else:  
        for fname in os.listdir(tempfile):  
            os.remove(os.path.join(tempfile, fname))  
    fileUtil.mwSplitFile(path, tempfile, chunksize)
```



## 优化完成情况

|系统优化>大文件传输

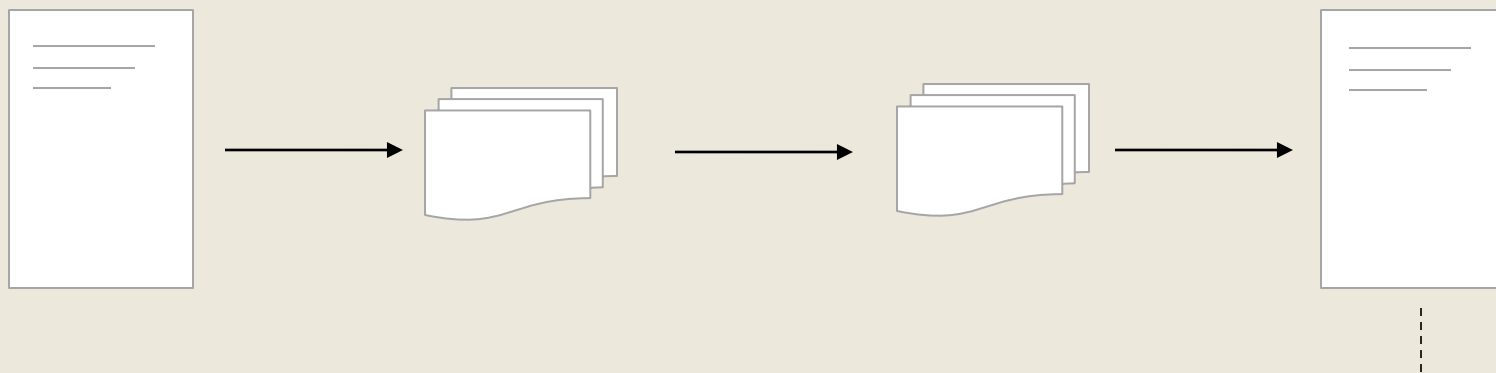


```
for filepart in files:
    newpath = os.path.join(tempfile, filepart)
    rq = rq.replace(path,newpath)
    rq = rq.replace(path.split("/")[-1] + "?op", filepart + "?op")
    result = self.upload(rq)
    temp_path = newpath
```



## 优化完成情况

|系统优化>大文件传输

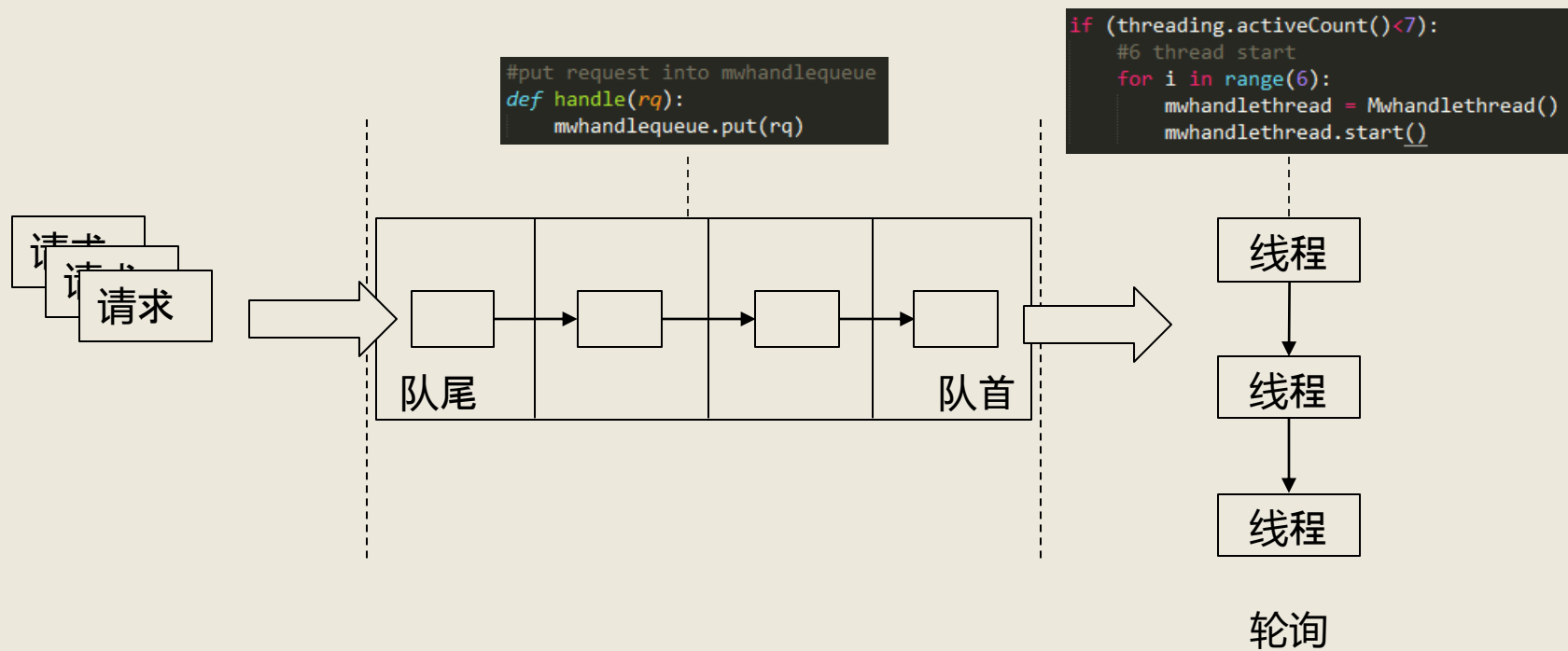


```
for fpath in chunk_file_path:
    chunk_file_attr.append(fileUtil.mwGetFileAttr(fpath, userid, token))
for fattr in chunk_file_attr:
    file_attr += fattr + ","
file_attr = file_attr[:-1]
join_req = __JOIN__.replace("#file_attr#", file_attr)
data = os.popen(join_req)
data = data.read()
```



## 优化完成情况

|系统优化>并发I/O





## 优化完成情况

|缓存管理

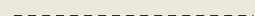


```
if(n.find('UPLOAD')!=-1 or n.find('DOWNLOAD')!=-1):  
    result = self.doCache(rq, n)
```



## 优化完成情况

|缓存管理



```
cx.execute('''create table if not exists fileCache(  
    user_id varchar[100] not null,  
    file_name varchar[100] not null,  
    length bigint not null,  
    MD5 varchar[100] not null,  
    server_path varchar[100] not null,  
    use_times int not null,  
    modified_time varchar[100] not null);''')
```

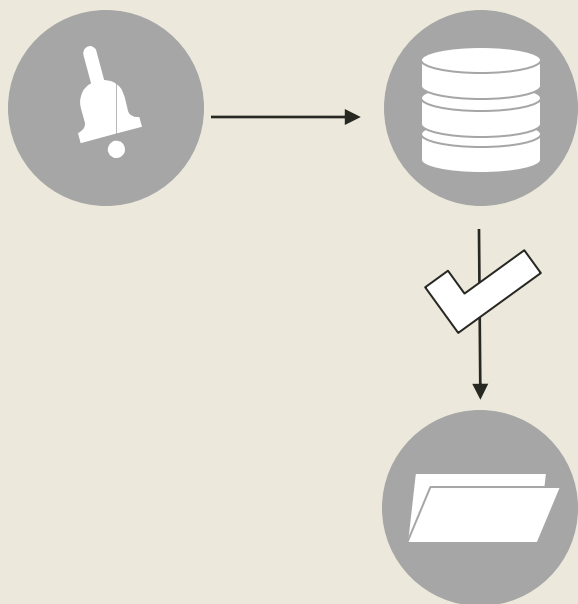
```
cx = sqlite3.connect('/var/log/mwcache.db',isolation_level=None)  
if(n.find('UPLOAD')!=-1):  
    result = fileUtil.mwUploadCache(rq)  
elif(n.find('DOWNLOAD')!=-1):  
    result = fileUtil.mwDownloadCache(rq)
```





## 优化完成情况

|缓存管理



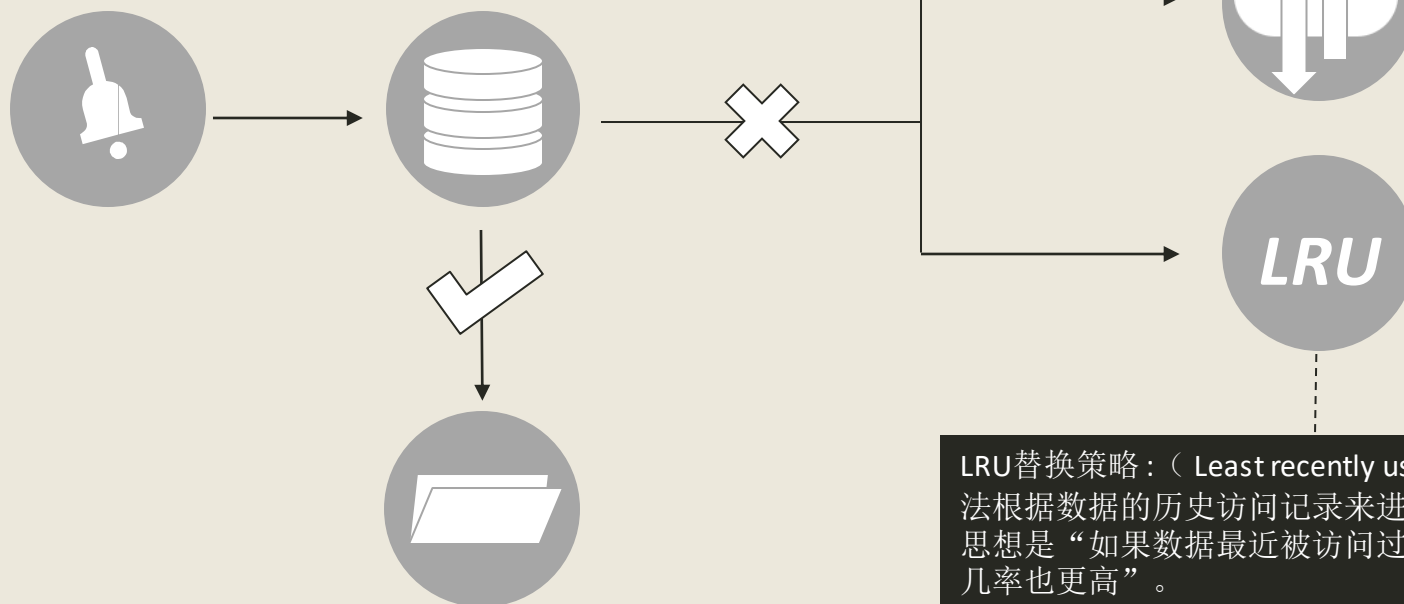
```
cp_req = __COPY__.replace("#userid#", userid)
cp_req = cp_req.replace("#src_path#", src_path)
cp_req = cp_req.replace("#token#", token)
cp_req = cp_req.replace("#des_path#", des_path)
data = os.popen(cp_req)
data = data.read()
```

```
cache_path = __CACHEPATH__ + "/" + fname[0]
des_start = rq.find('-o')
destination = rq[des_start+3:]
shutil.copy(cache_path, destination)
```



## 优化完成情况

| 缓存管理



```
result = self.doCache(rq, n)
if(result):
    result = 'mwresult = ' + result
    return result
else:
    pass
```

LRU替换策略：（Least recently used 最近最少使用）算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。



## 优化完成情况

|会话持久化管理



会话存储



会话恢复



会话加密



会话查询



会话撤销



## 优化完成情况

|会话持久化管理

request	uuid	state	time	userid	result
-k -X POST...	685cac89dd...	unfinished	1479113445	kaeyika163...	{json}
-k -X GET...	63d75b56d...	not return	1479113480	zhu__feng...	{json}
-k -X PUT...	db8e3ed4c...	finished	1479113497	kaeyia163...	{json}

会话管理数据库



# 优化完成情况

## |会话持久化管理



会话存储

```
try:
    sql_connection.execute("insert into Log values('"+clientrequest.replace("'", "#", 2) + "', '" + uuid + "', 'unfinished', '" + t + "', '" + userid + "', '');"
except:
    time.sleep(0.05)
    continue
```



会话恢复

```
recovery = RecoveryThread()
recovery.start()
```

```
sql_conn = sqlite3.connect('/var/log/mwlog.db', isolation_level=None)
result = sql_conn.execute("select request from Log where state = 'unfinished' and userid='" + userid + "' and time<' " + t + "';")
for row in result:
    data = row[0]
    handle(data)
result=sql_conn.execute("select result,uuid from Log where state = 'not return' and userid='" + userid + "' and time<' " + t + "';")
for row in result:
    data = row[0]
    sock.sendall(data)
```



会话加密

```
#encrypt client request before saving
encrypt_request = mwEncrypter.sAES_str_encrypt(clientrequest, getToken())
```



## 优化完成情况

### |会话持久化管理



会话查询

```
connect=sqlite3.connect('/var/log/mwlog.db',isolation_level=None)
rows = mcursor.execute("select * from Log where userid = '" + userid + "' order by time desc limit %d;"%(recent + 1))
for data in rows:
    mresult = data[5]
    decode_result = mwEncrypter.sAES_str_decrypt(mresult, getToken())
```



会话撤销

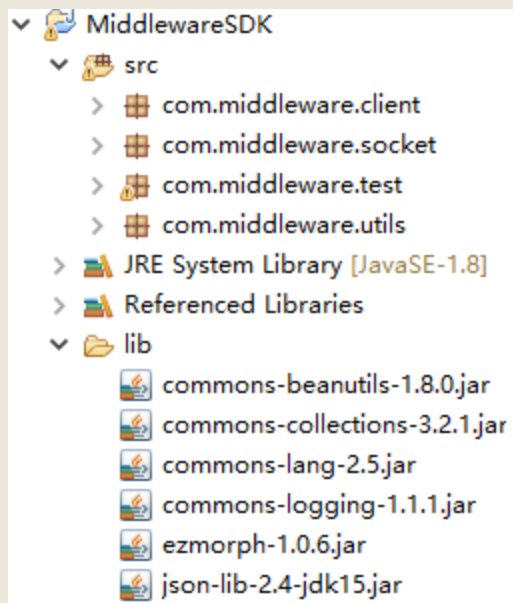
```
connect=sqlite3.connect('/var/log/mwlog.db',isolation_level=None)
mcursor = connect.cursor()
userid = userUtil.mwGetUserIdByReq(rq)[:userUtil.mwGetUserIdByReq(rq).find('?')]
mcursor.execute("delete from Log where userid = '" + userid + "' and uuid = '" + withdraw_uuid + "';")
```



## 优化完成情况

### | 多应用适配> java开发包

Java开发包提供给java开发者使用，开发者可以通过调用开发包提供的接口与中间件进行交互，从而使用云备份服务器提供的服务。其作用类似于各大云存储平台提供给开发者使用的SDK。



- com.middleware.client: 定义接口以及接口的实现。
- com.middleware.socket: socket类与中间件服务器进行通信。
- com.middleware.test: 测试接口。
- com.middleware.utils: 工具类提供通用的方法以及定义常量。

外部引用的包，用于解析和封装json数据。



# 优化完成情况

## |多应用适配>java开发包

### 1. 初始化接口对象

```
MiddlewareAPI middlewareAPI = new MiddlewareAPI("kaeyika@163.com", "123456");  
  
public class MiddlewareAPI implements IMiddlewareAPI{  
    private String userid;  
    private String access_token;  
    private ExecutorService exec;
```

ExecutorService：异步执行的机制，并且可以让任务在后台执行。一个ExecutorService 实例特别像一个线程池，事实上，在 java.util.concurrent 包中的 ExecutorService 的实现就是一个线程池的实现。

### 2. 调用接口

```
result = middlewareAPI.getContainer();  
  
private String sendRequest(String request){  
    String result = "";  
    MiddlewareSocket middlewareSocket = new MiddlewareSocket(request);  
    Future<String> mwFuture= exec.submit(middlewareSocket);  
    try {  
        result = mwFuture.get();  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }
```

发送请求时，首先创建一个socket对象，然后提交到线程池，最后根据线程池的返回对象Future获取结果。





## 优化完成情况

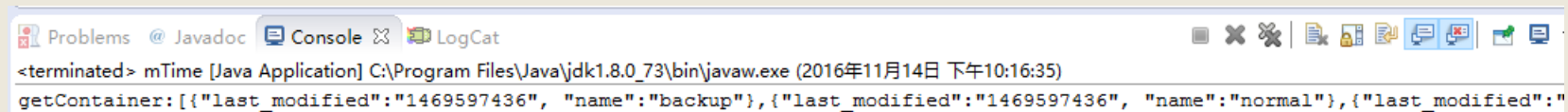
### | 多应用适配> java开发包

#### 3. 处理返回结果

```
jsonObject = JSONObject.fromObject(result);  
System.out.println("getContainer:" + jsonObject.get("result"));
```

测试中，结在果控制台显示。

#### 4. 运行结果



The screenshot shows a Java IDE console window with the following content:

```
<terminated> mTime [Java Application] C:\Program Files\Java\jdk1.8.0_73\bin\javaw.exe (2016年11月14日 下午10:16:35)  
getContainer:[{"last_modified":"1469597436", "name":"backup"}, {"last_modified":"1469597436", "name":"normal"}, {"last_modified":"
```



## 优化完成情况

### | 多应用适配>firefox插件

中间件firefox插件提供给firefox用户使用，用户可以通过插件实现与中间件的通信，能够实时获取用户在云备份服务器上的信息，如容器列表、配额等。此外，还能将页面中的图片、文本、音频等文件通过插件同步到云备份服务器中。

#### 01 WebExtensions

WebExtensions是跨浏览器开发的附加组件，与 Google Chrome 和 Opera 所支持的扩展 API 在很大程度兼容，大多数情况下为这些浏览器编写的扩展只需少许修改即可在 Firefox、Microsoft Edge 中运行。这种 API 与 多进程 Firefox 完全兼容。

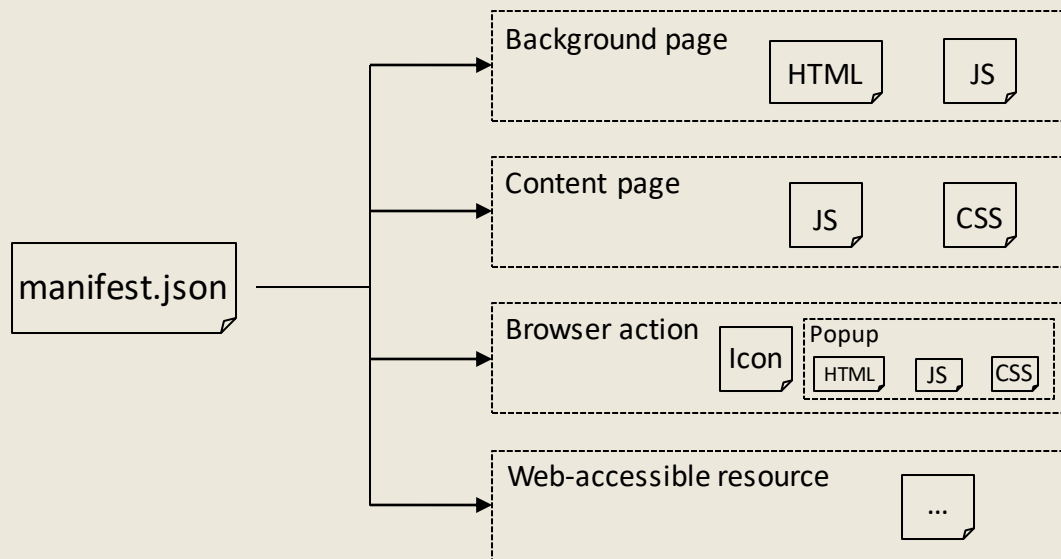
#### 02 WebSocket

WebSocket protocol 是HTML5一种新的协议。它实现了浏览器与服务器全双工通信(full-duplex)。一开始的握手需要借助HTTP请求完成。



## 优化完成情况

| 多应用适配 > firefox 插件 > Webextensions



后台脚本，独立于任何页面，在插件加载的时候运行以及插件失效或卸载时停止运行。

用来访问和操作页面的脚本。相当于在特点页面插入Js脚本。

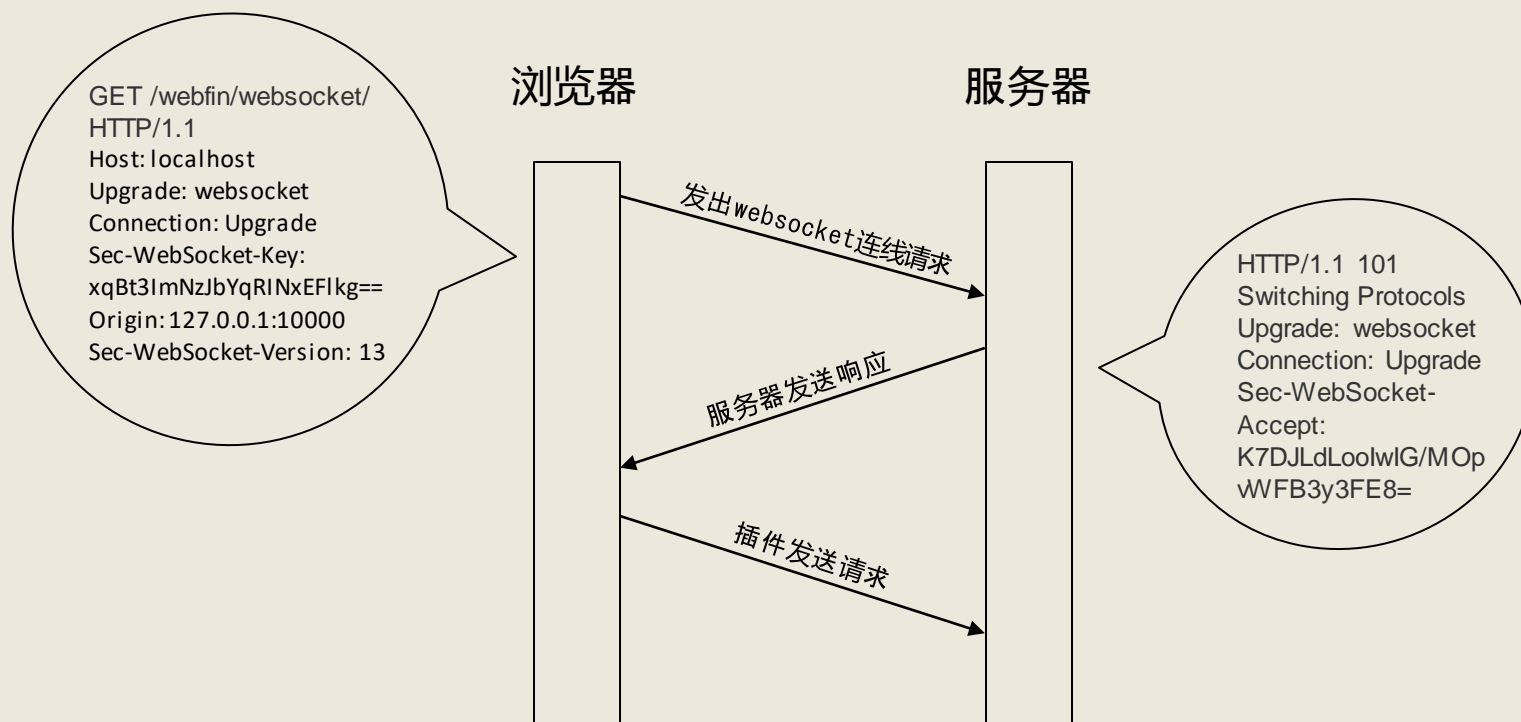
工具栏里的按钮，用户点击可与插件进行交互。

提供给content script和web pages使用的外部资源。



## 优化完成情况

| 多应用适配 > firefox 插件 > WebSocket





## 优化完成情况

| 多应用适配 > firefox 插件 > WebSocket

```
var host = "ws://127.0.0.1:10000/"
socket = new WebSocket(host);
try {

    socket.onopen = function (msg) {
        //do login while connect to the server at the first time
        doLogin(message.email, message.pwd);
    };

    socket.onmessage = function (msg) {
        var JResult = JSON.parse(msg.data);
        portFromCS.postMessage("JResult", JResult);
    };

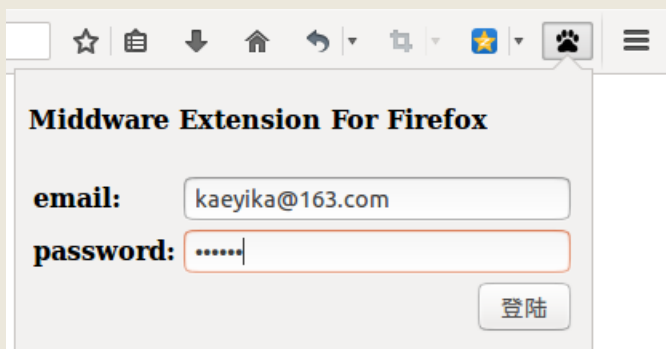
    socket.onclose = function (msg) {
        portFromCS.postMessage("onclose", "WebSocket closed");
    };
}
```

```
connection, address = sock.accept()
try:
    data = connection.recv(1024)
    headers = parse_headers(data)
    token = generate_token(headers['Sec-WebSocket-Key'])
    connection.send('\
        HTTP/1.1 101 WebSocket\r\n\
        Upgrade: WebSocket\r\n\
        Connection: Upgrade\r\n\
        Sec-WebSocket-Accept: %s\r\n\r\n' % token)
    thread = websocket_thread(connection)
```

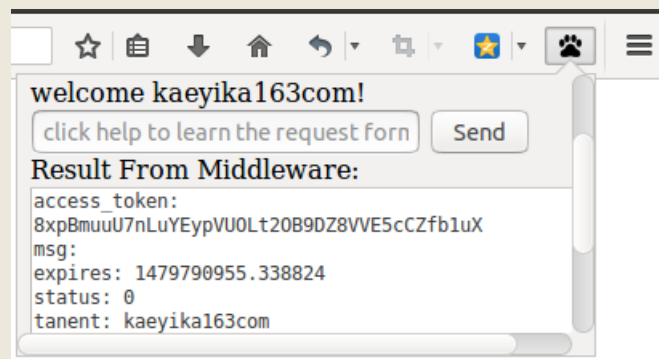


## 优化完成情况

| 多应用适配>firefox插件



登录



服务器返回结果

谢 谢

2016/11/23