功能扩展文档

编号	功能	功能描述	完成时间
1	功能扩展1	Server/Client保存所有收到的消息到文件、每天进行文档归档	2016.04.27
2	功能扩展2	消息组播、每周进行文件归档	2016.05.03
3	功能扩展3	输出文件控制、归档文件加密	2016.05.11
4	功能扩展4	维护同组成员、有序接收遗漏消息	2016.05.17
5	功能扩展5	统一日志功能、分级别、可动态配置，日志文件归档	2016.05.22
6	功能扩展6	分拆Server	2016.05.31

功能扩展1

讨论基于组件的开发

Server/Client保存所有收到的消息到文件

- 文件格式不限
- 文件路径可配置

解决：

- 在原有的PM构件基础上，增加 `writeFile(String filename,String content)` 接口，实现向文件中写入收到的消息功能
- Server中，日志全都由 `LoggerHandler` 这个handler来处理，所以只要在收到聊天消息的条件下，将该消息写入文件即可，将server端收到的消息保存在./messageRecords/server.txt中，格式如下： `[yyyy-MM-dd HH:mm:ss] 用户帐号 消息内容` ```java if(messageType == MessageType.CHATTING){ receivedMessageNumber.record() //记录开始 Date now = new Date(); SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//可以方便地修改日期格式 String snow = dateFormat.format(now); Log.writeFile("./messageRecords/server.txt","["+ snow+ "]" + " "+message.getChatContent().getAccount()+": "+message.getChatContent().toString()); //记录结束 if (messageStatus == MessageStatus.NEEDHANDLED) forwardMessageNumber.record(); else ignoredMessageNumber.record(); }

```
- Client中，日志全都由`ClientLoggerHandler`这个handler来处理，所以只要在确认消息类型为收到其他用户消息的条件下，将该消息记录下来即可。记录在messageRecords目录下以用户账户号为文件名的`.txt`文件中，格式如下：
`[yyyy-MM-dd HH:mm:ss] 用户帐号 消息内容`

```java
if(type == ACKType.OTHERSMESSAGE){
            receiveMsgNumber.record();
            //记录开始
            Date now = new Date();
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//可以方便地修改日期格式
            String snow = dateFormat.format(now);
            Log.writeFile("./messageRecords/"+account+".txt","["+ snow+ "]" + " " + ack.getChatContent().getAccount()+": " + ack.getChatCon
tent().toString());
            //记录结束
        }
```

每天所有的输出文件(Server/Client/PM)归档成一个压缩包

- 压缩格式不限
- 压缩包路径可配置

在Server和Client端收到消息这个部分，我们扩展了自己的Log模块部分，使它能够定时地完成压缩任务。

```
Log.setCompressPath("message_records")
//每天凌晨压缩一次
Log.startCompressSchedule()
//可以随时结束压缩计划
Log.endCompressSchedule()
```

在PM部分，我们则依然使用了第五组的PM模块，很方便地完成了pm记录文件的压缩。
```

```
LogReporter realtimeReporter = new RollingFileReporter("present/server-log-%d{yyyy-MM-dd_HH-mm}.log");
LogReporter archiveReporter = new RollingFileReporter("archive/server-log-%d{yyyy-MM-dd}.zip");

PerformanceMonitor monitor = new PerformanceMonitor(realtimeReporter, archiveReporter);
```

功能扩展2

1. 同组成员配置
2. 将每周的7个归档文件，重新生成一个压缩包

同组成员配置

配置同组成员并只向同组成员广播消息

解决

- 在Server端的AuthorityHandler（也即登陆验证模块）中，验证登陆成功的同时记录下该用户所属组号并将其保存在登陆消息中
- 在Server端的ChannelManagerHandler（也即用户管理模块）中，将登陆成功的channel和所对应用户的groupId同时进行维护
- 在Server端的Responser中，每次进行转发时无需连接数据库，只需要得到该channel所对应的groupId，再遍历Manager.clientChannels对同组用户进行转发即可

之前考虑的方案是，将用户的account和channel同时进行维护，而这样当每次进行消息转发时都需要连接数据库，进行两次查询，分别为 `getGroupIdByAccount(String account)` 和 `getAccountsByGroupId(int groupId)` ，这样会频繁地进行数据库连接，降低效率，因此采用将用户的groupId和channel同时进行维护的方法

功能扩展3

1. 输出文件控制
    - 时间
    - 文件大小控制
    - 总文件大小限制
    - 参数可配置
2. 归档文件加密

解决

识别并实现两个新的可复用构件，发布地址：

- COMPRESS https://github.com/bookish-component/TZCompressor

TZCompressor

Introduction

同意管理压缩任务的构件

Installation

`maven`

```
<dependency>
    <groupId>tongji.sse</groupId>
    <artifactId>compress</artifactId>
    <version>1.0.0</version>
</dependency>
```

Usage

- 初始化 `java TZCompressor tzCompressor = new TZCompressor();`
- 定义一个压缩任务,为压缩任务指定目标路径和资源目录

```
CompressTask task = new CompressTask("archive/qq","logs"))
```

- 为压缩任务设置参数

```
task.setDelay(0);
task.setInterval(60*1000);
```

- 执行压缩任务

```
task.start();
```

- 也可以把task交给TZCompressor管理

```
tzCompressor.addTask(task,"qq");
// 也可以添加多个
tzCompressor.addTask(anotherTask,"other")
              .addTask(...)
              .addTask(...);
```

- 执行压缩任务

```
//指定执行
tzCompressor.startTask("qq");
//全部执行
tzCompressor.startAllTask();
```

- 停止压缩任务

```
//指定执行
tzCompressor.stopTask("qq");
//全部执行
tzCompressor.stopAllTask();
```

- CIPHER https://github.com/bookish-component/CIPHER

CIPHER

Introduction

文件加密构件

- 可以对文件进行加密解密
- 使用DES算法加密

Installation

`maven`

```
<dependency>
    <groupId>sse.tongji.bookish-meme</groupId>
    <artifactId>cipher</artifactId>
    <version>1.0.0</version>
</dependency>
```

Usage

有两种使用方法 - 第一种： - 初始化,确定密钥 `java FileEncryptAndDecrypt fileEncryptAndDecrypt = new FileEncryptAndDecrypt("keyString");` - 加密 `java fileEncryptAndDecrypt.encrypt("./messageRecords/100.log","./messageRecords/100_加密.log");` - 解密 `java fileEncryptAndDecrypt.decrypt("./messageRecords/100_加密.log","./messageRecords/100_解密.log");` - 第二种： - 初始化 `java FileEncryptAndDecrypt fileEncryptAndDecrypt = new FileEncryptAndDecrypt();` - 生成密匙 `java Key key = fileEncryptAndDecrypt.generateKey("abc");` - 用密匙变量加密 `java fileEncryptAndDecrypt.encrypt("./messageRecords/100.log","./messageRecords/100_加密.log",key);` - 用密匙变量解密 `java fileEncryptAndDecrypt.decrypt("./messageRecords/100_加密.log","./messageRecords/100_解密.log",key);`

功能扩展4

1. Client维护其他已经Login的同组成员列表
2. Client Login后，有序接收所有遗漏的消息

用户登录时，能够查看到所有与他同组的在线的成员 其他同组成员上下线时，别的用户能够得到提示

用户登陆时，能够看到他所在的组的之前发送的消息

解决

1. Client维护其他已经Login的同组成员列表
    - 成功登录
        - 每次一个account成功登录后，Server向所有同组其他在线成员发送ACKTYPE.SOMEONEONLINE确认消息
          `java for(ClientChannel clientChannel:Manager.clientChannels) { if(clientChannel.getGroupId()==groupId&&clientChannel.getChanne`
            - 同时界面更新
              `java @Subscribe public void SomeOneOnline(SomeOneOnlineEvent someOneOnlineEvent) { String account=someOneOnlineEvent.getAc`
        - 且该登陆成功的用户得到同组所有在线用户 `java ack.setAccounts(sameGroupOnlineAccounts);`
        - 同时界面更新
          `java List<ChatContent> chatContents = loginSuccessEvent.getChatContents(); List<String> onlineAccounts =loginSuccessEvent.getO`
    - 用户下线
        - 用户下线时，Server向同组其他所有在线account发送ACKTYPE.SOMEONEOFFLINE确认消息

`java for (ClientChannel clientChannel : Manager.clientChannels){ if(clientChannel.getGroupId()==groupid){ ACK ack=new ACK();`

- 客户端界面更新

`java @Subscribe public void SomeOneOffline(SomeOneOfflineEvent someOneOfflineEvent) { String account=someOneOfflineEvent.getA`

2. Client Login后，有序接收所有遗漏的消息

- Server维护一个全局的静态Map －
`public static Map<Integer,Map<String,Integer>> groupClientsMissingNum = new HashMap<Integer, Map<String, Integer>>();`，该Map的key表示groupId，而value又是一个Map，其中这个map的key为accountId,value为遗漏的消息个数。
- 服务器启动时，初始化该全局的静态Map－groupClientsMissingNum，所有account的遗漏消息个数都为0 `ChatServer.java` ``` java try { Map uidAndGids = ServiceProvider.getDbServer().getGidAndUid(); for (String accountId:uidAndGids.keySet()){ int groupId = uidAndGids.get(accountId); if(!Manager.groupClientsMissingNum.containsKey(groupId)){

```
        Map<String,Integer> missingIndex = new HashMap<String, Integer>(){
            {
                put(accountId,0);
            }

        };
        Manager.groupClientsMissingNum.put(groupId,missingIndex);
    }else{
        Manager.groupClientsMissingNum.get(groupId).put(accountId,0);
    }
}
```

} catch (Exception e) { e.printStackTrace(); } ```

- 每次一个account 成功发送消息 时，将该消息存储到数据库中，同时将该组所有未在线account所对应的遗漏消息个数＋1 `LoggerHandler.java` ```java if(messageType == MessageType.CHATTING){ receivedMessageNumber.record(); //消息存储 ServiceProvider.getMessageStoreServer().store(message); if (messageStatus == MessageStatus.NEEDHANDLED) {

```
    forwardMessageNumber.record();

} else ignoredMessageNumber.record();
```

- 每次account 成功登陆 时，得到其遗漏消息数n，从数据库中取出该组倒数n个消息，返回给该account，将遗漏消息数清零，同时得到该account所在group中所有account的遗漏消息的最大个数maxValue，将数据库中所存该组的消息中前面count()-maxValue删去。 `Responser.java` ```java //同组在线account Set onlineClientsInSameGroup = new HashSet(); //转发给同组在线的其他成员 for (ClientChannel clientChannel : Manager.clientChannels){ if(clientChannel.getGroupId()==groupId && clientChannel.getChannel()!=incomingChannel){ onlineClientsInSameGroup.add(clientChannel.getAccount()); ACK toOthersACK = new ACK(); toOthersACK.setType(ACKType.OTHERSMESSAGE); toOthersACK.setChatContent(message.getChatContent()); String otherACKJson = gson.toJson(toOthersACK); clientChannel.getChannel().write(otherACKJson + "\n"); } }

```
//该组所有在线clients的account,包括发送方
onlineClientsInSameGroup.add(message.getChatContent().getSender());

Set<String> clientKeysSet = (Manager.groupClientsMissingNum.get(groupId)).keySet();

//该组所有clients的account
List<String> clientKeys = new ArrayList<String>();
clientKeys.addAll(clientKeysSet);
for(String clientKey : clientKeys){
    System.out.println(clientKey);

    //该组不在线成员遗漏消息数+1
    if(!onlineClientsInSameGroup.contains(clientKey)){
        int num = Manager.groupClientsMissingNum.get(groupId).get(clientKey);
        Manager.groupClientsMissingNum.get(groupId).remove(clientKey);
        Manager.groupClientsMissingNum.get(groupId).put(clientKey, num + 1);
        System.out.println(Manager.groupClientsMissingNum);
    }
}
```

- 由于数据库存储是有序的，所以可以保持消息的有序性。

功能扩展5

统一日志功能、分级别、可动态配置，日志文件归档

解决

日志分级为： - INFO - WARN - ERROR - FATAL

功能扩展6

分拆Server

- 鉴权 AuthorityServer
- 消息接收、消息转发 ChatServer
- 消息存储 MessageStoreServer
- 数据库Server DBServer（由于采用的是文件存储，即SQLite本来是无需服务器的，但是因为有两个Server，即ChatServer和MessageStoreServer需要用到该数据库所以将其拆分为一个单独的Server）

解决

```
服务器之间主要通过java RMI（远程方法调用） 来进行交互
```

Service Provider

```
工厂模式 + 单例模式
```

使用工厂方法，来为每个服务创建一个单例，其中各服务的主机地址和端口号在 `config/conf.json` 中配置

```
{

  "auth_server":{
    "host":"localhost",
    "port":2015
  },

  "store_server":{
    "host":"localhost",
    "port":2016
  }

  "db_server":{
    "host":"localhost",
    "port":2017
  }


}
```

```
// 用户授权服务器
    private static AuthorityServer authorityServer = null;


    public static synchronized AuthorityServer getAuthorityServer() throws Exception {
           if (authorityServer == null){
               config.readFile("config/conf.json");
               Registry reg = LocateRegistry.getRegistry(config.getConf("auth_server").getString("host"), config.getConf("auth_server").get
Int("port"));
               authorityServer = (AuthorityServer)(reg.lookup("authorityServer"));
           }
           return authorityServer;
       }
```

```
//消息存储服务器
private static MessageStoreServer messageStoreServer = null;

   public static synchronized MessageStoreServer getMessageStoreServer() throws Exception {
        if (messageStoreServer == null){
            config.readFile("config/conf.json");
            Registry reg = LocateRegistry.getRegistry(config.getConf("store_server").getString("host"), config.getConf("store_server").getI
nt("port"));
            messageStoreServer = (MessageStoreServer)(reg.lookup("messageStoreServer"));
        }
        return messageStoreServer;
    }
```

```
// sqlite数据库服务器
    private static DBServer dbServer = null;

    public static synchronized DBServer getDbServer() throws Exception {
        if (dbServer == null){
            config.readFile("config/conf.json");
            Registry reg = LocateRegistry.getRegistry(config.getConf("db_server").getString("host"), config.getConf("db_server").getInt("po
rt"));
            dbServer = (DBServer)(reg.lookup("DBServer"));
        }
        return dbServer;
    }
```