程序文档

小组成员

学号	姓名	Github
1352892	谭靖儒	tztztztztz
1352875	黄安娜	aaana
1352965	晁佳欢	wlmxjm1
1352923	马致远	maerye
1352965	林昌盛	apelyn

主要内容

- 项目描述
  - 基本功能
  - 功能扩展
    - 功能扩展1
    - 功能扩展2
    - 功能扩展3
    - 功能扩展4
    - 功能扩展5
    - 功能扩展6
- 项目展示
- 项目架构
  - 基础框架
  - 模块
  - 架构图
- 协议定义
- 数据流图
- 配置管理

项目描述

编号	功能	功能描述	完成时间
1	基本功能	用户登陆聊天并进行文件记录(复用组件重构应用程序之后)	2016.04.20
2	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
3	功能扩展2	消息组播、每周进行文件归档	2016.05.03
4	功能扩展3	输出文件控制、归档文件加密	2016.05.11
5	功能扩展4	维护同组成员、有序接收遗漏消息	2016.05.17
6	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
7	功能扩展6	分拆Server	2016.05.31

基本功能

该项目是客户端与服务端的一些通信过程，`客户端`能够与`服务端`进行连接.

1. 客户端能够申请登陆服务器, 并且登陆成功之后能够发送信息.
2. 服务端能够客户端发送的信息做一定的处理, 并且能够对登陆成功的客户端的信息转发给所有的登陆的客户端
3. 客户端和服务端都能记录活动信息到文件中

功能扩展

功能扩展1

1. Server/Client保存所有收到的消息到文件
2. 每天所有的输出文件归档至一个压缩包

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端收到的消息报存在./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
        }
```

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

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

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.getC`
   - 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(); a`
     - 客户端界面更新
       `java @Subscribe public void SomeOneOffline(SomeOneOfflineEvent someOneOfflineEvent) { String account=someOneOfflineEvent.getAc`
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;

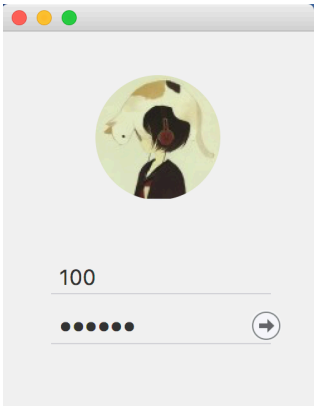
   public static synchronized MessageStoreServer getMessageStoreServer() throws Exception {
        if (messageStoreServer == null){
            config.readFile("config/conf.json");
            Registry reg = LocateRegistry.getRegistry(config.getConf("store_server").getString("host"), config.getConf("store_server").getI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

项目展示

- 登陆界面



- 聊天界面
    - 用户200和201次序登陆并聊天：

- 接着同组另一用户202登陆(次序接收所有遗漏消息):

- 用户202下线

- 登陆失败警告

项目架构

基础框架

项目直接使用了Netty这个基于socket的库来作为基础框架, 使用 `事件驱动` 以及 `异步` 和 `事件机制` 来构建整个项目

详见Netty工作流

主要服务

中央服务

主要负责消息的接受和转发

鉴权服务

主要负责登陆时的认证

消息存储服务

主要负责消息的存储

```
1. 服务器之间主要通过java RMI（远程方法调用） 来进行交互
2. 获取服务时使用**依赖注入**的方法来获取服务
```

数据库服务

主要提供数据库的存储和相关操作

模块

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;
    }
```

json解析模块

在客户端和服务端都有负责将json字符串转换为对象的模块，主要使用了Gson来进行转换。

管道管理模块

在客户端连接到服务器之后，将用户的信息（**目前只记录了用户管道所在的组**）与管道的信息加到到管道管理模块下，方便以后对用户管道的识别以及以后功能的扩展。

ClientChannel.java

```
public class ClientChannel {
    private Channel channel;
    private int groupId;
}
```

Manager.java

```
public class Manager {
    public static final ArrayList<ClientChannel> clientChannels = new ArrayList<ClientChannel>();
}
```

流量控制模块

根据需求，对客户端发送消息的频率和总数都有限制。对于发送总数的限制，在每个客户端的这一模块中会有一个变量保存已收到的消息数目，在每次收到消息时都会判断是否超过限制。对于每秒发送消息数的限制，这里用到了Google limit.limiter.RateLimiter，服务端每次收到某个客户端消息时都判断是否超过了频率限制。

日志模块

主要使用了Log4J 进行日志的记录, 其中客户端和服务器使用了不同的配置文件.

- 客户端 `config/log4j-client.property`

- 服务端 `config/log4j-server.property`

定时任务直接使用了 `java Timer` 来完成

数据库模块

数据库部分使用比较轻便的**Sqlite**来进行了数据的持久化存储

配置模块

使用了简单易懂的json来作为配置文件, 记录了服务端的IP和Port，以及每次登陆所允许发送的消息最大次数maxMsgNumber和每秒允许发送的消息最大次数maxMsgNumberPerSec

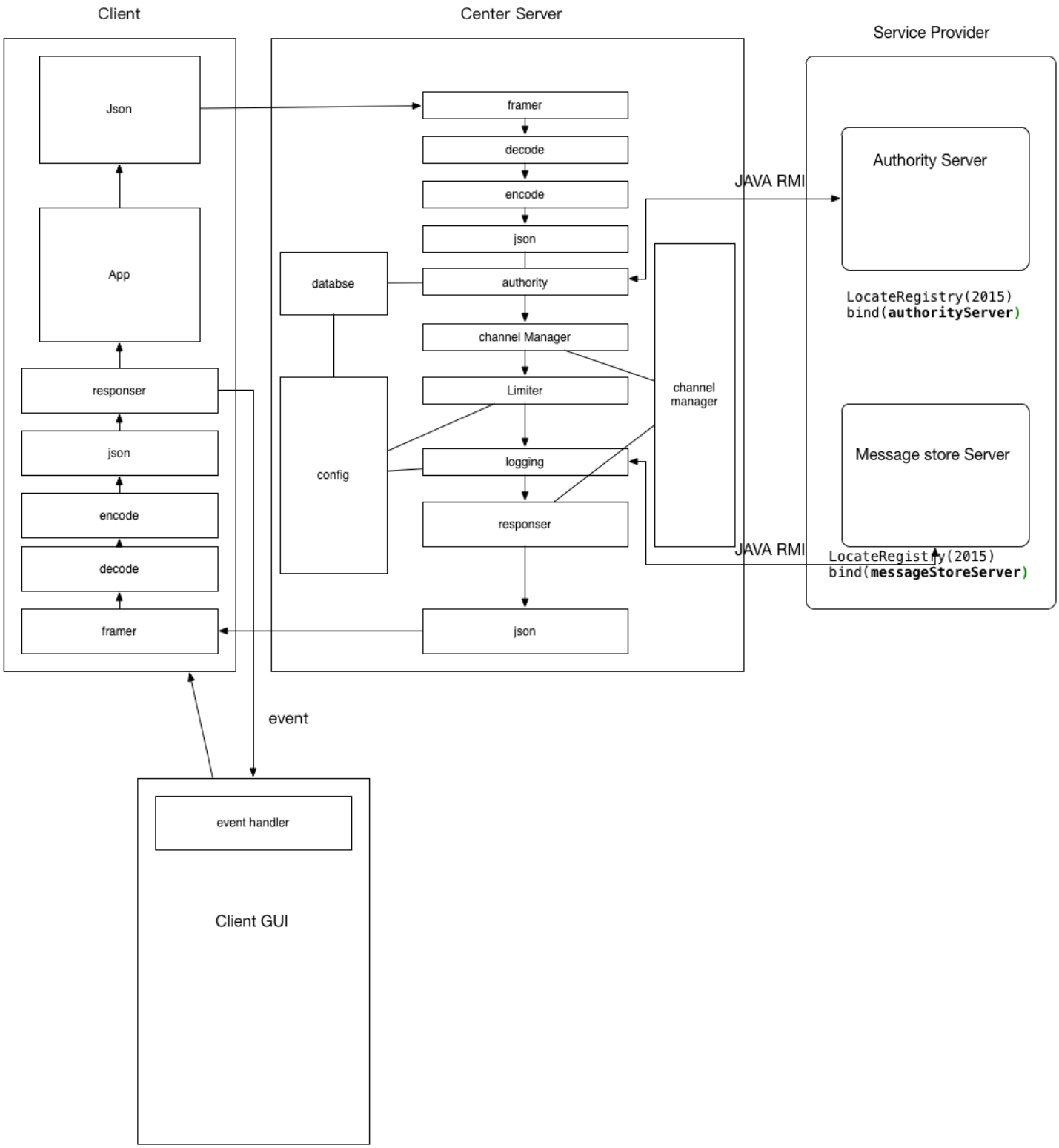
流水线模块

基于Netty工作流, 我们需要构建一系列的管道来对接收到的数据来进行的一个流水线式的操作.

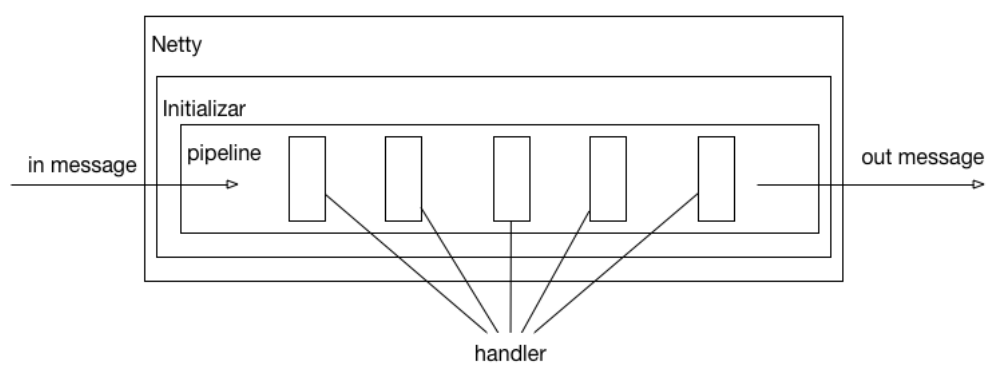
压缩模块

基于Zip4j封装了一个易于自己使用的压缩构建，负责处理每天每周生成的各类归档压缩加密。

架构图


Client
Center Server
Service Provider
Json
App
responser
json
encode
decode
framer
framer
decode
encode
json
databse
authority
channel Manager
Limiter
logging
responser
config
channel manager
json
JAVA RMI
Authority Server
LocateRegistry(2015)
bind(authorityServer)
Message store Server
JAVA RMI
LocateRegistry(2015)
bind(messageStoreServer)
event
event handler
Client GUI

Netty工作流



协议定义

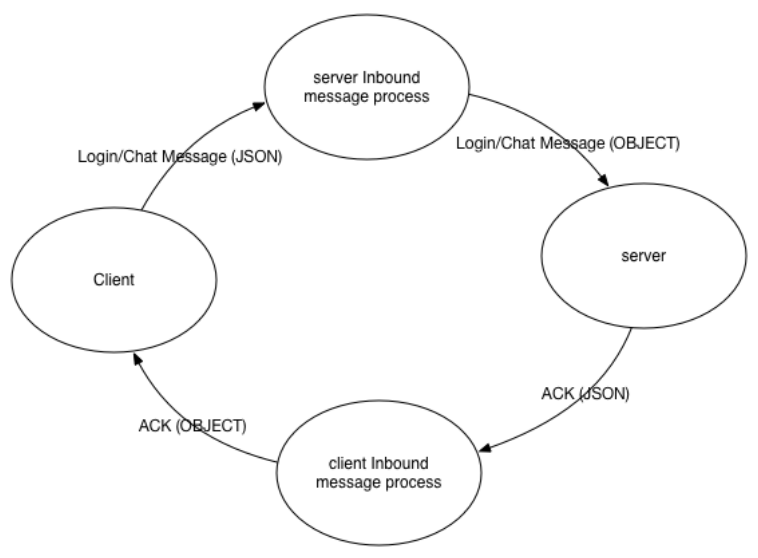
我们对客户端向服务端发送的消息和服务端回复客户端的消息预先进行了一些定义，具体如下：

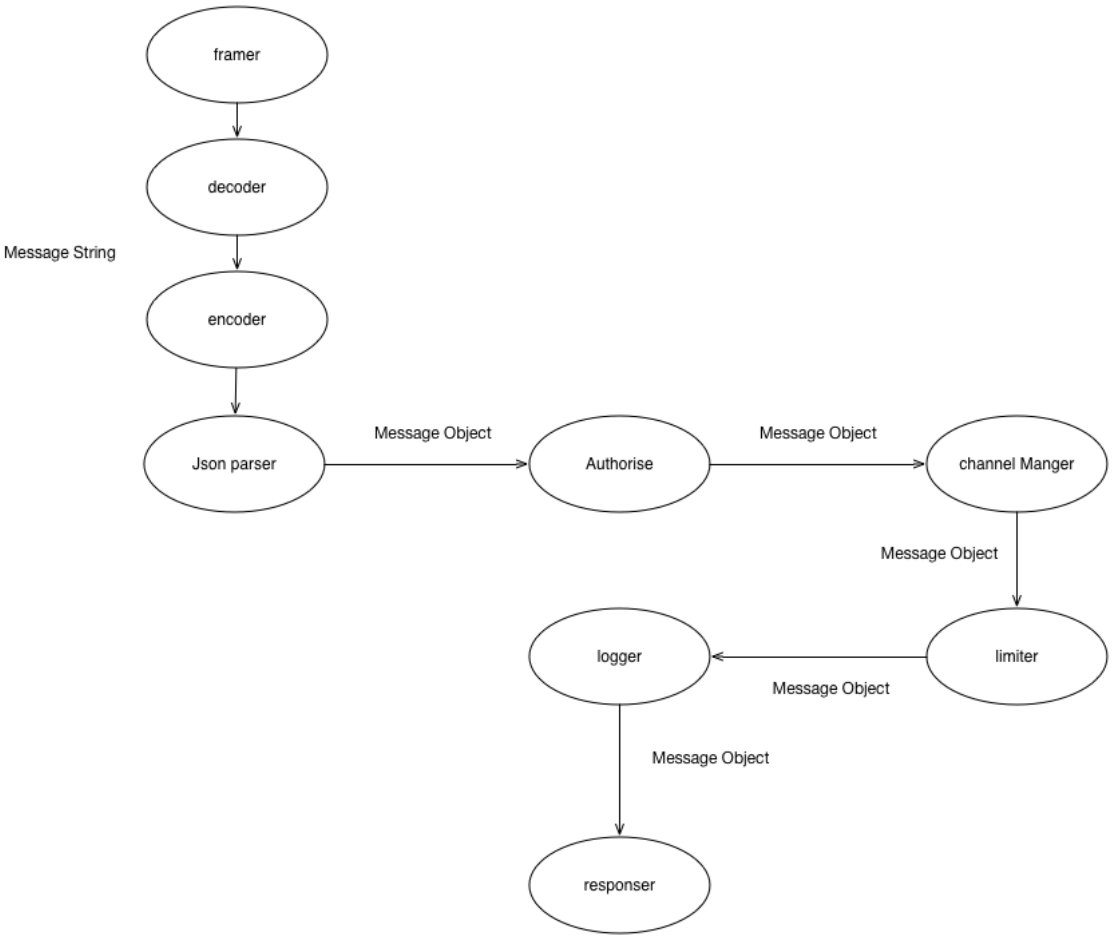
客户端发送的消息有两种类型： + AUTHORITY（登录消息） + CHATTING（聊天消息）

服务端回复给客户端的消息有六种类型： + LOGINFAIL(登录失败) + LOGINSUCCESS(登录成功) + RELOGIN(需要重新登录) + TOOFREQUENT（消息发送过于频繁） + SENDSUCCESS（发送成功） + OTHERMESSAGE（收到其他客户端发送的消息）

这样客户端和服务端在收到消息时就可以根据消息类型做相应的处理。

数据流图





配置管理

使用 github repository 进行版本控制，各成员在各自的分支上编写代码，在完成某一功能模块后合并至主分支。