程序文档

主要内容

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

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.getChatContent().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

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

模块

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<Cl
ientChannel>();
}
```

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

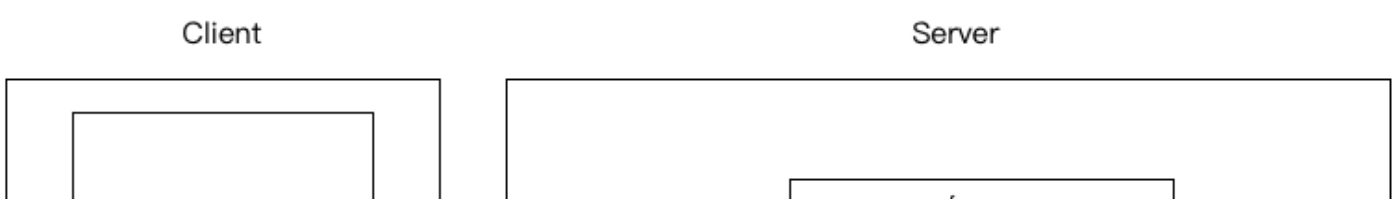
流水线模块

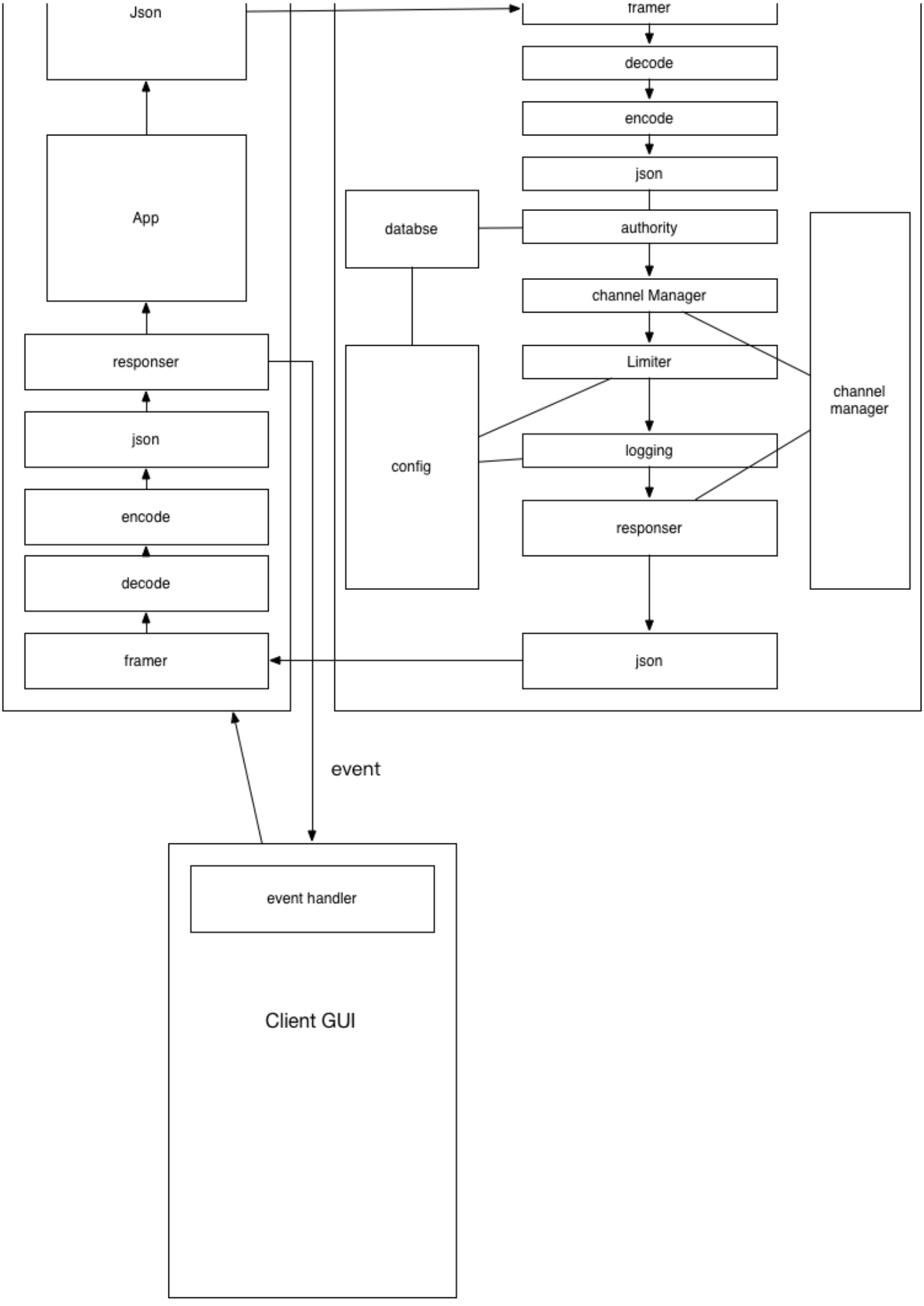
基于Netty工作流, 我们需要构建一系列的管道来对接收到的数据来进行的一个流水线式的操作.

压缩模块

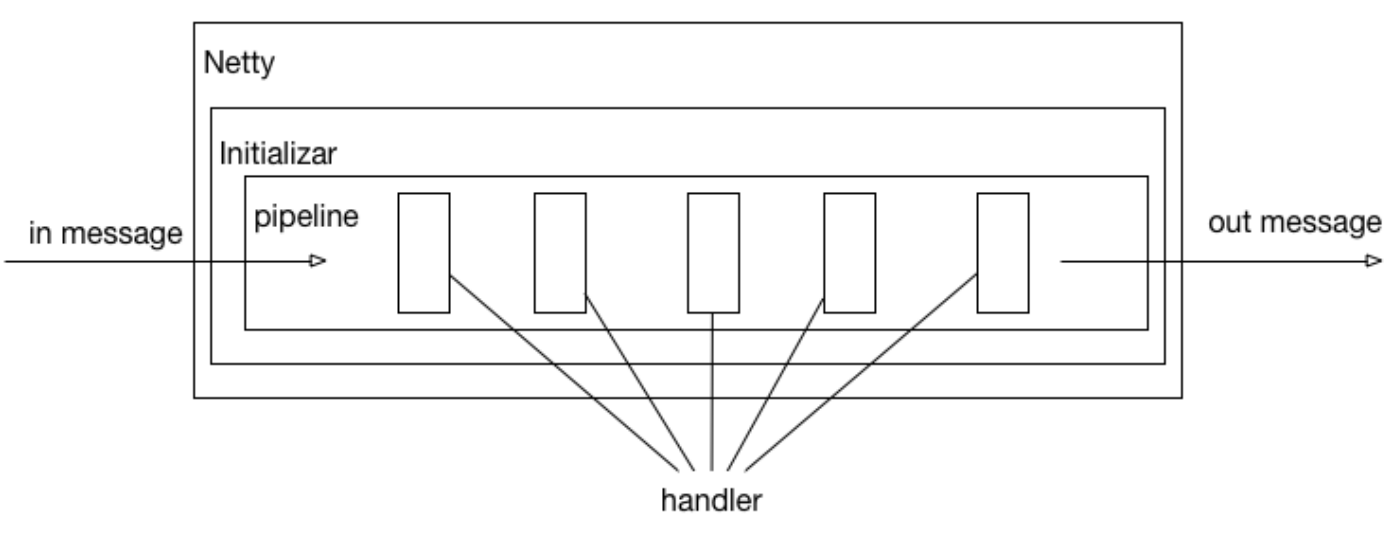
基于Zip4j封装了一个易于自己使用的压缩构建，负责处理每天每周生成的各类归档压缩加密。

架构图




Json
framer
decode
encode
json
App
databse
authority
channel Manager
responser
Limiter
channel
manager
config
logging
responser
json
framer
event
event handler
Client GUI

Netty工作流



协议定义

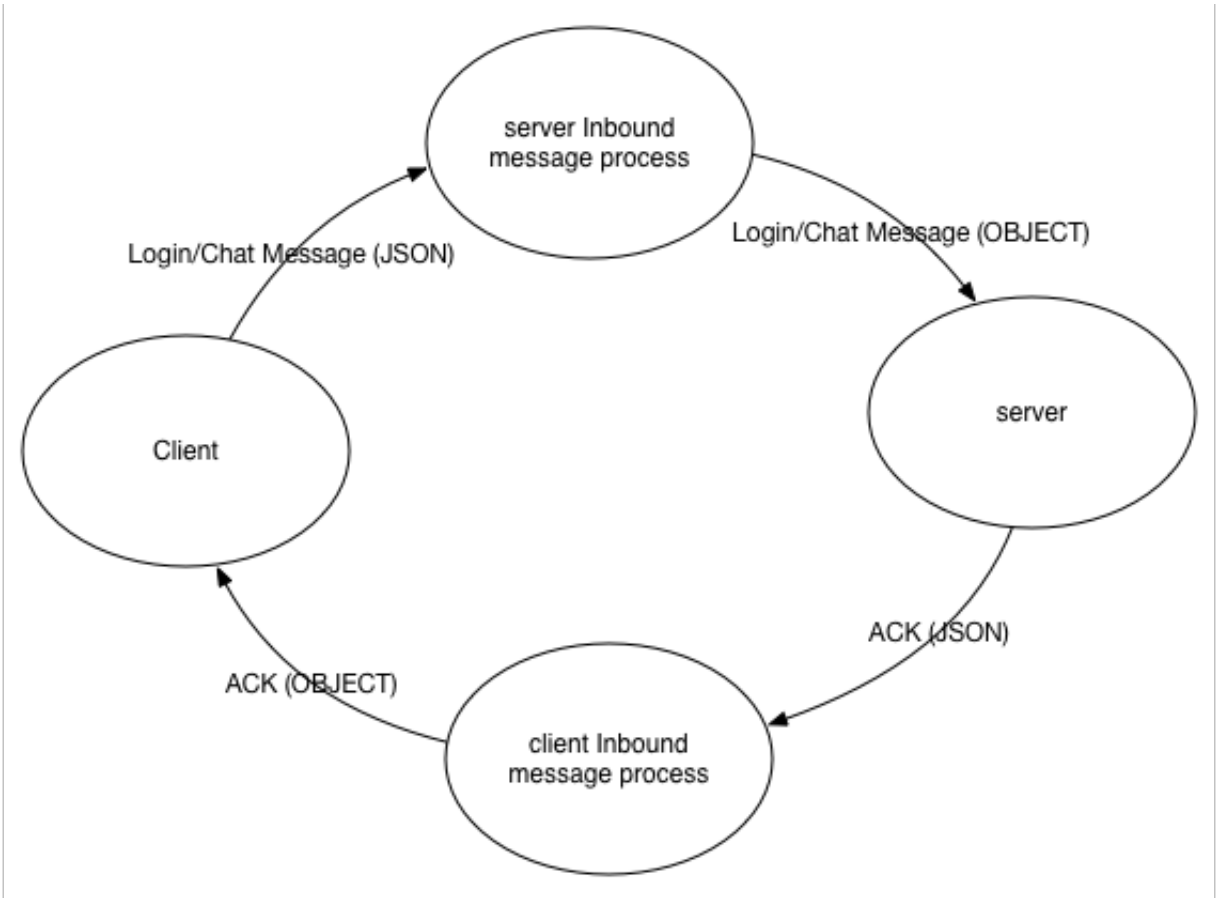
我们对客户端向服务端发送的消息和服务端回复客户端的消息预先进行了一些定义，具体如下：

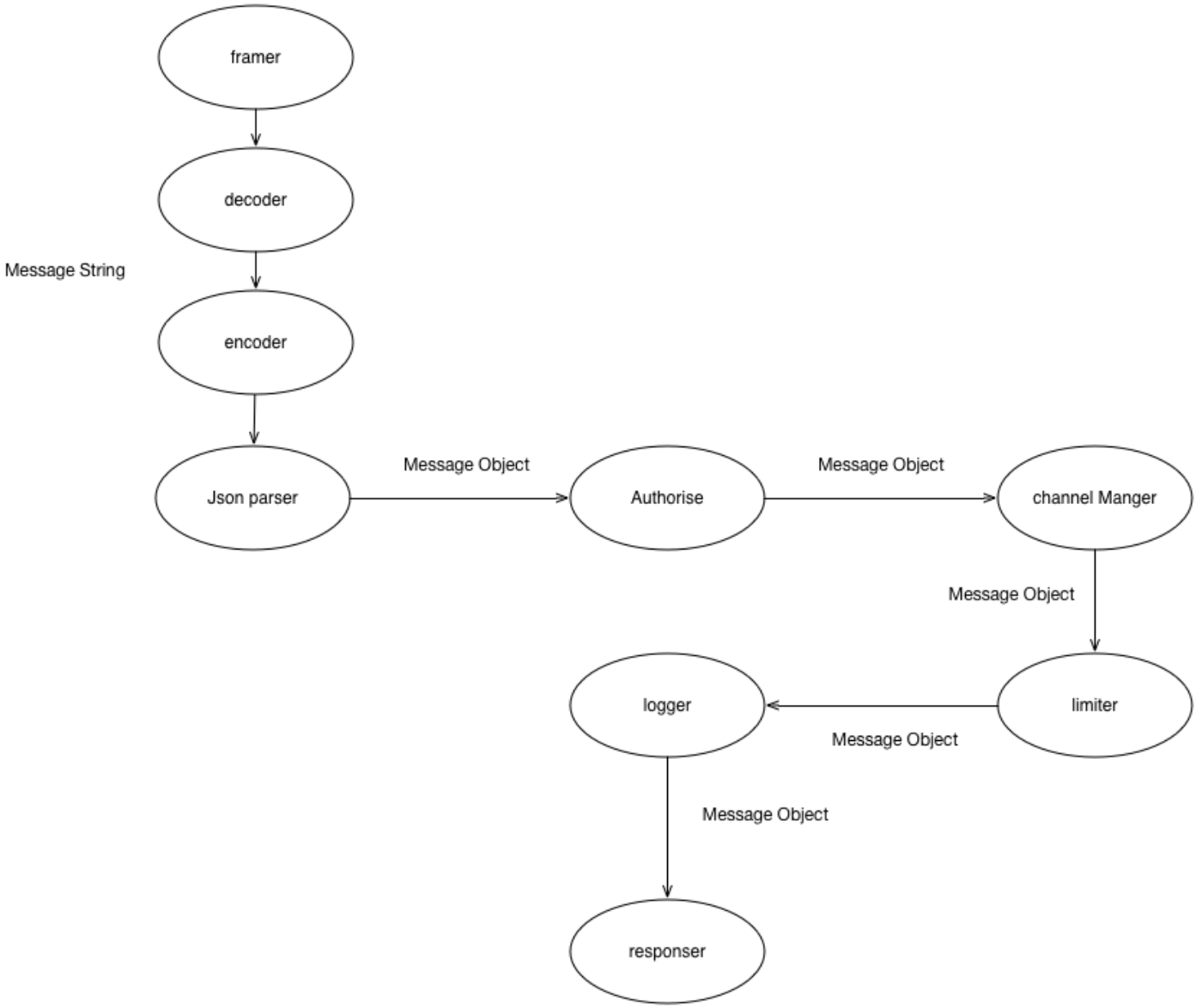
客户端发送的消息有两种类型： + AUTHORITY（登录消息） + CHATTING（聊天消息）

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


server Inbound
message process
Login/Chat Message (JSON)
Login/Chat Message (OBJECT)
Client
server
ACK (JSON)
ACK (OBJECT)
client Inbound
message process



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。