**CloudCate BE 聊天服务器说明文档**

1. **设计思路**

* **采用TCP的方式，用户与服务器建立长连接**
* **使用Protobuf作为通信的二进制协议**
* **链接层和消息层是解耦的，链接层的消息会通过channel通知给逻辑处理层，用户连接的所有状态都会通过channel通知给逻辑处理层。**
* **脏字屏蔽作为独立模块进行设计**
* **使用了比较规范的日志输出**
* **房间管理层（逻辑处理层），维护了一个大厅Hall和一房间字典RoomMap。大厅维护了所有用户的链接和状态。房间内维护了当前房间内所有的用户状态。**
* **房间管理层收到链接层的channel通知的消息，进行消息转换，并且回调初始化时候注册的处理函数**

1. **关键算法**

* **脏字屏蔽，使用了TrieTree字典树的一种结构，节省内存，在进行脏字查询和脏字过滤的时候提高效率**
* **最近N秒内频率最高的词汇，使用以当前Second为key的一个map进行存储。返回N秒内的最高词汇，则以当前秒的前N秒内的结果集合进行统计**

1. **部署和使用**

* **Go version 1.14**
* **Linux Centos 7**
* **git clone https://github.com/zjj9850/chatroom.git**
* **进入工程目录，执行go build .**
* **命令行./chatroom –port 1234**

1. **使用的第三方库**

* **Protobuf v1.26.0，Protobuf作为通信协议，是一种非常高效的数据定义结构，其自身的序列化和反序列化的压缩和效率也非常高。对应的golang的数据结构，理解和使用也非常的方便。不用自己进行消息格式的定义和消息的二进制化的繁琐操作**
* **gnet v1.4.3，gnet是一种基于epoll事件驱动的网络库，底层采用了主从多Reactors 和 goroutine 池的结构，提高了读写的效率。接口非常友好，不用自己造轮子进行TCP粘包等逻辑处理**

1. **可扩展的方面**

* **配置项可以做成yaml，并且做个setting的读取模块，将一些通过配置的变量写入到yaml中，通过setting的模块进行读取，减少代码中的hardcode**
* **最近N秒内频率最高的词汇，算法应该可以进行优化，现在的算法在效率上不是很理想**
* **发送消息在逻辑层中进行，现在逻辑处理层中关联了链接层的一个指针，这里应该可以进行解耦，通过channel进行消息的回包处理**

1. **不理想的地方**

* **有些模块的测试做的不是很到位**
* **一些变量命名上不是很规范**
* **欠缺说明文档的经验**
* **优化的空间还是有很多**