# MessageStoreServer

## 版权说明



对比特项目感兴趣，可以联系这个微信。

## 代码 & 板书链接

https://gitee.com/bitedu-tech/cpp-chatsystem

# 功能设计

消息管理子服务，主要用于管理消息的存储：

- 文本消息，储存在 ElasticSearch 文档搜索服务中。
- 文件/语音/图片，需要转储到文件管理子服务中。

除了管理消息的存储，还需要管理消息的搜索获取，因此需要对外提供以下接口：

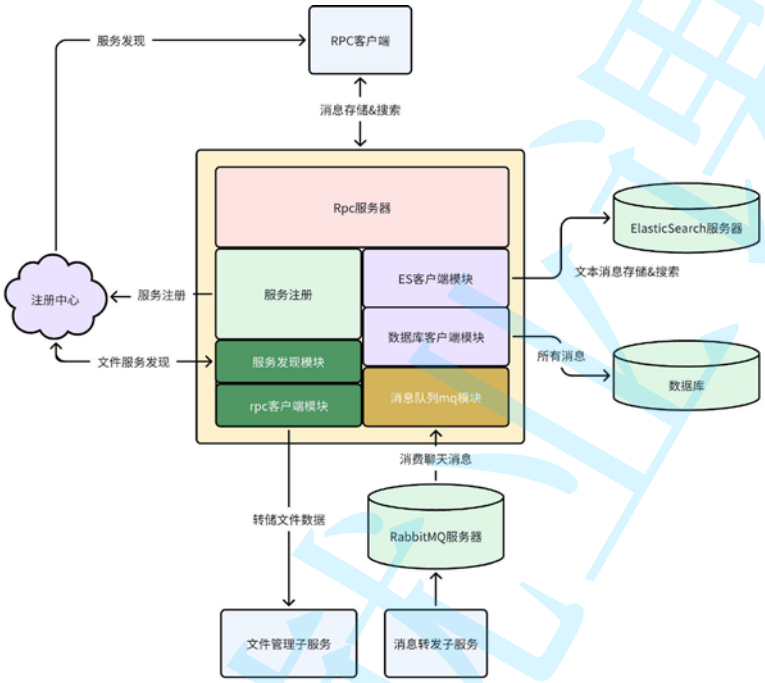
1. 获取历史消息：
    a. 获取最近 N 条消息：用于登录成功后，点击对方头像打开聊天框时显示最近的消息
    b. 获取指定时间段内的消息：用户可以进行聊天消息的按时间搜索
2. 关键字消息搜索：用户可以针对指定好友的聊天进行聊天消息的关键字搜索

# 模块划分

1. 参数/配置文件解析模块：基于 gflags 框架直接使用进行参数/配置文件解析。
2. 日志模块：基于 spdlog 框架封装的模块直接使用进行日志输出。
3. 服务注册模块：基于 etcd 框架封装的注册模块直接使用，进行聊天消息存储子服务的注册。
4. 数据库数据操作模块：基于 odb-mysql 数据管理封装的模块，进行数据库数据操作，用于从 MQ 中消费到消息后，向数据库中存储一份，以便于通过时间进行范围性查找。
    a. 从数据库根据指定用户的所有好友信息
5. rpc 服务模块：基于 brpc 框架搭建 rpc 服务器。
6. 服务发现与调用模块：基于 etcd 框架与 brpc 框架封装的服务发现与调用模块，
    a. 连接文件管理子服务：将文件/语音/图片类型的消息以及用户头像之类的文件数据转储到文件管理子服务。
    b. 连接用户管理子服务：在消息搜索时，根据发送用户的 ID 获取发送者用户信息
7. ES 客户端模块：基于 elasticsearch 框架实现访问客户端，向 es 服务器进行文本聊天消息的存储，以便于文本消息的关键字搜索。

8. MQ 消费模块：基于 rabbitmq-client 封装的消费者模块从消息队列服务器消费获取聊天消息，将文本消息存储到 ElasticSearch 服务，将文件消息转储到文件管理子服务，所有消息的简息都需要向数据库存储一份。

## 模块功能示意图：




# 数据管理

## 数据库消息管理：

在消息的存储管理中，所有的消息简息都要在数据库中存储一份，进行消息的持久化，以便于进行时间范围性查询和离线消息的实现。

消息类型有四种：文本，文件，语音，图片。

我们不可能将文件数据也存储到数据库中，因此数据库中只存储文本消息和其他类型消息的元信息即可。

## 数据库表结构：

- 消息 ID：唯一标识
- 消息产生时间：用于进行时间性搜索
- 消息发送者用户 ID：明确消息的发送者
- 消息产生会话 ID：明确消息属于哪个会话

- 消息类型：明确消息的类型
- 消息内容：只存储文本消息；文件/语音/图片数据不进行存储，或者说是存储在文件子服务中。
- 文件 ID：只有文件/语音/图片类消息会用到
- 文件大小：只有文件/语音/图片类消息会用到
- 文件名称：只有文件类消息会用到

## 数据库操作：

- 新增消息
- 通过消息 ID 获取消息信息
- 通过会话 ID，时间范围，获取指定时间段之内的消息，并按时间进行排序
- 通过会话 ID，消息数量，获取最近的 N 条消息（逆序+limit 即可）

## ODB 映射数据结构：

```
C++
#include <string>
#include <cstddef>
#include <odb/core.hxx>
#include <odb/nullable.hxx>
#include <boost/date_time/posix_time/posix_time.hpp>

#pragma db object
class message
{
public:
    message (){}
private:
    friend class odb::access;

    #pragma db id auto
    unsigned long _id;
    #pragma db unique type("VARCHAR(127)")
    std::string _message_id ;
    #pragma db type("TIMESTAMP") not_null
    boost::posix_time::ptime _created_time;
    #pragma db type("VARCHAR(127)")
    std::string _from_user_id ;
    #pragma db type("VARCHAR(127)")
```

```
    std::string _to_session_id ;
    #pragma db not_null
    signed char _message_type ;
    odb::nullable<std::string> _content;
    #pragma db type("VARCHAR(127)")
    odb::nullable<std::string> _file_id ;
    #pragma db type("VARCHAR(127)")
    odb::nullable<std::string> _filename ;
    odb::nullable<unsigned long> _filesize;
};
```

## ES 文本消息管理：

因为当前聊天室项目中，实现了聊天内容的关键字搜索功能，但是如果在数据库中进行关键字的模糊匹配，则效率会非常低，因此采用 ES 进行消息内容存储与搜索，但是在搜索的时候需要进行会话的过滤，因此这里也要考虑 ES 索引的构造。

## ES 文档 INDEX：

```HTTP
POST /message/_doc
{
    "settings" : {
        "analysis" : {
            "analyzer" : {
                "ik" : {
                    "tokenizer" : "ik_max_word"
                }
            }
        }
    },
    "mappings" : {
        "dynamic" : true,
        "properties" : {
            "chat_session_id" : {
                "type" : "keyword",
                "analyzer" : "standard"
            },
            "message_id" : {
                "type" : "keyword",
                "analyzer" : "standard"
            },
            "content" : {
```

```
                "type" : "text",
                "analyzer" : "ik_max_word"
            }
        }
    }
}
```

## ES 消息测试用例

### 新增数据

```
HTTP
POST /message/_doc/_bulk
{"index":{"_id":"1"}}
{"chat_session_id" : "会话 ID1","message_id" : "消息
ID1","content" : "吃饭了么？"}
{"index":{"_id":"2"}}
{"chat_session_id" : "会话 ID1","message_id" : "消息
ID2","content" : "吃的盖浇饭。"}
{"index":{"_id":"3"}}
{"chat_session_id" : "会话 ID2","message_id" : "消息
ID3","content" : "昨天吃饭了么？"}
{"index":{"_id":"4"}}
{"chat_session_id" : "会话 ID2","message_id" : "消息
ID4","content" : "昨天吃的盖浇饭。"}
```

### 查看数据

```
HTTP
GET /message/_doc/_search?pretty
{
    "query": {
        "match_all": {}
    }
}
```

### 搜索数据

```
HTTP
GET /message/_doc/_search?pretty
{
```

```
    "query" : {
        "bool" : {
            "must" : [
                {
                        "term" : { "chat_session_id.keyword" : "会
话 ID1" }
                },
                {
                        "match" : { "content" : "盖浇饭" }
                }
            ]
        }
    }
}
```

### 删除索引

```
HTTP
DELETE /message
```

# 接口实现流程

## 最近 N 条消息获取：

1. 从请求中，获取会话 ID， 要获取的消息数量
2. 访问数据库，从数据库中按时间排序，获取指定数量的消息简略信息（消息 ID，会话 ID，消息类型，产生时间，发送者用户 ID，文本消息内容，文件消息元信息）
3. 循环构造完整消息（从用户子服务获取消息的发送者用户信息，从文件子服务获取文件/语音/图片数据）
4. 组织响应返回给网关服务器。

## 指定时间段消息搜索：

1. 从请求中，获取会话 ID， 要获取的消息的起始时间与结束时间。
2. 访问数据库，从数据库中按时间进行范围查询，获取消息简略信息（消息 ID，会话 ID，消息类型，产生时间，发送者用户 ID，文本消息内容，文件消息元信息）
3. 循环构造完整消息（从用户子服务获取消息的发送者用户信息，从文件子服务获取文件/语音/图片数据）

4. 组织响应返回给网关服务器。

## 关键字消息搜索：

1. 从请求中，获取会话 ID， 搜索关键字。
2. 基于封装的 ES 客户端，访问 ES 服务器进行文本消息搜索（以消息内容进行搜索，以会话 ID 进行过滤），从 ES 服务器获取到消息简息（消息 ID，会话 ID， 文本消息内容）。
3. 循环从数据库根据消息 ID 获取消息简息（消息 ID，消息类型，会话 ID，发送者 ID，产生时间，文本消息内容，文件消息元数据）。
4. 循环从用户子服务获取所有消息的发送者用户信息，构造完整消息。
5. 组织响应返回给网关服务器。