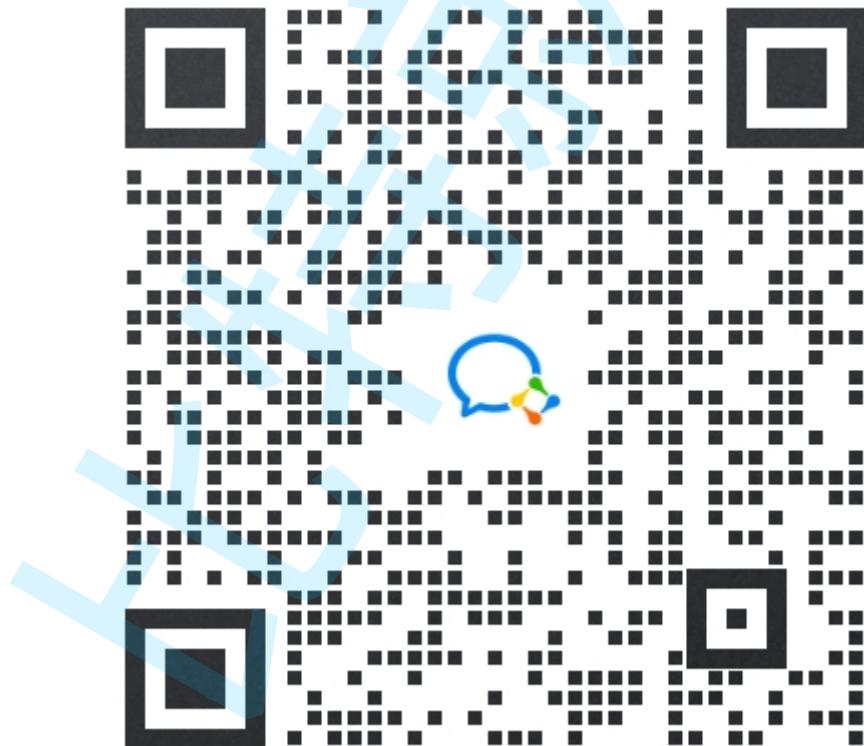


# 聊天室后台服务器实现

## 版权说明

本“比特就业课”项目（以下简称“本项目”）的所有内容，包括但不限于文字、图片、音频、视频、软件、程序、数据库、设计、布局、界面等，均由本项目的开发者或授权方拥有版权。我们鼓励个人学习者使用本项目进行学习和研究。在遵守相关法律法规的前提下，个人学习者可以下载、浏览、学习本项目的内容，并为了个人学习、研究或教学目的而使用其中的材料。但请注意，未经我们明确授权，个人学习者不得将本项目的内容用于任何商业目的，包括但不限于销售、转让、许可或以其他方式从中获利。此外，个人学习者也不得擅自修改、复制、传播、展示、表演或制作本项目内容的衍生作品。任何未经授权的使用均属侵权行为，我们将依法追究法律责任。如果您希望以其他方式使用本项目的内容，包括但不限于引用、转载、摘录、改编等，请事先与我们取得联系，获取书面授权。感谢您对“比特就业课”项目的关注与支持，我们将持续努力，为您提供更好的学习体验。特此说明。比特就业课版权所有方

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>

## 服务器功能设计

在聊天室项目的功能设计中，包含了以下功能：

1. 用户注册：用户输入用户名(昵称)，以及密码进行用户名的注册
2. 用户登录：用户通过用户名和密码进行登录
3. 短信验证码获取：当用户通过手机号注册或登录的时候，需要获取短信验证码
4. 手机号注册：用户输入手机号和短信验证码进行手机号的用户注册
5. 手机号登录：用户输入手机号和短信验证码进行手机号的用户登录
6. 用户信息获取：当用户登录之后，获取个人信息进行展示
7. 头像修改：设置用户头像
8. 昵称修改：设置用户昵称
9. 签名修改：设置用户签名
10. 手机号修改：修改用户的绑定手机号
11. 好友列表的获取：当用户登录成功之后，获取自己好友列表进行展示
12. 申请好友：搜索用户之后，点击申请好友，向对方发送好友申请
13. 待处理申请的获取：当用户登录成功之后，会获取离线的好友申请请求以待处理
14. 好友申请的处理：针对收到的好友申请进行同意/拒绝的处理
15. 删除好友：删除当前好友列表中的好友
16. 用户搜索：可以进行用户的搜索用于申请好友
17. 聊天会话列表的获取：每个单人/多人聊天都有一个聊天会话，在登录成功后可以获取聊天会话，查看历史的消息以及对方的各项信息
18. 多人聊天会话的创建：单人聊天会话在对方同意好友时创建，而多人会话需要调用该接口进行手动创建
19. 聊天成员列表的获取：多人聊天会话中，可以点击查看群成员按钮，查看群成员信息
20. 发送聊天消息：在聊天框输入内容后点击发送，则向服务器发送消息聊天请求
21. 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
22. 消息搜索：用户可以进行聊天消息的关键字搜索
23. 文件的上传
- a. 单个文件的上传：这个接口基本用于后台部分，收到文件消息后将文件数据转发给文件子服务进行存储
  - b. 多个文件的上传：这个接口基本用于后台部分，收到文件消息后将文件数据转发给文件子服务进行存储
24. 文件的下载
- a. 单个文件的下载：在后台用于获取用户头像文件数据，以及客户端用于获取文件/语音/图片消息的文件数据
  - b. 多个文件的下载：在后台用于大批量获取用户头像数据（比如获取用户列表的时候），以及前端的批量文件下载
25. 语音消息的文字转换：客户端进行语音消息的文字转换。

除了以上的与客户端之间交互的功能之外，还包含一些服务器后台内部所需的功能：

1. 消息的存储：用于将文本消息进行存储起来，以便于进行消息的搜索，以及离线消息的存储。
2. 文件的存储：用于存储用户的头像文件，以及消息中的文件/图片/语音文件数据。
3. 各项用户，好友，会话数据的存储管理

## 框架与微服务拆分设计

该项目在设计的时候采用微服务框架设计，指将一个大的业务拆分为多个子业务，分别在多台不同的机器节点上提供对应的服务，由网关服务统一接收多个客户端的各种不同请求，然后将请求分发到不同的子服务节点上进行处理，获取响应后，再转发给客户端。

微服务架构设计的思想主要包括以下几个方面：

1. **服务拆分**：将应用程序拆分成多个小型服务，每个服务负责一部分业务功能，具有独立的生命周期和部署。
2. **独立部署**：每个微服务可以独立于其他服务进行部署、更新和扩展。
3. **语言和数据的多样性**：不同的服务可以使用不同的编程语言和数据库，根据服务的特定需求进行技术选型。
4. **轻量级通信**：服务之间通过定义良好的 API 进行通信，通常使用 HTTP/REST、

gRPC 等协议。

5. **去中心化治理**: 每个服务可以有自己的开发团队，拥有自己的技术栈和开发流程。
6. **弹性和可扩展性**: 微服务架构支持服务的动态扩展和收缩，以适应负载的变化。
7. **容错性**: 设计时考虑到服务可能会失败，通过断路器、重试机制等手段提高系统的容错性。
8. **去中心化数据管理**: 每个服务管理自己的数据库，数据在服务之间是私有的，这有助于保持服务的独立性。
9. **自动化部署**: 通过持续集成和持续部署 (CI/CD) 流程自动化服务的构建、测试和部署。
10. **监控和日志**: 对每个服务进行监控和日志记录，以便于跟踪问题和性能瓶颈。
11. **服务发现**: 服务实例可能动态变化，需要服务发现机制来动态地找到服务实例。
12. **安全**: 每个服务需要考虑安全问题，包括认证、授权和数据传输的安全性。

基于微服务的思想，以及聊天室项目的业务功能，将聊天室项目进行服务拆分为以下几个子服务：

## 网关服务

网关服务，提供与客户端进行直接交互的作用，用于接收客户端的各项不同的请求，进行用户鉴权通过后，将请求分发到各个不同的子服务进行处理，接收到响应后，发送给客户端。

用户鉴权：客户端在登录成功后，后台会为客户端创建登录会话，并向客户端返回一个登录会话 ID，往后，客户端发送的所有请求中都必须带有对应的会话 ID 进行身份识别，否则视为未登录，不予提供除注册/登录/验证码获取以外的所有服务。

在网关服务中，基于不同的使用目的，向客户端提供两种不同的通信：

### HTTP 通信：

在项目的设计中客户端的大部分业务都是基于请求-响应模式进行的，因此基于便于扩展，设计简单的目的，采用 HTTP 协议作为与客户端进行基础的业务请求的通信协议，在 HTTP 通信中涵盖了上述所有的功能接口请求。

### WEBSOCKET 通信：

在聊天室项目中，不仅仅包含客户端主动请求的业务，还包含了一些需要服务器主动推送的通知，因为 HTTP 不支持服务器主动推送数据，因此采用 Websocket 协议进行长连接的通信，向客户端发送通知类型的数据。

- 好友申请的通知
- 好友申请处理结果的通知
- 好友删除的通知
- 聊天会话建立的通知
- 聊天新消息的通知

## 用户管理子服务

用户管理子服务，主要用于管理用户的数据，以及关于用户信息的各项操作，因此在上述项目功能中，用户子服务需要提供以下接口：

1. 用户注册：用户输入用户名(昵称)，以及密码进行用户名的注册
2. 用户登录：用户通过用户名和密码进行登录
3. 短信验证码获取：当用户通过手机号注册或登录的时候，需要获取短信验证码
4. 手机号注册：用户输入手机号和短信验证码进行手机号的用户注册
5. 手机号登录：用户输入手机号和短信验证码进行手机号的用户登录
6. 用户信息获取：当用户登录之后，获取个人信息进行展示
7. 头像修改：设置用户头像
8. 昵称修改：设置用户昵称
9. 签名修改：设置用户签名
10. 手机号修改：修改用户的绑定手机号

## 好友管理子服务

好友管理子服务，主要用于管理好友相关的数据与操作，因此主要负责以下接口：

1. 好友列表的获取：当用户登录成功之后，获取自己好友列表进行展示
2. 申请好友：搜索用户之后，点击申请好友，向对方发送好友申请
3. 待处理申请的获取：当用户登录成功之后，会获取离线的好友申请请求以待处理
4. 好友申请的处理：针对收到的好友申请进行同意/拒绝的处理
5. 删除好友：删除当前好友列表中的好友
6. 用户搜索：可以进行用户的搜索用于申请好友
7. 聊天会话列表的获取：每个单人/多人聊天都有一个聊天会话，在登录成功后可以获取聊天会话，查看历史的消息以及对方的各项信息

8. 多人聊天会话的创建：单人聊天会话在对方同意好友时创建，而多人会话需要调用该接口进行手动创建
9. 聊天成员列表的获取：多人聊天会话中，可以点击查看群成员按钮，查看群成员信息

## 文件管理子服务

文件管理子服务，主要用于管理用户的头像，以及消息中的文件存储，因此需要提供以下接口：

1. 文件的上传
  - a. 单个文件的上传：这个接口基本用于后台部分，收到文件消息后将文件数据转发给文件子服务进行存储
  - b. 多个文件的上传：这个接口基本用于后台部分，收到文件消息后将文件数据转发给文件子服务进行存储
2. 文件的下载
  - a. 单个文件的下载：在后台用于获取用户头像文件数据，以及客户端用于获取文件/语音/图片消息的文件数据
  - b. 多个文件的下载：在后台用于大批量获取用户头像数据（比如获取用户列表的时候），以及前端的批量文件下载

## 消息管理子服务

消息管理子服务，主要用于管理消息元信息的存储，因此需要提供以下接口：

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. 消息搜索：用户可以进行聊天消息的关键字搜索

## 转发管理子服务

转发子服务，主要用于针对一条消息内容，组织消息的 ID 以及各项所要素，然后告诉网关服务器一条消息应该发给谁。

通常消息都是以聊天会话为基础进行发送的，根据会话找到它的所有成员，这就是转发的目标。

除此之外，转发子服务将收到的消息，放入消息队列中，由文件子服务/消息子服务进行消费存储

1. 获取消息转发目标：针对消息内容，组织消息，并告知网关转发目标。

## 语音转换子服务

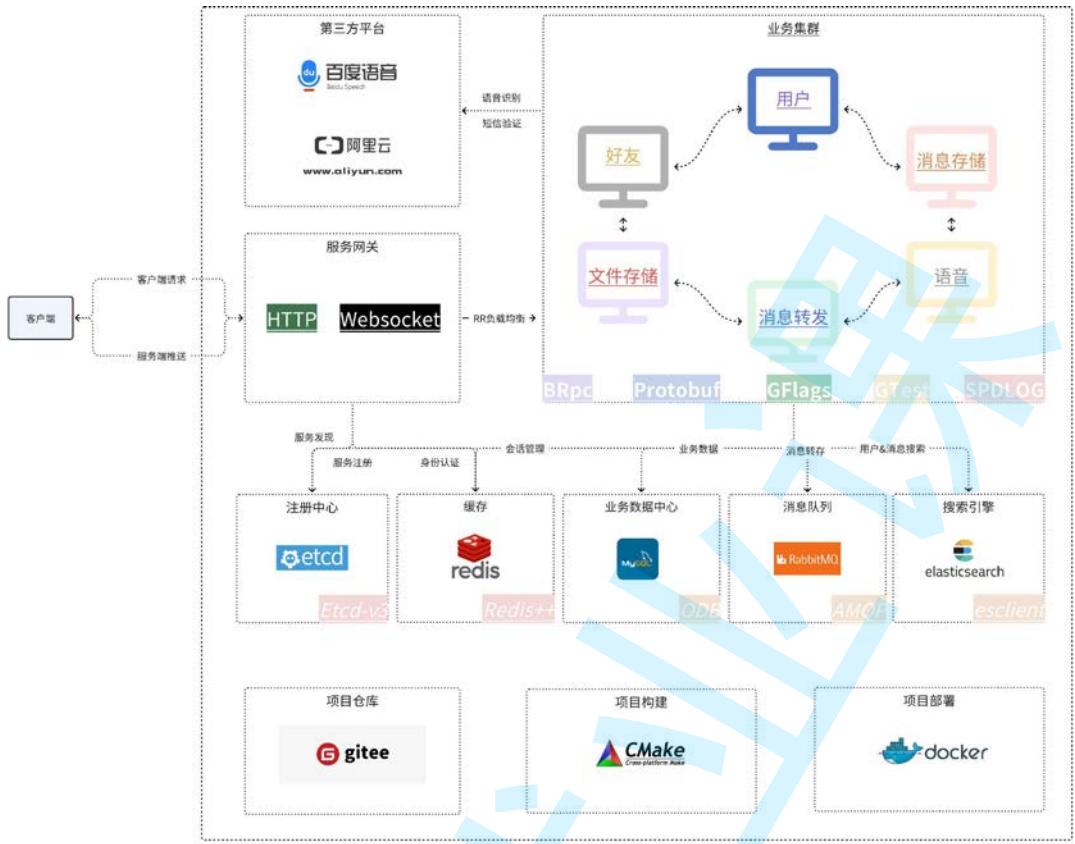
语音转换子服务，用于调用语音识别 SDK，进行语音识别，将语音转为文字后返回给网关。

1. 语音消息的文字转换：客户端进行语音消息的文字转换。

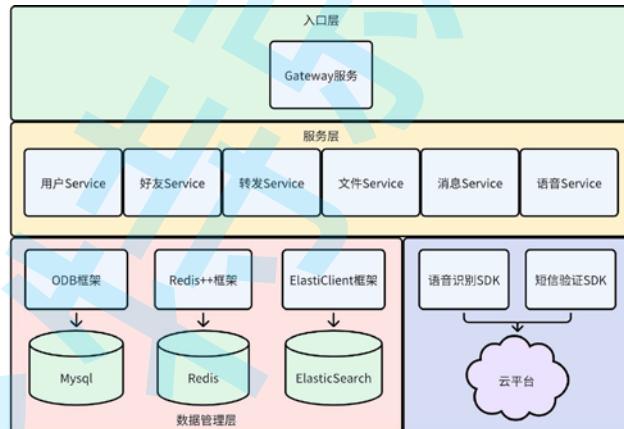
## 项目所使用到的框架/库：

- gflags：针对程序运行所需的运行参数解析/配置文件解析框架。
- gtest：针对程序编写到一定阶段后，进行的单元测试框架。
- spdlog：针对项目中进行日志输出的框架。
- protobuf：针对项目中的网络通信数据所采用的序列化和反序列化框架。
- brpc：项目中的 rpc 调用使用的框架。
- redis：高性能键值存储系统，用于项目中进行用户登录会话信息的存储管理。
- mysql：关系型数据库系统，用于项目中的业务数据的存储管理。
- ODB：项目中 mysql 数据库操作的 ORM 框架（Object-Relational Mapping，对象关系映射）
- Etcd：分布式、高可用的一致性键值存储系统，用于项目中实现服务注册与发现功能的框架。
- cpp-httplib：用于搭建简单轻量 HTTP 服务器的框架。
- websocketpp：用于搭建 Websocket 服务器的框架。
- rabbitMQ：用于搭建消息队列服务器，用于项目中持久化消息的转发消费。
- elasticsearch：用于搭建文档存储/搜索服务器，用于项目中历史消息的存储管理
- 语音云平台：采用百度语音识别技术云平台实现语音转文字功能。
- 短信云平台：采用阿里云短信云平台实现手机短信验证码通知功能。
- cmake：项目工程的构建工具。
- docker：项目工程的一键式部署工具。

## 后台服务技术框架图：

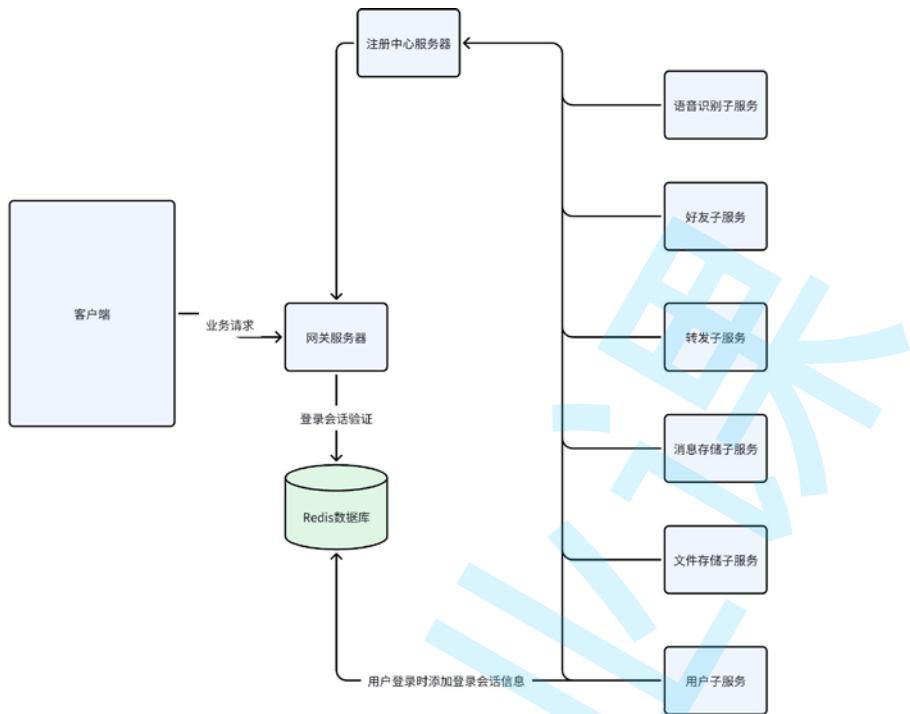


## 后台服务的模块层次图：

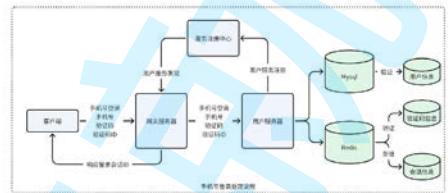
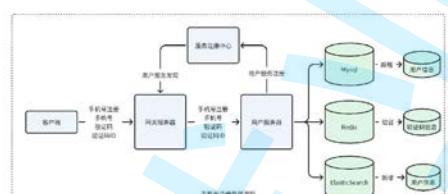
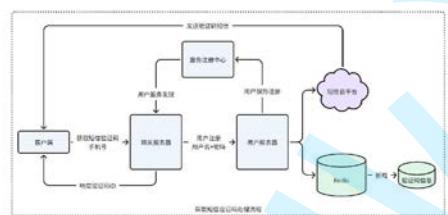
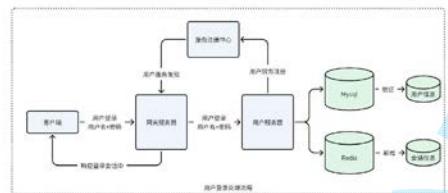
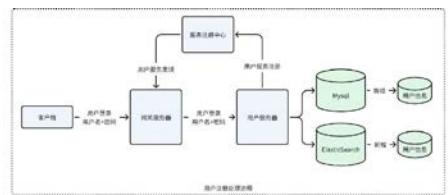


## 后台服务的通信流程图：

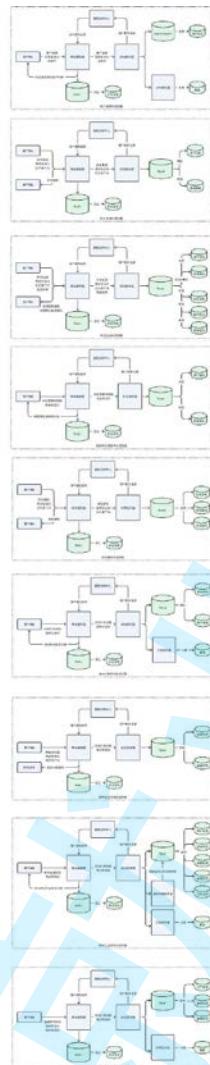
### 入口网关子服务业务接口：



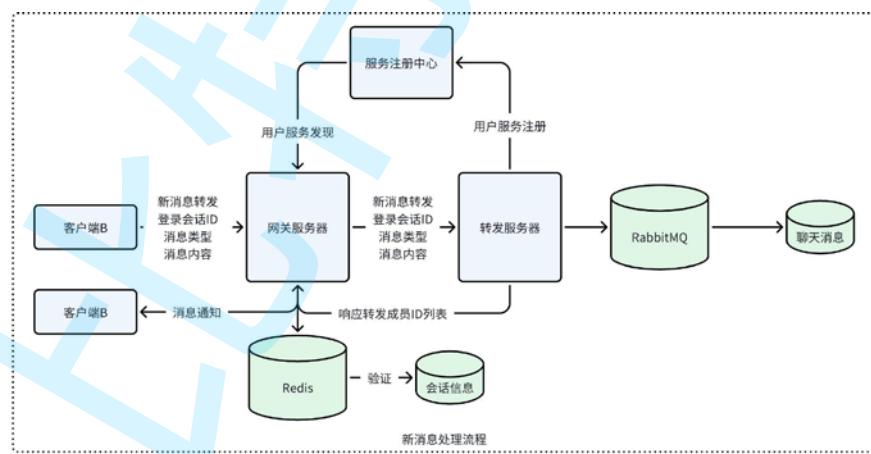
**用户管理子服务业务接口：**



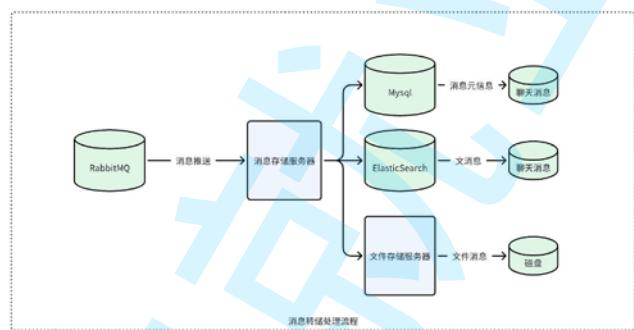
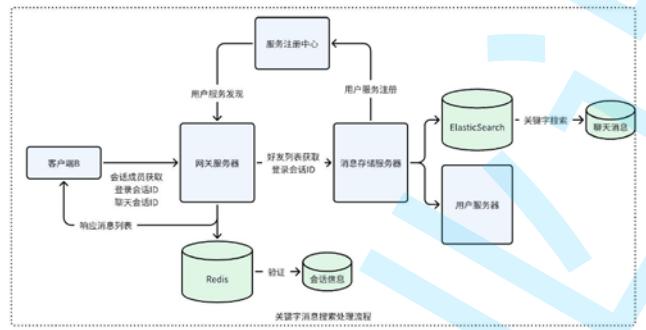
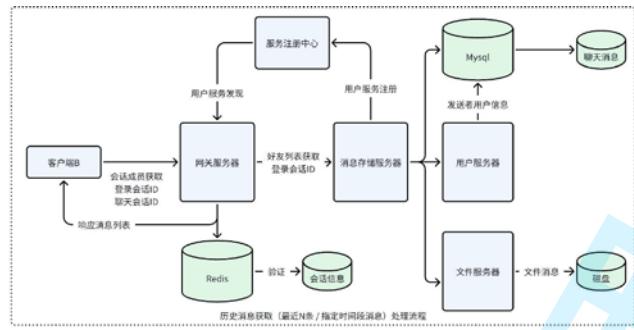
**好友管理子服务业务接口：**



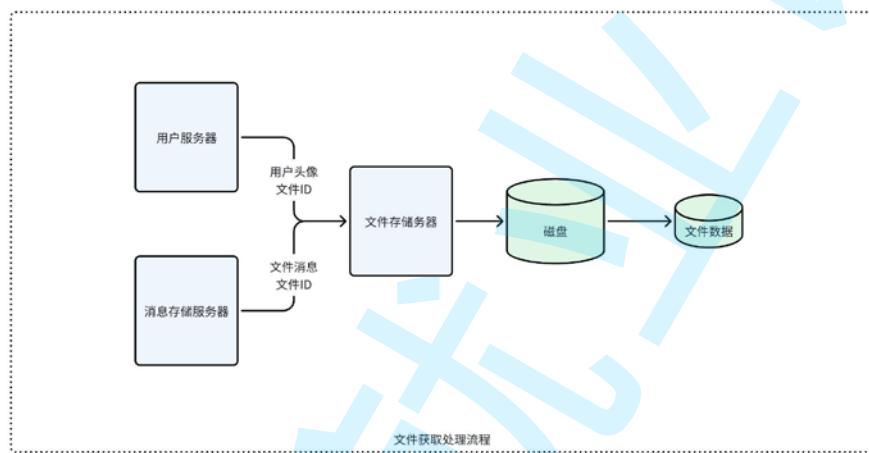
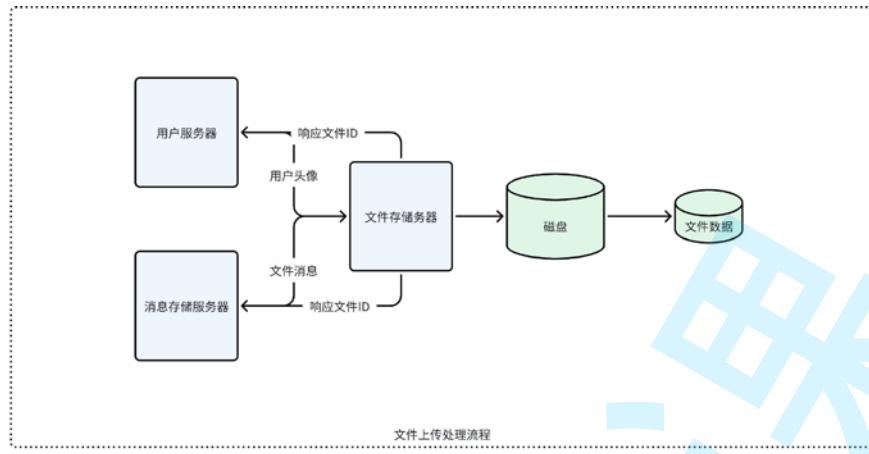
### 转发管理子服务业务接口：



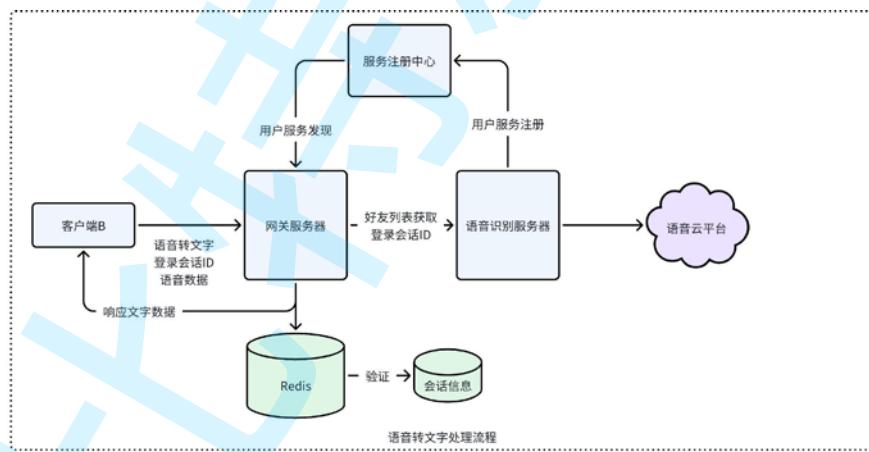
### 消息存储子服务业务接口：



## 文件存储子服务业务接口：



## 语音识别子服务业务接口：



## 微服务通信接口设计

因为微服务框架的思想是将业务拆分到不同的节点主机上提供服务，因此主机节点之间的通信就尤为重要，而在进行开发之前，首先要做的就是将通信接口定义出来，这样只要双方遵循约定，即可实现业务往来。

# 网关服务

网关负责直接与客户端进行通信，其基础业务请求使用 HTTP 协议进行通信，通知类业务使用 Websocket 协议进行通信，接口定义如下：

## 网关 HTTP 接口

HTTP 通信，分为首行，头部和正文三部分，首行中的 URI 明确了业务请求目标，头部进行正文或连接描述，正文中包含请求或响应的内容，在约定的内容中，首先需要定义出来的是 URI：

```
ProtoBuf
//在客户端与网关服务器的通信中，使用 HTTP 协议进行通信
// 通信时采用 POST 请求作为请求方法
// 通信时，正文采用 protobuf 作为正文协议格式，具体内容字段以前边各个文件中定义的字段格式为准
/* 以下是 HTTP 请求的功能与接口路径对应关系：
   SERVICE HTTP PATH:
   {
       获取随机验证码
       /service/user/get_random_verify_code
       获取短信验证码
       /service/user/get_phone_verify_code
       用户名密码注册
       /service/user/username_register
       用户名密码登录
       /service/user/username_login
       手机号码注册
       /service/user/phone_register
       手机号码登录
       获取个人信息
       修改头像
       修改昵称
       修改签名
       /service/user/set_description
       修改绑定手机
       /service/user/set_phone
       获取好友列表
       /service/friend/get_friend_list
       获取好友信息
       /service/friend/get_friend_info
       发送好友申请
   }
```

```
/service/friend/add_friend_apply  
    好友申请处理  
/service/friend/add_friend_process  
    删除好友  
/service/friend/remove_friend  
    搜索用户  
/service/friend/search_friend  
    获取指定用户的消息会话列表  
/service/friend/get_chat_session_list  
    创建消息会话  
/service/friend/create_chat_session  
    获取消息会话成员列表  
/service/friend/get_chat_session_member  
    获取待处理好友申请事件列表  
/service/friend/get_pending_friend_events  
  
    获取历史消息/离线消息列表  
/service/message_storage/get_history  
    获取最近 N 条消息列表  
/service/message_storage/get_recent  
    搜索历史消息  
/service/message_storage/search_history  
  
    发送消息  
/service/message_transmit/new_message  
  
    获取单个文件数据  
/service/file/get_single_file  
    获取多个文件数据  
/service/file/get_multi_file  
    发送单个文件  
/service/file/put_single_file  
    发送多个文件  
/service/file/put_multi_file  
  
    语音转文字  
}  
*/  
/service/speech/recognition
```

其次，在 HTTP 请求正文中，将采用 protobuf 协议作为正文的序列化方式，不同的请求正文与后台的请求基本上吻合，因此请求正文结构，将与后台服务之间复用同一套接口，具体接口格式在下列各项子服务中给出。

## 网关 Websocket 接口：

websocket 通信接口中，包含两方面内容：

- 连接的身份识别：

当用户登录成功后，向服务器发起 websocket 长连接请求，建立长连接。

长连接建立成功后，向服务器发送身份鉴权请求，请求内容为 protobuf 结构数据，主要内容为：

- 请求 ID
- 登录会话 ID：用于进行身份识别

该请求不需要服务端进行回复，鉴权成功则长连接保持，鉴权失败则断开长连接即可。

```
ProtoBuf
syntax = "proto3";
package bite_im;
import "base.proto";
option cc_generic_services = true;

message ClientAuthenticationReq {
    string request_id = 1;
    string session_id = 2;
}
```

- 事件通知的内容

因为事件通知在 websocket 长连接通信中进行，因此只需要定义出消息结构即可：

先将一些公共结构给提取出来进行定义，定义到一个 base.proto 文件中。

```
ProtoBuf
syntax = "proto3";
package bite_im;
option cc_generic_services = true;

//用户信息结构
message UserInfo {
    string user_id = 1;//用户 ID
    string nickname = 2;//昵称
    string description = 3;//个人签名/描述
```

```
    string phone = 4; //绑定手机号
    bytes avatar = 5;//头像照片，文件内容使用二进制
}

//聊天会话信息
message ChatSessionInfo {
    //群聊会话不需要设置，单聊会话设置为对方用户 ID
    optional string single_chat_friend_id = 1;
    string chat_session_id = 2; //会话 ID
    string chat_session_name = 3;//会话名称 git
    //会话上一条消息，新建的会话没有最新消息
    optional MessageInfo prev_message = 4;
    //会话头像 --群聊会话不需要，直接由前端固定渲染，单聊就是对方的头像
    optional bytes avatar = 5;
}

//消息类型
enum MessageType {
    STRING = 0;
    IMAGE = 1;
    FILE = 2;
    SPEECH = 3;
}
message StringMessageInfo {
    string content = 1;//文字聊天内容
}
message ImageMessageInfo {
    //图片文件 id，客户端发送的时候不用设置，由 transmit 服务器进行设置后
    //交给 storage 的时候设置
    optional string file_id = 1;
    //图片数据，在 ES 中存储消息的时候只要 id 不要文件数据，服务端转发的
    //时候需要原样转发
    optional bytes image_content = 2;
}
message FileMessageInfo {
    optional string file_id = 1;//文件 id，客户端发送的时候不用设置
    int64 file_size = 2;//文件大小
    string file_name = 3;//文件名称
    //文件数据，在 ES 中存储消息的时候只要 id 和元信息，不要文件数据，服
    //务端转发的时候也不需要填充
    optional bytes file_contents = 4;
}
```

```

message SpeechMessageInfo {
    //语音文件 id,客户端发送的时候不用设置
    optional string file_id = 1;
    //文件数据，在ES中存储消息的时候只要 id 不要文件数据，服务端转发的时候也不需要填充
    optional bytes file_contents = 2;
}
message MessageContent {
    MessageType message_type = 1; //消息类型
    oneof msg_content {
        StringMessageInfo string_message = 2;//文字消息
        FileMessageInfo file_message = 3;//文件消息
        SpeechMessageInfo speech_message = 4;//语音消息
        ImageMessageInfo image_message = 5;//图片消息
    };
}
//消息结构
message MessageInfo {
    string message_id = 1;//消息 ID
    string chat_session_id = 2;//消息所属聊天会话 ID
    int64 timestamp = 3;//消息产生时间
    UserInfo sender = 4;//消息发送者信息
    MessageContent message = 5;
}

message FileDownloadData {
    string file_id = 1;
    bytes file_content = 2;
}

message FileUploadData {
    string file_name = 1;
    int64 file_size = 2;
    bytes file_content = 3;
}

```

然后，开始定义通知内容结构：

```

Protobuf
syntax = "proto3";
package bite_im;
import "base.proto";
option cc_generic_services = true;

```

```
enum NotifyType {
    FRIEND_ADD_APPLY_NOTIFY = 0;
    FRIEND_ADD_PROCESS_NOTIFY = 1;
    CHAT_SESSION_CREATE_NOTIFY = 2;
    CHAT_MESSAGE_NOTIFY = 3;
    FRIEND_REMOVE_NOTIFY = 4;
}

message NotifyFriendAddApply {
    UserInfo user_info = 1; //申请人信息
}
message NotifyFriendAddProcess {
    bool agree = 1;
    UserInfo user_info = 2; //处理人信息
}
message NotifyFriendRemove {
    string user_id = 1; //删除自己的用户 ID
}
message NotifyNewChatSession {
    ChatSessionInfo chat_session_info = 1; //新建会话信息
}
message NotifyNewMessage {
    MessageInfo message_info = 1; //新消息
}

message NotifyMessage {
    optional string notify_event_id = 1;//通知事件操作 id (有则填无则忽略)
    NotifyType notify_type = 2;//通知事件类型
    oneof notify_remarks { //事件备注信息
        NotifyFriendAddApply friend_add_apply = 3;
        NotifyFriendAddProcess friend_process_result = 4;
        NotifyFriendRemove friend_remove = 7;
        NotifyNewChatSession new_chat_session_info = 5;//会话信息
        NotifyNewMessage new_message_info = 6;//消息信息
    }
}
```

## 用户管理子服务

```
/*
    用户操作服务器的子服务注册信息: /service/user/instance_id
        服务名称: /service/user
        实例 ID: instance_id      每个能够提供用户操作服务的子服务器唯一 ID
    当服务发现的时候, 通过 /service/user 进行服务发现, 就可以发现所有的能够提供用户操作的实例信息了
*/
syntax = "proto3";
package bite_im;
import "base.proto";
option cc_generic_services = true;

//-----
//用户名注册
message UserRegisterReq {
    string request_id = 1;
    string nickname = 2;
    string password = 3;
    string verify_code_id = 4;
    string verify_code = 5;
}
message UserRegisterRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}

//-----
//用户名登录
message UserLoginReq {
    string request_id = 1;
    string nickname = 2;
    string password = 3;
    string verify_code_id = 4;
    string verify_code = 5;
}
message UserLoginRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    string login_session_id = 4;
}
```

```
//手机号验证码获取
message PhoneVerifyCodeReq {
    string request_id = 1;
    string phone_number = 2;
}
message PhoneVerifyCodeRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    string verify_code_id = 4;
}
//-----
//手机号注册
message PhoneRegisterReq {
    string request_id = 1;
    string phone_number = 2;
    string verify_code_id = 3;
    string verify_code = 4;
}
message PhoneRegisterRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}
//-----
//手机号登录
message PhoneLoginReq {
    string request_id = 1;
    string phone_number = 2;
    string verify_code_id = 3;
    string verify_code = 4;
}
message PhoneLoginRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    string login_session_id = 4;
}
//个人信息获取-这个只用于获取当前登录用户的信息
// 客户端传递的时候只需要填充 session_id 即可
//其他个人/好友信息的获取在好友操作中完成
message GetUserInfoReq {
    string request_id = 1;
    optional string user_id = 2;
```

```
        optional string session_id = 3;
    }
message GetUserInfoRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    UserInfo user_info = 4;
}
//内部接口
message GetMultiUserInfoReq {
    string request_id = 1;
    repeated string users_id = 2;
}
message GetMultiUserInfoRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    map<string, UserInfo>users_info = 4;
}
//-----
//用户头像修改
message SetUserAvatarReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    bytes avatar = 4;
}
message SetUserAvatarRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}
//-----
//用户昵称修改
message SetUserNicknameReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    string nickname = 4;
}
message SetUserNicknameRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}
```

```
}

//-----
//用户签名修改
message SetUserDescriptionReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    string description = 4;
}
message SetUserDescriptionRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}

//-----
//用户手机修改
message SetUserPhoneNumberReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    string phone_number = 4;
    string phone_verify_code_id = 5;
    string phone_verify_code = 6;
}
message SetUserPhoneNumberRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}

service UserService {
    rpc UserRegister(UserRegisterReq) returns (UserRegisterRsp);
    rpc UserLogin(UserLoginReq) returns (UserLoginRsp);
    rpc GetPhoneVerifyCode(PhoneVerifyCodeReq) returns
(PhoneVerifyCodeRsp);
    rpc PhoneRegister(PhoneRegisterReq) returns
(PhoneRegisterRsp);
    rpc PhoneLogin(PhoneLoginReq) returns (PhoneLoginRsp);
    rpc GetUserInfo(GetUserInfoReq) returns (GetUserInfoRsp);
    rpc GetMultiUserInfo(GetMultiUserInfoReq) returns
(GetMultiUserInfoRsp);
    rpc SetUserAvatar(SetUserAvatarReq) returns
(SetUserAvatarRsp);
    rpc SetUserNickname(SetUserNicknameReq) returns
```

```
(SetUserNicknameRsp);
    rpc SetUserDescription(SetUserDescriptionReq) returns
(SetUserDescriptionRsp);
    rpc SetUserPhoneNumber(SetUserPhoneNumberReq) returns
(SetUserPhoneNumberRsp);
}
```

## 好友管理子服务

ProtoBuf

```
/*
    好友操作服务器的子服务注册信息: /service/friend/instance_id
        服务名称: /service/friend
        实例 ID: instance_id      每个能够提供用户操作服务的子服务器唯一 ID
    当服务发现的时候, 通过 /service/friend 进行服务发现, 就可以发现所有的能够提供用户操作的实例信息了
*/
syntax = "proto3";
package bite_im;
import "base.proto";

option cc_generic_services = true;

//-----
//好友列表获取
message GetFriendListReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
}
message GetFriendListRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated UserInfo friend_list = 4;
}

//-----
//好友删除
message FriendRemoveReq {
```

```
        string request_id = 1;
        optional string user_id = 2;
        optional string session_id = 3;
        string peer_id = 4;
    }
message FriendRemoveRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}
//-----
//添加好友--发送好友申请
message FriendAddReq {
    string request_id = 1;
    optional string session_id = 2;
    optional string user_id = 3;//申请人 id
    string respondent_id = 4;//被申请人 id
}
message FriendAddRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    string notify_event_id = 4;//通知事件 id
}
//-----
//好友申请的处理
message FriendAddProcessReq {
    string request_id = 1;
    string notify_event_id = 2;//通知事件 id
    bool agree = 3;//是否同意好友申请
    string apply_user_id = 4; //申请人的用户 id
    optional string session_id = 5;
    optional string user_id = 6;
}
// ++++++
message FriendAddProcessRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    // 同意后会创建会话，向网关返回会话信息，用于通知双方会话的建立，这个字段客户端不需要关注
    optional string new_session_id = 4;
}
```

```
//-----
//获取待处理的，申请自己好友的信息列表
message GetPendingFriendEventListReq {
    string request_id = 1;
    optional string session_id = 2;
    optional string user_id = 3;
}

message FriendEvent {
    string event_id = 1;
    UserInfo sender = 3;
}
message GetPendingFriendEventListRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated FriendEvent event = 4;
}

//-----
//好友搜索
message FriendSearchReq {
    string request_id = 1;
    string search_key = 2;//就是名称模糊匹配关键字
    optional string session_id = 3;
    optional string user_id = 4;
}
message FriendSearchRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated UserInfo user_info = 4;
}

//-----
//会话列表获取
message GetChatSessionListReq {
    string request_id = 1;
    optional string session_id = 2;
    optional string user_id = 3;
}
message GetChatSessionListRsp {
    string request_id = 1;
    bool success = 2;
```

```
        string errmsg = 3;
        repeated ChatSessionInfo chat_session_info_list = 4;
    }
//-----
//创建会话
message ChatSessionCreateReq {
    string request_id = 1;
    optional string session_id = 2;
    optional string user_id = 3;
    string chat_session_name = 4;
    //需要注意的是，这个列表中也必须包含创建者自己的用户 ID
    repeated string member_id_list = 5;
}
message ChatSessionCreateRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    //这个字段属于后台之间的数据，给前端回复的时候不需要这个字段，会话信息通过通知进行发送
    optional ChatSessionInfo chat_session_info = 4;
}
//-----
//获取会话成员列表
message GetChatSessionMemberReq {
    string request_id = 1;
    optional string session_id = 2;
    optional string user_id = 3;
    string chat_session_id = 4;
}
message GetChatSessionMemberRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated UserInfo member_info_list = 4;
}

service FriendService {
    rpc GetFriendList(GetFriendListReq) returns
(GetFriendListRsp);
    rpc FriendRemove(FriendRemoveReq) returns (FriendRemoveRsp);
    rpc FriendAdd(FriendAddReq) returns (FriendAddRsp);
    rpc FriendAddProcess(FriendAddProcessReq) returns
(FriendAddProcessRsp);
    rpc FriendSearch(FriendSearchReq) returns (FriendSearchRsp);
```

```
    rpc GetChatSessionList(GetChatSessionListReq) returns
    (GetChatSessionListRsp);
    rpc ChatSessionCreate(ChatSessionCreateReq) returns
    (ChatSessionCreateRsp);
    rpc GetChatSessionMember(GetChatSessionMemberReq) returns
    (GetChatSessionMemberRsp);
    rpc GetPendingFriendEventList(GetPendingFriendEventListReq)
returns (GetPendingFriendEventListRsp);
}
```

## 文件管理子服务

ProtoBuf

```
/*
    文件操作服务器的子服务注册信息: /service/file/instance_id
    服务名称: /service/file
    实例 ID: instance_id      每个能够提供用户操作服务的子服务器唯一 ID
当服务发现的时候, 通过 /service/file 进行服务发现, 就可以发现所有的能够提供用户操作的实例信息了
*/
syntax = "proto3";
package bite_im;
import "base.proto";

option cc_generic_services = true;

message GetSingleFileReq {
    string request_id = 1;
    string file_id = 2;
    optional string user_id = 3;
    optional string session_id = 4;
}

message GetSingleFileRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    FileDownloadData file_data = 4;
}

message GetMultiFileReq {
    string request_id = 1;
    optional string user_id = 2;
}
```

```
        optional string session_id = 3;
        repeated string file_id_list = 4;
    }
message GetMultiFileRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    map<string, FileDownloadData> file_data = 4;
}

message PutSingleFileReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    FileUploadData file_data = 4;
}
message PutSingleFileRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    FileMessageInfo file_info = 4;
}

message PutMultiFileReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    repeated FileUploadData file_data = 4;
}
message PutMultiFileRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated FileMessageInfo file_info = 4;
}

service FileService {
    rpc GetSingleFile(GetSingleFileReq) returns
(GetSingleFileRsp);
    rpc GetMultiFile(GetMultiFileReq) returns (GetMultiFileRsp);
    rpc PutSingleFile(PutSingleFileReq) returns
(PutSingleFileRsp);
    rpc PutMultiFile(PutMultiFileReq) returns (PutMultiFileRsp);
}
```

## 消息管理子服务

```
Protobuf
/*
    消息存储服务器的子服务注册信息:
/service/message_storage/instance_id
        服务名称: /service/message_storage
        实例 ID: instance_id      每个能够提供用户操作服务的子服务器唯一 ID
当服务发现的时候, 通过 /service/message_storage 进行服务发现, 就可以发现所有的能够提供用户操作的实例信息了
*/
syntax = "proto3";
package bite_im;
import "base.proto";

option cc_generic_services = true;

message GetHistoryMsgReq {
    string request_id = 1;
    string chat_session_id = 2;
    int64 start_time = 3;
    int64 over_time = 4;
    optional string user_id = 5;
    optional string session_id = 6;
}
message GetHistoryMsgRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated MessageInfo msg_list = 4;
}

message GetRecentMsgReq {
    string request_id = 1;
    string chat_session_id = 2;
    int64 msg_count = 3;
    optional int64 cur_time = 4; //用于扩展获取指定时间前的 n 条消息
    optional string user_id = 5;
    optional string session_id = 6;
}
```

```

}

message GetRecentMsgRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated MessageInfo msg_list = 4;
}

message MsgSearchReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    string chat_session_id = 4;
    string search_key = 5;
}

message MsgSearchRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    repeated MessageInfo msg_list = 4;
}

service MsgStorageService {
    rpc GetHistoryMsg(GetHistoryMsgReq) returns
(GetHistoryMsgRsp);
    rpc GetRecentMsg(GetRecentMsgReq) returns (GetRecentMsgRsp);
    rpc MsgSearch(MsgSearchReq) returns (MsgSearchRsp);
}

```

## 转发管理子服务

```

ProtoBuf
/*
消息转发服务器的子服务注册信息:
/service/message_transmit/instance_id
    服务名称: /service/message_transmit
    实例 ID: instance_id      每个能够提供用户操作服务的子服务器唯一 ID
当服务发现的时候, 通过 /service/message_transmit 进行服务发现, 就可以发现所有的能够提供用户操作的实例信息了
*/

```

```
//消息转发服务器接口
syntax = "proto3";
package bite_im;
import "base.proto";

option cc_generic_services = true;

//这个用于和网关进行通信
message NewMessageReq {
    string request_id = 1;
    optional string user_id = 2;
    optional string session_id = 3;
    string chat_session_id = 4;
    MessageContent message = 5;
}

message NewMessageRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
}

//这个用于内部的通信,生成完整的消息信息,并获取消息的转发人员列表
message GetTransmitTargetRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    MessageInfo message = 4;
    repeated string target_id_list = 5;
}

service MsgTransmitService {
    rpc GetTransmitTarget(NewMessageReq) returns
(GetTransmitTargetRsp);
}
```

## 语音转换子服务

```
ProtoBuf
/*
    语音识别服务器的子服务注册信息: /service/speech/instance_id
    服务名称: /service/speech
    实例 ID: instance_id      每个能够提供用户操作服务的子服务器唯
```

## 一 ID

当服务发现的时候，通过 `/service/speech` 进行服务发现，就可以发现所有的能够提供用户操作的实例信息了

```
/*
syntax = "proto3";
package bite_im;

option cc_generic_services = true;

message SpeechRecognitionReq {
    string request_id = 1;
    bytes speech_content = 2;
    optional string user_id = 3;
    optional string session_id = 4;
}

message SpeechRecognitionRsp {
    string request_id = 1;
    bool success = 2;
    string errmsg = 3;
    string recognition_result = 4;
}

service SpeechService {
    rpc SpeechRecognition(SpeechRecognitionReq) returns
    (SpeechRecognitionRsp);
}
```