

2-2-IM登录服务器和消息服务器设计

1 数据库说明

IMAdmin

功能

建表语句

字段说明

IMAudio

功能

建表语句

字段说明

IMDepart

功能

建表语句

字段说明

IMDiscovery

功能

建表语句

字段说明

IMGroup

功能

建表语句

字段说明

IMGroupMember

功能

建表语句

字段说明

IMGroupMessage_(x)

功能

建表语句

字段说明

IMMessage_(x)

功能

建表语句

字段说明

IMRecentSession

功能

建表语句

字段说明

IMRelationShip

功能

建表语句

字段说明

IMUser

功能

建表语句

字段说明

2 netlib

3 CImConn

4 login_server响应流程

5 msg_server响应流程

登录流程

单聊文字消息发送流程

登录请求响应过程

获取消息

新登录用户踢掉老用户

零声学院 <https://0voice.ke.qq.com>

讲师 Darren老师 QQ326873713

班主任 柚子老师 QQ2690491738

2022年6月21日

即时通讯总共6次课：

- 1.即时通讯框架分析和部署
- 2.登录服务器和消息服务器设计
- 3.消息服务器和路由服务器设计
- 4.数据库代理服务器设计
- 5.文件服务器和docker部署
- 6.性能测试和k8s上云发布

该节主要内容：

- 通信协议设计
- 数据库基本设计
- 网络模型reactor
- login_server流程分析
- msg_server流程分析

1 数据库说明

IMAdmin

功能

后台管理员表

建表语句

```
SQL | 复制代码

1 CREATE TABLE `IMAdmin` (
2   `id` mediumint(6) unsigned NOT NULL AUTO_INCREMENT,
3   `uname` varchar(40) NOT NULL COMMENT '用户名',
4   `pwd` char(32) NOT NULL COMMENT '密码',
5   `status` tinyint(2) unsigned NOT NULL DEFAULT '0' COMMENT '用户状态 0 :正
   常 1:删除 可扩展',
6   `created` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
7   `updated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '更新时间',
8   PRIMARY KEY (`id`)
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

字段说明

▼

Bash | 复制代码

1	id	管理员id, 自增, 唯一。
2	uname	用户名。
3	pwd	密码, 经过md5加密的密码。
4	status	状态, 0正常, 1删除。
5	created	创建时间。
6	updated	更新时间。

IMAudio

功能

存储语音地址

建表语句

▼

SQL | 复制代码

```
1 CREATE TABLE `IMAudio` (  
2     `id` int(11) NOT NULL AUTO_INCREMENT,  
3     `fromId` int(11) unsigned NOT NULL COMMENT '发送者Id',  
4     `toId` int(11) unsigned NOT NULL COMMENT '接收者Id',  
5     `path` varchar(255) COLLATE utf8mb4_bin DEFAULT '' COMMENT '语音存储的  
地址',  
6     `size` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '文件大小',  
7     `duration` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '语音时长',  
8     `created` int(11) unsigned NOT NULL COMMENT '创建时间',  
9     PRIMARY KEY (`id`),  
10    KEY `idx_fromId_toId` (`fromId`,`toId`)  
11 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

字段说明

1	id	语音id,自增,唯一。
2	fromId	发送语音用户id。
3	toId	接收语音用户id。
4	path	语音存储的路径。
5	size	语音文件大小。
6	duration	语音时长。
7	created	创建时间。

IMDepart

功能

存储部门信息

建表语句

```

1  CREATE TABLE `IMDepart` (
2      `id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '部门id',
3      `departName` varchar(64) COLLATE utf8mb4_bin NOT NULL DEFAULT ''
4      COMMENT '部门名称',
5      `priority` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '显示优先级',
6      `parentId` int(11) unsigned NOT NULL COMMENT '上级部门id',
7      `status` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '状态',
8      `created` int(11) unsigned NOT NULL COMMENT '创建时间',
9      `updated` int(11) unsigned NOT NULL COMMENT '更新时间',
10     PRIMARY KEY (`id`),
11     KEY `idx_departName` (`departName`),
12     KEY `idx_priority_status` (`priority`,`status`)
13 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin

```

字段说明

1	id	部门id, 自增, 唯一。
2	departName	部门名称
3	priority	部门显示优先级, 相同的优先级按照拼音先后显示
4	parentId	父部门id
5	status	状态, 0表示正常
6	created	创建时间
7	updated	更新时间

IMDiscovery

功能

发现配置表

建表语句

```

1 CREATE TABLE `IMDiscovery` (
2   `id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT 'id',
3   `itemName` varchar(64) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT
  '名称',
4   `itemUrl` varchar(64) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT
  'URL',
5   `itemPriority` int(11) unsigned NOT NULL COMMENT '显示优先级',
6   `status` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '状态',
7   `created` int(11) unsigned NOT NULL COMMENT '创建时间',
8   `updated` int(11) unsigned NOT NULL COMMENT '更新时间',
9   PRIMARY KEY (`id`),
10  KEY `idx_itemName` (`itemName`)
11 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;

```

字段说明

1	id	发现项id, 自增, 唯一。
2	itemName	发现项名称。
3	iteamUrl	发现项url连接。
4	iteamPriority	发现项显示优先级。
5	status	状态。
6	created	创建时间。
7	updated	更新时间。

IMGroup

功能

群组表

建表语句

```

1 CREATE TABLE `IMGroup` (
2   `id` int(11) NOT NULL AUTO_INCREMENT,
3   `name` varchar(256) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT
   '群名称',
4   `avatar` varchar(256) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT
   '群头像',
5   `creator` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建者用户id',
6   `type` tinyint(3) unsigned NOT NULL DEFAULT '1' COMMENT '群组类型, 1-固
   定; 2-临时群',
7   `userCnt` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '成员人数',
8   `status` tinyint(3) unsigned NOT NULL DEFAULT '1' COMMENT '是否删除, 0-
   正常, 1-删除',
9   `version` int(11) unsigned NOT NULL DEFAULT '1' COMMENT '群版本号',
10  `lastChated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '最后聊天时
   间',
11  `created` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
12  `updated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '更新时间',
13  PRIMARY KEY (`id`),
14  KEY `idx_name` (`name` (191)),
15  KEY `idx_creator` (`creator`),
16 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin COMMENT='IM群
   信息'
```

字段说明

1	id	群组id, 自增, 唯一。
2	name	群组名称。
3	avatar	群组头像, 目前没用, 群组头像是客户端自己合成的。
4	creator	群主。
5	type	群类型, 是临时群, 还是固定群。
6	userCnt	群成员数目。
7	status	状态。
8	version	群信息版本。
9	lastChated	最后聊天时间。
10	created	创建时间。
11	updated	最后更新时间。

IMGroupMember

功能

群成员表

建表语句

```

1  CREATE TABLE `IMGroupMember` (
2      `id` int(11) NOT NULL AUTO_INCREMENT,
3      `groupId` int(11) unsigned NOT NULL COMMENT '群Id',
4      `userId` int(11) unsigned NOT NULL COMMENT '用户id',
5      `status` tinyint(4) unsigned NOT NULL DEFAULT '1' COMMENT '是否退出群,
6      0-正常, 1-已退出',
7      `created` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
8      `updated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '更新时间',
9      PRIMARY KEY (`id`),
10     KEY `idx_groupId_userId_status` (`groupId`,`userId`,`status`),
11     KEY `idx_userId_status_updated` (`userId`,`status`,`updated`),
12     KEY `idx_groupId_updated` (`groupId`,`updated`)
13 ) ENGINE=InnoDB AUTO_INCREMENT=68 DEFAULT CHARSET=utf8 COMMENT='用户和群的关系表'
```

字段说明

1	id	自增，唯一。
2	groupId	群组id。
3	userId	用户id。
4	status	状态。
5	created	创建时间。
6	updated	更新时间。

IMGroupMessage_(x)

功能

群消息表,x代表第几张表，目前做了分表有8张:0-7.消息具体在哪张表中，是
groupId%IMGroupMessage表的数目

建表语句

```

1 CREATE TABLE `IMGroupMessage_(x)` (
2   `id` int(11) NOT NULL AUTO_INCREMENT,
3   `groupId` int(11) unsigned NOT NULL COMMENT '用户的关系id',
4   `userId` int(11) unsigned NOT NULL COMMENT '发送用户的id',
5   `msgId` int(11) unsigned NOT NULL COMMENT '消息ID',
6   `content` varchar(4096) COLLATE utf8mb4_bin NOT NULL DEFAULT ''
  COMMENT '消息内容',
7   `type` tinyint(3) unsigned NOT NULL DEFAULT '2' COMMENT '群消息类型,101
  为群语音,2为文本',
8   `status` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '消息状态',
9   `created` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
10  `updated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '更新时间',
11  PRIMARY KEY (`id`),
12  KEY `idx_groupId_status_created` (`groupId`,`status`,`created`),
13  KEY `idx_groupId_msgId_status_created`
  (`groupId`,`msgId`,`status`,`created`)
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin COMMENT='IM群
  消息表'

```

字段说明

1	id	自增，唯一。
2	groupId	收消息的群组id。
3	userId	发消息的用户id。
4	msgId	消息id，每个群组唯一。
5	content	消息内容，如果是语音消息则存储语音id。
6	type	消息类型，文本or语音。
7	status	状态。
8	created	创建时间。
9	updated	更新时间。

IMMessage_(x)

功能

单聊消息表，x代表第几张表，目前做了分表有8张:0-7.具体在那张表，是relateId%IMMessage表数目。

建表语句

```

1  CREATE TABLE `IMMessage_0` (
2      `id` int(11) NOT NULL AUTO_INCREMENT,
3      `relateId` int(11) unsigned NOT NULL COMMENT '用户的关系id',
4      `fromId` int(11) unsigned NOT NULL COMMENT '发送用户的id',
5      `toId` int(11) unsigned NOT NULL COMMENT '接收用户的id',
6      `msgId` int(11) unsigned NOT NULL COMMENT '消息ID',
7      `content` varchar(4096) COLLATE utf8mb4_bin DEFAULT '' COMMENT '消息内
   容',
8      `type` tinyint(2) unsigned NOT NULL DEFAULT '1' COMMENT '消息类型',
9      `status` tinyint(1) unsigned NOT NULL DEFAULT '0' COMMENT '0正常 1被删
   除',
10     `created` int(11) unsigned NOT NULL COMMENT '创建时间',
11     `updated` int(11) unsigned NOT NULL COMMENT '更新时间',          PRIMARY
   KEY (`id`),
12     KEY `idx_relateId_status_created` (`relateId`,`status`,`created`),
13     KEY `idx_relateId_status_msgId_created`
   (`relateId`,`status`,`msgId`,`created`),
14     KEY `idx_fromId_toId_created` (`fromId`,`toId`,`status`)
15 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin

```

字段说明

1	id	自增，唯一
2	relateId	用户与用户关系id，在IMrelationShop中。
3	fromId	发送消息用户id。
4	toId	接收消息用户id。
5	msgId	消息id，每个relation 唯一。
6	content	消息内容，如果是语音消息则存储语音id。
7	type	类型，文本or语音。
8	status	状态。
9	created	创建时间。
10	updated	更新时间。

IMRecentSession

功能

最近联系人(会话)表。

建表语句

```

1  CREATE TABLE `IMRecentSession` (
2      `id` int(11) NOT NULL AUTO_INCREMENT,
3      `userId` int(11) unsigned NOT NULL COMMENT '用户id',
4      `peerId` int(11) unsigned NOT NULL COMMENT '对方id',
5      `type` tinyint(1) unsigned DEFAULT '0' COMMENT '类型, 1-用户,2-群组',
6      `status` tinyint(1) unsigned DEFAULT '0' COMMENT '用户:0-正常, 1-用户A
删除,群组:0-正常, 1-被删除',
7      `created` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',
8      `updated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '更新时间',
9      PRIMARY KEY (`id`),
10     KEY `idx_userId_peerId_status_updated`
      (`userId`,`peerId`,`status`,`updated`),
11     KEY `idx_userId_peerId_type` (`userId`,`peerId`,`type`)
12 ) ENGINE=InnoDB DEFAULT CHARSET=utf8

```

字段说明

		Plain Text	复制代码
1	id	会话id, 自增, 唯一。	
2	userId	用户id。	
3	peerId	对方id。	
4	type	类型, 群组or用户。	
5	status	状态。	
6	created	创建时间。	
7	updated	更新时间。	

IMRelationship

功能

用户关系表, 标识两个用户之间的唯一关系id, 用于消息分表。relationId % 消息表数目。

建表语句

		SQL	复制代码
1	CREATE TABLE `IMRelationship` (
2	`id` int(11) NOT NULL AUTO_INCREMENT,		
3	`smallId` int(11) unsigned NOT NULL COMMENT '用户A的id',		
4	`bigId` int(11) unsigned NOT NULL COMMENT '用户B的id',		
5	`status` tinyint(1) unsigned DEFAULT '0' COMMENT '用户:0-正常, 1-用户A删除, 群组:0-正常, 1-被删除',		
6	`created` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '创建时间',		
7	`updated` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '更新时间',		
8	PRIMARY KEY (`id`),		
9	KEY `idx_smallId_bigId_status_updated`		
	(`smallId`, `bigId`, `status`, `updated`)		
10) ENGINE=InnoDB DEFAULT CHARSET=utf8		

字段说明

		Plain Text	复制代码
1	id	关系id, 自增, 唯一。	
2	smallId	两个用户中id小的。	
3	bigId	两个用户中id大的。	
4	status	状态。	
5	created	创建时间。	
6	updated	更新时间。	

IMUser

功能

用户表

建表语句

Plain Text | 复制代码

```
1 CREATE TABLE `IMUser` (  
2     `id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '用户id',  
3     `sex` tinyint(1) unsigned NOT NULL DEFAULT '0' COMMENT '1男2女0未知',  
4     `name` varchar(32) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT  
5     '用户名',  
6     `domain` varchar(32) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT  
7     '拼音',  
8     `nick` varchar(32) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT  
9     '花名,绰号等',  
10    `password` varchar(32) COLLATE utf8mb4_bin NOT NULL DEFAULT ''  
11    COMMENT '密码',  
12    `salt` varchar(4) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT '混  
13    淆码',  
14    `phone` varchar(11) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT  
15    '手机号码',  
16    `email` varchar(64) COLLATE utf8mb4_bin NOT NULL DEFAULT '' COMMENT  
17    'email',  
18    `avatar` varchar(255) COLLATE utf8mb4_bin DEFAULT '' COMMENT '自定义用  
19    户头像',  
20    `departId` int(11) unsigned NOT NULL COMMENT '所属部门Id',  
21    `status` tinyint(2) unsigned DEFAULT '0' COMMENT '1. 试用期 2. 正式 3.  
22    离职 4. 实习',  
23    `created` int(11) unsigned NOT NULL COMMENT '创建时间',  
24    `updated` int(11) unsigned NOT NULL COMMENT '更新时间',  
25    PRIMARY KEY (`id`),  
26    KEY `idx_domain` (`domain`),  
27    KEY `idx_name` (`name`),  
28    KEY `idx_phone` (`phone`)  
29 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

字段说明

1	id	用户id, 自增, 唯一。
2	sex	性别。
3	name	用户名。
4	domain	拼音。
5	nick	昵称。
6	password	密码, 规则md5(md5(passwd)+salt)。
7	salt	密码混淆。
8	phone	电话号码。
9	email	email。
10	avatar	用户头像。
11	departId	所属部门Id
12	status	状态
13	created	创建时间。
14	updated	更新时间。

2 netlib

网络库netlib

TeamTalk/server/src/base/netlib.h

TeamTalk/server/src/base/netlib.cpp

函数名称	说明	返回值	备注
netlib_init()	初始化网络连接	int	Linux系统无操作，返回NETLIB_OK；
netlib_destroy()	清除网络连接	int	Linux系统无操作，返回NETLIB_OK；
netlib_listen()	监听连接	int	底层实现：CBaseSocket
netlib_connect()	建立连接	net_handle_t	
netlib_send()	发送数据	int	
netlib_recv()	接收数据	int	
netlib_close()	关闭连接	int	
netlib_option()	配置连接信息	int	
netlib_register_timer()	添加定时器	int	
netlib_delete_timer()	删除定时器	int	
netlib_add_loop	添加事件循环	int	
netlib_eventloop()	进入事件循环	void	
netlib_stop_event()	停止事件	void	
netlib_is_running()	判断是否运行	bool	

TeamTalk使用CBaseSocket对socket进行了封装，其中包含了针对这个socket的一些回调操作；在Epoll返回时，针对每一个socket进行检索：

```
CBaseSocket* pSocket = FindBaseSocket(ev_fd);
```

根据之前保存的SocketMap来查询ev_fd对应的CBaseSocket，然后ev_fd就找到了自己对应的回调函数。

3 CImConn

OnTimer函数的工作原理：

- 是否调用OnTimer是由用户自己是否注册定时器决定的。由用户自己注册的回调函数去决定是否在里面处理OnTimer事件。
- 只有你将回调函数注册到定时器列表，当回调函数被触发后，然后你可以在自定义的回调函数里面设置是否要调用OnTimer，继承者可以在OnTimer可以自定义要处理的事务。

比如下列调用栈所示：

▼

C++ | 复制代码

```
1  #0  CMsgConn::OnTimer (this=0x84cca0, curr_tick=1585836238611)
2      at
    /home/ubuntu/0voice/im/0voice_im/server/src/msg_server/MsgConn.cpp:226
3  #1  0x000000000004e382 in msg_conn_timer_callback (callback_data=0x0,
    msg=6 '\006', handle=0, pParam=0x0)
4      at
    /home/ubuntu/0voice/im/0voice_im/server/src/msg_server/MsgConn.cpp:56
5  #2  0x0000000000051690 in CEventDispatch::_CheckTimer (this=0x84a2e0)
6      at
    /home/ubuntu/0voice/im/0voice_im/server/src/base/EventDispatch.cpp:90
7  #3  0x00000000000516e01 in CEventDispatch::StartDispatch (this=0x84a2e0,
    wait_timeout=100)
8      at
    /home/ubuntu/0voice/im/0voice_im/server/src/base/EventDispatch.cpp:404
9  #4  0x000000000005154b2 in netlib_eventloop (wait_timeout=100)
10     at /home/ubuntu/0voice/im/0voice_im/server/src/base/netlib.cpp:160
11  #5  0x000000000004fcf81 in main (argc=1, argv=0x7fffffff628)
12     at
    /home/ubuntu/0voice/im/0voice_im/server/src/msg_server/msg_server.cpp:148
13
```

是在msg_conn_timer_callback回调函数进行socket的检测，而msg_conn_timer_callback需要注册到网络库的定时器列表：netlib_register_timer(msg_conn_timer_callback, NULL, 1000);。

4 login_server响应流程

在<https://www.sojson.com/http/test.html>网站进行http测试请求。

请求url为 http://111.229.231.225:8080/msg_server （具体注意自己服务器的ip地址）

使用post的方式。

返回内容

▼ C++ 复制代码

```
1  {  
2      "backupIP": "111.229.231.225",  
3      "code": 0,  
4      "discovery": "http://127.0.0.1/api/discovery",  
5      "msfsBackup": "http://111.229.231.225:8700/",  
6      "msfsPrior": "http://111.229.231.225:8700/",  
7      "msg": "",  
8      "port": "8000",  
9      "priorIP": "111.229.231.225"  
10 }
```

<https://www.processon.com/embed/5e819e30e4b07ff5bb99bf60>

5 msg_server响应流程

登录流程

打断点

- **CMsgConn::HandlePdu** (msg_server模块, 处理客户端的请求的信息)
- **CMsgConn::_HandleLoginRequest** (msg_server模块, 处理客户端的登录请求信息, CID_LOGIN_REQ_USERLOGIN命令)
- **CImUserManager::AddImUserByLoginName** (msg_server模块, 将登录的user_name封装以user_name(login_name)为key封装成pImUser插入到**m_im_user_map_by_name**)
- **proxy_serv_callback** (db_proxy_server模块) 处理其他server的数据库操作请求, 每个请求new CProxyConn();
- **DB_PROXY::doLogin** (db_proxy_server模块) msg_server 往dbproxy_server发登录验证请求 (CID_OTHER_VALIDATE_REQ), 在db_proxy_server的**doLogin**进行处理, 主要流程:
 - 先检测是否密码经常错误
 - 调用CInterLoginStrategy::doLogin 验证账号和密码是否匹配
 - 如果账号密码匹配则返回正确, 如果错误则返回错误, 使用**CID_OTHER_VALIDATE_RSP**命令;
- **CProxyConn::AddResponsePdu** (db_proxy_server模块) 处理完其他server的请求后需要回发信息, 但不会直接调用send进行发送, 而是封装成ResponsePdu_t插入到s_response_pdu_list队列, 由另外的线程取出来进行发送。
- **CProxyConn::SendResponsePduList** (db_proxy_server模块) 负责回发 ResponsePdu_t。打个断点也方便分析是loop循环由谁发起, 实际上这里的loop和epoll所在的loop同属于一个大loop。init_proxy_conn时调用netlib_add_loop(**proxy_loop_callback**, NULL);进行注册loop, 而**proxy_loop_callback**实质是调用了CProxyConn::SendResponsePduList()
- **CDBServConn::HandlePdu** (msg_server模块, 处理dbproxy回发的数据), 根据**CID_OTHER_VALIDATE_RSP**找到对应的处理函数
- **CDBServConn::_HandleValidateResponse** (msg_server模块), 使用**CID_LOGIN_RES_USERLOGIN**命令回应客户端。
- **CRouteServConn::_HandlePCLoginStatusNotify** (msg_server模块) 客户端回复CID_OTHER_LOGIN_STATUS_NOTIFY, 通知其他端目前自己的登录情况。

单聊文字消息发送流程

每个CImUser对应一个登陆用户, CMsgConn对应一个端的登录, CImUser和CMsgConn是1:n的对应关系。

- **CMsgConn::HandlePdu** (msg_server模块, 处理客户端的请求的信息)
- **CMsgConn::_HandleClientMsgData** (msg_server模块, 处理客户端的消息发送, **CID_MSG_DATA**命令), 重新拼装pdu, 主要是增加handle作为attach数据, 然后发送给

db_proxy_server

- DB_PROXY::sendMessage (db_proxy_server模块)，
 - 获取消息FromId, Told, MsgType等，并先验证消息类型MsgType是否有效（这里主要先分析单聊的情况）
 - nSessionId 服务器分配会话id：通过CSessionModel::getSessionId查询两个人直接的聊天是否已经建立最近会话记录(从IMRecentSession表)，如果没有记录则调用CSessionModel::addSession创建
 - nPeerSessionId 服务器分配对端会话id：通过CSessionModel::getSessionId查询两个人直接的聊天是否已经建立最近会话记录(从IMRecentSession表)，如果没有记录则调用CSessionModel::addSession创建，**需要注意的是nPeerSessionId和nSessionId的FromId和Told是相反的。**
 - nRelateId：获取通话人之间的关系id，如果两者之前没有关系则调用CRelationModel::getRelationId进行添加（操作IMRelationship表）
 - nMsgId 服务器分配消息id，CMessageModel::getMsgId根据nRelateId映射进行获取，（FromId和Told相互之间的nRelateId是唯一的，不分方向性，进而保证相互之间发送消息时消息的顺序性），msgId存储在redis中，通过key为"msg_id_" + int2string(nRelateId)进行获取，每次进行+1的递增操作
 - CMessageModel::sendMessage 将消息插入到数据库（操作IMMessage_x表），发送消息和要读取消息之间存储的是同一条消息：nRelateId, nFromId, nTold, nMsgType, nCreateTime, nMsgId, msg_data
 - 然后封装响应pPduResp，最重要的是附带nMsgId和msg回发给msg_server，使用CID_MSG_DATA命令。一样是以AddResponsePdu插入队列，然后SendResponsePduList进行回发的套路。
- CDBServConn::HandlePdu (msg_server模块，处理dbproxy回发的数据)，根据CID_MSG_DATA找到对应的处理函数
- CDBServConn::_HandleMsgData (msg_server模块)
 - 根据attach_data的handle查找到对应的socket通路，使用CID_MSG_DATA_ACK告知客户端消息已经发送到服务器。
 - get_route_serv_conn，将pdu发送给route_server，CRouteConn::HandlePdu进行响应，然后调用CRouteConn::_BroadcastMsg转发给其他msg_server。
 - CImUser::BroadcastClientMsgData：广播给消息发起者，对于发起者不需要广播给自己的，只需要广播给其他端（比如多端登录时，PC端发送的数据，则广播给Android、IOS端，不用再广播给PC端），并将该消息插入到m_send_msg_list
 - CImUser::BroadcastClientMsgData：广播给消息接收者，有几端登录同一个账号就广播给几个端，并将该消息插入到m_send_msg_list
 - CID_OTHER_GET_DEVICE_TOKEN_REQ：消息推送请求，主要是针对Android和IOS，此时由新发回给db_proxy_server，在setDevicesToken进行响应，我们这里不继续关注它。
- 接收的客户端写入消息的回应
- 作为接收者的客户端读取消息后回应CMsgConn::_HandleClientMsgReadAck (msg_server模块)，使用CID_MSG_READ_ACK命令。

- 使用CID_MSG_READ_NOTIFY通知其他多端登录的客户端，已经有客户端读取了该消息。
- 将该msg从m_send_msg_list移除。
- 如果客户端没有回应，则CMsgConn::OnTimer定时器定时check消息是否已经正常发送给客户端，没有收到响应则认为g_down_msg_miss_cnt++，该详细下行失败。

登录请求响应过程

这里略过登录流程，即是略过CID_LOGIN_REQ_USERLOGIN到CID_OTHER_LOGIN_STATUS_NOTIFY。

更新部门列表的部门信息（时间匹配是否要拉取最新的where updated > nLastTime）

- CMsgConn::_HandleClientDepartmentRequest：（msg_server模块）
CID_BUDDY_LIST_DEPARTMENT_REQUEST，拉取部门信息
- 发送给db_proxy_server，DB_PROXY::getChgedDepart进行响应
 - CDepartModel::getChgedDeptId，IMDepart存储的是部门id信息，通过对比本地客户端更新的时间和服务器更新的时间进行对比，或者到已经更新了的部门ID信息
 - CDepartModel::getDepts，还是操作IMDepart，此时是读取出新更新部门的所有信息
 - 将更新的部门信息封装成pdu回发给msg_server，使用
CID_BUDDY_LIST_DEPARTMENT_RESPONSE命令
- CDBServConn::_HandleDepartmentResponse（msg_server模块）响应，然后也以
CID_BUDDY_LIST_DEPARTMENT_RESPONSE回发给客户端。

更新用户列表的用户信息

- CMsgConn::_HandleClientAllUserRequest（msg_server模块）用户信息请求，使用
CID_BUDDY_LIST_ALL_USER_REQUEST命令，并转发给db_proxy_server
- DB_PROXY::getChangedUser（db_proxy_server模块）（会把IM库里面所有人的用户信息都会回发，假如公司有个1万人，第一次安装的时候直接拉取一万人的信息）
 - 检测是否有用户信息更新，主要是通过对比客户端本地的最近更新时间和服务器的最新更新时间
 - 如果有更新 CUserModel::getChangedId获取更新的用户id
 - 获取有更新的用户信息CUserModel::getUsers
 - 封装到pdu使用CID_BUDDY_LIST_ALL_USER_RESPONSE命令回发给msg_server
- CDBServConn::_HandleAllUserResponse 进行响应，然后回发给客户端

查询用户列表的用户当前的在线状态

- CMsgConn::_HandleClientUsersStatusRequest（msg_server模块），去route_server查询用户列表里面用户的状态，使用CID_BUDDY_LIST_USERS_STATUS_REQUEST命令，并由route_server进行广播
- CRouteConn::_HandleUsersStatusRequest（route_server模块），从route_server的UserInfoMap_t g_user_map;里面获取user的当前状态。
 - 回复给msg_server CID_BUDDY_LIST_USERS_STATUS_RESPONSE

- CRouteServConn::_HandleUsersStatusResponse进行响应，并以CID_BUDDY_LIST_USERS_STATUS_RESPONSE回复客户端。

获取最近联系会话

- CMsgConn::_HandleClientRecentContactSessionRequest，使用CID_BUDDY_LIST_RECENT_CONTACT_SESSION_REQUEST命令，并转发给db_proxy_server
- DB_PROXY::getRecentSession (db_proxy_server模块)进行响应
 - 使用getRecentSession在IMRecentSession查询最近联系人列表信息，以及对应的详细信息，比如最后的消息msgId
 - CID_BUDDY_LIST_RECENT_CONTACT_SESSION_RESPONSE回复给msg_server
- CDBServConn::_HandleRecentSessionResponse (msg_server模块)，回复给客户端。

获取未读消息数量

- CMsgConn::_HandleClientUnreadMsgCntRequest (msg_server模块)，使用CID_MSG_UNREAD_CNT_REQUEST命令，并转发给db_proxy_server
- DB_PROXY::getUnreadMsgCounter (db_proxy_server模块) (根据CID_MSG_UNREAD_CNT_REQUEST命令查找)，未读消息数量包括单聊和群聊消息
 - CMessageModel::getUnreadMsgCount获取单聊未读消息数量
 - 未读消息数量存储在redis，以列表的方式进行存储，列表list包括<session_id,unread_cnt>
 - 调用CMessageModel::getLastMsg读取最新的消息
 - CMessageModel::getUnreadMsgCount获取群聊未读消息数量
 - 封装成pdu用CID_MSG_UNREAD_CNT_RESPONSE命令进行回发
- CDBServConn::_HandleUnreadMsgCountResponse (msg_server模块) 进行响应，然后回发给客户端

获取消息

- CMsgConn::_HandleClientGetMsgListRequest (msg_server模块) 响应客户端的CID_MSG_LIST_REQUEST，并使用CID_MSG_LIST_REQUEST转发给db_proxy_server
- DB_PROXY::getMessage(db_proxy_server模块):
 - 单聊消息使用CMessageModel::getMessage读取消息，查询IMMessage_x表
 - 群聊消息使用CMessageModel::getMessage读取消息，查询IMGroupMessage_x表
- 使用CID_MSG_LIST_RESPONSE发回给msg_server
- CDBServConn::_HandleGetMsgListResponse (msg_server模块)，发回给客户端。

新登录用户踢掉老用户

CDBServConn::_HandleValidateResponse 检测到新登录成功时则踢掉老登录。
使用CID_OTHER_SERVER_KICK_USER命令，通过广播的方式进行

- CRouteConn::HandlePdu (route_server模块) 响应CID_OTHER_SERVER_KICK_USER后原封不动转发到各个msg_server
- CRouteServConn::_HandleKickUser (msg_server模块) 响应CID_OTHER_SERVER_KICK_USER, 如果该user在当前msg_server, 则CImUser::KickOutSameClientType查找是否有重复的登录, 并通过对应的CImConn发送回给客户端。