C/C++Linux服务器开发

高级架构师课程

三年课程沉淀 | 五次精益升级 | 十年行业积累 | 百个实战项目 | 十万内容受众

讲师:darren/326873713

扫一扫 升职加薪

班主任:柚子/2690491738

讲师介绍--专业来自专注和实力

King老师

系统架构师，曾供职著名创业公司系统架构师，微软亚洲研究院、创维集团全球研发中心。国内第一代商业Paas平台开发者。著有多个软件专利，参与多个开源软件维护。在全球化，高可用的物联网云平台架构与智能硬件设计方面有丰富的研发与实战经验。

Darren老师

曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

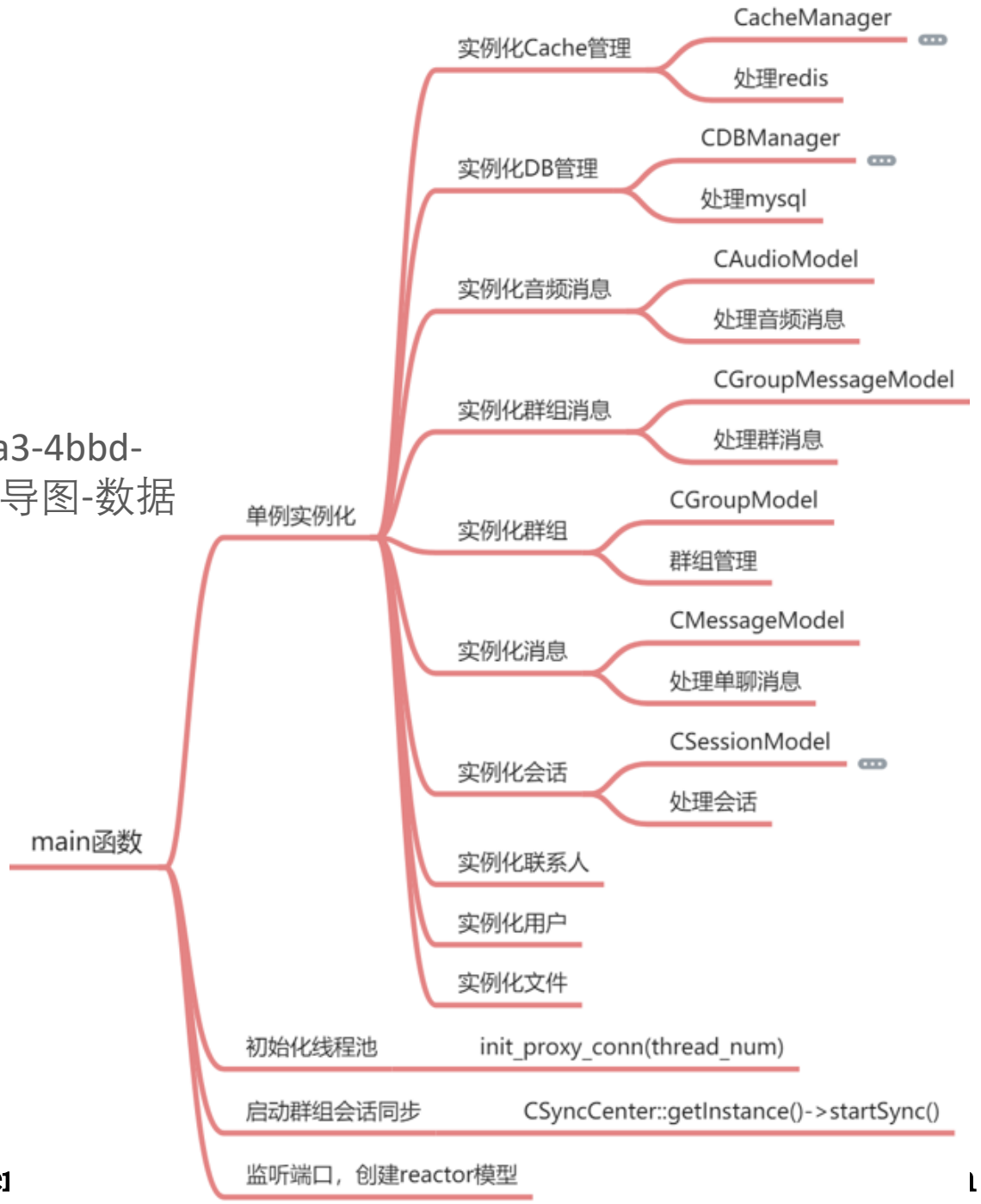
4 数据库代理服务器设计

1 main函数主流程
2 reactor响应流程
3 redis缓存
4 消息计数（单聊和群聊）
5 未读消息机制
6 群成员管理
7 单聊 群聊

1 main函数主流程

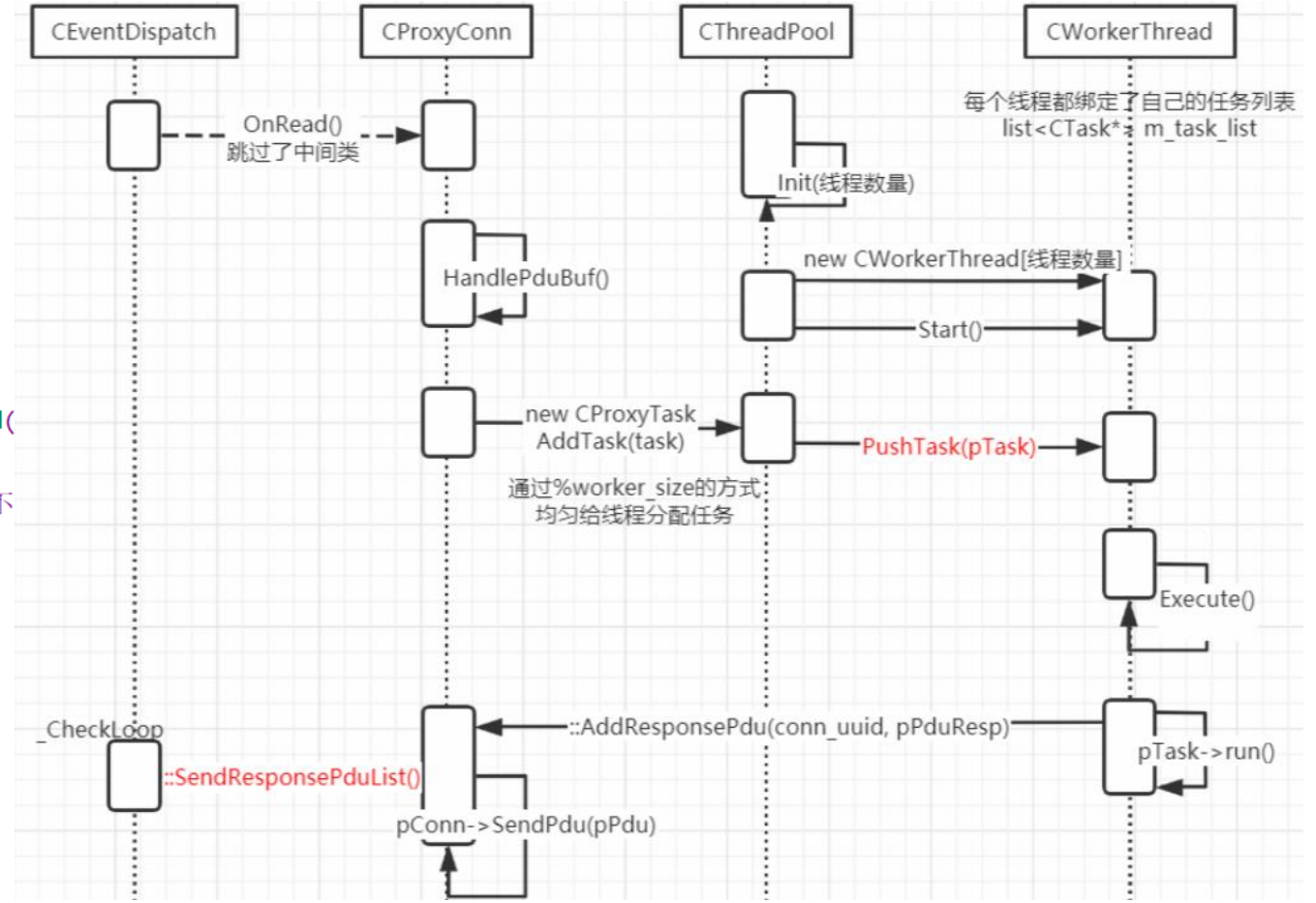
完整导图:

https://www.yuque.com/docs/share/97599556-cca3-4bbd-9199-501568114685?#（密码：mexp） 《4 思维导图-数据库代理服务器》



2 响应流程

```
199: void CProxyConn::HandlePduBuf(uchar_t* pdu_buf, uint32_t pdu_len)
200: {
201:     CImPdu* pPdu = NULL;
202:     pPdu = CImPdu::ReadPdu(pdu_buf, pdu_len);
203:     if (pPdu->GetCommandId() == IM::BaseDefine::CID_OTHER_HEARTBEAT) {
204:         return;
205:     }
206:     // 查找对应的 command id对应的处理函数
207:     pdu_handler_t handler = s_handler_map->GetHandler(pPdu->GetCommandId(
208:
209:     if (handler) {
210:         CTask* pTask = new CProxyTask(m_uuid, handler, pPdu);   // 根据不
211:         g_thread_pool.AddTask(pTask);
212:     } else {
213:         log("no handler for packet type: %d", pPdu->GetCommandId());
214:     }
215: }
```




```
29: void CProxyTask::run()
30: {
31:
32:     if (!m_pPdu) {
33:         // tell CProxyConn to close connection with m_conn_uuid
34:         CProxyConn::AddResponsePdu(m_conn_uuid, NULL);
35:     } else {
36:         if (m_pdu_handler) {
37:             m_pdu_handler(m_pPdu, m_conn_uuid); // 执行
38:         }
39:     }
40: }
```

```
m_handler_map.insert(make_pair(uint32_t(CID_OTHER_VALIDATE_REQ), DB_PROXY::doLogin
m_handler_map.insert(make_pair(uint32_t(CID_LOGIN_REQ_PUSH_SHIELD), DB_PROXY::doPu
```

```
79: int init_proxy_conn(uint32_t thread_num)
80: {
81:     s_handler_map = CHandlerMap::getInstance();
82:     g_thread_pool.Init(thread_num);
83:
84:     netlib_add_loop(proxy_loop_callback, NULL);
85:     //4.1 signal(SIGTERM, sig_handler); 信号设置，让db_proxy_server能够平滑退出
86:     signal(SIGTERM, sig_handler);
87:
88:     return netlib_register_timer(proxy_timer_callback, NULL, 1000);
89: }
90:
```

回发数据包

3 redis缓存

db index	名称	最大连接数 根据实际情况设置	主机	说明
1	unread	16	127.0.0.1:6379	未读消息计数器
2	group_set	16	127.0.0.1:6379	群组设置
3	sync	16	127.0.0.1:6379	同步控制
4	token	16	127.0.0.1:6379	推送相关的token
5	group_member	16	127.0.0.1:6379	群组成员

查看配置文件

select index

```
CacheConn* pCacheConn = pCacheManager->GetCacheConn("unread");
CacheConn* pCacheConn = pCacheManager->GetCacheConn("group_member");
```

```
#configure for unread
CacheInstances=unread,group_set,token,sync,group_member
#未读消息计数器的redis
unread_host=127.0.0.1
unread_port=6379
unread_db=1
unread_maxconncnt=16

#群组设置redis
group_set_host=127.0.0.1
group_set_port=6379
group_set_db=2
group_set_maxconncnt=16
```

4 消息计数（单聊和群聊）

存储在unread连接池所在的数据库

1. 单聊：
key设计： "msg_id_“ + nRelateId
函数： uint32_t CMessageModel::getMsgId(uint32_t nRelateId)

2. 群聊：
key设计： "group_msg_id_" + group_id
函数： uint32_t CGroupMessageModel::getMsgId(uint32_t nGroupId)

Db 1

```
1) "msg_id_1"
2) "group_msg_id_1"
3) "last_update_group"
4) "msg_id_2"
5) "1_1_im_user_group"
6) "unread_2"
7) "1_im_group_msg"
8) "total_user_update"
127.0.0.1:6379[1]>
```

群聊消息ID

单聊消息ID

```
roupId);
 nMsgId = pGroupMsgModel->getMsgId(nToId);
 nMsgId = pGroupMsgModel->getMsgId(nToId);
 nMsgId = pMsgModel->getMsgId(nRelateId);
 nMsgId = pMsgModel->getMsgId(nRelateId);
Id)
```

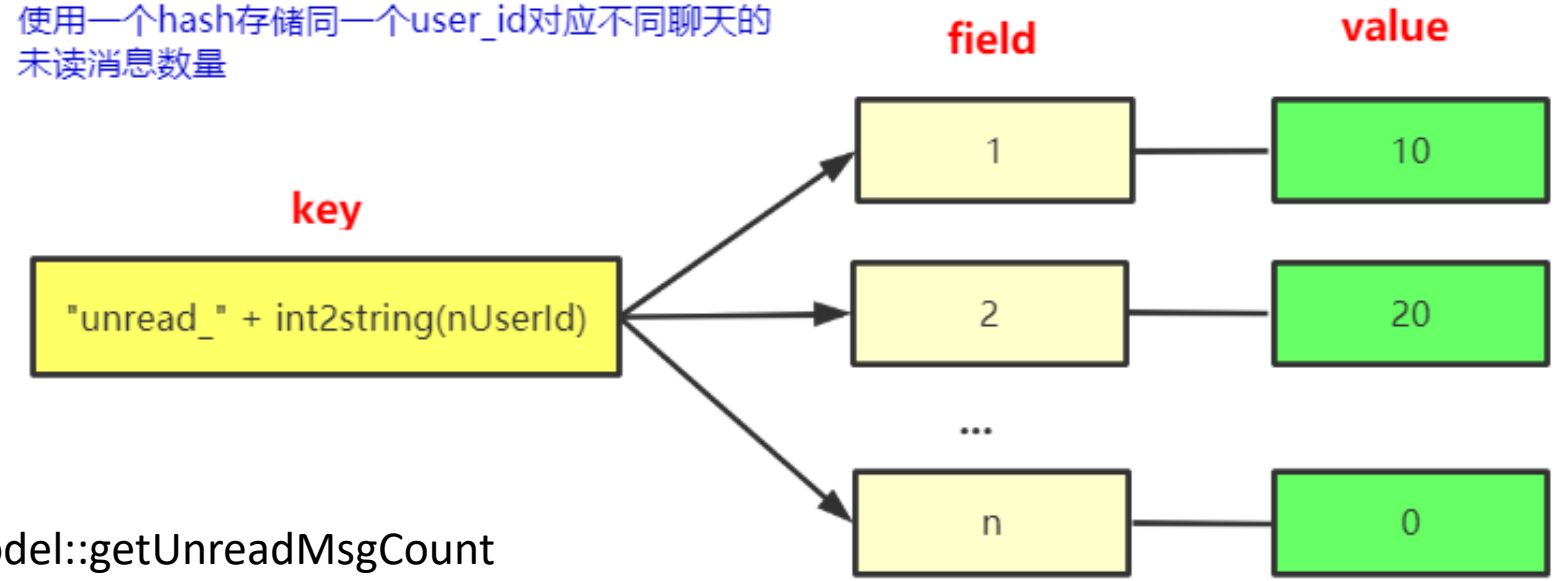
sendMessage 发送消息的时候使用

```
(gdb) bt
#0  CMessageModel::getMsgId (this=0x219a000, nRelateId=2)
    at /home/lqf/im/0voice_im/server/src/db_proxy_server/business/MessageModel.cpp:248
#1  0x0000000000550eb3 in DB_PROXY::sendMessage (pPdu=0x293f1d0, conn_uuid=137)
    at /home/lqf/im/0voice_im/server/src/db_proxy_server/business/MessageContent.cpp:177
#2  0x000000000051f6bf in CProxyTask::run (this=0x293f200)
    at /home/lqf/im/0voice_im/server/src/db_proxy_server/ProxyTask.cpp:37
#3  0x0000000000555674 in CWorkerThread::Execute (this=0x223c588)
    at /home/lqf/im/0voice_im/server/src/base/ThreadPool.cpp:51
#4  0x00000000005555ae in CWorkerThread::StartRoutine (arg=0x223c588)
    at /home/lqf/im/0voice_im/server/src/base/ThreadPool.cpp:27
#5  0x00007fbe6612e6ba in start_thread (arg=0x7fbe55506700) at pthread_create.c:333
#6  0x00007fbe6475051d in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:109
```

5.1 未读消息机制-单聊

未读消息计数
key设计："unread_" + int2string(nUserId)

使用一个hash存储同一个user_id对应不同聊天的未读消息数量



每个filed对应一个联系人的未读消息计数

1 . 读取 CMessageModel::getUnreadMsgCount

2. 写入 CMessageModel::incMsgCount

```
127.0.0.1:6379[1]> KEYS *
1) "msg_id_1"
2) "group_msg_id_1"
3) "last_update_group"
4) "msg_id_2"
5) "1_1_im_user_group"
6) "unread_2"
7) "1_im_group_msg"
8) "total_user_update"
127.0.0.1:6379[1]>
```

未读消息计数

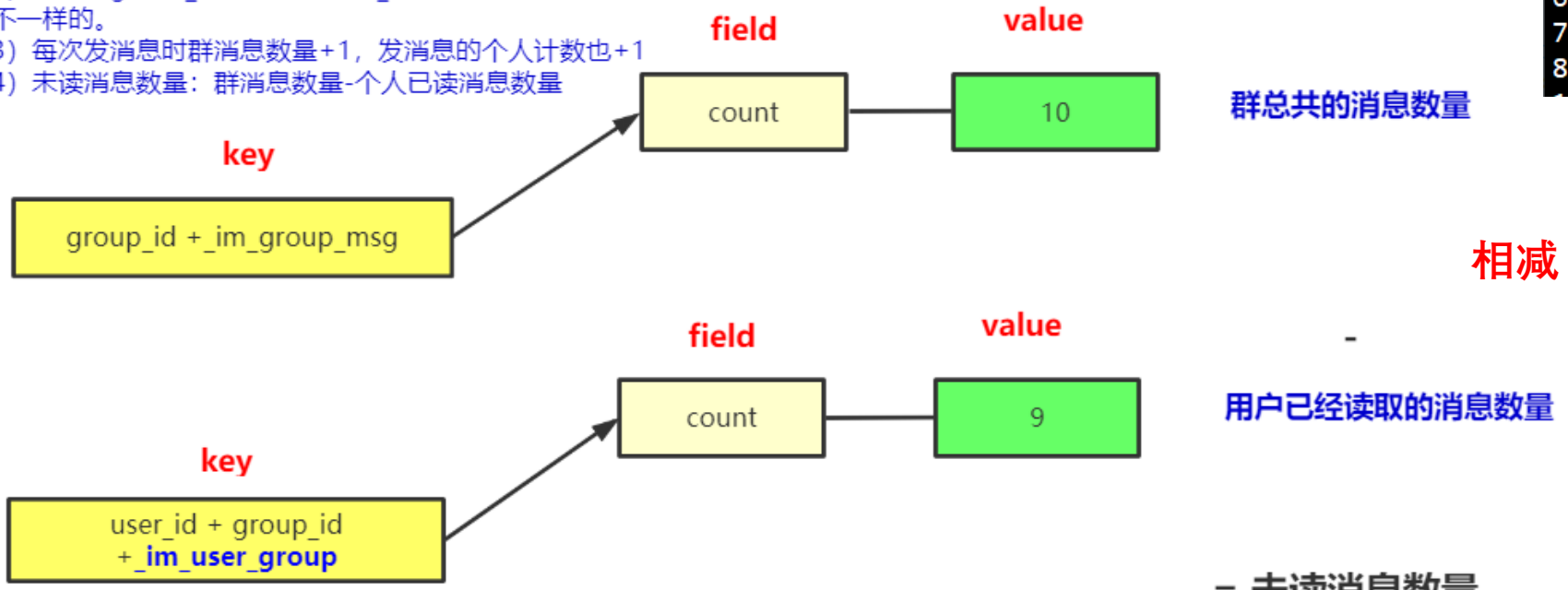
```
127.0.0.1:6379[1]> HGETALL unread_2
1) "1"
2) "6"
3) "3"
4) "1"
127.0.0.1:6379[1]>
```

from id

unread cnt

5.2 未读消息机制–群聊

群未读消息计数：
（1）一个群groud_id对应多个user_id
（2）同一个groud_id，不同的user_id对应的未读消息数量是不一样的。
（3）每次发消息时群消息数量+1，发消息的个人计数也+1
（4）未读消息数量：群消息数量-个人已读消息数量




```
127.0.0.1:6379[1]> KEYS *
1) "msg_id_1"
2) "group_msg_id_1"
3) "last_update_group"
4) "msg_id_2"
5) "1_1_im_user_group"
6) "unread_2"
7) "1_im_group_msg"
8) "total_user_update"
```

user1->group1用户已读计数

id=1的群总共的消息数量

1 . 读取
CGroupMessageModel::getUnreadMsgCount
群总消息数量 – 自己已经读取的消息数量

```
1_1_im_user_group
```

2. 写入 CGroupMessageModel::incMessageCount
增加群消息总共的消息数量

3. 清除未读消息

作为消息发送者，在发送群聊的时候也要把自己的消息计数设置成和群消息数量一样。

5.3 清除未读消息

一对一聊天、群聊天都是调用该函数:
void CUserModel::clearUserCounter(uint32_t nUserId, uint32_t nPeerId,
IM::BaseDefine::SessionType nSessionType)

单聊: 直接删除unread_ + userId的key

群聊: 更新user_id + group_id +_im_user_group对应的value和group_id +_im_group_msg一致

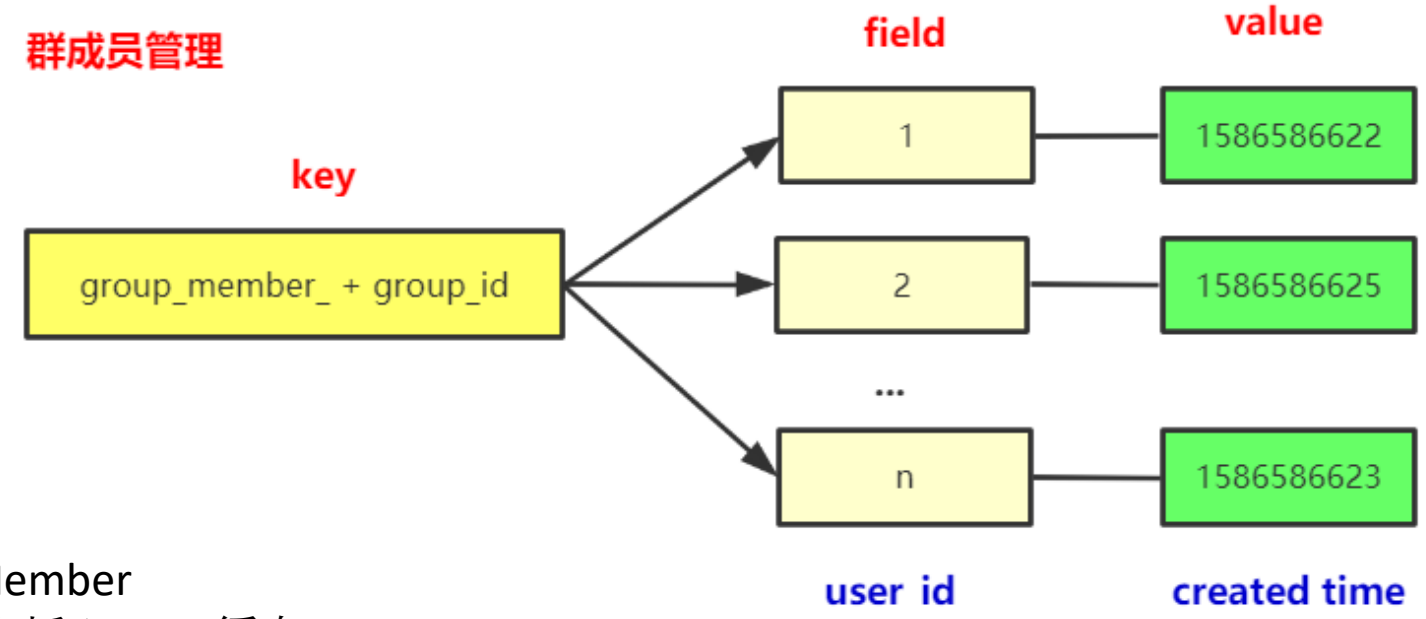
```
// message content
m_handler_map.insert(make_pair(uint32_t(CID_MSG_DATA), DB_PROXY::sendMessage));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_LIST_REQUEST), DB_PROXY::getMessage));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_UNREAD_CNT_REQUEST), DB_PROXY::getUnreadMsgCounter));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_READ_ACK), DB_PROXY::clearUnreadMsgCounter));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_GET_BY_MSG_ID_REQ), DB_PROXY::getMessageById));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_GET_LATEST_MSG_ID_REQ), DB_PROXY::getLatestMsgId));
```

读取消息后应答

6 群成员管理1

群成员管理的redis缓存设计，以hash为存储结构。key使用group_*member* + group_id，hash里面的field使用user_id，value则对应创建时间

```
127.0.0.1:6379[5]> KEYS *
1) "group_member_2"
2) "group_member_1"
```



Db 5

1. 加入成员： insertNewMember
插入mysql数据库的同时也插入redis缓存

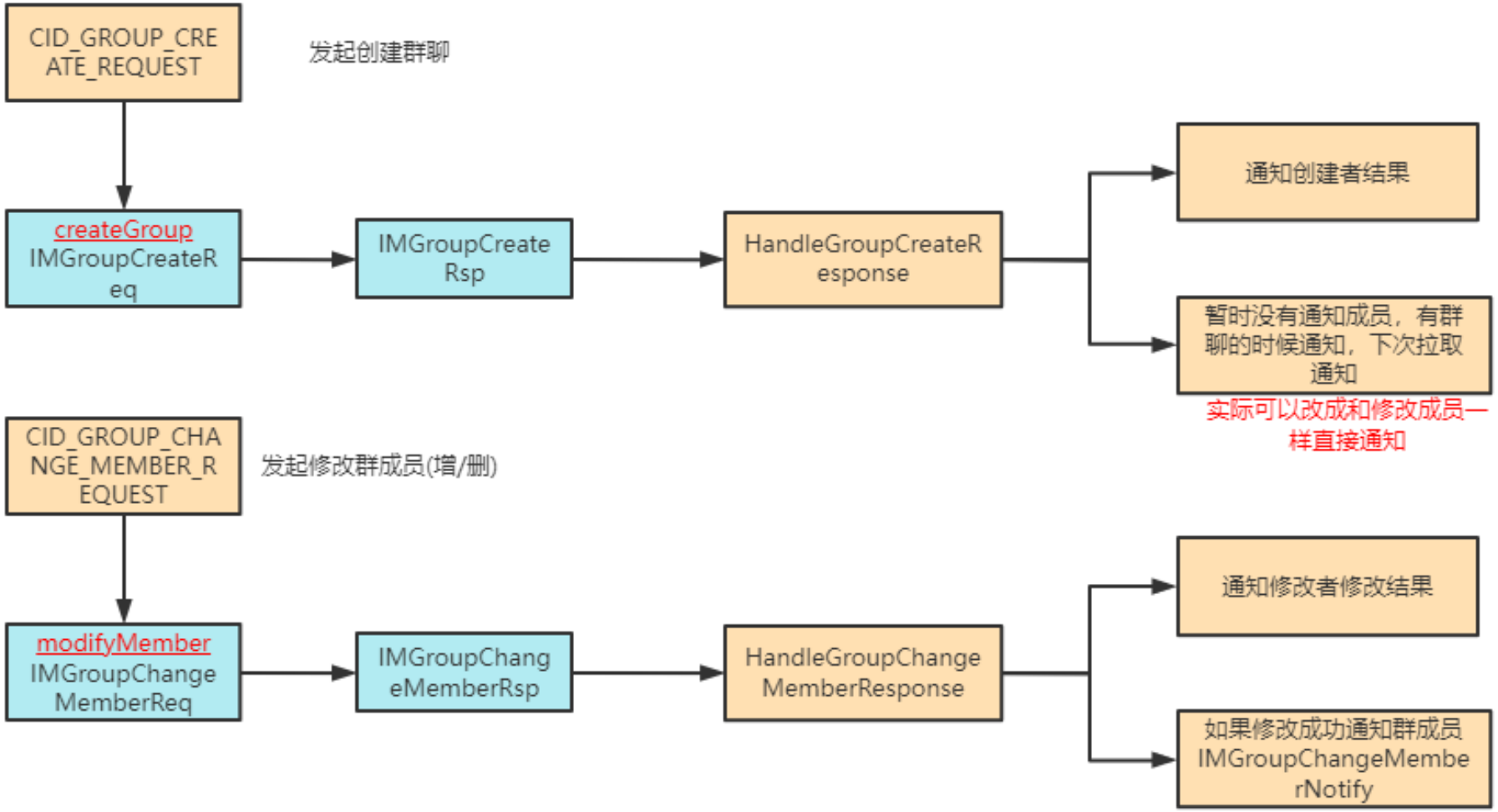
2. 删除成员： removeMember
从mysql数据库删除的同时，也从redis缓存删除

```
127.0.0.1:6379[5]> HGETALL group_member_1
1) "1"
2) "1655990722"
3) "2"
4) "1655990725"
5) "3"
6) "1655990726"
```

user id

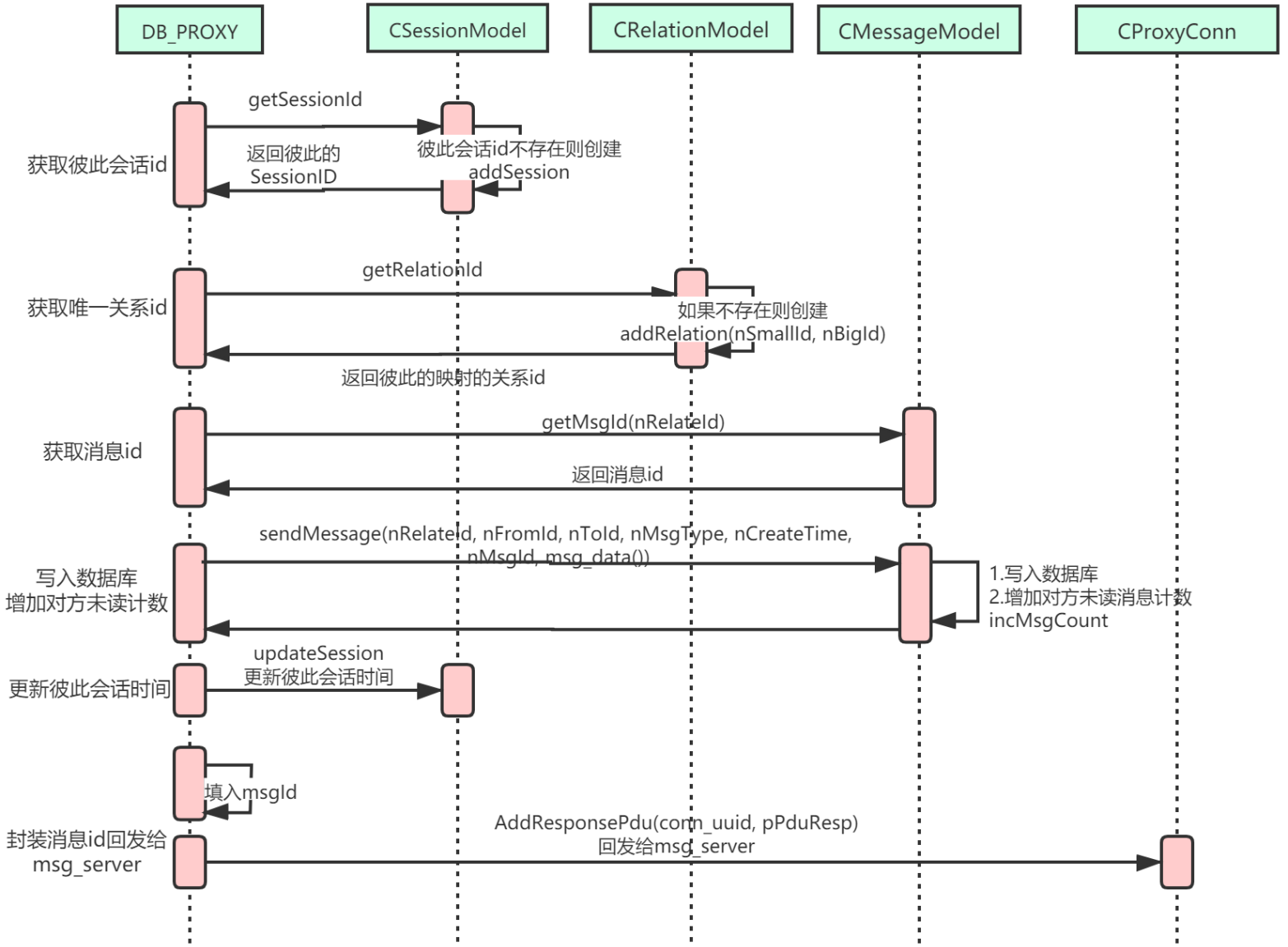
创建时间

6 群成员管理2



createGroup
modifyMember

7.1 单聊消息



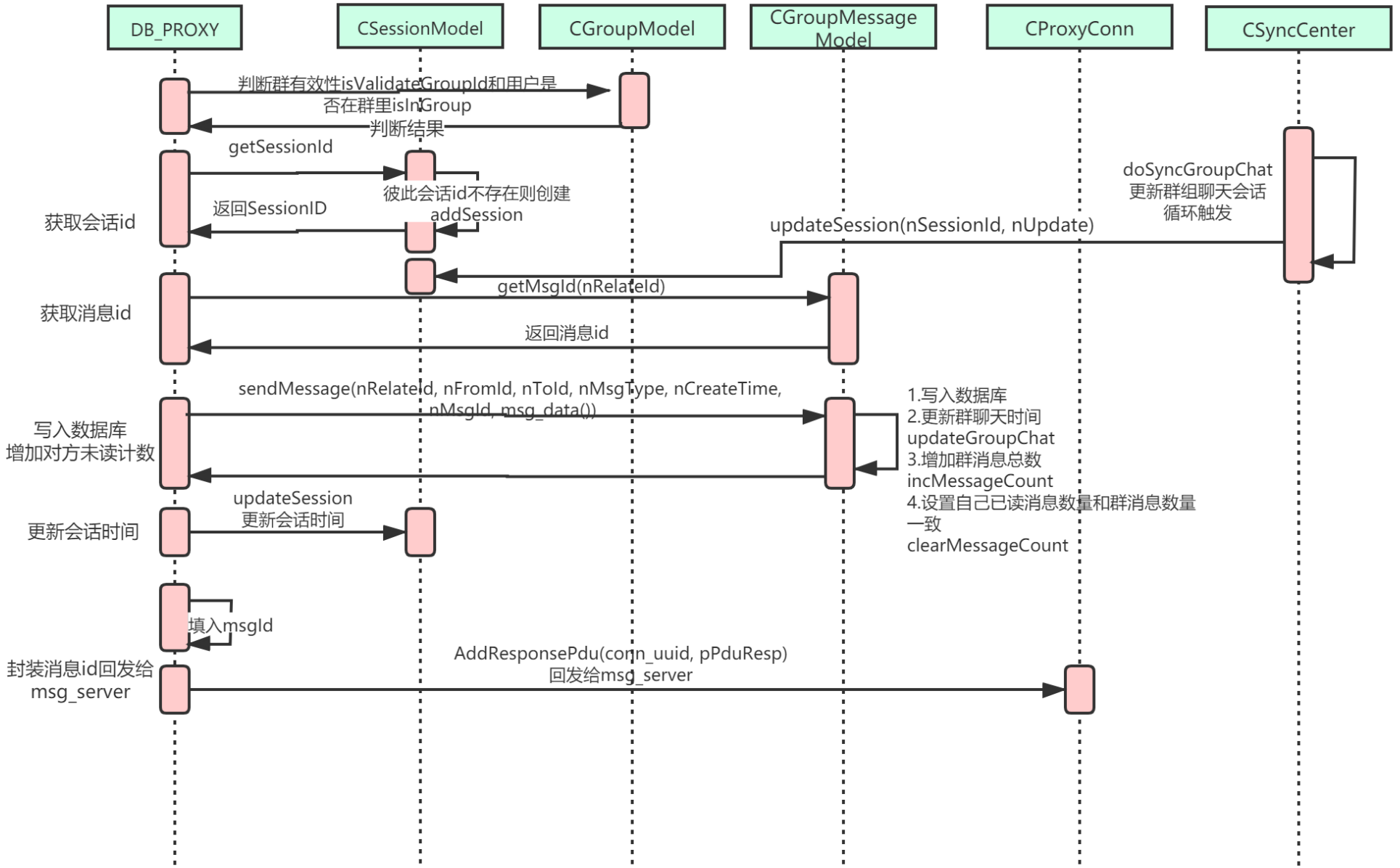
消息读取后回应
CID_MSG_READ_ACK

```
// message content
m_handler_map.insert(make_pair(uint32_t(CID_MSG_DATA), DB_PROXY::sendMessage));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_LIST_REQUEST), DB_PROXY::getMessage));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_UNREAD_CNT_REQUEST), DB_PROXY::getUnreadMsgCounter));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_READ_ACK), DB_PROXY::clearUnreadMsgCounter));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_GET_BY_MSG_ID_REQ), DB_PROXY::getMessageById));
m_handler_map.insert(make_pair(uint32_t(CID_MSG_GET_LATEST_MSG_ID_REQ), DB_PROXY::getLatestMsgId));
```

读取消息后应答

7.2 群聊消息

群聊的时候，实时更新每个user的会话不现实



7.3 群聊消息转发

void CDBServConn::_HandleMsgData(CImPdu *pPdu)

```
487: void CDBServConn::_HandleMsgData(CImPdu *pPdu)
488: {
489:     IM::Message::IMMsgData msg;
490:     CHECK_PB_PARSE_MSG(msg.ParseFromArray(pPdu->GetBodyData(), pPdu->GetBodyLength()));
491:     if (CHECK_MSG_TYPE_GROUP(msg.msg_type())) {
492:         s_group_chat->HandleGroupMessage(pPdu);
493:         return;
494:     }
495:
```

1. CID_GROUP_INFO_REQUEST
2. void getGroupInfo(CImPdu* pPdu, uint32_t conn_uuid)
3. case CID_GROUP_INFO_RESPONSE
   HandleGroupInfoResponse 发送数据

```
m_handler_map.insert(make_pair(uint32_t(CID_GROUP_INFO_REQUEST), DB_PROXY::getGroupInfo));
```