



下载APP

## 11 | 期中实战：动手写一个简易版的IM系统

2019-09-20 袁武林

即时消息技术剖析与实战

[进入课程 >](#)



讲述：袁武林

时长 08:53 大小 12.21M



你好，我是袁武林。

到上一讲为止，IM 的相关课程已经进行过半，在前面的课程中，我们讨论的大部分内容都比较偏理论，你理解起来可能会比较抽象。为了让你对前面讲到的知识有更深入的理解，今天我们就来回顾、梳理近期学习的内容，一起尝试搭建一个简单的 IM 聊天系统。

**在开始之前呢，我先来说明一下 IM 课程的期中、期末实战的课程计划和设计思路。**

期中和期末实战是希望你以自己动手实现为主，提供的 Demo 主要作为参考，在设计层面上，并不能让你直接用于线上使用。

本次期中实战 Demo 的主要关注点是：消息的存储、未读数的设计，并以“短轮询”的方式来实现消息的实时触达。希望你能从用户的使用场景出发，来理解消息存储设计的思路，以及未读数独立两套存储的必要性。

另外，在期末实战中，我会从“短轮询”方式调整为 WebSocket 长连接的方式，并且加上 ACK 机制、应用层心跳等特性。希望你能在两次实战中，通过对比，真正理解长连接方式相比“短轮询”方式的优势，并且通过 ACK 机制和应用层心跳，真正理解为什么它们能够解决“数据丢失”和“连接可靠性”的问题。

OK，下面我们说回本次实战。

这个聊天系统要求并不复杂，只需要构建简单的 Web 界面（没有界面通过命令行能实现也行）。我在这里写了一个简易版的 Demo，供你参考。

示例代码你可以在[GitHub](#)上下载。

## 需求梳理

这个简易聊天系统的大概要求有以下几点：

支持用户登录；

双方支持简单的文本聊天；

支持消息未读数（包括总未读和会话未读）；

支持联系人页和未读数有新消息的自动更新；

支持聊天页有新消息时自动更新。

## 需求分析

我们先来分析一下整体需求。

**首先，要支持用户登录，先要有“用户”。**对应的数据底层需要有一个用户表，用户表的设计可以比较简单，能够支持唯一的 UID 和密码用于登录即可。当然，如果有用户头像信息，聊天时的体验会更好，所以这里我们也加一下。简单的库表设计可以是这样的：

```
1 CREATE TABLE IM_USER (
2   uid INT PRIMARY KEY,
3   username VARCHAR(500) NOT NULL,
4   password VARCHAR(500) NOT NULL,
5   email VARCHAR(250) DEFAULT NULL,
6   avatar VARCHAR(500) NOT NULL
7 );
```

◀ ▶

对应的实体类 User 字段和库表一致，这里就不罗列了，我们需要设计用户通过邮箱和密码来登录。因为课程主要是涉及 IM 相关的知识，所以这里对用户信息的维护可以不做要求，启动时内置几个默认用户即可。

**有了用户后，接下来就是互动了，这一期我们只需要关注简单的文本聊天即可。**在设计中，我们需要对具体的聊天消息进行存储，便于在 Web 端使用，因此可以简单地按照 “[02 | 消息收发架构：为你的 App，加上实时通信功能](#)” 中讲到的消息存储来实现此项功能。

消息的存储大概分为消息内容表、消息索引表、联系人列表，这里我用最基础的字段来给你演示一下。单库单表的设计如下：

复制代码

```
1 消息内容表:
2
3 CREATE TABLE IM_MSG_CONTENT (
4   mid INT AUTO_INCREMENT PRIMARY KEY,
5   content VARCHAR(1000) NOT NULL,
6   sender_id INT NOT NULL,
7   recipient_id INT NOT NULL,
8   msg_type INT NOT NULL,
9   create_time TIMESTAMP NOT NULL
10 );
```

◀ ▶

复制代码

```
1 消息索引表:
2
3 CREATE TABLE IM_MSG_RELATION (
4   owner_uid INT NOT NULL,
5   other_uid INT NOT NULL,
6   mid INT NOT NULL,
7   type INT NOT NULL,
8   create_time TIMESTAMP NOT NULL,
```

```
9 PRIMARY KEY (`owner_uid`, `mid`)
10 );
11 CREATE INDEX `idx_owneruid_otheruid_msgid` ON IM_MSG_RELATION(`owner_uid`, `other_uid`,
```

 复制代码

```
1 联系人列表:
2
3 CREATE TABLE IM_MSG_CONTACT (
4 owner_uid INT NOT NULL,
5 other_uid INT NOT NULL,
6 mid INT NOT NULL,
7 type INT NOT NULL,
8 create_time TIMESTAMP NOT NULL,
9 PRIMARY KEY (`owner_uid`, `other_uid`)
10 );
```

**消息内容表：**由于只是单库单表，消息 ID 采用自增主键，主要包括消息 ID 和消息内容。

**消息索引表：**使用了索引“归属人”和消息 ID 作为联合主键，可以避免重复写入，并增加了“归属人”和“关联人”及消息 ID 的索引，用于查询加速。

**联系人列表：**字段和索引表一致，不同点在于采用“归属人”和“关联人”作为主键，可以避免同一个会话有超过一条的联系人记录。

消息相关实体层类的数据结构和库表的字段基本一致，这里不再列出，需要注意的是：为了演示的简单方便，这里并没有采用分库分表的设计，所以分库的 Sharding 规则你需要结合用户消息收发和查看的场景，多加考虑一下库表的设计。

**OK，有了用户和消息存储功能，现在来看如何支持消息未读数。**

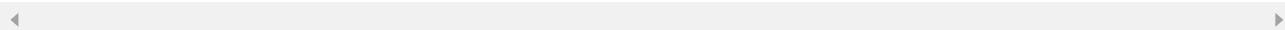
在“[07 | 分布式锁和原子性：你看到的未读消息提醒是真的吗？](#)”一讲中，我讲到了消息未读数在聊天场景中的重要性，这里我们也把未读数相关的功能加上来。

未读数分为总未读和会话未读，总未读虽然是会话未读之和，但由于使用频率很高，会话很多时候聚合起来性能比较差，所以冗余了总未读来单独存储。比如，你可以采用 Redis 来

进行存储，总未读可以使用简单的 K-V (Key-Value) 结构存储，会话未读使用 Hash 结构存储。大概的存储格式如下：

 复制代码

```
1 总未读:  
2 owneruid_T, 2  
3  
4 会话未读:  
5 owneruid_C, otheruid1, 1  
6 owneruid_C, otheruid2, 1
```



最后，我们一起来看看如何支持消息和未读自动更新。

在 “[03 | 轮询与长连接：如何解决消息的实时到达问题？](#)” 一讲中，我讲到了保证消息实时性的三种常见方式：短轮询、长轮询、长连接。对于消息和未读的自动更新的设计，你可以采用其中任意一种，我实现的简版代码里就是采用的“短轮询”来在联系人页面和聊天页面轮询未读和新消息的。实现上比较简单，Web 端核心代码和服务端核心代码如下。

 复制代码

```
1 Web 端核心代码:  
2  
3 newMsgLoop = setInterval(queryNewcomingMsg, 3000);  
4 $.get(  
5     '/queryMsgSinceMid',  
6     {  
7         ownerUid: ownerUid,  
8         otherUid: otherUid,  
9         lastMid: lastMid  
10    },  
11    function (msgsJson) {  
12        var jsonarray = $.parseJSON(msgsJson);  
13        var ul_pane = $('.chat-thread');  
14        var owner_uid_avatar, other_uid_avatar;  
15        $.each(jsonarray, function (i, msg) {  
16            var relation_type = msg.type;  
17            owner_uid_avatar = msg.ownerUidAvatar;  
18            other_uid_avatar = msg.otherUidAvatar;  
19  
20            var ul_pane = $('.chat-thread');  
21            var li_current = $('- </li>'); // 创建一个 li  
22            li_current.text(msg.content);  
23            ul_pane.append(li_current);  
24        });  
25    });

```

```
25         });
26     });
27 );
```

 复制代码

```
1 服务端核心代码:
2
3 List<MessageVO> msgList = Lists.newArrayList();
4 List<MessageRelation> relationList =     relationRepository.findAllByOwnerUidAndOtherUid(
5
6 /** 先拼接消息索引和内容 */
7 User self = userRepository.findOne(ownerUid);
8 User other = userRepository.findOne(otherUid);
9 relationList.stream().forEach(relation -> {
10     Long mid = relation.getMid();
11     MessageContent contentVO = contentRepository.findOne(mid);
12     if (null != contentVO) {
13         String content = contentVO.getContent();
14         MessageVO messageVO = new MessageVO(mid, content,     relation.getOwnerUid(), relati
15         msgList.add(messageVO);
16     }
17 });
18
19 /** 再变更未读 */
20 Object convUnreadObj = redisTemplate.opsForHash().get(ownerUid + Constants.CONVERSION_UN
21 if (null != convUnreadObj) {
22     long convUnread = Long.parseLong((String) convUnreadObj);
23     redisTemplate.opsForHash().delete(ownerUid +     Constants.CONVERSION_UNREAD_SUFFIX,
24     long afterCleanUnread =     redisTemplate.opsForValue().increment(ownerUid +     Con
25
26 /** 修正总未读 */
27 if (afterCleanUnread <= 0) {
28     redisTemplate.delete(ownerUid + Constants.TOTAL_UNREAD_SUFFIX);
29 }
30 return msgList;
```

## 这里需要注意两个地方：

其一，由于业务上使用的“消息对象”和存储层并不是一一对应关系，所以一般遇到这种情况，你可以为展现层在实体层基础上创建一些 VO 对象来承接展示需要的数据。比如，我在这里创建了一个用于消息展现的 VO 对象 MessageVO。

其二，在未读数的实现上，由于只是演示，这里并没有做到两个未读的原子变更，所以可能存在两个未读不一致的情况，因此上方代码最后部分我简单对总未读做了一个为负后的修正。

## 其他实现上的注意点

前面我们从需求梳理出发，把这个简易聊天系统核心部分的设计思路和核心代码讲了一下，也强调了一下这套代码实现中容易出现问题的地方。除此之外，我希望在代码实现中，你还能考虑以下这些问题，虽然这些不是 IM 系统实现中的核心问题，但对于代码的整洁和设计的习惯培养也是作为一个程序员非常重要的要求。

### 代码分层

首先是代码分层方面，我们在代码实现上，应该尽量让表现层、业务层和持久化层能分离清楚，做到每一层的职责清晰明确，也隔离了每一层的实现细节，便于多人协作开发。

表现层控制数据输入和输出的校验和格式，不涉及业务处理；业务层不涉及展现相关的代码，只负责业务逻辑的组合；持久化层负责和底层资源的交互，只负责数据的写入、变更和获取，不涉及具体的业务逻辑。

### 依赖资源

另外，对于代码中使用到的 Web 服务器、DB 资源、Redis 资源等，我们尽量通过 Embedding 的方式来提供，这样便于在 IDE 中执行和代码的迁移使用。

### 小结

这一节课主要是安排一次实战测试，让你来实现一个简易的聊天系统，涉及的 IM 相关知识包括：消息存储的设计、消息未读的设计、消息实时性的具体实现，等等。

在实现过程中，希望你能做到合理的代码分层规划，通过 Embedding 方式提升代码对 IDE 的友好性及代码的可迁移性。相信通过这一次理论与实践相结合的动手实战，能够帮助你进一步加深对 IM 系统核心技术的理解。

千里之行，积于跬步。通过脚踏实地的代码实现，将理论知识变成真正可用的系统，你的成就感也是会倍增的。

期中实战的演示代码并没有完全覆盖之前课程讲到的内容，所以如果你有兴趣，也非常欢迎你在代码实现时加入更多的特性，比如长连接、心跳、ACK 机制、未读原子化变更等。如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。感谢你的收听，我们下期再见。



# 即时消息技术剖析与实战

10 周精通 IM 后端架构技术点

袁武林

微博研发中心技术专家



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 自动智能扩缩容：直播互动场景中峰值流量的应对

下一篇 12 | 服务高可用：保证核心链路稳定性的流控和熔断机制

## 精选留言 (7)

 写留言



杨成龙

2019-09-20

只是基于redis吗？没用到netty吗？

展开 ▼

作者回复：期中实战只是个简易版demo哈，期末的实战会基于目前轮询的方式进行长连接改造，再加上ack和心跳等功能，这样也让大家能对“消息推送实时性”的实现方案有一个比对。

另外，实战的目的是让大家自己尝试结合课程前面的内容来练手加深对知识的理解，所以提供的demo并不具备线上可用性，只是一个简单的参考。



Geek\_71a2ee

2019-09-21

消息索引表：

```
CREATE TABLE IM_MSG_RELATION (
    owner_uid INT NOT NULL,
    other_uid INT NOT NULL,...
```

展开 ▼



2



给我点阳光就灿烂

2019-09-23

即使通讯的消息可不可以不存在数据库中而已消息队列的形式代替呢



1



leslie

2019-09-23

期中考试、、、、还是等期末考试的时候一起做吧，看的懂写不来，出去的都是伪代码：忙起来就发现写这个东西自己的Coding太差了，被Coding能力拖后腿了：谁让这是DBA和OPS的通病呢、、、



飞翔

2019-09-23

老师 有一个问题  
对于redis 事务

```
redisTemplate.multi();
redisTemplate.opsForValue().increment(recipientUid + "_T", 1); //加总未读
redisTemplate.opsForHash().increment(recipientUid + "_C", senderUid, 1); //...
```

展开 ▼



飞翔

2019-09-22

```
CREATE INDEX `idx_owneruid_otheruid_msgid` ON IM_MSG_RELATION(`owner_uid`,'`other_uid`,`mid`);
```

老师 消息索引表中 为什么要创作(`owner\_uid`,`other\_uid`,`mid`); 这三个的联合索引 呀  
展开▼



飞翔

2019-09-22

消息内容表：

```
CREATE TABLE IM_MSG_CONTENT (
mid INT AUTO_INCREMENT PRIMARY KEY,
content VARCHAR(1000) NOT NULL,...
```

展开▼

