适合新手：从零开发一个IM服务端（基于Netty，有完整源码）

JackJiang Lv.9 18 天前 只看大图

0、引言

站长提示： 本文适合IM新手阅读，但最好有一定的网络编程经验，必竟实践性的代码上手就是网络编程。如果你对网络编程，以及IM的一些理论知识知之甚少，请务必首先阅读：《新手入门一篇就够：从零开发移动端IM》，该文为IM小白分类整理了详尽的理论资料，请按需补充相关知识。

配套源码： 本文写的比较浅显但不太易懂，建议结合代码一起来读，文章配套的完整源码 请从本文文末 “11、完整源码下载” 处下载!

本站的另两篇同类代码你可能也喜欢：

- 《自己开发IM有那么难吗？手把手教你自撸一个Andriod版简易IM (有源码)》
- 《拿起键盘就是干：跟我一起徒手开发一套分布式IM系统》

1、内容概述

首先讲讲IM（即时通讯）技术可以用来做什么：

- 1）聊天：qq、微信；
- 2）直播：斗鱼直播、抖音；
- 3）实时位置共享、游戏多人互动等等。

可以说几乎所有高实时性的应用场景都需要用到IM技术。

本篇将带大家从零开始搭建一个轻量级的IM服务端。

麻雀虽小，五脏俱全，我们搭建的IM服务端实现以下功能：

- 1）一对一的文本消息、文件消息通信；
- 2）每个消息有“已发送”/“已送达”/“已读”回执；
- 3）存储离线消息；
- 4）支持用户登录，好友关系等基本功能；
- 5）能够方便地水平扩展。

通过这个项目能学到很多后端必备知识：

- 1）rpc通信；
- 2）数据库；
- 3）缓存；
- 4）消息队列；
- 5）分布式、高并发的架构设计；
- 6）docker部署。

2、相关文章

更多实践性代码参考：

- 《开源移动端IM技术框架MobileIMSDK》（* 推荐）
- 《自己开发IM有那么难吗？手把手教你自撸一个Andriod版简易IM (有源码)》
- 《一种Android端IM智能心跳算法的设计与实现探讨（含样例代码）》
- 《手把手教你用Netty实现网络通信程序的心跳机制、断线重连机制》
- 《NIO框架入门(一)：服务端基于Netty4的UDP双向通信Demo演示 [附件下载]》
- 《NIO框架入门(二)：服务端基于MINA2的UDP双向通信Demo演示 [附件下载]》
- 《NIO框架入门(三)：iOS与MINA2、Netty4的跨平台UDP双向通信实战 [附件下载]》
- 《NIO框架入门(四)：Android与MINA2、Netty4的跨平台UDP双向通信实战 [附件下载]》
- 《一个WebSocket实时聊天室Demo：基于node.js+socket.io [附件下载]》

相关IM架构方面的文章：

- 《浅谈IM系统的架构设计》
- 《简述移动端IM开发的那些坑：架构设计、通信协议和客户端》
- 《一套海量在线用户的移动端IM架构设计实践分享(含详细图文)》
- 《一套原创分布式即时通讯(IM)系统理论架构方案》
- 《从零到卓越：京东客服即时通讯系统的技术架构演进历程》
- 《蘑菇街即时通讯/IM服务器开发之架构选择》
- 《一套高可用、易伸缩、高并发的IM群聊、单聊架构方案设计实践》

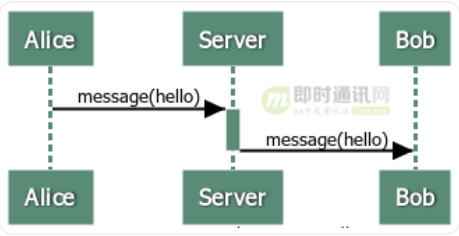
3、消息通信

3.1 文本消息

我们先从最简单的特性开始实现：一个普通消息的发送。

消息格式如下：

```
message ChatMsg{
    id = 1;
    //消息id
    fromId = Alice
    //发送者userId
    destId = Bob
    //接收者userId
    msgBody = hello
    //消息体
}
```



如上图，我们现在有两个用户：Alice和Bob连接到了服务器，当Alice发送消息message(hello)给Bob，服务端接收到消息，根据消息的destId进行转发，转发给Bob。

3.2 发送回执

那我们要怎么来实现回执的发送呢？

我们定义一种回执数据格式ACK，MsgType有三种，分别是sent（已发送）,delivered（已送达）, read（已读）。

消息格式如下：

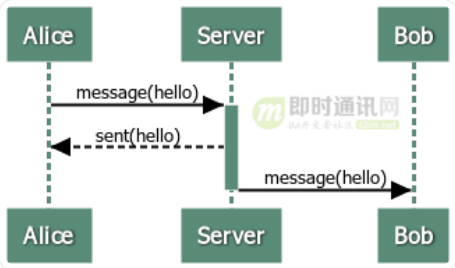
```
message AckMsg {
    id;
    //消息id
    fromId;
    //发送者id
    destId;
    //接收者id
    msgType;
    //消息类型
    ackMsgId;
    //确认的消息id
}

enum MsgType {
    DELIVERED;
    READ;
}
```

当服务端接受到Alice发来的消息时：

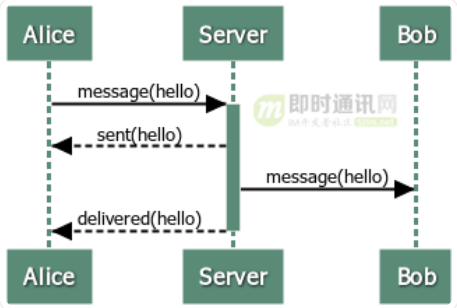
1）向Alice发送一个sent(hello)表示消息已经被发送到服务器：

```
message AckMsg {
    id = 2;
    fromId = Alice;
    destId = Bob;
    msgType = SENT;
    ackMsgId = 1;
}
```



2）服务器把hello转发给Bob后，立刻向Alice发送delivered(hello)表示消息已经发送给Bob：

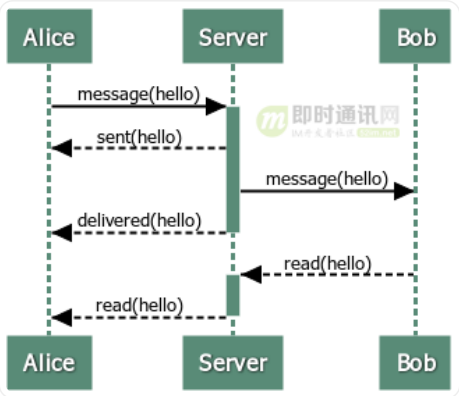
```
message AckMsg {
    id = 3;
    fromId = Bob;
    destId = Alice;
    msgType = DELIVERED;
    ackMsgId = 1;
}
```



3）Bob阅读消息后，客户端向服务器发送read(hello)表示消息已读：

```
message AckMsg {
```

```
    id = 4;
    fromId = Bob;
    destId = Alice;
    msgType = READ;
    ackMsgId = 1;
}
```



这个消息会像一个普通聊天消息一样被服务器处理，最终发送给Alice。

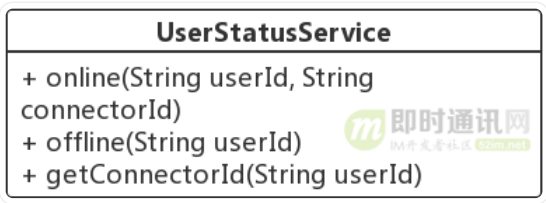
在服务器这里不区分ChatMsg和AckMsg，处理过程都是一样的：解析消息的destId并进行转发。

4、水平扩展

当用户量越来越大，必然需要增加服务器的数量，用户的连接被分散在不同的机器上。此时，就需要存储用户连接在哪台机器上。

我们引入一个新的模块来管理用户的连接信息。

4.1 管理用户状态



模块叫做user status，共有三个接口：

```
public interface UserStatusService {

    /**
     * 用户上线，存储userId与机器id的关系
     *
     * @param userId
     * @param connectorId
     * @return 如果当前用户在线，则返回他连接的机器id，否则返回null
     */
    String online(String userId, String connectorId);

    /**
     * 用户下线
     *
     * @param userId
     */
    void offline(String userId);

    /**
     * 通过用户id查找他当前连接的机器id
     *
```

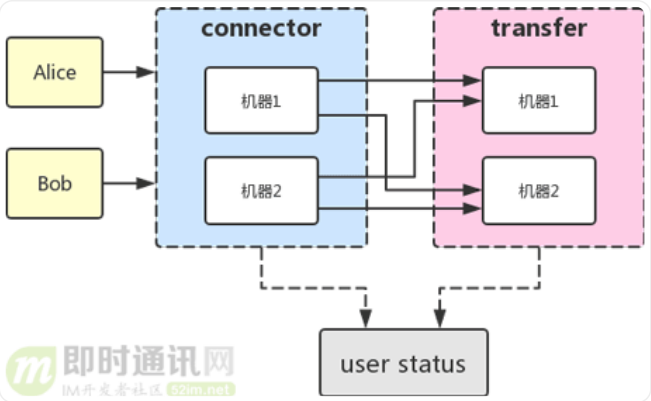
```
22      * @param userId
23      * @return
24      */
25     String getConnectorId(String userId);
26 }
```

这样我们就能够对用户连接状态进行管理了，具体的实现应考虑服务的用户量、期望性能等进行实现。

此处我们使用redis来实现，将userId和connectorId的关系以key-value的形式存储。

4.2 消息转发

除此之外，还需要一个模块在不同的机器上转发消息，如下结构：



此时我们的服务被拆分成了connector和transfer两个模块，connector模块用于维持用户的长链接，而transfer的作用是将消息在多个connector之间转发。

现在Alice和Bob连接到了两台connector上，那么消息要如何传递呢？

1）Alice上线，连接到机器[1]上时：

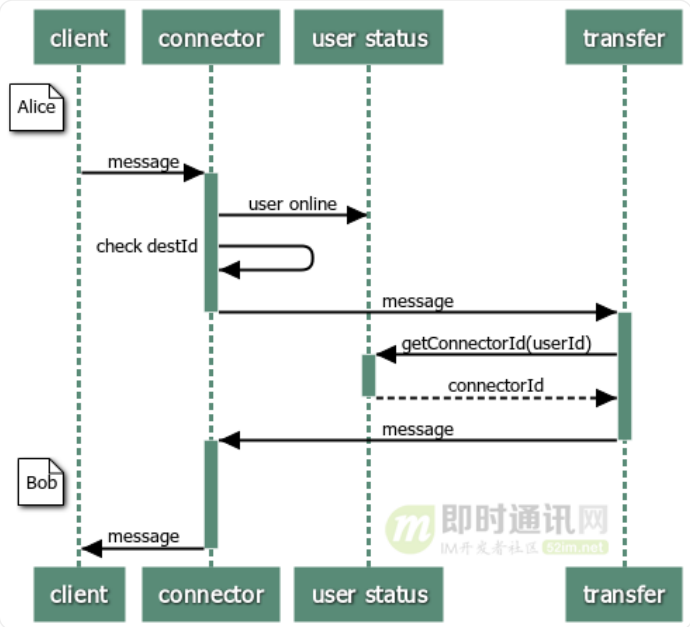
- 1.1）将Alice和它的连接存入内存中。
- 1.2）调用user status的online方法记录Alice上线。

2）Alice发送了一条消息给Bob：

- 2.1）机器[1]收到消息后，解析destId，在内存中查找是否有Bob。
- 2.2）如果没有，代表Bob未连接到这台机器，则转发给transfer。

3）transfer调用user status的getConnectorId(Bob)方法找到Bob所连接的connector，返回机器[2]，则转发给机器[2]。

流程图：



4.3 总结

引入user status模块管理用户连接，transfer模块在不同的机器之间转发，使服务可以水平扩展。为了满足实时转发，transfer需要和每台connector机器都保持长链接。

5、离线消息

如果用户当前不在线，就必须把消息持久化下来，等待用户下次上线再推送，这里使用mysql存储离线消息。

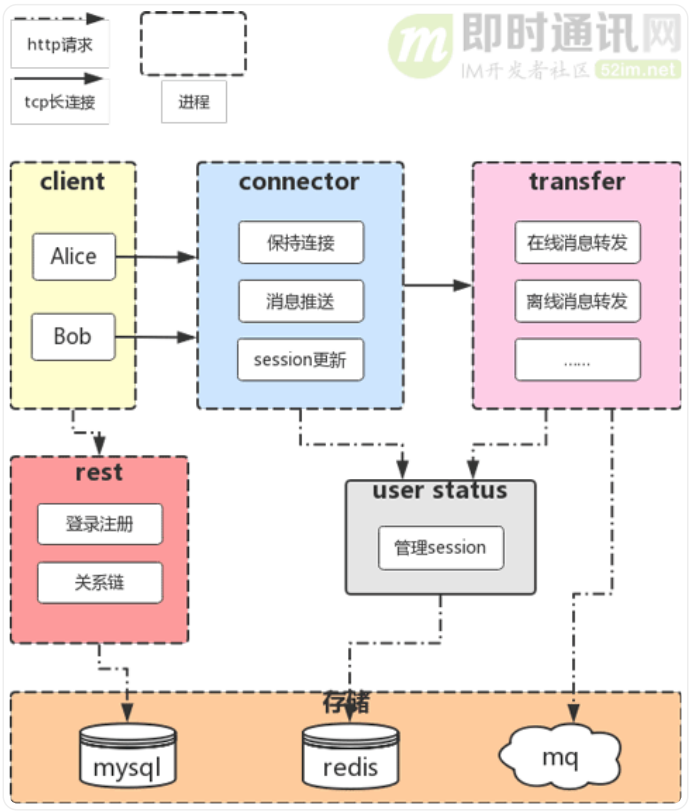
为了方便地水平扩展，我们使用消息队列进行解耦：

- 1）transfer接收到消息后如果发现用户不在线，就发送给消息队列入库；
- 2）用户登录时，服务器从库里拉取离线消息进行推送。

6、用户登录、好友关系

用户的注册登录、账户管理、好友关系链等功能更适合使用http协议，因此我们将这个模块做成一个restful服务，对外暴露http接口供客户端调用。

至此服务端的基本架构就完成了：



7、中场休息 ... ...

本文以上内容，本篇帮大家构建了IM服务端的架构，但还有很多细节需要我们去思考。

例如：

- 1）如何保证消息的顺序和唯一
- 2）多个设备在线如何保证消息一致性
- 3）如何处理消息发送失败
- 4）消息的安全性
- 5）如果要存储聊天记录要怎么做
- 6）数据库分表分库
- 7）服务高可用
- ……

更多细节实现请继续读下半部分啦～

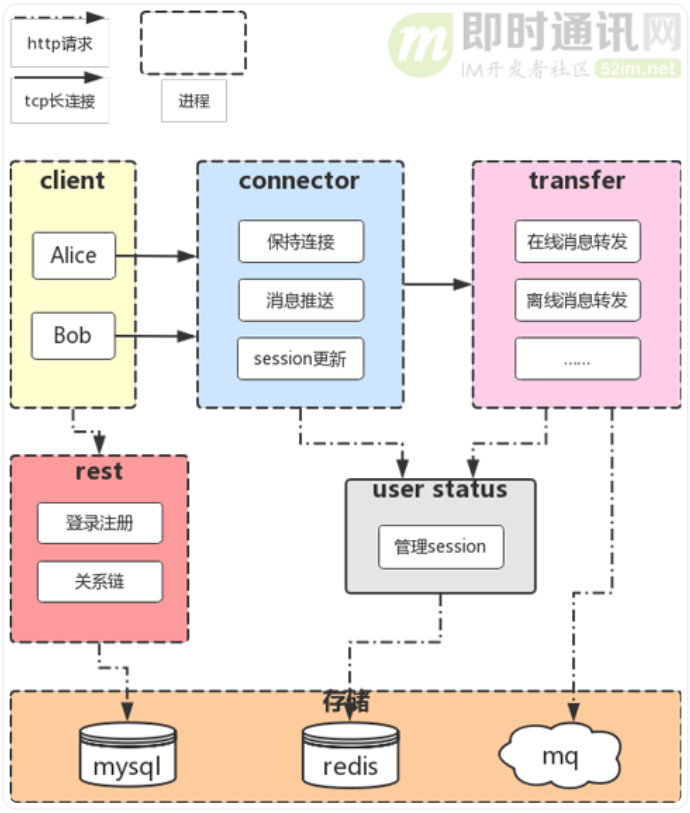
8、可靠性

什么是可靠性？对于一个IM系统来说，可靠的定义至少是不丢消息、消息不重复、不乱序，满足这三点，才能说有一个好的聊天体验。

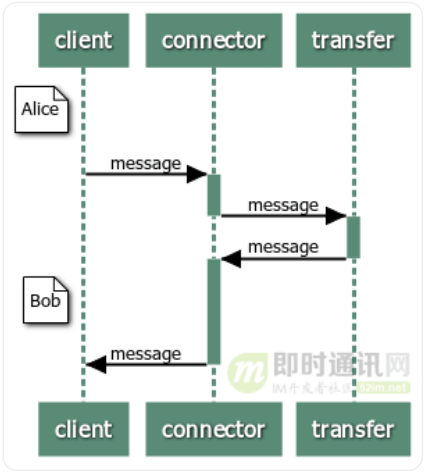
8.1 不丢消息

我们先从不丢消息开始讲起。

首先复习一下上面章节中设计的服务端架构：



我们先从一个简单例子开始思考： 当Alice给Bob发送一条消息时，可能要经过这样一条链路：



- 1）client-->connecter
- 2）connector-->transfer
- 3）transfer-->connector
- 4）connector-->client

在这整个链路中的每个环节都有可能出问题，虽然tcp协议是可靠的，但是它只能保证链路层的可靠，无法保证应用层的可靠。

例如在第一步中，connector收到了从client发出的消息，但是转发给transfer失败，那么这条消息Bob就无法收到，而Alice也不会意识到消息发送失败了。

如果Bob状态是离线，那么消息链路就是：

- 1）client-->connector
- 2）connector-->transfer
- 3）transfer-->mq

如果在第三步中，transfer收到了来自connector的消息，但是离线消息入库失败，那么这个消息也是传递失败了。

为了保证应用层的可靠，我们必须要有一个ack机制，使发送方能够确认对方收到了这条消息。

具体的实现，我们模仿tcp协议做一个应用层的ack机制。

tcp的报文是以字节（byte）为单位的，而我们以message单位。

发送方每次发送一个消息，就要等待对方的ack回应，在ack确认消息中应该带有收到的id以便发送方识别。

其次，发送方需要维护一个等待ack的队列。 每次发送一个消息之后，就将消息和一个计时器入队。

另外存在一个线程一直轮询队列，如果有超时未收到ack的，就取出消息重发。

超时未收到ack的消息有两种处理方式：

- 1）和tcp一样不断发送直到收到ack为止。
- 2）设定一个最大重试次数，超过这个次数还没收到ack，就使用失败机制处理，节约资源。例如如果是connector长时间未收到client的ack，那么可以主动断开和客户端的连接，剩下未发送的消息就作为离线消息入库，客户端断连后尝试重连服务器即可。

8.2 不重复、不乱序

有的时候因为网络原因可能导致ack收到较慢，发送方就会重复发送，那么接收方必须有一个去重机制。

去重的方式是给每个消息增加一个唯一id。这个唯一id并不一定是全局的，只需要在一个会话中唯一即可。例如某两个人的会话，或者某一个群。如果网络断连了，重新连接后，就是新的会话了，id会重新从0开始。

关于消息ID的生成算法方面的文章，请详细参考：

- 《融云技术分享：解密融云IM产品的聊天消息ID生成策略》
- 《微信技术分享：微信的海量IM聊天消息序列号生成实践（算法原理篇）》
- 《微信技术分享：微信的海量IM聊天消息序列号生成实践（容灾方案篇）》
- 《美团技术分享：深度解密美团的分布式ID生成算法》

接收方需要在当前会话中维护收到的最后一个消息的id，叫做lastId。

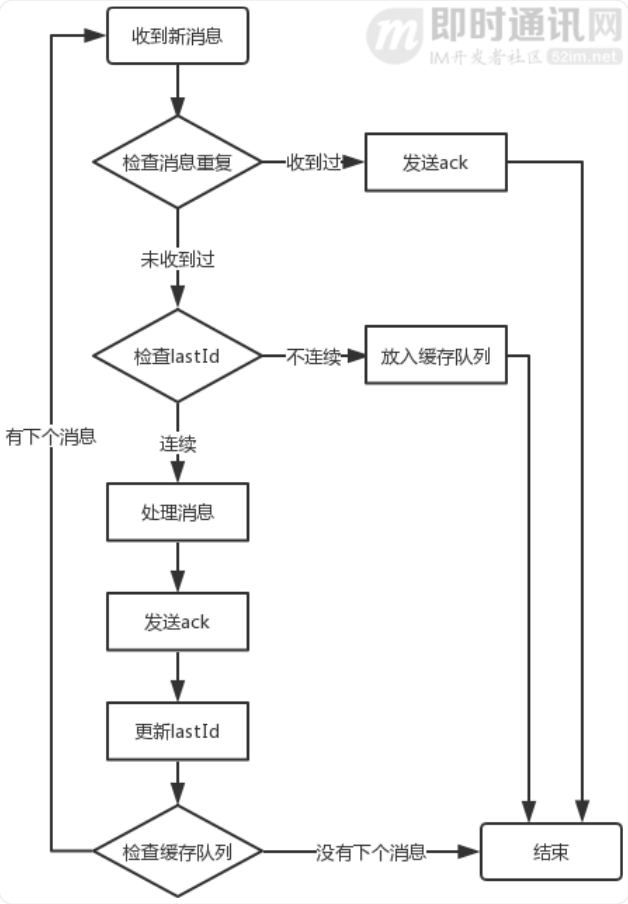
每次收到一个新消息， 就将id与lastId作比较看是否连续，如果不连续，就放入一个暂存队列 queue中稍后处理。

例如：

- 1）当前会话的lastId=1，接着服务器收到了消息msg(id=2)，可以判断收到的消息是连续的，就处理消息，将lastId修改为2；
- 2）但是如果服务器收到消息msg(id=3)，就说明消息乱序到达了，那么就将这个消息入队，等待lastId变为2后，（即服务器收到消息msg(id=2)并处理完了），再取出这个消息处理。

因此，判断消息是否重复只需要判断msgId>lastId && !queue.contains(msgId)即可。如果收到重复的消息，可以判断是ack未送达，就再发送一次ack。

接收方收到消息后完整的处理流程如下：



伪代码如下：

```
class ProcessMsgNode{
    /**
     * 接收到的消息
     */
    private Message message;
    /**
     * 处理消息的方法
     */
    private Consumer<Message> consumer;
}

public CompletableFuture<Void> offer(Long id,Message     message,Consumer<Message> consumer) {
    if (isRepeat(id)) {
    //消息重复
        sendAck(id);
        return null;
    }
    if (!isConsist(id)) {
    //消息不连续
        notConsistMsgMap.put(id, new ProcessMsgNode(message, consumer));
        return null;
    }
    //处理消息
    return process(id, message, consumer);
}

private CompletableFuture<Void> process(Long id, Message message, Consumer<Message> consumer) {
    return CompletableFuture
        .runAsync(() -> consumer.accept(message))
        .thenAccept(v -> sendAck(id))
        .thenAccept(v -> lastId.set(id))
        .thenComposeAsync(v -> {
            Long nextId = nextId(id);
```

```
            if (notConsistMsgMap.containsKey(nextId)) {
                //队列中有下个消息
                ProcessMsgNode node = notConsistMsgMap.get(nextId);
                return process(nextId, node.getMessage(), consumer);
            } else {
                //队列中没有下个消息
                CompletableFuture<Void> future = new CompletableFuture<>();
                future.complete(null);
                return future;
            }
        })
        .exceptionally(e -> {
            logger.error("[process received msg] has error", e);
            return null;
        });
}
```

9、安全性

无论是聊天记录还是离线消息，肯定都会在服务端存储备份，那么消息的安全性，保护客户的隐私也至关重要。

因此所有的消息都必须要加密处理。

在存储模块里，维护用户信息和关系链有两张基础表，分别是im_user用户表和im_relation关系链表。

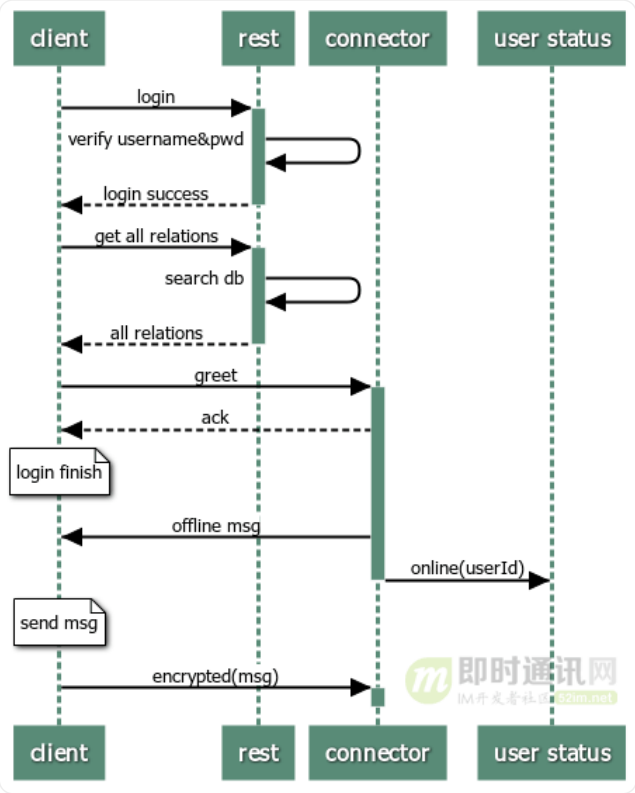
- im_user表用于存放用户常规信息，例如用户名密码等，结构比较简单。
- im_relation表用于记录好友关系。

结构如下：

```
CREATE TABLE `im_relation` (
  `id` bigint(20) COMMENT '关系id',
  `user_id1` varchar(100) COMMENT '用户1id',
  `user_id2` varchar(100) COMMENT '用户2id',
  `encrypt_key` char(33) COMMENT 'aes密钥',
  `gmt_create` timestamp DEFAULT CURRENT_TIMESTAMP,
  `gmt_update` timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `USERID1_USERID2` (`user_id1`,`user_id2`)
);
```

- 1）user_id1和user_id2是互为好友的用户id，为了避免重复，存储时按照user_id1<user_id2的顺序存，并且加上联合索引；
- 2）encrypt_key是随机生成的密钥。当客户端登录时，就会从数据库中获取该用户的所有的relation，存在内存中，以便后续加密解密；
- 3）当客户端给某个好友发送消息时，取出内存中该关系的密钥，加密后发送。同样，当收到一条消息时，取出相应的密钥解密。

客户端完整登录流程如下：



- 1）client调用rest接口登录；
- 2）client调用rest接口获取该用户所有relation；
- 3）client向connector发送greet消息，通知上线；
- 4）connector拉取离线消息推送给client；
- 5）connector更新用户session。

那为什么connector要先推送离线消息再更新session呢？

我们思考一下如果顺序倒过来会发生什么：

- 1）用户Alice登录服务器；
- 2）connector更新session；
- 3）推送离线消息；
- 4）此时Bob发送了一条消息给Alice。

如果离线消息还在推送的过程中，Bob发送了新消息给Alice，服务器获取到Alice的session，就会立刻推送。这时新消息就有可能夹在一堆离线消息当中推过去了，那这时，Alice收到的消息就乱序了。

而我们必须保证离线消息的顺序在新消息之前。

那么如果先推送离线消息，之后才更新session。在离线消息推送的过程中，Alice的状态就是“未上线”，这时Bob新发送的消息只会入库im_offline，im_offline表中的数据被读完之后才会“上线”开始接受新消息。这也就避免了乱序。

10、存储设计

10.1 存储离线消息

当用户不在线时，离线消息必然要存储在服务端，等待用户上线再推送。理解了上一个小节后，离线消息的存储就非常容易了。

增加一张离线消息表im_offline，表结构如下：

```
CREATE TABLE `im_offline` (
  `id` int(11) COMMENT '主键',
  `msg_id` bigint(20) COMMENT '消息id',
  `msg_type` int(2) COMMENT '消息类型',
  `content` varbinary(5000) COMMENT '消息内容',
  `to_user_id` varchar(100) COMMENT '收件人id',
  `has_read` tinyint(1) COMMENT '是否阅读',
```

```
  `gmt_create` timestamp COMMENT '创建时间',
  PRIMARY KEY (`id`)
);
```

msg_type用于区分消息类型（chat,ack），content加密后的消息内容以byte数组的形式存储。

用户上线时按照条件to_user_id=用户id拉取记录即可。

10.2 防止离线消息重复推送

我们思考一下多端登录的情况，Alice有两台设备同时登陆，在这种并发的情况下，我们就需要某种机制来保证离线消息只被读取一次。

这里利用CAS机制来实现：

- 1）首先取出所有has_read=false的字段；
- 2）检查每条消息的has_read值是否为false，如果是，则改为true。这是原子操作：

```
update im_offline set has_read = true where id = ${msg_id} and has_read = false
```

- 3）修改成功则推送，失败则不推送。

相信到这里，同学们已经可以自己动手搭建一个完整可用的IM服务端了。

11、完整源码下载

从零开发一个IM服务端（完整源码）(52im.net).zip (266.94 KB，下载次数: 22，售价: 1 金币)
（**或自行从github下载：** https://github.com/52im/IM）

附录：更多IM开发文章

[1] 更多IM代码实践(适合新手)：
《自己开发IM有那么难吗？手把手教你自撸一个Andriod版简易IM (有源码)》
《一种Android端IM智能心跳算法的设计与实现探讨（含样例代码）》
《手把手教你用Netty实现网络通信程序的心跳机制、断线重连机制》
《详解Netty的安全性：原理介绍、代码演示（上篇）》
《详解Netty的安全性：原理介绍、代码演示（下篇）》
《微信本地数据库破解版(含iOS、Android)，仅供学习研究 [附件下载]》
《Java NIO基础视频教程、MINA视频教程、Netty快速入门视频 [有源码]》
《轻量级即时通讯框架MobileIMSDK的iOS源码（开源版）[附件下载]》
《开源IM工程“蘑菇街TeamTalk”2015年5月前未删减版完整代码 [附件下载]》
《微信本地数据库破解版(含iOS、Android)，仅供学习研究 [附件下载]》
《NIO框架入门(一)：服务端基于Netty4的UDP双向通信Demo演示 [附件下载]》
《NIO框架入门(二)：服务端基于MINA2的UDP双向通信Demo演示 [附件下载]》
《NIO框架入门(三)：iOS与MINA2、Netty4的跨平台UDP双向通信实战 [附件下载]》
《NIO框架入门(四)：Android与MINA2、Netty4的跨平台UDP双向通信实战 [附件下载]》
《用于IM中图片压缩的Android工具类源码，效果可媲美微信 [附件下载]》
《高仿Android版手机QQ可拖拽未读数小气泡源码 [附件下载]》
《一个WebSocket实时聊天室Demo：基于node.js+socket.io [附件下载]》
《Android聊天界面源码：实现了聊天气泡、表情图标(可翻页) [附件下载]》
《高仿Android版手机QQ首页侧滑菜单源码 [附件下载]》
《开源libco库：单机千万连接、支撑微信8亿用户的后台框架基石 [源码下载]》
《分享java AMR音频文件合并源码，全网最全》
《微信团队原创Android资源混淆工具：AndResGuard [有源码]》
《一个基于MQTT通信协议的完整Android推送Demo [附件下载]》
《Android版高仿微信聊天界面源码 [附件下载]》
《高仿手机QQ的Android版锁屏聊天消息提醒功能 [附件下载]》
《高仿iOS版手机QQ录音及振幅动画完整实现 [源码下载]》
《Android端社交应用中的评论和回复功能实战分享[图文+源码]》
《Android端IM应用中的@人功能实现：仿微博、QQ、微信，零入侵、高可扩展[图文+源码]》
《仿微信的IM聊天时间显示格式(含iOS/Android/Web实现)[图文+源码]》
《Android版仿微信朋友圈图片拖拽返回效果 [源码下载]》
《适合新手：从零开发一个IM服务端（基于Netty，有完整源码）》
>> 更多同类文章 ……

[2] IM群聊相关的技术文章：

《快速裂变：见证微信强大后台架构从0到1的演进历程（一）》
《如何保证IM实时消息的“时序性”与“一致性”？》
《IM单聊和群聊中的在线状态同步应该用“推”还是“拉”？》
《IM群聊消息如此复杂，如何保证不丢不重？》
《微信后台团队：微信后台异步消息队列的优化升级实践分享》
《移动端IM中大规模群消息的推送如何保证效率、实时性？》
《现代IM系统中聊天消息的同步和存储方案探讨》
《关于IM即时通讯群聊消息的乱序问题讨论》
《IM群聊消息的已读回执功能该怎么实现？》
《IM群聊消息究竟是存1份(即扩散读)还是存多份(即扩散写)？》
《一套高可用、易伸缩、高并发的IM群聊、单聊架构方案设计实践》
《[技术脑洞] 如果把14亿中国人拉到一个微信群里技术上能实现吗？》
《IM群聊机制，除了循环去发消息还有什么方式？如何优化？》
《网易云信技术分享：IM中的万人群聊技术方案实践总结》
>> 更多同类文章 ……

[3] 有关IM架构设计的文章：

《浅谈IM系统的架构设计》
《简述移动端IM开发的那些坑：架构设计、通信协议和客户端》
《一套海量在线用户的移动端IM架构设计实践分享(含详细图文)》
《一套原创分布式即时通讯(IM)系统理论架构方案》
《从零到卓越：京东客服即时通讯系统的技术架构演进历程》
《蘑菇街即时通讯/IM服务器开发之架构选择》
《腾讯QQ1.4亿在线用户的技术挑战和架构演进之路PPT》
《微信后台基于时间序的海量数据冷热分级架构设计实践》
《微信技术总监谈架构：微信之道——大道至简(演讲全文)》
《如何解读《微信技术总监谈架构：微信之道——大道至简》》
《快速裂变：见证微信强大后台架构从0到1的演进历程（一）》
《17年的实践：腾讯海量产品的技术方法论》
《移动端IM中大规模群消息的推送如何保证效率、实时性？》
《现代IM系统中聊天消息的同步和存储方案探讨》
《IM开发基础知识补课(二)：如何设计大量图片文件的服务端存储架构？》
《IM开发基础知识补课(三)：快速理解服务端数据库读写分离原理及实践建议》
《IM开发基础知识补课(四)：正确理解HTTP短连接中的Cookie、Session和Token》
《WhatsApp技术实践分享：32人工程团队创造的技术神话》
《微信朋友圈千亿访问量背后的技术挑战和实践总结》
《王者荣耀2亿用户量的背后：产品定位、技术架构、网络方案等》
《IM系统的MQ消息中间件选型：Kafka还是RabbitMQ？》
《腾讯资深架构师干货总结：一文读懂大型分布式系统设计的方方面面》
《以微博类应用场景为例，总结海量社交系统的架构设计步骤》
《快速理解高性能HTTP服务端的负载均衡技术原理》
《子弹短信光鲜的背后：网易云信首席架构师分享亿级IM平台的技术实践》
《知乎技术分享：从单机到2000万QPS并发的Redis高性能缓存实践之路》
《IM开发基础知识补课(五)：通俗易懂，正确理解并用好MQ消息队列》
《微信技术分享：微信的海量IM聊天消息序列号生成实践（算法原理篇）》
《微信技术分享：微信的海量IM聊天消息序列号生成实践（容灾方案篇）》
《新手入门：零基础理解大型分布式架构的演进历史、技术原理、最佳实践》
《一套高可用、易伸缩、高并发的IM群聊、单聊架构方案设计实践》
《阿里技术分享：深度揭秘阿里数据库技术方案的10年变迁史》
《阿里技术分享：阿里自研金融级数据库OceanBase的艰辛成长之路》
《社交软件红包技术解密(一)：全面解密QQ红包技术方案——架构、技术实现等》
《社交软件红包技术解密(二)：解密微信摇一摇红包从0到1的技术演进》
《社交软件红包技术解密(三)：微信摇一摇红包雨背后的技术细节》
《社交软件红包技术解密(四)：微信红包系统是如何应对高并发的》
《社交软件红包技术解密(五)：微信红包系统是如何实现高可用性的》
《社交软件红包技术解密(六)：微信红包系统的存储层架构演进实践》
《社交软件红包技术解密(七)：支付宝红包的海量高并发技术实践》
《社交软件红包技术解密(八)：全面解密微博红包技术方案》
《社交软件红包技术解密(九)：谈谈手Q红包的功能逻辑、容灾、运维、架构等》
《即时通讯新手入门：一文读懂什么是Nginx？它能否实现IM的负载均衡？》
《即时通讯新手入门：快速理解RPC技术——基本概念、原理和用途》
《多维度对比5款主流分布式MQ消息队列，妈妈再也不担心我的技术选型了》
《从游击队到正规军：马蜂窝旅游网的IM系统架构演进之路》

《IM开发基础知识补课(六)：数据库用NoSQL还是SQL？读这篇就够了！》
>> 更多同类文章 ……

来源：即时通讯网 - 即时通讯开发者社区!

标签： 新手入门 | IM开发 | 网络编程

本主题由 JackJiang 于 18 天前 加入精华

上一篇：求教关于IM离线聊天消息同步策略的的一些疑惑 ▪ 下一篇：拿起键盘就是干：跟我一起徒手开发一套分布式IM系统

本帖已收录至以下技术专辑

IM综合资料 | 主题 46 · 关注 6 | **IM代码实践(适合新手)** | 主题 26 · 关注 0

相关文章

- 不为人知的网络编程(九)：理论联系实际，全方位深入理解DNS
- 融云技术分享：解密融云IM产品的聊天消息ID生成策略
- 拿起键盘就是干：跟我一起徒手开发一套分布式IM系统
- 求教关于IM应用中，聊天记录数据存储的一些疑问
- 求教关于IM离线聊天消息同步策略的的一些疑惑
- 正确理解IM长连接的心跳及重连机制，并动手实现（有完整IM源码）

推荐方案

MobileIMSDK (v4.0精编版)
轻量级开源移动端即时通讯框架。
快速入门 / 性能 / 指南 / 提问

MobileIMSDK-Web (有偿开源)
轻量级Web端即时通讯框架。
详细介绍 / 精编源码 / 手册教程

RainbowAV new (有偿开源)
移动端实时音视频框架。
详细介绍 / 性能测试 / 安装体验

RainbowChat (技术转让)
基于MobileIMSDK的移动IM系统。
详细介绍 / 产品截图 / 安装体验

RainbowChat-Web (技术转让)
一套产品级Web端IM系统。
详细介绍 / 产品截图 / 演示视频

评论 5

2 楼: clark.li *Lv.3* 16 天前 引用此评论
文章还行
签名: 秋天到了，终于凉快了

3 楼: 只是路过 *Lv.7* 16 天前 引用此评论
有客户端可以跑跑吗
签名: 星期六，那又怎样，还是得上班

4 楼: 睡痴 *Lv.1* 8 天前 引用此评论
感谢

5 楼: Chowing *Lv.1* 昨天 14:18 引用此评论
还没学过netty看不懂
签名: 来看看咯

6 楼: JackJiang *Lv.9* 楼主 昨天 14:21 引用此评论
> **引用：** Chowing 发表于 2019-10-25 14:18
> 还没学过netty看不懂

先学netty再来看不迟
签名: 好想早点退休

+ 发新帖 发表评论 返回列表

即时通讯网
实时推送、IM等即时通讯相关技术的学习、交流与分享的平台。专业的资料、专业的人、专业的社区！让即时通讯技术能更好传播与分享。
平等 开放 分享 传承

友情链接 [友链交换]
ZEGO即构科技
一起开源网
容联云通讯
OpenSNS

关于
关于我们
活跃QQ群
在线文档
网址导航

手机访问本站

微信公众号 new