Atitit im即时聊天的系统的实现

原理理论部分

目录

[1. 3 2](#_Toc6530)

[2. 概念 2](#_Toc21993)

[2.1. Message 2](#_Toc32752)

[2.2. presence 2](#_Toc26258)

[3. 2](#_Toc15674)

[4. 2](#_Toc12170)

[5. 、消息通信 2](#_Toc29677)

[5.1. 3.1 文本消息 2](#_Toc6718)

[5.2. 3.2 发送回执 3](#_Toc15505)

[5.3. 4、水平扩展 4](#_Toc1713)

[5.3.1. 5、离线消息 6](#_Toc567)

[6. 6、用户登录、好友关系 7](#_Toc31338)

[7. 特性 7](#_Toc10287)

[7.1. 多个设备在线如何保证消息一致性3 7](#_Toc6070)

[7.2. 10、存储设计 7](#_Toc27345)

[7.3. 10.1 存储离线消息 7](#_Toc10208)

[7.4. 10.2 防止离线消息重复推送 8](#_Toc15171)

# ****3****

# 概念

## Message

## presence

。RFC2778是一个资料性质的草案，定义了所有presence和IM服务 的原理。RFC2779定义了IMPP的最小需求条件。另外，这个草案还就presence服务定义了一些条款，如运行的命令、信息的格式，以及 presence服务器如何把presence的状态变化通知给客户。

# 

# 

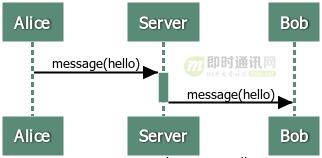
# ****、消息通信****

## ****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）。

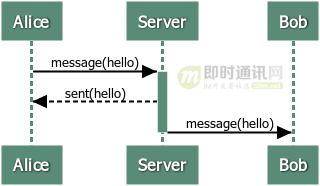
**消息格式如下：**

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：**

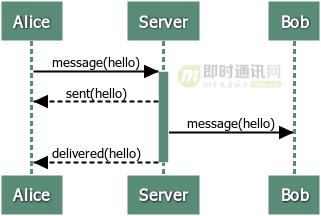
1）向Alice发送一个sent(hello)表示消息已经被发送到服务器：

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



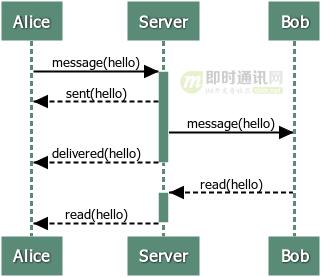
2）服务器把hello转发给Bob后，立刻向Alice发送delivered(hello)表示消息已经发送给Bob：

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。

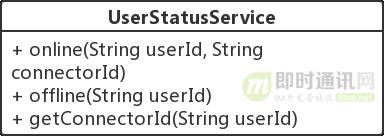
在服务器这里不区分ChatMsg和AckMsg，处理过程都是一样的：解析消息的destId并进行转发。

## ****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：**

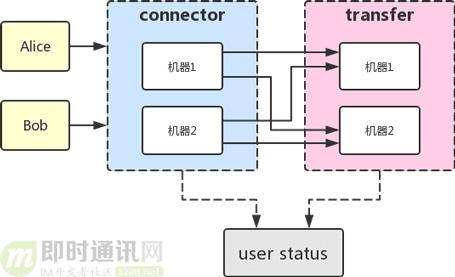
public interface UserStatusService { /\*\* \* 用户上线，存储userId与机器id的关系 \* \* @param userId \* @param connectorId \* @return 如果当前用户在线，则返回他连接的机器id，否则返回null \*/ String online(String userId, String connectorId); /\*\* \* 用户下线 \* \* @param userId \*/ voidoffline(String userId); /\*\* \* 通过用户id查找他当前连接的机器id \* \* @param userId \* @return \*/ String getConnectorId(String userId);}

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。

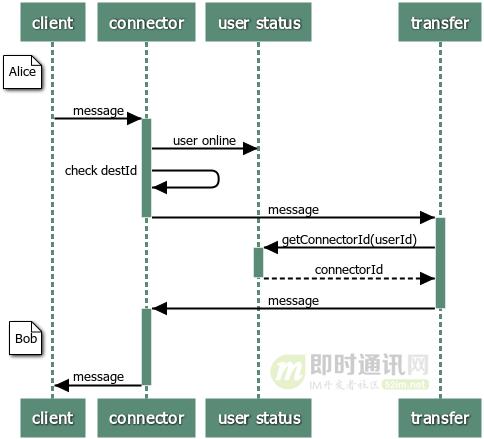
**2）Alice发送了一条消息给Bob：**

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：**

# 特性

## 多个设备在线如何保证消息一致性3

**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。

## ****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：

1updateim\_offline sethas\_read = truewhereid = ${msg\_id} andhas\_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。