# 1.登陆IM

Connection.DEBUG\_ENABLED = true;//首先激活调试模式

## 1.1建立连接

首先，在启动DSM Message时，客户端通过XMPPConnection与服务器建立连接。建立连接的方式有两种：

1.直接连到服务器

Connection conn = **new** XMPPConnection("localhost");//创建连接

//其中“localhost”是服务器地址，由于我用的是本机，所以是“localhost”。

conn.connect();//接通连接

2.根据配置连接

ConnectionConfiguration config = ConnectionConfiguration();

config.set[ServiceName](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/ConnectionConfiguration.html" \l "getServiceName%28%29)("localhost")；//还可以设置很多其他属性，如隐身登陆

Connection conn1 = **new** XMPPConnection(config);

conn1.connect();

## 1.2用户登陆

建立连接之后就是用户登陆了，openfire是支持多终端登陆的，下面的resource就是指的是终端名称，如Smack、Spark等。登陆的方法有两个

[login](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/Connection.html" \l "login%28java.lang.String,%20java.lang.String,%20java.lang.String%29)([String](http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html?is-external=true" \o "class or interface in java.lang) username, [String](http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html?is-external=true" \o "class or interface in java.lang) password)和

[login](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/Connection.html" \l "login%28java.lang.String,%20java.lang.String,%20java.lang.String%29)([String](http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html?is-external=true" \o "class or interface in java.lang) username, [String](http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html?is-external=true" \o "class or interface in java.lang) password, [String](http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html?is-external=true" \o "class or interface in java.lang) resource)

具体例子如下，livsun与livsun1是两个用户，密码都是111

conn.login("livsun", "111","Smack");

resource也可以缺省不设置

conn1.login("livsun1", "111");

本文后面默认livsun与livsun1已经登陆，且与服务器的连接分别conn和conn1.

## 1.3断开连接

断开连接只需调用disconnect()方法，如conn.disconnect();

# 2.单人聊天

## 2.1发起会话请求

作为一款IM软件的通信协议，首要功能就是聊天。聊天包含两种方式，主动向其他人发起会话；也可以接受别人的会话请求。下面是主动对别人发起会话的实现：

conn.getChatManager.createChat(String userID, MessageListener listener)

其中userID是本次对话请求的接收方的标志，如UserID是livsun1@z00189374/Smack，livsun1@z00189374是对话请求的接收方，/Smack说明用户是在哪个终端登陆的，可以缺省不写。

### 2.1.1消息监听

当想一个用户发起会话请求，对方接受请求并创建会话后，对方可能发送消息回来，这时需要对接受的消息进行处理。这里要用到MessageListener。

下例将创建一个会话并对收到的消息进行输出：

Chat chat = conn.getChatManager().createChat("livsun1@z00189374/Smack", **new** MessageListener(){

**public** **void** processMessage(Chat chat0, Message message)

{

System.*out*.println("来自livsun1的消息，内容为"+message.getBody());

}

});

红色chat是这次创建的对话对象，它在livsun跟livsun1之间建立一个会话连接；同事处理来自livsun1的消息。如果livsun想要发消息，需要用chat来调用sendMessage方法。

具体可见下例：

chat.sendMessage("Hello,I’m livsun");

### 2.1.2会话请求监听

通过上面例子可以知道livsun怎样发送了一个消息给livusn1。但是livsun1需要知道何时livsun发了一个消息给自己，同时还要对这个消息进行处理。Smack提供了会话请求监听接口ChatManagerListener，每个用户通过实现该接口，来监听没一个会话的创建请求。这里再以livsun1为例来展示一下：

chat.getChatManager. [addChatListener](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/ChatManager.html" \l "addChatListener%28org.jivesoftware.smack.ChatManagerListener%29)([ChatManagerListener](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/ChatManagerListener.html" \o "interface in org.jivesoftware.smack) listener)

ChatManagerListener只有一个方法[chatCreated](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/ChatManagerListener.html" \l "chatCreated%28org.jivesoftware.smack.Chat,%20boolean%29)([Chat](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/Chat.html" \o "class in org.jivesoftware.smack) chat, boolean createdLocally)，它用来处理当chat创建时的具体操作。

具体请看下面例子：

conn1.getChatManager().addChatListener(

**new** ChatManagerListener(){

**public** **void** chatCreated(Chat chat, **boolean** createdLocal)

{

//自己创建是指自己调用createChat方法，它也会触发这个listener

//在创建对话时就会制定MessageListener，这样判断主要是为了避免重复定义//MessageLister

**if**(createdLocal)

{

System.*out*.println("这个chat是自己创建的");

}

//创建与livsun的对话，这里面可以通过message来获取发送聊天请求的人信息

**else**

{

System.*out*.println("别人想跟我聊天");

chat.addMessageListener(**new** MessageListener(){

**public** **void** processMessage(Chat chat, Message message)

{

System.*out*.println(message.getFrom+"想跟我聊天)

chat.sendMessage("你好");

}

});

}

}

});

### 2.1.3会话状态监听

通常用户在参加一个会话时，想知道另一个用户是否正在输入。会话状态有五种：active（参加会话）, composing（正在输入）, gone（离开）, inactive（没有参加会话）, paused（暂停输入）。

如果想要对会话状态监听，需要在用户登陆之后，会话建立之前，添加ChatStateManager对象，方法为ChatStateManager.getInstance(XPMMConnectio conn);譬如获取livsun连接的ChatStateManager代码是ChatStateManager csm = ChatStateManager(conn);

在客户端，根据用户对于一个会话的不同操作，通过ChatStateManager调用方法[setCurrentState](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/ChatStateManager.html" \l "setCurrentState%28org.jivesoftware.smackx.ChatState,%20org.jivesoftware.smack.Chat%29)([ChatState](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/ChatState.html" \o "enum in org.jivesoftware.smackx) newState, [Chat](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/Chat.html" \o "class in org.jivesoftware.smack) chat)来设置会话的即时状态。如当livsun正在打字输入，则可以通过csm.setCurrentState(ChatState.composing, chat)来设置状态为正在输入。

在客户端，还需要对会话状态改变进行监听，ChatStateListener继承MessageListener，

方法是[stateChanged](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/ChatStateListener.html" \l "stateChanged%28org.jivesoftware.smack.Chat,%20org.jivesoftware.smackx.ChatState%29)([Chat](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/Chat.html" \o "class in org.jivesoftware.smack) chat, [ChatState](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/ChatState.html" \o "enum in org.jivesoftware.smackx) state)，由于Chat类没有直接添加ChatStateListener的方法，因此需要自己编写一个内部类实现ChatStateListener和MessageListener两个接口。

请看例子：

private class SelfListener implements ChatStateListener, MessageListener{

public void stateChanged(Chat arg0, ChatState arg1){

……

}

Public void processMessage(Chat arg0, Message arg1){

……

}

}

MessageListener messageListener = new SelfListener();

这样当会话对方的ChatState发生改变时，stateChanged方法就被触发。实际上是接收到一个内容为空的Message，Message里通知ChatState改变。

## 2.2消息

### 2.2.1设置消息的属性

发送消息的方法sendMessage()，它有两种实现方式，sendMessage(String text) 和sendMessage(Message message)。第一种就是上面例子中所用的，直接以String为对象，发送一个消息。

为了满足用户自定义需求，Smack提供了第二种方式，它可以为message添加一些附加属性，在message中他们只显示为String字段。在客户端接收时，可以通过处理来让这些字段有特定的意义，譬如设置发送文字的大小，字体，颜色等。

这里需要了解Message这个类，它有三种构造函数，Message()、Message(String text)和Message(String text, Message.Type type)其中text是要发送的内容，Message.Type共有五种，常用的为chat和groupchat。

Message中有很多方法，通过这些方法可以设置或者取得消息的属性，如addBody()添加消息内容，getBody()获得消息内容，getFrom()获取消息的发送者等。Message还有API中没有提到的方法：setProperty(String name, String value)。通过这个方法我们可以设置很多自己定义的属性，可参见下面例子：

Message message = new Message();

message.setBody(“This is a Message text”);

message.setProperty(“color”,”red”);

message.setProperty(“size”, “big”);

message.setProperty(“isPictureIn”,”false”);

chat.sendMessage(message);

这样就发送了一个包含一些特定属性的消息。当用户收到这样的消息后，通过调用message. getPropertyNames()方法来获取所有属性的一个Collection。再通过message.getProprety(String name)来获取名为name属性的值，根据用户自己的定义，可以实现API没有提供的功能，如改变消息显示字体颜色。

### 2.2.2消息状态跟踪

对于发送出去的消息，有时候需要获取消息发送的情况，如是否发送成功，对方是够处理等，这些功能需要用到MessageEventManager，通过调用MessageEventManager. [addNotificationsRequests](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/MessageEventManager.html" \l "addNotificationsRequests%28org.jivesoftware.smack.packet.Message,%20boolean,%20boolean,%20boolean,%20boolean%29)([Message](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smack/packet/Message.html" \o "class in org.jivesoftware.smack.packet) message, boolean offline, boolean delivered, boolean displayed, boolean composing)来对消息的状态进行标记跟踪，message为将要发送的消息，剩余四项为可选择的跟踪选项，true为跟踪。

当对其中部分选项进行跟踪后，下一步操作就是对跟踪响应结果进行监控。比如你想知道自己发送消息的标记结果，需要创建一个 MessageEvenetManager对象，并调用方法[addMessageEventNotificationListener](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/MessageEventManager.html" \l "addMessageEventNotificationListener%28org.jivesoftware.smackx.MessageEventNotificationListener%29)([MessageEventNotificationListener](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/MessageEventNotificationListener.html" \o "interface in org.jivesoftware.smackx)  messageEventNotificationListener)。

如果你想监控是否有人给自己发送消息跟踪请求，这时候需要调用方法[addMessageEventRequestListener](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/MessageEventManager.html" \l "addMessageEventRequestListener%28org.jivesoftware.smackx.MessageEventRequestListener%29)([MessageEventRequestListener](http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/org/jivesoftware/smackx/MessageEventRequestListener.html" \o "interface in org.jivesoftware.smackx) messageEventRequestListener)。

### 2.2.3离线消息

发送消息时，用户不在线，系统会自动保存这些消息。当用户登录后，用户需要主动去服务器获取离线消息。主要用到的接口是OfflineMessageMananger。

这里有一个需要注意的地方，那就是用户login时不能发送Presence（用户状态，这个下章讲），否则收不到离线消息。

具体可以看下面例子：

ConnectionConfiguration connConfig = new ConnectionConfiguration("localhost");

connConfig.setSendPresence(false);//设置不发送Presence

Connection conn2 = new XMPPConnection(connConfig);

Conn2.connect();

Conn2.login("zhang", "111");

OfflineMessageManager omm = new OfflineMessageManager(conn2);//创建一个对象

Iterator<Message> ommlist = omm.getMessages();//获取消息，结果为一个Message迭代器

omm.deleteMessages();//将服务器上的离线消息删除。

如果不执行最后一步操作的话，下次登录这些离线消息还在。

## 2.3文件传输

用户可能希望向其它用户发送文件。其它用户有接受，拒绝，或忽略用户的请求。Smack为用户轻松发送文件提供了一个简单的接口。暂只实现文件传输，没有实现文件夹传输。

### 2.3.1发送文件

要发送或接受文件首先必须创建FileTransferManager类的一个对象，创建方法是FileTransferManager(XMPPConnection connection)。假设现在livsun创建了一个文件传输管理对象fileTransferManager = new FileTransferManager(conn);现在需要创建一个输出的文件传输对象来发送文件。

OutgoingFileTransfer outfile = fileTransferManager(UserID)，UserID为完全ID，如果要发送文件给livsun1的话，这里UserID为”livsun1@z00189374/Smack”。

通过调用sendFile(file, description)来发送文件，file是文件对象，description是对文件的描述，可以在接受时获取，并提示用户，可以根据描述来选择是否接受。

一个完整的发送文件的过程如下：

FileTransferManager fileTransferManager = new FileTransferManager(conn);

OutgoingFileTransfer outfile =

manager.createOutgoingFileTransfer("livsun1@z00189374/Smack ");

outfile.sendFile(new File("fileToBeSended.txt"), "You won't believe this!");

### 2.3.2接受文件

如果想知道是否有人发送文件给自己，首先同样需要创建一个FileTransferManager对象，如FileTransferManager fileTransferManager1 = new FileTransferManager(conn1);然后对该对象注册一个监听器FileTransferListener，来监听文件传输请求。这个监听器在有文件传输时被触发，它只有一个方法fileTransferRequest(FileTransferRequest request)，这个方法来决定文件时接受还是拒绝。

如果要拒绝接受，只要简单的request.reject()就可以了。

如果要接受，通过request.accept()创建一个IncomingFileTransfer对象，代码实现：

IncomingFileTransfer infile = request.accept();然后infile调用recieveFile(file)来讲文件接受到file中。同时还可以对文件传输的状态进行监听。

监听及接受文件的代码实现：

final FileTransferManager fileTransferManager1 = new FileTransferManager(conn);

fileTransferManager1.addFileTransferListener(new FileTransferListener() {

public void fileTransferRequest(FileTransferRequest request) {

// 监查此请求是否应该被接受,shouldAccept由自己实现

if(shouldAccept(request)) {//接受则接受文件

IncomingFileTransfer infile = request.accept();

infile.recieveFile(new File("FILE\_PATH"+infile.getName()));

} else {

request.reject();

}

}

});

### 2.3.3文件传输状态监听

在文件传输过程中，发送方活接收方需要知道文件传输的进度和状态，需要通过FileTransfer类来实现。

IncomingFileTransfer和OutgoingFileTransfer都继承了FileTransfer类，此类提供一些方法监视文件传输的进展：

getStatus() - 文件传输可以有多种状态，协商，拒绝，取消，传输过程中，错误和完成。这个方法会文件传输的当前状态。

getProgress() - 如果文件在传输过程中，这个方法会返回一个0到1的数字，0表示传输还没开始，1表示传输完成。如果没在传输过程中也可能返回-1。

isDone() - 和getProgress()方法类似，不同的是它返回boolean。如果状态为拒绝，取消，错误或完成返回真，否则返回假。

getError() - 如果在传输过程中发生错误，这个方法将会返回所发生的错误的类型。

如在接收方监视文件状态，代码实现如下：

while(!infile.isDone()) {

if(infile.getStatus().equals(Status.ERROR)) {

System.out.println("ERROR!!! " + transfer.getError());

} else {

System.out.println(infile.getStatus());

System.out.println(infile.getProgress());

}

}

# 3.多人聊天

MultiUserChat，即多人聊天，通过一个用户创建群组，并邀请其他用户进入群组，或者其他用户可以自由进入群组，并在群组里聊天。功能有创建房间、邀请、监听邀请或拒绝、权限更改、身份改变等。

## 3.1创建多人聊天房间

用户可以创建两种多人聊天房间：即时房间和永久房间。即时房间按照默认的设置立马生成，但是在所有参与用户下线后，该房间注销。永久房间是创建者通过自己的设置生成，创建者是第一个参与者，并且是该房间的Owner。

要想创建一个房间，首先需要创建一个MultiUserChat的对象 ，MultiUserChat类的构造函数需要两个参数，当前用户的连接和房间的JID。如

MultiUserChat chatRoom = new MultiUserChat(conn, “myGroup@conference .z00189374”);

通过chatRoom调用create(“nickname”)来创建房间，其中nickName为用户在房间里显示的昵称。

为了使房间成为永久房间并且更加符合用户的习惯，用户需要对房间进行一些设置。第一步是获取房间的配置表格，代码为

Form form = chatRoom.getConfigurationForm();

再根据配置表格生成一个用来提交配置的表格，代码为

Form submitForm = form.createAnswerForm();

在对submifForm进行设置一些属性，属性包含但不限于以下几个：

//房间的名称

submitForm.setAnswer("muc#roomconfig\_roomname", "TestUserNumber");

//保证只有注册的昵称才能进入房间

submitForm.setAnswer("x-muc#roomconfig\_reservednick",**true**);

//设置为永久房间

submitForm.setAnswer("muc#roomconfig\_persistentroom",**true**);

//设置房间人数上限，注意，参数不是int！！！！！

submitForm.setAnswer("muc#roomconfig\_maxusers",list);

最后通过chatRoom调用sendConfigurationForm(submitForm)来完成对房间的配置。

## 3.2加入聊天室

为了在聊天室里接受或发送消息，首先需要进入聊天室。进入房间之前，先通过用户连接和房间JID来创建一个MultiUserChat的对象。

MultiUserChat chatRoom1 = new MultiUserChat(conn1, “myGroup@ conference .z00189374”);

有三种进入房间的方式：

一、直接加入不需要密码的房间chatRoom1.join(“nickname”);

二、加入需要密码的房间chatRoom1.join(“nickName”,”password”);

三、加入需要密码的房间，且同时设置接受聊天记录的数目，在一、二中，没有设置这项，接受的数目有服务器决定。

DiscussionHistory history = new DiscussionHistory();

history.setMaxStanzas(5);

muc2.join("testbot2","password",history, SmackConfiguration.getPacketReplyTimeout());

## 3.3 群组邀请及群组邀请监听

如果你建了一个群，你可能希望你的某些好友也加入到这个群，你可以通过对他们发送邀请并监听结果来实现这个功能。

首先加入一个房间，muc.join(“livsun”)，并且确认你有这个房间的邀请资格。然后通过muc调用invite(user,reason)方法，其中user是被邀请人的JID，reason是邀请原因。为了知道邀请结果，这里只能获取对面拒绝的结果，在发出邀请之前，需要对multiuserChat对象注册一个监听器muc.addInvitationRejectionListener(new InvitationRejectionListener() )；

为了获取来自其他人的群组邀请，用户可以直接通过自己的连接对MultiUserChat注册一个监听器。如MultiUserChat.addInvitationListener(conn1, new InvitationListener() )。

下面看一个livsun对livsun1发起邀请，livsun1拒绝对例子：

MultiUserChat muc = new MultiUserChat(conn, “maGroup@conference.z00189374”);

muc.join("livsun");

muc.addInvitationRejectionListener(new InvitationRejectionListener() {

public void invitationDeclined(String invitee, String reason) {

// 被拒绝了

}

});

//发出邀请

muc.invite("livsun1@z00189374/Smack", "For Dota");

//下面是livsun1的处理

MultiUserChat.addInvitationListener(conn1, new InvitationListener() {

public void invitationReceived(Connection conn, String room, String inviter, String reason, String password) {

// 对请求进行处理，这里是拒绝了

MultiUserChat.decline(conn, room, inviter, "I'm working delay");

//接受并加入

//MultiUserChat muc1 = new MultiUserChat(conn, room);

//muc1.join(“livsun1”);

}

});

## 3.4 在群组中开始一个单人聊天

在群组列表中，你可能想对某个人发起单独对话，这时可以使用下面方法实现：

Chat chat = muc.createPrivateChat("myGroup@conference.z00189374/livsun1");

chat.sendMessage("Hello there");

不过这样livsun1收到的消息并不是来自于livsun，而是来自myGroup@conference.z00189374

## 3.5更改房间成员权限

一个典型的群组具有三种身份的人：创建者、管理员和成员。房间创建者可以改变房间配置、授予用户所有权和管理权限以及毁掉此房间。房间管理员可以禁止或授予用户权限和主持者权限。房间成员仅能允许用户加入房间。这里用到的方法都是MultiUserChat的方法，具体可以去看Smack的API文档。

还可以对房间其他成员或自己的权限改变进行监听，方法分别为[addParticipantStatusListener](http://www.igniterealtime.org/builds/smack/docs/3.1.0/javadoc/org/jivesoftware/smackx/muc/MultiUserChat.html" \l "addParticipantStatusListener%28org.jivesoftware.smackx.muc.ParticipantStatusListener%29)([ParticipantStatusListener](http://www.igniterealtime.org/builds/smack/docs/3.1.0/javadoc/org/jivesoftware/smackx/muc/ParticipantStatusListener.html" \o "interface in org.jivesoftware.smackx.muc) listener)和[addUserStatusListener](http://www.igniterealtime.org/builds/smack/docs/3.1.0/javadoc/org/jivesoftware/smackx/muc/MultiUserChat.html" \l "addUserStatusListener%28org.jivesoftware.smackx.muc.UserStatusListener%29)([UserStatusListener](http://www.igniterealtime.org/builds/smack/docs/3.1.0/javadoc/org/jivesoftware/smackx/muc/UserStatusListener.html" \o "interface in org.jivesoftware.smackx.muc) listener)

## 3.6获取自己曾经加入的房间

MultiUserChat没有提供方法获取自己曾经加入的房间，它的getJoinedRooms只能获取一个好友当前正在加入的房间。而当一个用户登陆时，用户需要获取自己加入过的房间列表，这里需要用到收藏夹Bookmarks。

### 3.6.1保存房间信息

首先需要创建一个Bookmarks的对象，BookeMarks bm = new BookeMarks()。然后创建BookmarkedConference对象bmConference来保存群组信息。

通过bm调用[addBookmarkedConference](http://www.igniterealtime.org/builds/smack/docs/3.1.0/javadoc/org/jivesoftware/smackx/bookmark/Bookmarks.html" \l "addBookmarkedConference%28org.jivesoftware.smackx.bookmark.BookmarkedConference%29)([BookmarkedConference](http://www.igniterealtime.org/builds/smack/docs/3.1.0/javadoc/org/jivesoftware/smackx/bookmark/BookmarkedConference.html" \o "class in org.jivesoftware.smackx.bookmark) bookmarkedConference)方法，来讲一个BookmarkedConference对象加入Bookmarks中。

最后要把Bookmarks的信息保存到私人专有数据中。先创建PrivateDataManager的一个对象pdmanager，通过pdmanager调用方法setPrivateData(bm)来将数据保存到服务器。

具体代码实现如下：

//创建收藏夹对象

Bookmarks bookmarks = **new** Bookmarks();

//创建聊天群组收藏,参数分别为房间的名字，JID，是否自动进入房间，用户昵称，密码

BookmarkedConference conference1 = **new** BookMarkMain("groupName","myGroup@conference.z00189374",**true**,"nickname",**null**);

bookmarks.addBookmarkedConference(conference1);

PrivateDataManager manager = **new** PrivateDataManager(conn);

manager.setPrivateData(bookmarks);

### 3.6.2获取房间信息

当用户登录时，用户去privateData获取自己的房间信息。用户需要提供Bookmarks的解析方法来正确获取保存在PrivateData中的Bookmarks信息。具体方法如下

Bookmarks bm = new Bookmarks();

PrivateDataManager manager = new PrivateDataManager(conn);

//提供对Bookmarks的IQ包的解析方法

manager.addPrivateDataProvider(bm.getElementName(),

bm.getNamespace(), new Bookmarks.Provider());

//获取Bookmarks信息

bm = (Bookmarks)manager.getPrivateData(bm.getElementName(),

bm.getNamespace());

//获取Bookmarks中群组的信息。

Collection<BookmarkedConference> c = bm.getBookmarkedConferences();

# 4.联系人

每一个用户都有一个Roster列表，包含多个RosterEntry。

在roster中每个用户用一个RosterEntry表示，它包括：

一个XMPP地址(例如 [livsun@z00189374](mailto:livsun@z00189374))。

一个您分配给用户的昵称(例如 "2b")。

登陆所属的roster组列表。如果roster登陆不属于任何组，它将被称为“unfiled entry”。 在openfire中一个RosterEntry可以同时属于多个分组。

## 4.1获取联系人

当用户通过一个连接登录服务器后，用户可以从服务器获取自己的Roster列表。

Roster roster = conn.getRoster();

获取Roster后可以通过调用roster.getEntries()方法来获取所有RosterEntry对象，返回类型为Collection<RosterEntry>。

Collection<RosterEntry> rosterentrys = roster.getEntries();

如果想获得分组信息，需要调用roster.getGroups(), 返回结果是一个Collection<RosterGroup>。可以在通过rosterGroup.getEntries()获取每个分组的成员。

对于每个成员的状态信息，如是否在线，签名等，可通过roster.gerPresence(RosterEntry)获取。

## 4.2管理好友

用户可能需要添加其他用户到自己的Roster中，并可以获取这些用户的状态更新。Smack使用了一种订阅的Presence的方式来获取状态，这样确保用户隐私，因为只有允许订阅才能获取状态。

对于订阅请求，用户有三种处理方式：接受所有、拒绝所有和手动处理。这需要通过通过Roster.setSubscriptionMode(int subscriptionMode)方法来设置。

添加好友就是一个互相发送状态订阅消息的过程。通过调用roster.createEntry(JID,nickname,group)来将一个用户添加到自己的roster中，并向这个用户发送一个订阅presence的请求。

删除好友只需roster.removeEntry(RosterEntry)就可以将用户从自己的roster中删除。

如果用户想把某位好友从一个群组移动到另外一个群组里，需要用到RosterGroup的removeEntry()和addEntry()方法。假设当前群组名称为groupNow，即将移入的群组名称为groupTo，则代码实现如下

//假设被移动的成员为一个RosterEntry对象rosterEntryMove

RosterGroup gourpNow = roster.getGroup(“groupNow”);

RosterGroup gourpTo = roster.getGroup(“groupTo”);

groupNow.removeEntry(rosterEntryMove);

groupTo.addEntry(rosterEntryMove);

## 4.2状态修改

Presence是用来管理用户的在线状态，它有在线和不在线两种状态。如果在线又可以包含很多其他信息，如忙碌、离开等，还可以获取签名。

用户可以通过发送Presence包来改变自己的状态。首先需要创建一个Presence对象。

Presence presence = new Presence(Presence.type.available);

presence.setStatus(“吃饭去”);

presence.setMode(“away”);

conn.sendPacket(presence);

这里用户发送了一个Presence包，更新自己状态为离开，且发表即时心情为“吃饭去”。.

## 4.3隐私设置与黑名单的实现

Privacy是管理其他用户与自己通信的方法。它是由用户定义，可以获取、修改或删除的，保存在服务器端的隐私设置。它基于三种不同的范围JID、Group和签名类型来管理隐私。其中设置包含消息、上线通知、IQ包或所有通信。要实现隐私管理首先需要了解他的三个API

PrivacyListManager: 这是重新获得并处理服务器隐私列表的主API类。

PrivacyList: 代表一个隐私列表，有一个名字，一组隐私项目。例如，可见或不可见列表。PrivacyItem:阻挡或允许隐私的某个方面。例如，允许我的的朋友看到我们的出席状态。

首先要从服务器上获取用户的PrivacyListManager

PrivacyListManager plm = PrivacyListManager.getInstanceFor(conn);

再创建一个隐私列表

List<PrivacyItem> privacyItems = new ArrayList<PrivacyItem>();

然后向这个列表中添加隐私设置

//在线对其隐身

PrivacyItem item = new PrivacyItem(“jid”, false, 1);

item.setValue(userJID);

item.setFilterPresenceout(true);

privacyItems.add(item);

最后要将这个隐私列表创建到服务器上，并设置为激活状态。

plm.createPrivacyList(“myList”, privacyItems);

plm.setActiveListName(“myList”);

黑名单实现

PrivacyItem item1 = new PrivacyItem(“group”, false, 1);

item1.setValue(“blacklist”);

item1.setMessage(true);

privacyItems.add(item1);

其他同JID隐私设置一样。

# 5.用户信息管理

用户信息包含用户的姓名、昵称、住址、电话、头像和密码等。

## 5.1 用户信息

VCard是用来管理用户信息的类。为了获取自己的信息，首先需要新建一个VCard对象，然后通过该对象来加载自己的个人信息。

VCard vCard = new VCard();

vCard.load(conn);

通过get方法可以从VCard对象中获取自己想要的信息。如vCard.getNickName()来获取自己的昵称。

如果想设置或修改自己的信息，需要用VCard的set方法。如vCard.setNickName(“2b”).

在设置完成后，需要将vCard的数据保到服务器中，方法是vCard.save(conn)。

上传头像需要向对图片文件进行处理，将其转化为byte[]数组。假设图片文件已被转化成bytes数组。在将图片用set方法保存前，还需要对其进行编码，具体如下：

//调用smack接口对bytes编码

String encodedImage = StringUtils.encodeBase64(bytes);

//设置

card.setAvatar(bytes, encodedImage);

//保存

card.save(conn);

## 5.2账户信息

账户信息是由AccountManager来管理。它可以创建用户，修改密码，获取用户信息等。

首先根据连接来创建AccountManager对象，再调用它的changePassword(String password)的方法即完成密码修改.

AccountManager accountmanager = new Accountmanager(conn);

accountmanager.changePassword(“new password”);

# 6.与其他IM绑定

Smcak是基于XMPP通信协议的接口，支持与其他基于XMPP的IM软件互通。

## 6.1在服务器端安装插件

从网上下载gateway插件，推荐使用Kraken IM Gateway  ，将Kraken IM Gateway  单独jar包拷贝到openfire的plugin目录下，重启openfire即可；记住，这样会导致服务暂时终止。

## 6.2 在客户端获取支持绑定的IM

首先通过ServiceDiscoveryManager来查询当前服务器上的提供服务主体。创建方法如下：

ServiceDiscoveryManager sdm = ServiceDiscoveryManager.getInstanceFor(conn);

再通过sdm调用discoverItems(entityID)，其中entityID代表a given XMPP entity addressed by its JID，本例中为“z00189374”，来获取包含所有items的DiscoveryItems对象 。代码为：

DiscoveryItems discoveryItems = sdm.discoveryItems(“z00189374”);

再通过discoveryItems.getItems()来获取关于所有Items 的迭代器。

Iterator<DiscoverItems.Item> i = di.getItems();

在对每个Item进行判断，观察是否是一种IM绑定支持。

String entityName = item.getEntityID();

if (entityName.startsWith("msn."))