Android之基于XMPP协议即时通讯软件(三)

分类： Android 2013-12-22 23:55 1657人阅读 评论(17) 收藏 举报

CSDN博客之星投票请移驾：http://vote.blog.csdn.net/blogstaritem/blogstar2013/weidi1989

本文主要介绍本应用的控制层具体实现。如需了解项目结构与框架，请移步之前系列文章：

Android之基于XMPP协议即时通讯软件(一)

Android之基于XMPP协议即时通讯软件(二)

另外，本项目已经升级到V1.0.1，已同步到开源中国代码托管:http://git.oschina.net/way/XMPP

今后更新也只会在此处同步，不会再打包上传到csdn，敬请悉知！

之前给大家介绍过，该小应用采用的是MVC设计模式，所以今天就跟大家分享一下控制层的具体实现。控制层担当一个非常重要的角色，既要处理界面传递过来的任务：点击发送消息、切换在线状态等，又要处理服务器发送过来的消息：有好友上线、收到新消息、保持长连接、掉线自动连接等。概括的说，总共分为以下四步：

①.实例化对象，作一些参数配置。

②.开始连接服务器，实现登陆。

③.注册各种事件监听，比如联系人动态变化、各种消息状态监听、开启长连接任务、掉线自动连接等。

④.用户主动退出，注销登录，断开连接。

第一步很简单，当用户启动该应用时，即启动本应用关健服务，并与界面Activity完成绑定，同时完成xmpp的参数配置，我这里是放在类的静态块里面完成的：

```java
static {
    registerSmackProviders();
}

// 做一些基本的配置
static void registerSmackProviders() {
    ProviderManager pm = ProviderManager.getInstance();
    // add IQ handling
    pm.addIQProvider("query", "http://jabber.org/protocol/disco#info",
            new DiscoverInfoProvider());
    // add delayed delivery notifications
    pm.addExtensionProvider("delay", "urn:xmpp:delay",
            new DelayInfoProvider());
    pm.addExtensionProvider("x", "jabber:x:delay", new DelayInfoProvider());
    // add carbons and forwarding
    pm.addExtensionProvider("forwarded", Forwarded.NAMESPACE,
            new Forwarded.Provider());
    pm.addExtensionProvider("sent", Carbon.NAMESPACE, new Carbon.Provider());
    pm.addExtensionProvider("received", Carbon.NAMESPACE,
            new Carbon.Provider());
    // add delivery receipts
```

个人资料

weidi1989

访问： 273862次
积分： 5493分
排名： 第1002名

原创： 144篇 转载： 3篇
译文： 1篇 评论： 1070条

2014年01月 (1)
2013年12月 (5)
2013年11月 (2)
2013年10月 (4)
展开

阅读排行

Android之高仿手机QQ聊 (20132)
Android之基于百度云推送 (11998)
Android之高仿墨迹天气 (11345)
Android之消息推送实现 (9524)
Android之网络学习资源 (6065)
Android之高仿微信聊天 (5935)
Android之BaseExpanda (5064)
Android将程序崩溃信息 (4870)
Java之Socket简单聊天实 (4737)
Android之指南针学习 (4210)

评论排行

Android之高仿手机QQ聊 (228)
Android之基于百度云推送 (130)
Android之网络学习资源 (50)
Android之简洁天气 (47)
Android之极致拟物化空 (36)
Android之基于XMPP协议 (34)
Java之Socket简单聊天实 (34)
Android之高仿墨迹天气 (32)
Android之基于XMPP协议 (22)
Android之仿网易V3.5新 (22)

最新评论

Android4.0设置界面修改总结
tiaozhanmaidi: 楼主 请问一下 你的settings的标题栏的背景是如何修改的，急用，谢谢了。

Android4.0设置界面修改总结
weidi1989: @skyinmyheart:学Android，首先就需要入门咯，入门之后，可以在网上下一些简单的例子...

Android4.0设置界面修改总结
weidi1989: @manzer_zhou:没有源码，但是关键代码都已经分享，相信你认真看看，研究一下，绝对能实现这样...

Android4.0设置界面修改总结
weidi1989: @tiaozhanmaidi:关键代码已经分享出来了，源码分享出来，你也编不过。

Android4.0设置界面修改总结
tiaozhanmaidi: 确实非常不错，请问楼主可以共享一下源代码吗

Android之高仿手机QQ聊天
qq530918474: 大神，我把客户端安装在手机上，无法运行，请问这个问题该怎么解决？

Java之Socket简单聊天实现（QC
fangqingyin: 不错 看上去很Nb 马上试试

ViewPager切换动画PageTransf
各种代码各种敲: 为什么我跑起来没效果呢？

Android4.0设置界面修改总结
manzer_zhou: 有源代码吗？有发一份

Android之基于XMPP协议即时通
北鸣游羽: 楼主 账号密码不是很

```
    pm.addExtensionProvider(DeliveryReceipt.ELEMENT,
            DeliveryReceipt.NAMESPACE, new DeliveryReceipt.Provider());
    pm.addExtensionProvider(DeliveryReceiptRequest.ELEMENT,
            DeliveryReceipt.NAMESPACE,
            new DeliveryReceiptRequest.Provider());
    // add XMPP Ping (XEP-0199)
    pm.addIQProvider("ping", "urn:xmpp:ping", new PingProvider());

    ServiceDiscoveryManager.setIdentityName(XMPP_IDENTITY_NAME);
    ServiceDiscoveryManager.setIdentityType(XMPP_IDENTITY_TYPE);
}
```

第二步，当用户输入账号密码时，在服务中开启新线程启动连接服务器，传入参数信息(服务器、账号密码等)实现登录，同时会将登陆成功与否信息通过回调函数通知界面。也是比较简单的：

```
[java]
public boolean login(String account, String password) throws XXException {// 登陆实现
        try {
            if (mXMPPConnection.isConnected()) {// 首先判断是否还连接着服务器，需要先断开
                try {
                    mXMPPConnection.disconnect();
                } catch (Exception e) {
                    L.d("conn.disconnect() failed: " + e);
                }
            }
            SmackConfiguration.setPacketReplyTimeout(PACKET_TIMEOUT);// 设置超时时间
            SmackConfiguration.setKeepAliveInterval(-1);
            SmackConfiguration.setDefaultPingInterval(0);
            registerRosterListener();// 监听联系人动态变化
            mXMPPConnection.connect();
            if (!mXMPPConnection.isConnected()) {
                throw new XXException("SMACK connect failed without exception!");
            }
            mXMPPConnection.addConnectionListener(new ConnectionListener() {
                public void connectionClosedOnError(Exception e) {
                    mService.postConnectionFailed(e.getMessage());// 连接关闭时，动态反馈给服务

                }

                public void connectionClosed() {
                }

                public void reconnectingIn(int seconds) {
                }

                public void reconnectionFailed(Exception e) {
                }

                public void reconnectionSuccessful() {
                }
            });
            initServiceDiscovery();// 与服务器交互消息监听,发送消息需要回执，判断是否发送成功
            // SMACK auto-logins if we were authenticated before
            if (!mXMPPConnection.isAuthenticated()) {
                String ressource = PreferenceUtils.getPrefString(mService,
                        PreferenceConstants.RESSOURCE, XMPP_IDENTITY_NAME);
                mXMPPConnection.login(account, password, ressource);
            }
            setStatusFromConfig();// 更新在线状态

        } catch (XMPPException e) {
            throw new XXException(e.getLocalizedMessage(),
                    e.getWrappedThrowable());
        } catch (Exception e) {
            // actually we just care for IllegalState or NullPointer or XMPPEx.
            L.e(SmackImpl.class, "login(): " + Log.getStackTraceString(e));
            throw new XXException(e.getLocalizedMessage(), e.getCause());
        }
        registerAllListener();// 注册监听其他的事件，比如新消息
        return mXMPPConnection.isAuthenticated();
    }
```

第三步比较关键，登陆成功后，我们就必须要监听服务器的各种消息状态变化，以及要维持自身的一个稳定

理解 什么意思。。自定义的吗 还是用你的

性，即保持长连接和掉线自动重连。下面是注册所有监听的函数：

```java
private void registerAllListener() {
    // actually, authenticated must be true now, or an exception must have
    // been thrown.
    if (isAuthenticated()) {
        registerMessageListener();// 注册新消息监听
        registerMessageSendFailureListener();// 注册消息发送失败监听
        registerPongListener();// 注册服务器回应ping消息监听
        sendOfflineMessages();// 发送离线消息
        if (mService == null) {
            mXMPPConnection.disconnect();
            return;
        }
        // we need to "ping" the service to let it know we are actually
        // connected, even when no roster entries will come in
        mService.rosterChanged();
    }
}
```

①.注册联系人动态变化监听：第一次登陆时要同步本地数据库与服务器数据库的联系人，同时处理连接过程中联系人动态变化，比如说好友切换在线状态、有人申请加好友等。我这里没有将动态变化直接通知到界面线程，而是直接更新联系人数据库Roster.db，因为：我在界面线程监听了联系人数据库的动态变化，这就是ContentProvider的好处，下篇文章细说，这里就只提及一下。下面是关键部分代码：

```java
private void registerRosterListener() {
    mRoster = mXMPPConnection.getRoster();
    mRosterListener = new RosterListener() {
        private boolean isFristRoter;

        @Override
        public void presenceChanged(Presence presence) {// 联系人状态改变，比如在线或离开、隐身之类
            L.i("presenceChanged(" + presence.getFrom() + "): " + presence);
            String jabberID = getJabberID(presence.getFrom());
            RosterEntry rosterEntry = mRoster.getEntry(jabberID);
            updateRosterEntryInDB(rosterEntry);// 更新联系人数据库
            mService.rosterChanged();// 回调通知服务，主要是用来判断一下是否掉线
        }

        @Override
        public void entriesUpdated(Collection<String> entries) {// 更新数据库，第一次登陆
            // TODO Auto-generated method stub
            L.i("entriesUpdated(" + entries + ")");
            for (String entry : entries) {
                RosterEntry rosterEntry = mRoster.getEntry(entry);
                updateRosterEntryInDB(rosterEntry);
            }
            mService.rosterChanged();// 回调通知服务，主要是用来判断一下是否掉线
        }

        @Override
        public void entriesDeleted(Collection<String> entries) {// 有好友删除时，
            L.i("entriesDeleted(" + entries + ")");
            for (String entry : entries) {
                deleteRosterEntryFromDB(entry);
            }
            mService.rosterChanged();// 回调通知服务，主要是用来判断一下是否掉线
        }

        @Override
        public void entriesAdded(Collection<String> entries) {// 有人添加好友时，我这里没有弹出对话框确认，直接添加到数据库
            L.i("entriesAdded(" + entries + ")");
            ContentValues[] cvs = new ContentValues[entries.size()];
            int i = 0;
            for (String entry : entries) {
                RosterEntry rosterEntry = mRoster.getEntry(entry);
                cvs[i++] = getContentValuesForRosterEntry(rosterEntry);
            }
            mContentResolver.bulkInsert(RosterProvider.CONTENT_URI, cvs);
            if (isFristRoter) {
```

```
                isFristRoter = false;
                mService.rosterChanged();// 回调通知服务，主要是用来判断一下是否掉线
            }
        }
    };
    mRoster.addRosterListener(mRosterListener);
}
```

②.注册消息监听，也跟联系人动态监听是一样的处理方式，将消息的动态变化同步到消息数据库Chat.db，并未直接通知界面，界面也是通过监听数据库变化来作出动态变化的。下面是关键代码：

[java]

```
private void registerMessageListener() {
        // do not register multiple packet listeners
        if (mPacketListener != null)
            mXMPPConnection.removePacketListener(mPacketListener);

        PacketTypeFilter filter = new PacketTypeFilter(Message.class);

        mPacketListener = new PacketListener() {
            public void processPacket(Packet packet) {
                try {
                    if (packet instanceof Message) {// 如果是消息类型
                        Message msg = (Message) packet;
                        String chatMessage = msg.getBody();

                        // try to extract a carbon
                        Carbon cc = CarbonManager.getCarbon(msg);
                        if (cc != null
                                && cc.getDirection() == Carbon.Direction.received) {// 收到的消息
                            L.d("carbon: " + cc.toXML());
                            msg = (Message) cc.getForwarded()
                                    .getForwardedPacket();
                            chatMessage = msg.getBody();
                            // fall through
                        } else if (cc != null
                                && cc.getDirection() == Carbon.Direction.sent) {// 如果是自己发送的消息，则添加到数据库后直接返回
                            L.d("carbon: " + cc.toXML());
                            msg = (Message) cc.getForwarded()
                                    .getForwardedPacket();
                            chatMessage = msg.getBody();
                            if (chatMessage == null)
                                return;
                            String fromJID = getJabberID(msg.getTo());

                            addChatMessageToDB(ChatConstants.OUTGOING, fromJID,
                                    chatMessage, ChatConstants.DS_SENT_OR_READ,
                                    System.currentTimeMillis(),
                                    msg.getPacketID());
                            // always return after adding
                            return;// 记得要返回
                        }

                        if (chatMessage == null) {
                            return;// 如果消息为空，直接返回了
                        }

                        if (msg.getType() == Message.Type.error) {
                            chatMessage = "<Error> " + chatMessage;// 错误的消息类型
                        }

                        long ts;// 消息时间戳
                        DelayInfo timestamp = (DelayInfo) msg.getExtension(
                                "delay", "urn:xmpp:delay");
                        if (timestamp == null)
                            timestamp = (DelayInfo) msg.getExtension("x",
                                    "jabber:x:delay");
                        if (timestamp != null)
                            ts = timestamp.getStamp().getTime();
                        else
                            ts = System.currentTimeMillis();

                        String fromJID = getJabberID(msg.getFrom());// 消息来自对象

```

```
                        addChatMessageToDB(ChatConstants.INCOMING, fromJID,
                                chatMessage, ChatConstants.DS_NEW, ts,
                                msg.getPacketID());// 存入数据库，并标记为新消息DS_NEW
                        mService.newMessage(fromJID, chatMessage);// 通知service，处理是否需要显示通知栏，
                    }
                } catch (Exception e) {
                    // SMACK silently discards exceptions dropped from
                    // processPacket :(
                    L.e("failed to process packet:");
                    e.printStackTrace();
                }
            }
        };

        mXMPPConnection.addPacketListener(mPacketListener, filter);// 这是最关健的了，少了这句，前面的都是白费功夫
    }
```

③.启动保持长连接任务。我这里与服务器保持长连接，其实是通过每隔一段时间(本应用是15分钟)去ping一次服务器，服务器收到此ping消息，会对应的回复一个pong消息，完成一次ping-pong的过程，我们暂且叫它为心跳。此ping-pong过程有一个唯一的id，用来区分每一次的ping-pong记录。为了保证应用在系统休眠时也能启动ping的任务，我们使用了闹钟服务，而不是定时器，关于闹钟服务具体使用，请参看我之前的博客：Android中的定时器AlarmManager 。具体操作是：

从连上服务器完成登录**15**分钟后，闹钟响起，开始给服务器发送一条**ping**消息**(**随机生成一唯一**ID)**，同时启动超时闹钟**(**本应用是**30+3**秒**)**，如果服务器在**30+3**秒内回复了一条**pong**消息**(**与之前发送的**ping**消息**ID**相同**)**，代表与服务器任然保持连接，则取消超时闹钟，完成一次**ping-pong**过程。如果在**30+3**秒内服务器未响应，或者回复的**pong**消息与之前发送的**ping**消息**ID**不一致，则认为与服务器已经断开。此时，将此消息反馈给界面，同时启动重连任务。实现长连接。

关健代码如下：

```
[java]
    /***************** start 处理ping服务器消息 ***********************/
    private void registerPongListener() {
        // reset ping expectation on new connection
        mPingID = null;// 初始化ping的id

        if (mPongListener != null)
            mXMPPConnection.removePacketListener(mPongListener);// 先移除之前监听对象

        mPongListener = new PacketListener() {

            @Override
            public void processPacket(Packet packet) {
                if (packet == null)
                    return;

                if (packet.getPacketID().equals(mPingID)) {// 如果服务器返回的消息为ping服务器时的消息，说明没有掉线
                    L.i(String.format(
                            "Ping: server latency %1.3fs",
                            (System.currentTimeMillis() - mPingTimestamp) / 1000.));
                    mPingID = null;
                    ((AlarmManager) mService
                            .getSystemService(Context.ALARM_SERVICE))
                            .cancel(mPongTimeoutAlarmPendIntent);// 取消超时闹钟
                }
            }

        };

        mXMPPConnection.addPacketListener(mPongListener, new PacketTypeFilter(
                IQ.class));// 正式开始监听
        mPingAlarmPendIntent = PendingIntent.getBroadcast(
                mService.getApplicationContext(), 0, mPingAlarmIntent,
                PendingIntent.FLAG_UPDATE_CURRENT);// 定时ping服务器，以此来确定是否掉线
        mPongTimeoutAlarmPendIntent = PendingIntent.getBroadcast(
                mService.getApplicationContext(), 0, mPongTimeoutAlarmIntent,
                PendingIntent.FLAG_UPDATE_CURRENT);// 超时闹钟
        mService.registerReceiver(mPingAlarmReceiver, new IntentFilter(
                PING_ALARM));// 注册定时ping服务器广播接收者
        mService.registerReceiver(mPongTimeoutAlarmReceiver, new IntentFilter(
```

```
                PONG_TIMEOUT_ALARM));// 注册连接超时广播接收者
        ((AlarmManager) mService.getSystemService(Context.ALARM_SERVICE))
                .setInexactRepeating(AlarmManager.RTC_WAKEUP,
                        System.currentTimeMillis()
                                + AlarmManager.INTERVAL_FIFTEEN_MINUTES,
                        AlarmManager.INTERVAL_FIFTEEN_MINUTES,
                        mPingAlarmPendIntent);// 15分钟ping以此服务器
    }

    /**
     * BroadcastReceiver to trigger reconnect on pong timeout.
     */
    private class PongTimeoutAlarmReceiver extends BroadcastReceiver {
        public void onReceive(Context ctx, Intent i) {
            L.d("Ping: timeout for " + mPingID);
            mService.postConnectionFailed(XXService.PONG_TIMEOUT);
            //logout();// 超时就断开连接
        }
    }

    /**
     * BroadcastReceiver to trigger sending pings to the server
     */
    private class PingAlarmReceiver extends BroadcastReceiver {
        public void onReceive(Context ctx, Intent i) {
            if (mXMPPConnection.isAuthenticated()) {
                sendServerPing();// 收到ping服务器的闹钟，即ping一下服务器
            } else
                L.d("Ping: alarm received, but not connected to server.");
        }
    }
public void sendServerPing() {
        if (mPingID != null) {// 此时说明上一次ping服务器还未回应，直接返回，直到连接超时
            L.d("Ping: requested, but still waiting for " + mPingID);
            return; // a ping is still on its way
        }
        Ping ping = new Ping();
        ping.setType(Type.GET);
        ping.setTo(PreferenceUtils.getPrefString(mService,
                PreferenceConstants.Server, PreferenceConstants.GMAIL_SERVER));
        mPingID = ping.getPacketID();// 此id其实是随机生成，但是唯一的
        mPingTimestamp = System.currentTimeMillis();
        L.d("Ping: sending ping " + mPingID);
        mXMPPConnection.sendPacket(ping);// 发送ping消息

        // register ping timeout handler: PACKET_TIMEOUT(30s) + 3s
        ((AlarmManager) mService.getSystemService(Context.ALARM_SERVICE)).set(
                AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
                        + PACKET_TIMEOUT + 3000, mPongTimeoutAlarmPendIntent);// 此时需要
启动超时判断的闹钟了，时间间隔为30+3秒
    }
```

④.如果与服务器连接超时，则进入了我们掉线重连的任务了，因为mService.postConnectionFailed(XXService.PONG_TIMEOUT);回反馈到服务中，此时，我们会判断使用是否开启了掉线重连，关健代码如下，首先将消息由子线程发送到界面线程，文章开头说了，我们的连接是在新的线程中执行的：

```
[java]
public void postConnectionFailed(final String reason) {
        mMainHandler.post(new Runnable() {
            public void run() {
                connectionFailed(reason);
            }
        });
    }

    private void connectionFailed(String reason) {
        L.i(XXService.class, "connectionFailed: " + reason);
        mConnectedState = DISCONNECTED;// 更新当前连接状态
        if (mSmackable != null)
            mSmackable.setStatusOffline();// 将所有联系人标记为离线
        if (TextUtils.equals(reason, LOGOUT)) {// 如果是手动退出
            ((AlarmManager) getSystemService(Context.ALARM_SERVICE))
                    .cancel(mPAlarmIntent);
            return;
```

```
        }
        // 回调
        if (mConnectionStatusCallback != null) {
            mConnectionStatusCallback.connectionStatusChanged(mConnectedState,
                    reason);
            if (mIsFirstLoginAction)// 如果是第一次登录,就算登录失败也不需要继续
                return;
        }

        // 无网络连接时,直接返回
        if (NetUtil.getNetworkState(this) == NetUtil.NETWORN_NONE) {
            ((AlarmManager) getSystemService(Context.ALARM_SERVICE))
                    .cancel(mPAlarmIntent);
            return;
        }

        String account = PreferenceUtils.getPrefString(XXService.this,
                PreferenceConstants.ACCOUNT, "");
        String password = PreferenceUtils.getPrefString(XXService.this,
                PreferenceConstants.PASSWORD, "");
        // 无保存的帐号密码时，也直接返回
        if (TextUtils.isEmpty(account) || TextUtils.isEmpty(password)) {
            L.d("account = null || password = null");
            return;
        }
        // 如果不是手动退出并且需要重新连接，则开启重连闹钟
        if (PreferenceUtils.getPrefBoolean(this,
                PreferenceConstants.AUTO_RECONNECT, true)) {
            L.d("connectionFailed(): registering reconnect in "
                    + mReconnectTimeout + "s");
            ((AlarmManager) getSystemService(Context.ALARM_SERVICE)).set(
                    AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
                            + mReconnectTimeout * 1000, mPAlarmIntent);
            mReconnectTimeout = mReconnectTimeout * 2;
            if (mReconnectTimeout > RECONNECT_MAXIMUM)
                mReconnectTimeout = RECONNECT_MAXIMUM;
        } else {
            ((AlarmManager) getSystemService(Context.ALARM_SERVICE))
                    .cancel(mPAlarmIntent);
        }

    }
```

从上述代码中可以看出，在connectionFailed函数中，我们除了将此消息通知界面，同时会根据不同的reason来判断是否需要重连，如果是用户手动退出reason=LOGOUT，则直接返回咯，否则也是开启一个闹钟，启动重新连接任务，下面是该闹钟的接收处理：

```
[java]
// 自动重连广播接收者
private class ReconnectAlarmReceiver extends BroadcastReceiver {
    public void onReceive(Context ctx, Intent i) {
        L.d("Alarm received.");
        if (!PreferenceUtils.getPrefBoolean(XXService.this,
                PreferenceConstants.AUTO_RECONNECT, true)) {
            return;
        }
        if (mConnectedState != DISCONNECTED) {
            L.d("Reconnect attempt aborted: we are connected again!");
            return;
        }
        String account = PreferenceUtils.getPrefString(XXService.this,
                PreferenceConstants.ACCOUNT, "");
        String password = PreferenceUtils.getPrefString(XXService.this,
                PreferenceConstants.PASSWORD, "");
        if (TextUtils.isEmpty(account) || TextUtils.isEmpty(password)) {
            L.d("account = null || password = null");
            return;
        }
        Login(account, password);
    }
}
```

是不是这样就实现了长连接呢？也许高兴得太早了，我们还有一个重要的因素没有考虑到，对了，就是手机网

络，因为很多手机在系统休眠的时候是会断开网络连接的(应该是为了省电吧)，所以，我们必须要动态监听网络变化，来做出处理,以下是关键代码：

```java
public void onNetChange() {
        if (NetUtil.getNetworkState(this) == NetUtil.NETWORN_NONE) {// 如果是网络断开，不作处理
            connectionFailed(NETWORK_ERROR);
            return;
        }
        if (isAuthenticated())// 如果已经连接上，直接返回
            return;
        String account = PreferenceUtils.getPrefString(XXService.this,
                PreferenceConstants.ACCOUNT, "");
        String password = PreferenceUtils.getPrefString(XXService.this,
                PreferenceConstants.PASSWORD, "");
        if (TextUtils.isEmpty(account) || TextUtils.isEmpty(password))// 如果没有帐号，也直接返回
            return;
        if (!PreferenceUtils.getPrefBoolean(this,
                PreferenceConstants.AUTO_RECONNECT, true))// 不需要重连
            return;
        Login(account, password);// 重连
    }
```

OK，与服务器保持长连接，基本上就是这样了，其实还有一些问题没有考虑到，比如说内存过低，服务被系统回收，我们是没有考虑到的，这个就留个读者一个思考吧，我的想法是：在用户唤醒系统时也启动一次服务，接收此广播：

```
<action android:name="android.intent.action.USER_PRESENT" />
```

⑤.实现服务在前台运行：这个我在之前的文章中有介绍过：Android之后台服务判断本应用Activity是否处于栈顶，这里就不在赘述了。

第四步是用户主动退出，注销登陆，这个好像没有多少需要介绍的，无法是释放掉一些资源，关闭一些服务等等，也无需多说。看看代码即可：

```java
// 退出
    public boolean logout() {
        // mIsNeedReConnection = false;// 手动退出就不需要重连闹钟了
        boolean isLogout = false;
        if (mConnectingThread != null) {
            synchronized (mConnectingThread) {
                try {
                    mConnectingThread.interrupt();
                    mConnectingThread.join(50);
                } catch (InterruptedException e) {
                    L.e("doDisconnect: failed catching connecting thread");
                } finally {
                    mConnectingThread = null;
                }
            }
        }
        if (mSmackable != null) {
            isLogout = mSmackable.logout();
            mSmackable = null;
        }
        connectionFailed(LOGOUT);// 手动退出
        return isLogout;
    }
```

好了，整个控制层大概就讲到这里，总结一下：

重要的是第三步：注册监听和长连接的处理，其中长连接处理也是最为关键和麻烦的。

文章比较长，其实也花了我几个小时的时间，首先感谢你看到了文章末尾，由于个人水平限制，难免会有一些失

误或者不准确的地方，欢迎大家批评指出。

更多 0

上一篇：Android之基于XMPP协议即时通讯软件(二)
下一篇：Android之数据库异步加载利器--Loaders

顶 8 踩 0

查看评论

9楼 北鸣游羽 2014-02-08 10:57发表

openfire 能对后台数据进行 保存操作么？ 比如聊天记录··

8楼 huazai963184709 2014-01-17 14:09发表

> 引用"wulovey"的评论：
> 已投票...
> 你好,现在这个版本的是添加好友,对方不会有提醒吗?
> 是一方申请加为好友,对方也自动把申...

我也出现了 1楼和四楼所说的情况，不知道为什么 博主能指导一下吗？

Re: huazai963184709 2014-01-17 14:11发表

qq:963184709

7楼 诗风悠存 2014-01-07 11:05发表

群主你的jar包好像有点小问题，文件传输与群组的一些功能都没法使用。。

6楼 诗风悠存 2013-12-26 18:34发表

已投票。

5楼 jianhao025 2013-12-24 15:29发表

投了你了，希望以后增加群聊，嘿嘿

4楼 wulovey 2013-12-23 19:30发表

rosterID username jid sub ask recv nick
7 a b 0 0 -1
8 b a 0 0 -1

数据库的表里已经是好友状态了...
但是显示离线,且不能收消息

Re: weidi1989 2013-12-23 19:33发表

回复wulovey：长按好友，重新再发出申请试试。可能中间出现丢包的问题。

Re: wulovey 2013-12-23 19:53发表

回复weidi1989：还是离线...

Re: weidi1989 2013-12-23 23:16发表

回复wulovey：换谷歌的服务器时时，我的mac搭建的服务器也是无法保存好友信息和状态。但是ubuntu和windows都可以

Re: wulovey 2013-12-24 10:00发表

回复weidi1989：我现在的是windows+openfire...
从服务器openfire那发送消息给所有在线的客户端,这个能收到消息..
两个客户端之间就不行

3楼 wulovey 2013-12-23 18:47发表

找到了...CUSTOM_SERVER

2楼 wulovey 2013-12-23 18:40发表

现在连接服务器是自动获取@后面的IP地址,
如果我要直接代码里写死,那应该修改哪里呢,
找了好久,没找到...

Re: weidi1989 2013-12-23 18:46发表

回复wulovey：有一个常量类里面，把google服务器改一下就可以了。

1楼 wulovey 2013-12-23 17:22发表

已投票...
你好,现在这个版本的是添加好友,对方不会有提醒吗?
是一方申请加为好友,对方也自动把申请的加为好友了吗?

Re: weidi1989 2013-12-23 18:46发表

回复wulovey：一直都是没有提示的。

Re: wulovey 2013-12-23 19:13发表

回复weidi1989：你好,感谢你的回复...
我现在在试加好友,
我的Openfire里有a,b两个用户
我分别用虚拟机和手机登陆这两个用户
在a的添加好友的界面,我输入b,然后转到好友列表,可是提示的是离线呀...而且发消息对方也收不到...

发表评论

用户名： u013295109

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

Android 游戏教程:让人物...
Linux 下的 UltraEd...
关于HTML怎样用图片做背景
Android 游戏教程:让人物...
HTML5应用性能调优工具WAP...
Android应用开发

<< >>

更多招聘职位

【深圳市职帮人力资源有限公司】Java
【浪腾信息】asp.net 程序员
【英特尔亚太研发中心（上海）】Big
【猎头中国】虚拟化资深设计师/架构师
【畅捷通信息技术股份有限公司】JAVA
【北京搜狗科技发展有限公司】大数据

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK