链滴

MQTT实现消息推送

作者：jetlan

原文链接：https://ld246.com/article/1375620402722

来源网站：链滴

MQTT实现消息接收(接收消息需实现MqttSimpleCallback接口并实现它的publishArrived方法)须注册接收消息方法
<pre>[java]

mqttClient.registerSimpleHandler(simpleCallbackHandler);
// 册接收消息方法

[/java]</pre>
和订阅接主题
<pre>[java]</pre>
mqttClient.subscribe(TOPICS, QOS_VALUES);

// 阅接主题
<pre>[/java]</pre>
服务端：
<pre>[java]</pre>
package com.gmcc.kuchuan.business;

import org.pache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.ibm.mqtt.MqttClient;
import com.ibm.mqtt.MqttException;
impot com.ibm.mqtt.MqttSimpleCallback;

/**
 * MQTT消息发送与接收
 * @uthor Join
 *
 */
public class MqttBroker {
 private final static Log logger = LogFactory.getLog(MqttBroker.class);// 日志对象
 // 连接参数
 private final static String CONNECTION_STRING = "tcp://localhost:9901";
 private final static boolean CLEAN_START = true;
 private final static short KEEP_ALIVE = 30;// 低耗网络，是又需要及时获取数据，心跳30s
 private final static String CLIENT_ID = "master";// 客户端标识
 private final static int[] QOS_VALUES = { 0, 0, 2, 0 };// 对应主题的消息别
 private final static String[] TOPICS = { "Test/TestTopics/Topic1",
 quot;Test/TestTopics/Topic2", "Test/TestTopics/Topic3",
 "clien/keepalive" };
 private static MqttBroker instance = new MqttBroker();

 private MqttClient mqttClient;

/**
 * 返回实例对象
 *
 * @return
 */
 public static MqttBroker getInstance() {
 return instance;
 }

/**
 * 重新连接服务
 */
 private void connect() throws MqttException {
 logger.info("connect to mqtt broker.");
 mqttClient = new MqttClient(CNNECTION_STRING);
 logger.info("**********register Simple Handler**********");
 SimpleCallbackHandler simpleCallbackHandler = new SimpleCallbackHandler();
 mqttClient.registerSimpleHandler(simpleCallbackHandler);// 注册接收消息方法
 mqttClient.connect(CLIENT_ID, CLEAN_START, KEEP_ALIVE);
 logger.info("**********subscribe receiver topics**********");
 mqttClient.subscribe(TOPICS, QOS_VALUES);// 订阅接主题

logger.info("**********CLIENT_ID:" + CLIENT_ID);
 /*
 * 完成订阅后，可以增加心跳，保持网络通畅，也可以发布自己的消息
 */
 mqttClient.publish("keepalive", "keepalive".getBytes(), QOS_VALUES[0],
 true);// 增加心跳，保持网络通畅
 }

/**
 * 发送消息
 *
 * @param clientId
 * @param messageId
 */
 public void sendMessage(String clientId, String message) {
 try {
 if (mqttClient == null || !mqttClient.isConnected()) {
 connect();
 }

logger.info("send message to " + clientId + ", message is "
 + message);
 // 发布自己的消息
 mqttClient.publish("GMCC/client/" + clientId, message.getBytes(),
 0, false);
 } catch (MqttException e) {
 logger.error(e.getCause());
 e.printStackTrace();
 }
 }

/**
 * 简单回调函数，处理server接收到的主题消息
 *
 * @author Join
 *
 */
 class SimpleCallbackHandler implements MqttSimpleCallback {

/**
 * 当客户机和broker意外断开时触发 可以再此处理重新订阅
 */
 Override
 public void connectionLost() throws Exception {
 // TODO Auto-generatd method stub
 System.out.println("客户机和broker已经断开");
 }

/**
 * 客户端订阅消息后，该方法负责回调接收处理消息
 */
 @Override
 public void publishArrived(String topicName, byte[] payload, int Qos,
 boolean retaned) throws Exception {
 // TODO Auto-generated method stub
 System.out.println("订阅主题: " + topicName);
 System.out.println("消息数据: " + ew String(payload));
 System.out.println("消息级别(0,1,2): " + Qos);
 System.out.println("是否是实时发送的消息(false=实时，true=服务器上保留的最后消息): &quot
 + retained);
 }

}

public static void main(String[] args) {
 new MqttBroker().sendMessage("client", "message");
 }
}
<pre>
[/java]</pre>

Android客户端：
核心代码：MQTTConnection内部类
<pre>[java]

<pre class="java" name="code">impo
t java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.RandomAccessFile;
import java.net.Ht
pURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java
util.Timer;
import java.util.TimerTask;

import android.app.AlarmManager;<
r />import android.app.Notification;
import android.app.NotificationManager;
im
ort android.app.PendingIntent;
import android.app.Service;
import android.conte
t.BroadcastReceiver;
import android.content.Context;
import android.content.Inte
t;
import android.content.IntentFilter;
import android.content.SharedPreferences;
br />import android.database.Cursor;
import android.net.ConnectivityManager;
i
port android.net.NetworkInfo;
import android.os.Binder;
import android.os.Bundle

import android.os.IBinder;
import android.provider.ContactsContract;
impo
t android.util.Log;
//此部分项目导包已被删除</pre><pre class="java"
name="code">import com.ibm.mqtt.IMqttClient;
import com.ibm.mqtt
MqttClient;
import com.ibm.mqtt.MqttException;
import com.ibm.mqtt.MqttPersi
tence;
import com.ibm.mqtt.MqttPersistenceException;
import com.ibm.mqtt.Mqt
SimpleCallback;

/*
 * PushService that does all of the work.
 * Most of
the logic is borrowed from KeepAliveService.
 * http://code.google.com/p/android-ran
om/source/browse/trunk/TestKeepAlive/src/org/devtcg/demo/keepalive/KeepAliveService.ja
a?r=219
 */
public class PushService extends Service {
    private MyBinder
Binder = new MyBinder();
    // this is the log tag
    public static final String TAG =
"PushService";

    // the IP address, where your MQTT broker is runni
g.
    private static final String MQTT_HOST = "120.197.230.53"; // "20
.124.50.174";//
    // the port at which the broker is running.
    private static
nt MQTT_BROKER_PORT_NUM = 9901;
    // Let's not use the MQTT persistence.
private static MqttPersistence MQTT_PERSISTENCE = null;
    // We don't need to rem
mber any state between the connections, so we use a
    // clean start.
    private s
atic boolean MQTT_CLEAN_START = true;
    // Let's set the internal keep alive for MQTT
to 15 mins. I haven't tested
    // this value much. It could probably be increased.
private static short MQTT_KEEP_ALIVE = 60 * 15;
    // Set quality of services to 0 (at m
st once delivery), since we don't want
    // push notifications
    // arrive more th
n once. However, this means that some messages might get
    // lost (delivery is not g
aranteed)
    private static int[] MQTT_QUALITIES_OF_SERVICE = { 0 };
    private st
tic int MQTT_QUALITY_OF_SERVICE = 0;
    // The broker should not retain any message
.
    private static boolean MQTT_RETAINED_PUBLISH = false;

    // MQTT cl
ent ID, which is given the broker. In this example, I also use
    // this for the topic heade
.
    // You can use this to run push notifications for multiple apps with one
    //
QTT broker.
    public static String MQTT_CLIENT_ID = "client";

// These are the actions for the service (name are descriptive enough)
    public static f
nal String ACTION_START = MQTT_CLIENT_ID + ".START";
    private static f
nal String ACTION_STOP = MQTT_CLIENT_ID + ".STOP";
    private static fin
l String ACTION_KEEPALIVE = MQTT_CLIENT_ID
            + ".KEEP_ALIVE";
    private static final String ACTION_RECONNECT = MQTT_CLIENT_ID
            + &quo
;.RECONNECT";

    // Connection log for the push service. Good for debugg
ng.
    private ConnectionLog mLog;

    // Connectivity manager to determi
ing, when the phone loses connection
    private ConnectivityManager mConnMan;
    // Notification manager to displaying arrived push notifications
    private Notificati
nManager mNotifMan;

    // Whether or not the service has been started.
private boolean mStarted;

    // This the application level keep-alive interval, that
is used by the
    // AlarmManager
    // to keep the connection active, even when
the device goes to sleep.
    private static final long KEEP_ALIVE_INTERVAL = 1000 * 60 *
28;

    // Retry intervals, when the connection is lost.
    private static final l

ng INITIAL_RETRY_INTERVAL = 1000 * 10;
  private static final long MAXIMUM_RETRY_NTERVAL = 1000 * 60 * 30;

  // Preferences instance
  private SharedPrefrences mPrefs;
  // We store in the preferences, whether or not the service has been strted
  public static final String PREF_STARTED = "isStarted";
  // We also store the deviceID (target)
  public static final String PREF_DEVICE_ID = "deviceID";
  // We store the last retry interval
  public static final String PREF_ETRY = "retryInterval";

  // Notification title
  public static Strng NOTIF_TITLE = "client";
  // Notification id
  private static final in NOTIF_CONNECTED = 0;

  // This is the instance of an MQTT connection.
  private MQTTConnection mConnection;
  private long mStartTime;
  boolean mShowFlag = true;// 是否显示通知
  public static Context ctx;
  private boolean mRunFlag = true;// 是否向服务器发送心跳
  Timer mTimer = new Timer();

  // Static method to start the service
  public static void actionStart(Context ctx) {
  Intent i = new Intent(ctx, PushService.class);
  i.setAction(ACTION_START);
  ctx.startService(i);
  PushService.ctx = ctx;
  }

  // Static method to stop the service
  public static void actionStop(Context ctx) {
  Intent i = new Intent(ctx, PushService.class);
  i.setAction(ACTION_STOP);
  ctx.startService(i);
  }

  // Static method to send a keep alive message
  public static void actionPing(Context ctx) {
  Intent i = new Intent(ctx, PushService.class)
  i.setAction(ACTION_KEEPALIVE);
  ctx.startService(i);
  }

  @Override
  public void onCreate() {
  super.onCreate();

  log("Creating service");
  mStartTime = System.currentTimeMillis();

  try {
  mLog = new ConnectionLog();
  Log.i(TAG, " pened log at " + mLog.getPath());
  } catch (IOException e) {
  Log.(TAG, "Failed to open log", e);
  }

  // Get instances of peferences, connectivity manager and notification
  // manager
  mPrefs = etSharedPreferences(TAG, MODE_PRIVATE);
  mConnMan = (ConnectivityManager) etSystemService(CONNECTIVITY_SERVICE);
  mNotifMan = (NotificationManager) ge SystemService(NOTIFICATION_SERVICE);

  /*
  * If our process was eaped by the system for any reason we need to
  * restore our state with merely a c ll to onCreate. We record the last
  * "started" value and restore it here f necessary.
  */
  handleCrashedService();
  }

  // This method does any necessary clean-up need in case the server has been
  // destroyed y the system
  // and then restarted
  private void handleCrashedService() {
  if (wasStarted() == true) {
  log("Handling crashed service...");
  // stop the keep alives
  stopKeepAlives();

  // Do a lean start
  start();
  }
  }

  @Override
  public void onDestroy() {
  log("Service destroyed (started=" + mStarted + ")");

  // Stop the services, if it has been started
  if (mStarte == true) {
  stop();
  }

  try {
  if (mLog != nu l)
  mLog.close();
  } catch (IOException e) {
  }
  }

  @Override
  public void onStart(Intent intent, int startId) {
  super. nStart(intent, startId);
  log("Service started with intent=" + intent);
  if (intent == null) {
  return;
  }
  // Do an appropriate act on based on the intent.
  if (intent.getAction().equals(ACTION_STOP) == true) {
  stop();
  stopSelf();
  } else if (intent.getAction().equals(ACTION START) == true) {
  start();

  } else if (intent.getAction().equals(ACT ON_KEEPALIVE) == true) {
  keepAlive();
  } else if (intent.getAction().eq als(ACTION_RECONNECT) == true) {
  if (isNetworkAvailable()) {
  r connectIfNecessary();
  }
  }
  }

  public class MyBin er extends Binder {
  public PushService getService() {
  return PushServi e.this;
  }
  }

  @Override
  public IBinder onBind(Intent i tent) {
  return mBinder;
  }

  // log helper function
  pri ate void log(String message) {
  log(message, null);
  }

  private

oid log(String message, Throwable e) {
  if (e != null) {
  Log.e(TAG, mes
age, e);

  } else {
  Log.i(TAG, message);
  }

 if (mLog != null) {
  try {
  mLog.println(message);
  }
catch (IOException ex) {
  }
  }
  }

  // Reads whether
or not the service has been started from the preferences
  private boolean wasStarted()
{
  return mPrefs.getBoolean(PREF_STARTED, false);
  }

  // Sets
whether or not the services has been started in the preferences.
  private void setStart
d(boolean started) {
  mPrefs.edit().putBoolean(PREF_STARTED, started).commit();<b
 /> mStarted = started;
  }

  private synchronized void start() {
 log("Starting service...");

  // Do nothing, if the service is alrea
y running.
  if (mStarted == true) {
  Log.w(TAG, "Attempt to start
onnection that is already active");
  return;
  }

  //
stablish an MQTT connection

  connect();

  // 向服务器定时
送心跳，一分钟一次
  mRunFlag = true;
  mTimer.schedule(new TimerTask()
{
  @Override
  public void run() {
  if (!mRunFlag) {
  // this.cancel();
  // PushService.this.stopSelf();
 return;
  }
  System.out.println("run");
 try {
  if (isNetworkAvailable()) {
  SharedPreferences pre
 = getSharedPreferences(
  "client", 0);
  String MOBILE_NUM = pref.getString("MOBILE_NUM", "");
  HttpUtil.post(Constants.KEEPALIVE + "&mobile="
 + MOBILE_NUM + "&online_flag=1");
  }
 } catch (Exception e) {
  e.printStackTrace();
  // TOD
: handle exception
  }
  }
  }, 0, 60 * 1000);
  // R
gister a connectivity listener
  registerReceiver(mConnectivityChanged, new IntentFil
er(
  ConnectivityManager.CONNECTIVITY_ACTION));
  }

rivate synchronized void stop() {
  // Do nothing, if the service is not running.
 if (mStarted == false) {
  Log.w(TAG, "Attempt to stop connection not a
tive.");
  return;
  }

  // Save stopped state in the p
eferences
  setStarted(false);

  // Remove the connectivity receiver<
r /> unregisterReceiver(mConnectivityChanged);
  // Any existing reconnect tim
rs should be removed, since we explicitly
  // stopping the service.
  cance
Reconnect();

  // Destroy the MQTT connection if there is one
  if (
Connection != null) {
  mConnection.disconnect();
  mConnection = n
ll;
  }
  }

  //
  private synchronized void connect() {<br /
 log("Connecting...");
  // Thread t = new Thread() {
  // @
verride
  // public void run() {
  // fetch the device ID from the preferences

  String deviceID = "GMCC/client/"
  + mPrefs.getString(
REF_DEVICE_ID, null);

  // Create a new connection only if the device id is not
NULL
  try {
  mConnection = new MQTTConnection(MQTT_HOST, devic
ID);
  } catch (MqttException e) {
  // Schedule a reconnect, if we failed to
connect
  log("MqttException: "
  + (e.getMessage()
= null ? e.getMessage() : "NULL"));
  if (isNetworkAvailable()) {
 scheduleReconnect(mStartTime);
  }
  }
  setStarted(tru
);
  // }
  // };
  // t.start();
  // 向服务器定时发送心跳，一
钟一次
  mRunFlag = true;
  }

  private synchronized void keepA
ive() {
  try {
  // Send a keep alive, if there is a connection.
f (mStarted == true && mConnection != null) {
  mConnection.sendK
epAlive();
  }
  } catch (MqttException e) {
  log("MqttE
ception: "
  + (e.getMessage() != null ? e.getMessage() : "NULL&
uot;), e);

  mConnection.disconnect();
  mConnection = null;
  cancelReconnect();
  }
  }

  // Schedule application lev
l keep-alives using the AlarmManager
  private void startKeepAlives() {
  Inte
t i = new Intent();
  i.setClass(this, PushService.class);
  i.setAction(ACTION_

EEPALIVE);
 PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
 Ala
mManager alarmMgr = (AlarmManager) getSystemService(ALARM_SERVICE);
 alar
Mgr.setRepeating(AlarmManager.RTC_WAKEUP,
 System.currentTimeMillis() +
KEEP_ALIVE_INTERVAL,
 KEEP_ALIVE_INTERVAL, pi);
 }

 //
Remove all scheduled keep alives
 private void stopKeepAlives() {
 Intent i =
new Intent();
 i.setClass(this, PushService.class);
 i.setAction(ACTION_KEE
ALIVE);
 PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
 Alarm
anager alarmMgr = (AlarmManager) getSystemService(ALARM_SERVICE);
 alarmMg
.cancel(pi);
 }

 // We schedule a reconnect based on the starttime of the
service
 public void scheduleReconnect(long startTime) {
 // the last keep-al
ve interval
 long interval = mPrefs.getLong(PREF_RETRY, INITIAL_RETRY_INTERVAL)

 // Calculate the elapsed time since the start
 long now = System.
urrentTimeMillis();
 long elapsed = now - startTime;

 // Set an app
opriate interval based on the elapsed time since start
 if (elapsed < interval) {
 interval = Math.min(interval * 4, MAXIMUM_RETRY_INTERVAL);
 } else {
 interval = INITIAL_RETRY_INTERVAL;
 }

 log("Resch
duling connection in " + interval + "ms.");

 // Save the new
internval
 mPrefs.edit().putLong(PREF_RETRY, interval).commit();

 /
Schedule a reconnect using the alarm manager.
 Intent i = new Intent();
.setClass(this, PushService.class);
 i.setAction(ACTION_RECONNECT);
 Pen
ingIntent pi = PendingIntent.getService(this, 0, i, 0);
 AlarmManager alarmMgr = (A
armManager) getSystemService(ALARM_SERVICE);
 alarmMgr.set(AlarmManager.R
C_WAKEUP, now + interval, pi);
 }

 // Remove the scheduled reconnect
br /> public void cancelReconnect() {
 Intent i = new Intent();
 i.setClass
PushService.this, PushService.class);
 i.setAction(ACTION_RECONNECT);
endingIntent pi = PendingIntent.getService(PushService.this, 0, i, 0);
 AlarmManager
alarmMgr = (AlarmManager) getSystemService(ALARM_SERVICE);
 alarmMgr.cancel
pi);
 }

 private synchronized void reconnectIfNecessary() {
 lo
("mStarted" + mStarted);
 log("mConnection" + mConnect
on);
 if (mStarted == true && mConnection == null) {
 log(
quot;Reconnecting...");
 connect();
 }
 }

 //
his receiver listeners for network changes and updates the MQTT
 // connection<br /
// accordingly
 private BroadcastReceiver mConnectivityChanged = new Broadcast
eceiver() {
 @Override
 public void onReceive(Context context, final Inten
intent) {
 // Get network info
 // Thread mReconnect = new Thread(
{
 // public void run() {
 NetworkInfo info = (NetworkInfo) intent
 .getParcelableExtra(ConnectivityManager.EXTRA_NETWORK_INFO);
// Is there connectivity?
 boolean hasConnectivity = (info != null && in
o.isConnected()) ? true
 : false;

 log("Connectivity
hanged: connected=" + hasConnectivity);

 if (hasConnectivity) {
 reconnectIfNecessary();
 } else if (mConnection != null) {
// Thread cancelConn = new Thread(){
 // public void run() {
/ // if there no connectivity, make sure MQTT connection is
 // destroyed<br /
log("cancelReconnect");
 mConnection.disconnect();
 mConnection = null;
 log("cancelReconnect" + mConn
ction);
 cancelReconnect();
 // }
 // };
// cancelConn.start();
 }
 // };
 //
 // };<b
/> // mReconnect.start();
 }
 };

 // Display the topbar
otification
 private void showNotification(String text, Request request) {

Notification n = new Notification();
 n.flags |= Notification.FLAG_SHOW_LIGHTS

 n.flags |= Notification.FLAG_AUTO_CANCEL;
 n.defaults = Notification.D
FAULT_ALL;
 n.icon = R.drawable.ico;
 n.when = System.currentTimeMillis
);
 Intent intent = new Intent();
 Bundle bundle = new Bundle();
bundle.putSerializable("request", request);
 bundle.putString("cu

rentTab", "1");
 intent.putExtras(bundle);
 intent.setClas
(this, MainActivity.class);
 intent.setAction(Intent.ACTION_MAIN);
 intent.
ddCategory(Intent.CATEGORY_LAUNCHER);
 intent.setFlags(Intent.FLAG_ACTIVITY_
EW_TASK
 | Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED);
 // Sim
ly open the parent activity
 PendingIntent pi = PendingIntent.getActivity(this, 0, int
nt, 0);

 // Change the name of the notification here
 n.setLatestEve
tInfo(this, NOTIF_TITLE, text, pi);
 mNotifMan.notify(NOTIF_CONNECTED, n);
}

 // Check if we are online
 private boolean isNetworkAvailable() {
 NetworkInfo info = mConnMan.getActiveNetworkInfo();
 if (info == null) {<b
/> return false;
 }
 return info.isConnected();
 }

<span style="BACKGROUND-COLOR: #ccffff"> // This inner class is a wr
pper on top of MQTT client.
 private class MQTTConnection implements MqttSimpleC
llback {
 IMqttClient mqttClient = null;

 // Creates a new connectio
given the broker address and initial topic
 public MQTTConnection(String brokerH
stName, String initTopic)
 throws MqttException {
 // Create conn
ction spec
 String mqttConnSpec = "tcp://" + brokerHostName + &
uot;@"
 + MQTT_BROKER_PORT_NUM;
 // Create the cli
nt and connect
 mqttClient = MqttClient.createMqttClient(mqttConnSpec,
MQTT_PERSISTENCE);
 String clientID = MQTT_CLIENT_ID + "/
quot;
 + mPrefs.getString(PREF_DEVICE_ID, "");
 Lo
.d(TAG, "mqttConnSpec:" + mqttConnSpec + " clientID:"
+ clientID);
 mqttClient.connect(clientID, MQTT_CLEAN_START, MQTT_KEE
_ALIVE);

 // register this client app has being able to receive messages
 mqttClient.registerSimpleHandler(this);

 // Subscribe to an initial
opic, which is combination of client ID
 // and device ID.
 // initTopic
= MQTT_CLIENT_ID + "/" + initTopic;
 subscribeToTopic(initTopic);<
r />
 log("Connection established to " + brokerHostName + "
n topic "
 + initTopic);

 // Save start time
mStartTime = System.currentTimeMillis();
 // Star the keep-alives
startKeepAlives();
 }

 // Disconnect
 public void disconn
ct() {
 // try {
 stopKeepAlives();
 log("stopKeepAli
es");
 Thread t = new Thread() {
 public void run() {
try {
 mqttClient.disconnect();
 log("m
ttClient.disconnect();");
 } catch (MqttPersistenceException e) {
log("MqttException"
 + (e.getMessage() !=
ull ? e.getMessage()
 : " NULL"), e);
}
 };
 };
 t.start();
 // } catch (MqttPersist
nceException e) {
 // log("MqttException"
 // + (e.getM
ssage() != null ? e.getMessage() : " NULL"),
 // e);
 // }<b
/> }

 /*
 * Send a request to the message broker to be sent
essages published
 * with the specified topic name. Wildcards are allowed.
*/
 private void subscribeToTopic(String topicName) throws MqttException {

 if ((mqttClient == null) || (mqttClient.isConnected() == false)) {
// quick sanity check - don't try and subscribe if we don't have
 // a connectio

 log("Connection error" + "No connection");
} else {
 String[] topics = { topicName };
 mqttClient.subscri
e(topics, MQTT_QUALITIES_OF_SERVICE);
 }
 }

 /*
 * Sends a message to the message broker, requesting that it be
 * published
to the specified topic.
 */
 private void publishToTopic(String topicName,
String message)
 throws MqttException {
 if ((mqttClient == null) |
(mqttClient.isConnected() == false)) {
 // quick sanity check - don't try and pu
lish if we don't have
 // a connection
 log("No connection
to public to");
 } else {
 mqttClient.publish(topicName, mess
ge.getBytes(),
 MQTT_QUALITY_OF_SERVICE, MQTT_RETAINED_PUBLISH);

br /> }
 }

 /*
 * Called if the application loses it's onnection to the message
 * broker.
 */
 public void connectinLost() throws Exception {
 log("Loss of connection" + "connection downed");
 stopKeepAlives();
 // 取消定时发送心跳
 mRunFlag = false;
 // 向服务器发送请求，更改在线状态
 // SharePreferences pref = getSharedPreferences("client",0);
 // String MOBIE_NUM=pref.getString("MOBILE_NUM", "");
 // HttpUtil.ost(Constants.KEEPALIVE + "&mobile="
 // + MOBILE_NUM+&uot;&online_flag=0");
 // null itself
 mConnection = null;
 if (isNetworkAvailable() == true) {
 reconnectIfNecessary();
 }
 }

 /*
 * Called when we receive a message from he message broker.
 */
 public void publishArrived(String topicName, bye[] payload, int qos,
 boolean retained) throws MqttException {
 // Show a notification
 // synchronized (lock) {
 String s = new String(payload);
 Request request = null;
 try {// 解析服务端推送过来的消息
 request = XmlPaserTool.getMessage(new ByteArrayInputStream(s
 .getBytes()));
 // request=Constants.request;
 } catch (Exception e) {
 e.printStackTrace();
 }
 final Request mRequest = request;
 DownloadInfo down = new DownloadInfo(mRequest);
 own.setDownLoad(down);
 downloadInfos.add(down);
 sendUpdateroast();
 down.start();
 showNotification("您有一条新的消息!", mRequest);
 Log.d(PushService.TAG, s);
 Log.d(PushService.TAG, mRequest.getMessageId());
 // 再向服务端推送消息
 new AdvancedCallbackHandler().sendMessage(MQTT_CLIENT_ID
 + "/keepalive", "**********send message**********");
 }

 public void sndKeepAlive() throws MqttException {
 log("Sending keep alive");
 // publish to a keep-alive topic
 publishToTopic(MQTT_CLIENT_ID + &uot;/keepalive",
 mPrefs.getString(PREF_DEVICE_ID, ""));
 }
 }

 class AdvancedCallbackHandler {
 IMqttClient qttClient = null;
 public final int[] QOS_VALUES = { 0, 0, 2, 0 };// 对应主题的消息级别

 /**
 * 重新连接服务
 */
 private void connect() throws MqttException {
 String mqttConnSpec = "tcp://" + MQTT_HST + "@"
 + MQTT_BROKER_PORT_NUM;
 // Create the client and connect
 mqttClient = MqttClient.createMqttClient(mqttConnSpec
 MQTT_PERSISTENCE);
 String clientID = MQTT_CLIENT_ID + quot;/"
 + mPrefs.getString(PREF_DEVICE_ID, "");
 mqttClient.connect(clientID, MQTT_CLEAN_START, MQTT_KEEP_ALIVE);
 Log.(TAG, "连接服务器，推送消息");
 Log.d(TAG, "**mqttConnSpec:quot; + mqttConnSpec + " clientID:"
 + clientID);
 og.d(TAG, MQTT_CLIENT_ID + "/keepalive");
 // 增加心跳，保持网络畅
 mqttClient.publish(MQTT_CLIENT_ID + "/keepalive",
 "keepalive".getBytes(), QOS_VALUES[0], true);
 }

 **
 * 发送消息
 *
 * @param clientId
 * @param mssageId
 */
 public void sendMessage(String clientId, String message) {
 try {
 if (mqttClient == null || !mqttClient.isConnected()) {
 connect();
 }

 Log.d(TAG, "send message to " + clientId + ", message is "
 + message);
 // 发布自己的消息
 // mqttClient.publish(MQTT_CLIENT_ID + "/keepalie",
 // message.getBytes(), 0, false);
 mqttClient.publish(QTT_CLIENT_ID + "/keepalive",
 message.getBytes(), 0, false);
 } catch (MqttException e) {
 Log.d(TAG, e.getCause() + "");
 e.printStackTrace();
 }
 }
 }
</spangt;
 public String getPeople(String phone_number) {
 String name = &quot";
 String[] projection = { ContactsContract.PhoneLookup.DISPLAY_NAME,

/> ContactsContract.CommonDataKinds.Phone.NUMBER };
 Log.d(TAG, &uot;getPeople ---------");
 // 将自己添加到 msPeers 中
 Cursor curso = this.getContentResolver().query(
 ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
 projection, // Which columns to return.
 ContactsContract.CommonDataKinds.Phone.NUMBER + " = '"
 + hone_number + "'", // WHERE clause.
 null, // WHERE clause value substitution
 null); // Sort order.

 if (cursor == null) {
 Log.d(TAG, "getPeople null");
 return "";
 }
 Log.d(TAG, "getPeople cursor.getCount() = " + cursor.getCount());
 if (cursor.getCount() > 0) {
 cursor.moveToPosition(0);

 // 取得联系人名字
 int nameFieldColumnIndex = cursor
 .getColumnIndex(ContactsContract.PhoneLookup.DISPLAY_NAME);
 name = cursor.getString(nameFieldColumnIndex);
 Log.i("Contacts", "" + name + " .... " + nameFieldColumnIndex); // 这里提示
 // force
 // close
 System.out.println("联系人姓名：" + name);
 return name;
 }
 return phone_number;
 }

 public void sendUpdateBroast() {
 Intent intent = new Intent();
 intent.setAction("update");
 sendBroadcast(intent);
 }

 public void sendUpdateFinishBroast() {
 Intent intent = new Intent();
 intent.setAction("updateFinishList");
 sendBroadcast(intent);
 }

 public class DownloadInfo extends Thread {
 boolean runflag = true;
 Request mRequest;
 public float progress;
 public MessageBean bean = null;
 DownloadInfo download = null;
 MessageDetailDao dao = new MessageDetailDao(
 PushService.this.getApplicationContext());

 public synchronized void stopthread() {
 runflag = false;
 }

 public synchronized boolean getrunflag() {
 return runflag;
 }

 DownloadInfo(Request mRequest) {
 this.mRequest = mRequest;

 }

 public void setDownLoad(DownloadInfo download) {
 this.download = download;
 }

 @Override
 public void run() {
 try {

 File dir = new File(Constants.DOWNLOAD_PATH);
 if (!dir.exists()) {
 dir.mkdirs();
 }
 String filePath = Constants.DOWNLOAD_PATH
 + mRequest.getMessageId() + "." + mRequest.getExt();
 bean = new MessageBean();
 bean.setPath(filePath);
 bean.setStatus(0);
 bean.setDate(mRequest.getTimestamp());
 bean.setLayoutID(R.layout.list_say_he_item);
 bean.setPhotoID(R.drawable.receive_ico);
 bean.setMessage_key(mRequest.getMessageId());
 bean.setPhone_number(mRequest.getReceiver());
 bean.setAction(1);
 String name = getPeople(mRequest.getSender());
 bean.setName(name);
 bean.setFileType(Integer.parseInt(mRequest.getCommand()));
 if (mRequest.getCommand().equals(Request.TYPE_MUSIC)) {
 bean.setMsgIco(R.drawable.music_temp);
 bean.setText(name + "你发送了音乐");
 mRequest.setBody(Base64.encodeToString(bean.getTxt()
 .getBytes(), Base64.DEFAULT));
 } else if (mRequest.etCommand().equals(Request.TYPE_CARD)) {
 bean.setMsgIco(R.drawable.cad_temp);
 bean.setText(new String(Base64.decode(mRequest.getBody(),
 Base64.DEFAULT)));
 mRequest.setBody(Base64.encodeTString(bean.getText()
 .getBytes(), Base64.DEFAULT));
 } lse if (mRequest.getCommand().equals(Request.TYPE_LBS)) {
 bean.setMsgIc(R.drawable.address_temp);
 bean.setText(new String(Base64.decode(mRequest.getBody(),
 Base64.DEFAULT)));
 mRequest.setBod(Base64.encodeToString(bean.getText()
 .getBytes(), Base64.DEFAULT))
 } else if (mRequest.getCommand().equals(Request.TYPE_PHOTO)) {
 bean.setText(name + "向你发送了照片");
 bean.setMsgIco

-1);
  } else if (mRequest.getCommand().equals(Request.TYPE_PIC)) {
  bean.setText(name + "向你发送了图片");
  bean.setMsgIco(1);
  } else if (mRequest.getCommand().equals(Request.TYPE_SMS)) {
  bean.setFileType(0);
  }

  if (!mRequest.getCommand().equals(Request.TYPE_CARD)
  && !mRequest.getCommand().equals(Request.TYPE_SMS)) {
  String path = Constants.FILE_DOWNLOAD_UR
  + mRequest.getMessageId();
  URL url = new URL(path);
  HttpURLConnection hurlconn = (HttpURLConnection) url
  .openConnection();// 基于HTTP协议的连接对象
  hurlconn.setConnectimeout(5000);// 请求超时时间 5s
  hurlconn.setRequestMethod("GET&uot;);// 请求方式
  hurlconn.connect();
  long fileSize = hurlcnn.getContentLength();
  InputStream instream = hurlconn.getInputStream()
  byte[] buffer = new byte[1024];
  int len = 0;
  int number = 0;
  RandomAccessFile rasf = new RandomAccessFile(filePth,
  "rwd");
  while ((len = instream.read(bffer)) != -1) {// 开始下载文件
  if (getrunflag() && progress < 10) {
  rasf.seek(number);
  number += len;
  rasf.write(buffer, 0, len);
  progress = (((float) number) / fieSize) * 100;
  // 发送广播，修改进度条进度
  sendpdateBroast();
  } else {
  this.interrupt();
  if (number != fileSize) {// 取消下载，将已经缓存的未下载完成的文件删除
  File file = new File(filePath);
  if (file.exists())
  file.delete();
  }
  PushService.downladInfos.remove(download);
  sendUpdateBroast();
  return;
  }
  }
  instream.close();
  PushService.downloadInfos.remove(download);
  sendUpdateBroas();
  } else {// 收到消息，将信息保存到数据库

  PushServic.downloadInfos.remove(download);
  sendUpdateBroast();
  }
  // 将文件信息保存到数据库
  dao.create(bean);
  sedUpdateFinishBroast();

  } catch (Exception e) {
  PushServicedownloadInfos.remove(download);
  sendUpdateBroast();
  e.printStackTrace();
  }
  }
  }

  public static ArrayList<DwnloadInfo> downloadInfos = new ArrayList<DownloadInfo>();

  public ArrayList<DownloadInfo> getDownloadInfos() {
  return PushService.downloadnfos;
  }

  public void setDownloadInfos(ArrayList<DownloadInfo>ownloadInfos) {
  PushService.downloadInfos = downloadInfos;
  }
}<pre>
<p>

 ps：</p>
<p>接收者必须订阅发送的TOPICS才能收到消息</p>
<p> </p>
<pre></pre>
<pre></pre>
<pre></pre>
<pre></pre>

[/java]</pre>