建立连接成功之后，应该是客户端发起握手，或者快连；
客户端接收到握手成功消息，向推送服务端发送心跳、用户绑定、保存会话信息到本地缓存中；
客户端接收到快连成功消息，向推送服务端发送心跳、用户绑定；

ClientConfig中设置监听器ClientListener；
在监听器中，客户端需要实现各事件方法　：
(具体实现，参考mpush工程中的ConnClientChannelHandler、或者mpush-android)

```
public static class L implements ClientListener {
        private final ScheduledExecutorService scheduledExecutor;
        boolean flag = true;

        public L(ScheduledExecutorService scheduledExecutor) {
            this.scheduledExecutor = scheduledExecutor;
        }

        @Override
        public void onConnected(Client client) {
            flag = true;
        }

        @Override
        public void onDisConnected(Client client) {
            flag = false;
        }

        @Override
        public void onHandshakeOk(final Client client, final int heartbeat) {
            scheduledExecutor.scheduleAtFixedRate(new Runnable() {
                @Override
                public void run() {
                    client.healthCheck();
                }
            }, 10, 10, TimeUnit.SECONDS);

            //client.push(PushContext.build("test"));

        }

        @Override
        public void onReceivePush(Client client, byte[] content, int messageId) {
            if (messageId > 0) client.ack(messageId);
        }

        @Override
        public void onKickUser(String deviceId, String userId) {

        }

        @Override
        public void onBind(boolean success, String userId) {

        }

        @Override
        public void onUnbind(boolean success, String userId) {

        }
    }
```

onConnected

处理类TcpConnection#onConnected

建立连接， 需要向推送服务端发起握手，或者快连；

onDisConnected

处理类TcpConnection#doClose

断开连接，发起重连？？（也许在mpush-android中会有答案）

onHandshakeOk

处理器HandshakeOkHandler、FastConnectOkHandler

握手成功，向推送服务端发送心跳、用户绑定、保存会话信息到本地缓存中；

onReceivePush

处理器PushMessageHandler

接收PUSH，auto_ack自动发送ACK确认消息，biz_ack需要客户端根据业务处理ACK

onKickUser

处理器KickUserHandler

接收被踢消息，客户端下线退出登录？？

onBind

处理器OkMessageHandler

绑定用户成功，客户端可以做些什么事？

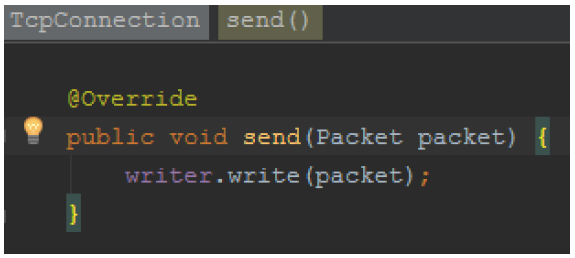
onUnbind

处理器OkMessageHandler

绑定用户失败，客户端可以做些什么事？重新发起用户绑定？

发送消息

握手、快连都会调用此方法发送消息给推送服务端



```java
public final class AsyncPacketWriter implements PacketWriter {
    private final Executor executor = ExecutorManager.INSTANCE.getWriteThread();
    private final Logger logger;
    private final Connection connection;
    private final EventLock connLock;
    private final ByteBuf buffer;

    public AsyncPacketWriter(Connection connection, EventLock connLock) {
        this.connection = connection;
        this.connLock = connLock;
        this.buffer = ByteBuf.allocateDirect(1024);//默认写buffer为1k
        this.logger = ClientConfig.I.getLogger();
    }

    public void write(Packet packet) {
        executor.execute(new WriteTask(packet));
    }

    private class WriteTask implements Runnable {
        private final long sendTime = System.currentTimeMillis();
        private final Packet packet;

        private WriteTask(Packet packet) {
            this.packet = packet;
        }

        @Override
        public void run() {
            buffer.clear();
            PacketEncoder.encode(packet, buffer);
            buffer.flip();
            ByteBuffer out = buffer.nioBuffer();
            while (out.hasRemaining()) {
                if (connection.isConnected()) {
                    try {
                        connection.getChannel().write(out);
                        connection.setLastWriteTime();
                    } catch (IOException e) {
                        logger.e(e, "write packet ex, do reconnect, packet=%s", packet);

                        if (isTimeout()) {
                            logger.w("ignored timeout packet=%s, sendTime=%d", packet, sendTime);

                            return;
                        }
                        connection.reconnect();
                    }
                } else if (isTimeout()) {
                    logger.w("ignored timeout packet=%s, sendTime=%d", packet, sendTime);

                    return;
                } else {
                    connLock.await(DEFAULT_WRITE_TIMEOUT);
                }
            }
            logger.d("write packet end, packet=%s, costTime=%d", packet.cmd, (System.currentTimeMillis() - sendTime));

        }

        public boolean isTimeout() {
            return System.currentTimeMillis() - sendTime > DEFAULT_WRITE_TIMEOUT;
        }
    }
}
```

```
AsyncPacketWriter  WriteTask  run()

    @Override
    public void run() {
        buffer.clear();
        PacketEncoder.encode(packet, buffer); 1
        buffer.flip(); 2
        ByteBuffer out = buffer.nioBuffer(); 3
        while (out.hasRemaining()) {
            if (connection.isConnected()) {
                try {
                    connection.getChannel().write(out); 4
                    connection.setLastWriteTime();
                } catch (IOException e) {
                    logger.e(e, "write packet ex, do reconnect, packet=%s", packet);
                    if (isTimeout()) { 5
                        logger.w("ignored timeout packet=%s, sendTime=%d", packet, sendTime);
                        return;
                    }
                    connection.reconnect(); 6
                }
            } else if (isTimeout()) { 7
                logger.w("ignored timeout packet=%s, sendTime=%d", packet, sendTime);
                return;
            } else {
                connLock.await(DEFAULT_WRITE_TIMEOUT); 8
            }
        }
        logger.d("write packet end, packet=%s, costTime=%d", packet.cmd, (System.currentTimeMillis() - sendTime));
    }

    public boolean isTimeout() {
        return System.currentTimeMillis() - sendTime > DEFAULT_WRITE_TIMEOUT;
    }
```

1、消息编码，将数据Put到bytebuf中

```
public final class PacketEncoder {

    public static void encode(Packet packet, ByteBuf out) {

        if (packet.cmd == Command.HEARTBEAT.cmd) {
            out.put(Packet.HB_PACKET_BYTE);
        } else {
            out.putInt(packet.getBodyLength());
            out.put(packet.cmd);
            out.putShort(packet.cc);
            out.put(packet.flags);
            out.putInt(packet.sessionId);
            out.put(packet.lrc);
            if (packet.getBodyLength() > 0) {
                out.put(packet.body);
            }
        }
    }
}
```

2、调用buffer.flip()，写模式切换成读模式，limit=position, position=0

3、获取NIO buffer对象

4、把buffer中的数据写入channel通道中

5、判断是否超时，默认10S

6、重连，重新连接到推送服务端

7、如果连接断开，判断是否超时，如果超时，则忽略

8、如果连接断开，判断没有超时，await等待超时