详细说明

1、所有的消息均是继承自 AbstractMessage，然后用 datagram 包装，datagram 里面放入加密后的 abstractmessage 和加密类型（RSA 或者 AES）

2、客户端在一开始是有一个 RSA 的公钥 A 的，服务器端有一个私钥 B，相当于证书的认证。

3、整个 message 的传递不仅有 RSA 和 AES 的加密，还有时间戳的判断，这是为了防止重放攻击。

4、整个项目没有使用数据库，而是直接存在本地，引入数据库也是可行的。

5、客户端的行为主要定义在 entity 包下的 client 类中，服务器端的行为定义在 server 包下的 server 类中。

6、服务器端有一个 serveruser 对象，里面有

```
private Key publicKey;
private ArrayList<Friend> friendList;
```

朋友列表和这个人的用户公钥（用户公钥会在后面注册的时候说明）。所有的用户，只要登录成功，都有一个这个对象，用来保存他的这两个信息。

一、注册

用户输入邮箱，检查合法性后，会在客户端生成一堆 RSA 秘钥，其中的私钥存在本地，将其中的公钥放入 message（RegisterRequest）中，邮箱也会存入进去，然后将用刚刚生成的秘钥中的私钥加密时间戳，也放入 message 中。

服务器端接到后首先用私钥 B 解密 datagram 中的 message 字节码，然后用 message 其中的包含的用户公钥解密时间戳进行时间判断。如果注册成功，则会在服务器端存入这个用户的公钥，并在一个 xml 文件中添加这个用户。

二、登录

用户输入邮箱，并导入之前存在客户端的用户私钥，对登录 LoginRequest 的时

间戳，用用户私钥加密，然后放入 message 中，对整个 message 用公钥 A 加密。

服务器端收到登录报文后，依次用私钥 B，用户公钥解密，还要判断时间戳，之后，决定是否登录成功，如果登录成功，会在服务器端维护一个 Server 的 arraylist（onLineUserThreadList），加入这个 server，这个 server 中含有这个用户的 serveruser 和一些其他信息，便于以后查询。

下图是 server 对象中的东西

```
private Socket socket;
private Socket chatsocket;
private ServerUser threadUser;
private Key kcs;
private Key chatkcs;
```

同时，一旦登录成功，会在服务器端生成一个 AES 的对称秘钥 kcs，用于之后通信。然后返回一个 AcceptLoginResonse，里面有这次登录 kcs。每次生成的 kcs 都是与这次登录用的 socket 唯一绑定，所以它不是恒久不变的，会随着登录的不同而变化。

三、拉取朋友列表

在进入 MainActivity 的时候会执行这个操作，就是从服务器拉一下有哪些朋友，并在本地维护一个 arraylist（里面的对象是 Friend）存一下。

Friend 对象中有两种钥匙，一个是好友的公钥，是服务器发过来的，一个是之后对话时候生成的 sessionkey

```
private Key publicKey;
private Key sessionKey;
```

四、加好友

加好友是必须要加的那个人在线，这样对方才能收到请求。这个 message 直接用之前登录成功返回的 kcs 加密时间戳和 message。Message 中有发起请求的人的邮箱和被加好友的人的邮箱。

服务器端负责处理这个报文，用其中的被加好友的人的邮箱，先到整个的 user 列表中找（这是一个 xml），没有说明不存在这个用户，在到 onLineUserThreadList

里找，没有说明用户没在线，这两种情况分别直接返回给发起加好友请求的客户端用户不存在和用户不在线两种 message。
如果用户在线，那么通过这个 onLineUserThreadList，得到对应的 server 对象，获得其中的 socket，serveruser 等，总之就是一切的信息。然后创建一个转发报文，发给这个人。

这个人会收到这个由服务器发来的转发报文，然后决定同意还是不同意，然后发回给服务器，服务器再发给发起请求的客户端。

五、发消息和发文件
首先客户端在用户点击某个朋友的时候，会先发起一个分发秘钥的操作，叫作（sendsessionkey）。首先在客户端生成一个 AES 对称秘钥，然后把和这个秘钥存在三中创建的 list 中，它是一个两层的 message，第一层是 SendSessionKeyRequest，第二层是 ForwardMessage，其中第一层的 message 中的时间戳用自己的用户私钥加密，message 用好友的用户公钥加密，第二层 ForwardMessage 用本客户端与服务器端连接 socket 中的 kcs 加密。

服务器端收到这个 forwardmessage 后，直接用 kcs 解密，再发给对方用户，对方用户用自己的私钥解密，再用发送方的公钥解密时间戳，然后就可以得到这个 sessionkey 了。如果对方用户不同意建立这个对话，就返回拒绝消息，如果同意，则返回同意消息，之后双方进入 ChatActivity（聊天界面）。

在聊天界面的一开始，用户会请求与服务器重新建立一个 socket 连接，作为专用 socket。
（这个 socket 由于是聊天界面生成的，一旦用户退出聊天界面，这个 socket 就没有了，之前的商量的 sessionkey 也失效了，因为下一次再进入聊天界面会重新 sendsessionkey，重新生成 socket，所以即使服务器端被某些人劫持，也不能获得其中的消息）
这个专用 socket 会存在之前 arraylist（onLineUserThreadList）中的对应 server 对象（邮箱 ID 一样）中的 chatsocket 中，Kcs 存在 chatkcs 中。

以后发消息都是两层消息，第一层是 ChatMessage，第二层是 ForwardMessage，服务器只负责先解密出 ChatMessage，然后利用

对方的 chatkcs 加密后，再利用 chatsocket 转发。

以后发文件也都是两层消息，第一层是 FileMessage，里面有这个文件的字节码 fi 和 maccode，第二层是 ForwardMessage，服务器只负责先解密出 ChatMessage，然后利用对方的 chatkcs 加密后，再利用 chatsocket 转发。对方收到后会重新用 fi 生成一个 maccode，再与传过来的 maccode 对比，每一位都一样则接收成功。