

《物联网安全》

设计文档

组 长： 魏来 班号： 193182

组 员： 郭鸿绅、 段依琳、 刘畅海

院（系）： 计算机学院 专业： 网络工程

指导教师： 姚 宏 职称： 教授

2021 年 5 月

[1. 序言 1](#_Toc21685)

[1.1.程序目的 1](#_Toc18533)

[1.2.定义 2](#_Toc6496)

[2. 需求分析 4](#_Toc12349)

[2.1. 任务概述 4](#_Toc25704)

[2.2. 数据流图 5](#_Toc23318)

[2.3. 状态机设计 9](#_Toc1329)

[2.4. UI布局设计 9](#_Toc17758)

[2.4.1. App UI界面 9](#_Toc20105)

[2.4.2. Web UI界面 13](#_Toc1635)

[3. 系统概要设计 15](#_Toc4978)

[3.1. 系统概述 15](#_Toc608)

[3.1.1功能需求规定： 15](#_Toc6006)

[3.1.2运行环境： 15](#_Toc1545)

[3.2. 系统结构 15](#_Toc30335)

[3.3. 功能模块设计 16](#_Toc24447)

*[3.4.](#_Toc26757)* [拓扑结构设计 18](#_Toc26757)

*[4. 数据](#_Toc26574)* [19](#_Toc26574)

[4.1. 数据库设计 19](#_Toc10129)

[4.1.1. 概念结构设计 19](#_Toc18783)

[4.1.2. 表结构设计 21](#_Toc13024)

[4.2. 时序图 24](#_Toc29450)

[4.2.1. 证书时序图 24](#_Toc16437)

[4.2.2. Kerberos认证时序图 25](#_Toc25772)

[4.2.3. 应用时序图 26](#_Toc29844)

[4.3. 数据报文 29](#_Toc10599)

[5. 详细设计 31](#_Toc2045)

[5.1. 数据结构 31](#_Toc13565)

[5.2. 详细流程图 34](#_Toc21476)

[5.3. 模块详细设计 40](#_Toc17637)

[5.3.1. Web服务器 40](#_Toc2677)

[5.3.2. 服务器框架 41](#_Toc11007)

[5.3.3. 服务逻辑： 42](#_Toc1251)

[5.3.4. Client界面： 48](#_Toc22305)

[5.3.5. Json报文 51](#_Toc18776)

[5.4. 甘特图 52](#_Toc24015)

[6. 任务分工 53](#_Toc11586)

[7. 调试设计 53](#_Toc28118)

# 

# 序言

## 1.1.程序目的

在学习了对称加密—DES，和非对称加密—RSA之后，目前，课程设计需要将上述两种工具，以及Kerberos的认证体系结合，并应用于自定义的网络应用中。

在近期的讨论中，我们小组认为本次课程设计的重点并不在于Kerberos认证过程，而是实现一套逻辑严密，结构完整，内容丰富的网络应用，并在应用的使用过程中体现出Kerberos认证过程以及两套加解密内容；经过小组讨论并结合了现有水平能力，时间限制后，我们小组从简单的网络游戏，模拟美团点餐系统，IM（即时通讯）中敲定了本次课程设计的大致方向，即设计一套能够满足一定量并发需求的C/S模式的通讯应用，通过端到端的通讯过程，体现Kerberos认证流程与RSA加密过程，在通讯内容中体现DES加密过程。

本次应用设计中，我们将尽可能将程序模块拆开掰碎，使得模块分工细致完整，同时降低函数间，模块间的耦合度，提高程序的可拓展性；同时在实现基础通讯功能外，还构想了邀约游戏等功能，可以理解为一套新的程序，作为一个黑盒子添加到通讯程序的拓展模块，使得程序内容更加丰富。

魏来

2021年4月28日

## 1.2.定义

|  |  |
| --- | --- |
| **专业术语** | **定义** |
| **Kerberos** | 计算机认证协议 |
| **数据库** | 用于存放数据，租借的阿里云MySQL数据库  rm-uf6t4cbyfz681x569.mysql.rds.aliyuncs.com 端口：3306 |
| **终端** | 4台设备模拟启动6个终端，即6个设备，分为KDC和User两种类别 |
| **KDC** | 密钥分配中心，由两个终端扮演，在本次程序中分为AS终端和TGS终端 |
| **AS** | 身份验证服务，为client生成TGT的服务 |
| **TGS** | 票务发放服务，为client生成某个服务的ticket |
| **TGT** | 用于获取ticket的票据 |
| **ticket** | 用于发送信息的凭证 |
| **User** | 实际运行应用程序的设备，又分为Client，Server |
| **Server** | 通讯服务的服务器设备 |
| **Client** | 需要通讯的用户的设备 |
| **K（）** | 密钥，用于加解密工具 |
| **聊天用户** | 使用本程序的用户 |
| **Session key** | 发送应用信息时，DES加密时用到的秘钥 |
| **Easy Chat** | 本程序名 |

表 1 名词定义

|  |  |
| --- | --- |
| **密钥表** | **意义** |
|  | User证书中的的公钥，用于发送对称钥和数字签名 |
|  | 对称钥，由AS生成，通过加密发送给User |
|  | TGS和另一个User之间的对称钥，与等价 |
|  | c和TGS之间的对称钥，用于发送session key |
|  | c和v之间的session key |
|  | AS和TGS之间的对称钥，写入文本。 |

表 2 密钥定义

# 需求分析

## 任务概述

本程序为基于 Kerberos 认证（DES加密）和 RSA加密的分布式通讯工具，需要实现下述部分：

工具：

1. 加密工具：使用DES与RSA两种加密方式，根据报文的不同需求，分别使用不同的加密方式。
2. RSA加密中的证书：约定好证书格式后，每个设备启动时，都会生成自己的证书，同时我们认为这个证书在系统内，独一无二，绝对真实，绝对安全，因此，先通过明文通讯，使得所有User设备与KDC设备间得到对方的公钥，User中，Client与Server得到对方公钥，即证书互信。同时可以认为系统内的Client与Server间都有对方的公钥，传输后续数组签名用到的密钥。
3. 数字签名，通过2）中获得的公钥，并用RSA给每条数据报文都加上数字签名。

Kerberos认证：

1. 在Kerberos认证之前：由于Kerberos本身无RSA加解密需求，但为了将该功能引入Kerberos中，我们将最开始认证中出现的作为明文，根据最初已经获得的证书信息，使用RSA加密，通过通信功能传输。
2. Kerberos认证体系：当所有设备启动后，其中两台设备扮演TGS和AS的角色，一台设备扮演通讯工具中的Server，其他设备扮演Client；所有Client根据自己在AS中的ID信息与Server执行Kerberos认证，AS和TGS为每一个Client与Server间提供认证服务（由Client发起），Client与Client间没有必要提供认证服务。
3. 通过上述顺序为4）---> 5）--->2）的逻辑，实现了Kerberos认证功能后，认为除了出现新的User设备，整个系统内无需再使用Kerberos认证，AS和TGS设备监听新设备，不执行后续逻辑功能。

应用：

①客户端：提供登录、主窗体及聊天等界面及对应的业务逻辑，向服务器发送相应的服务请求，并接受相应的处理结果。客户端是轻量级的软件，只负责链接远程服务器，并发出相应的服务请求，并不进行核心业务逻辑的处理。具体的处理交给服务器，而客户端只接收服务器处理的结果并显示给用户。

②服务器：监控登录信息及在线用户信息，接收客户端的服务请求，并做相应的处理，然后将处理结果发送给客户端。服务器负责处理核心的业务逻辑，并负责连接数据库，保存和读取数据。因此，服务器端设计的好坏也直接影响即时通信软件的质量

用例描述：



图 2.1 用户（服务器）用例图

## 数据流图



图 2.2 顶层数据流图



图 2.3 一层数据流图



图 2.4 P1二层数据流图



图 2.5 P2二层数据流图



图 2.6 P3二层数据流图

## 状态机设计



图 2.7 状态机

## UI布局设计

### App UI界面



图 2.8 登录界面

1. 消息列表



图 2.9 消息列表界面

1. 查看联系人界面



图 2.10 查看联系人界面

1. 查看个人信息界面



图 2-11 个人信息

1. 添加界面

点击+号，弹出功能

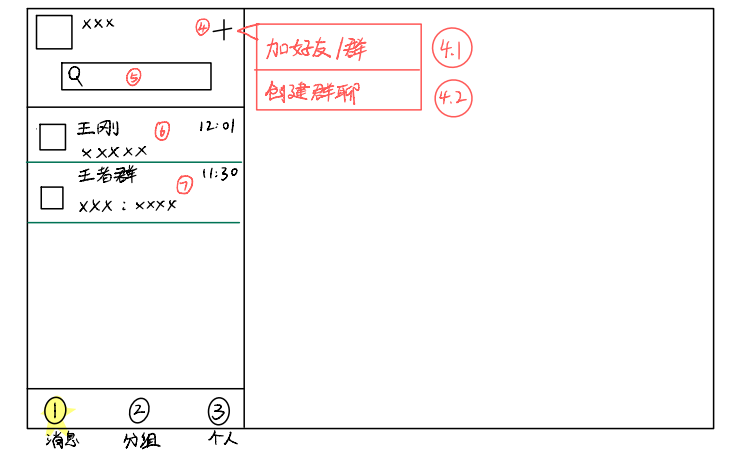


图 2-12 添加界面

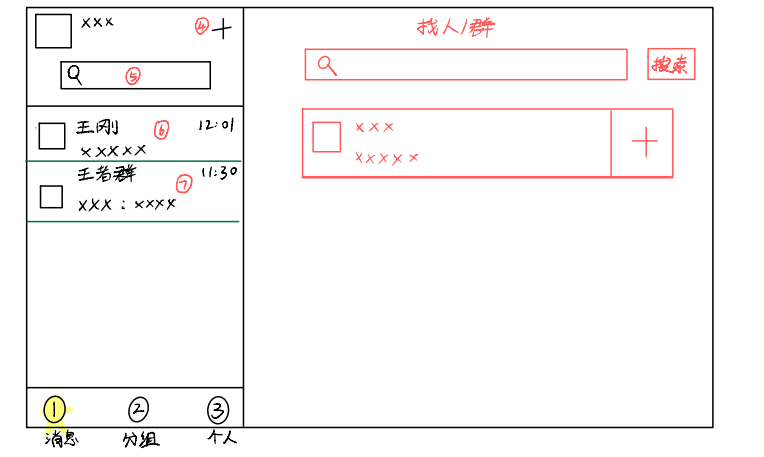


图 2-13 添加好友、群聊界面

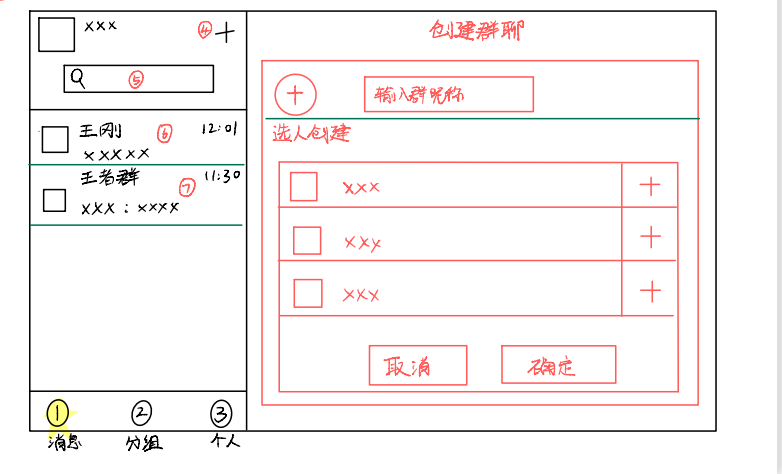


图 2-14 创建群聊界面

1. 聊天界面

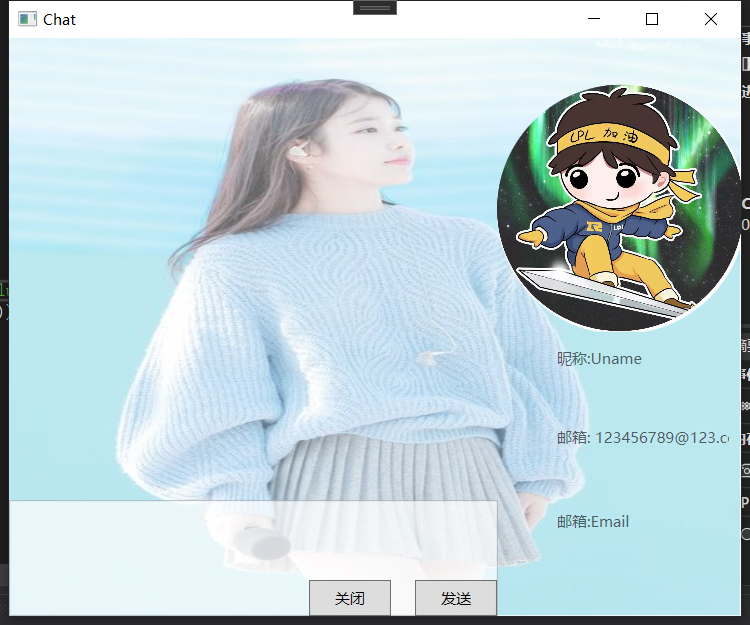


图 2-15 聊天界面

### Web UI界面

页面上方导航栏，注册按钮进入注册界面，登录按钮进入登录界面。首页可提供APP安装包下载。

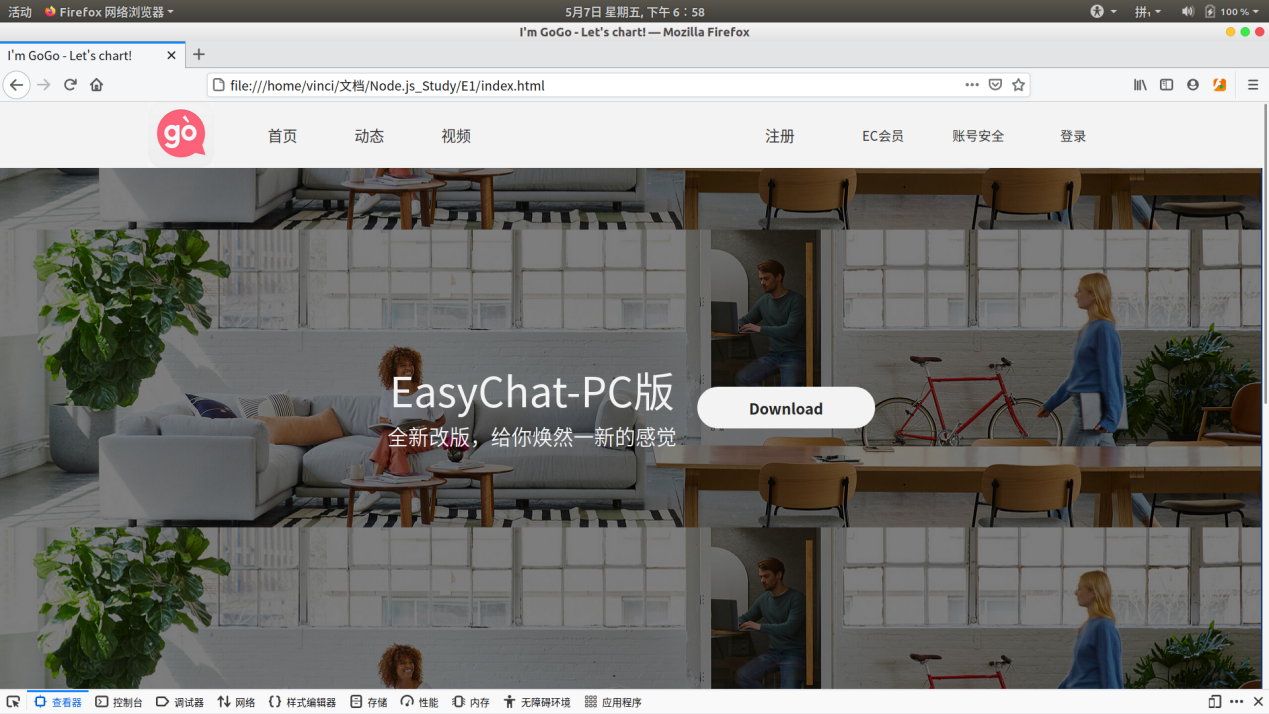


图 2-16 Web首页

（1）填写注册信息

用户输入账号、密码、邮箱，本地JavaScript程序会进行验证并显示输入错误。获取邮箱验证码的冷却时间为60秒。点击确认进行注册，点击取消返回主页。

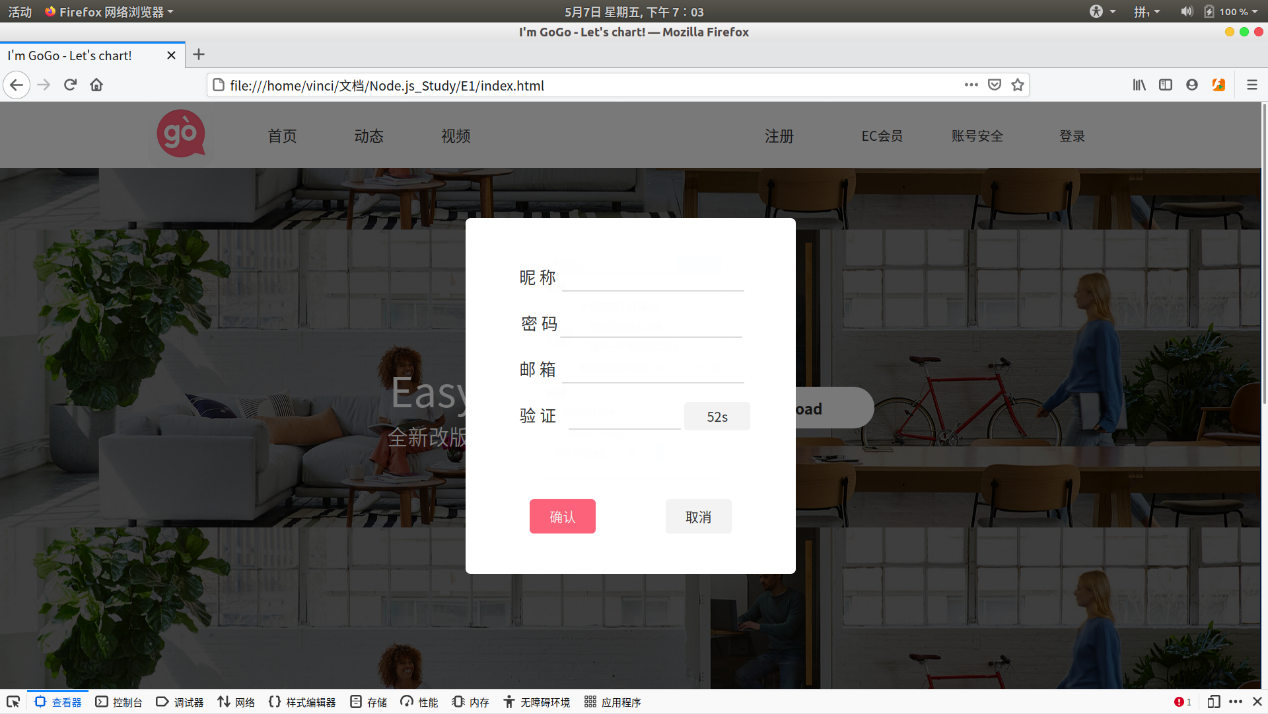


图 2-17 填写注册信息

（2）注册结果

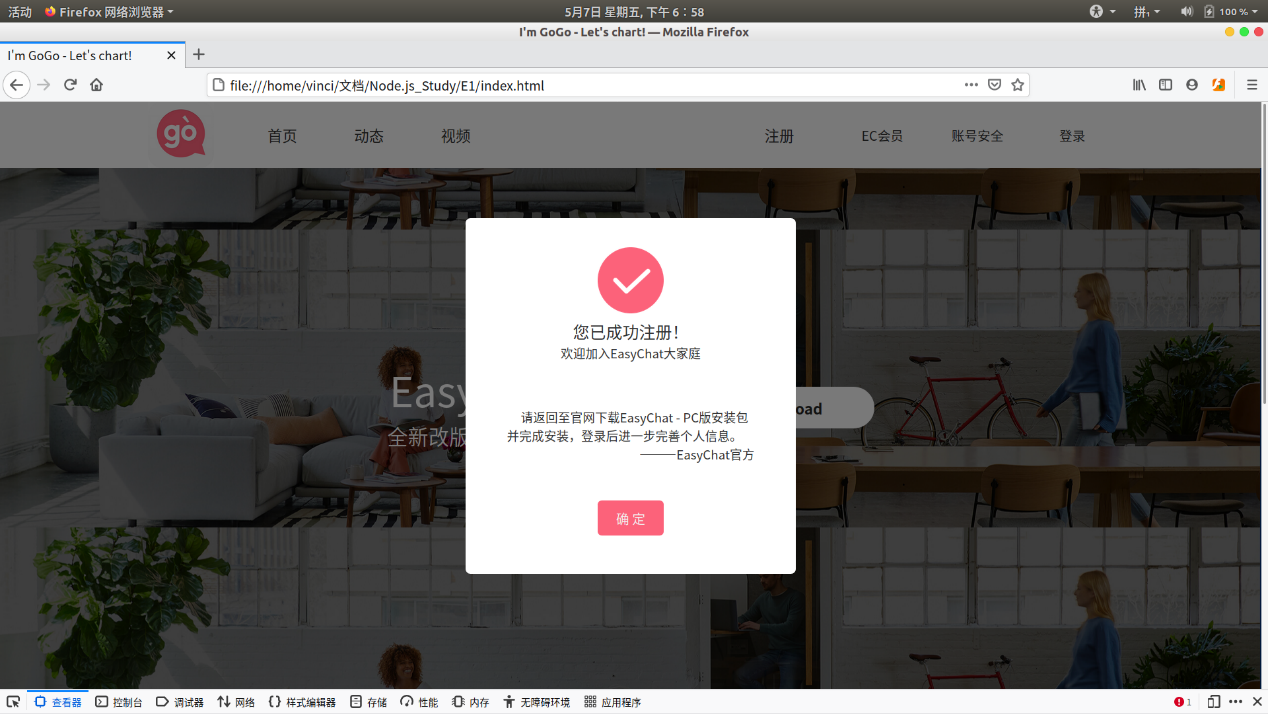


图 2-18 注册结果

3、找回密码模块

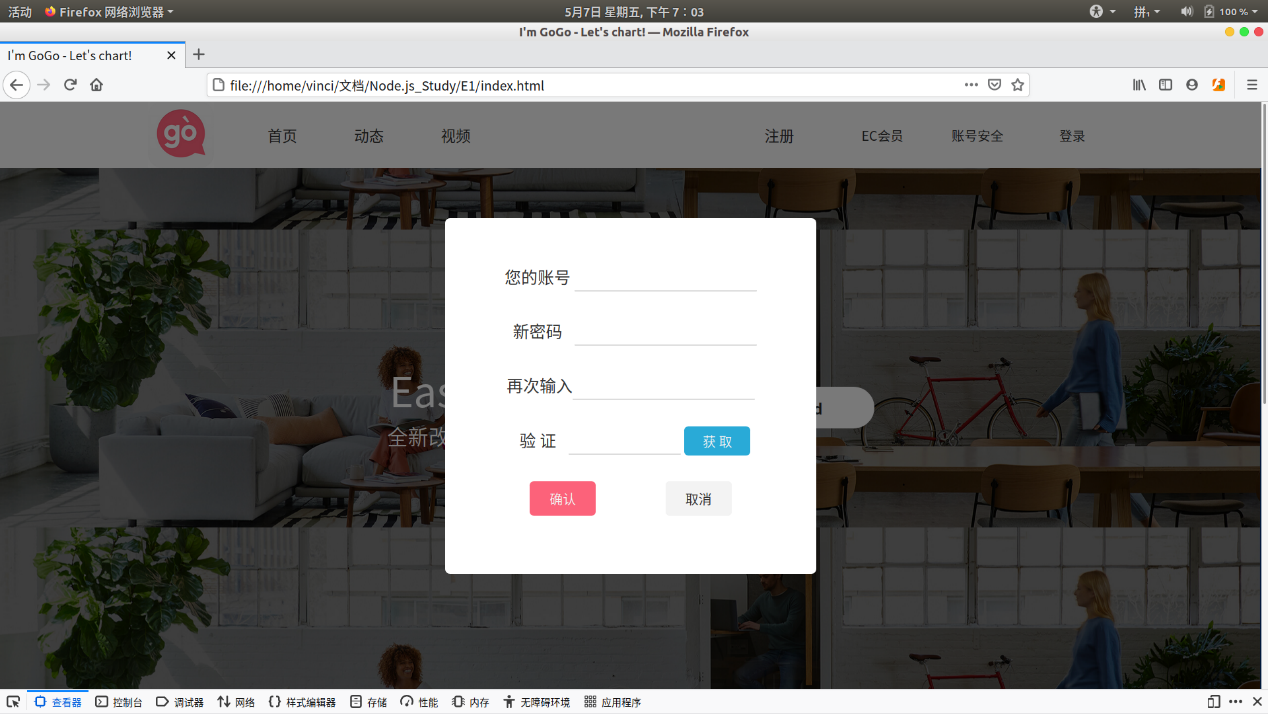


图 2-19 找回密码

# 系统概要设计

## 系统概述

### 3.1.1功能需求规定：

AS系统负责，接收User传过来的证书以及向User端发放TGT；

TGS系统验证User的TGT，生成并发放密钥。

Client系统，完成证书发送；Kerberos系统认证：向AS请求TGT，凭TGT向TGS请求密钥，与Server系统完成双向验证；完成与Server端的交互，包括，聊天，添加好友、群，删除好友，退出群聊，登录，注册，查看个人信息。

Server系统，完成证书发送；接受Client系统的交互请求，转发信息。

### 3.1.2运行环境：

运行平台：Windows10（AS、TGS、Client），Linux（Server）

数据库：MySQL

Web：Node.js平台下的JavaScript

开发语言：C#、Java、JavaScript

## 系统结构

结构描述：

系统最先部署AS与TGS服务器，两者通过共享数据库的方式建立联系，其中以在本地代码中做好约定。其次部署Server服务器，最后Client上线。

1. 当Server或者Client上线后，都向AS服务器发送证书，AS根据接受到的证书信息，提取公钥PK，并加密作为后续对称钥的密钥，AS将对称钥信息写入数据库，供TGS使用；其后每当Client上线，都会向Server发送证书，Server也会回复证书，作为RSA加密的公钥。
2. 约定好公钥、对称钥后，Client向AS发送Kerberos认证请求，最终与Server建立认证。
3. 认证后，直到Client程序终止，都无需在进行上述两个过程，通过Des加密完成数据报文传输，实现通讯应用。



图 3-1 系统结构流程图

## 功能模块设计



图 3-2 AS功能图



图 3-3 TGS功能图



图 3-4 Server功能图



图 3-5 Client功能图

## 拓扑结构设计

**

图 3-6 拓扑图

# *数据*

## 数据库设计

### 概念结构设计

本数据库将反映的现实世界中的实体，属性和它们之间的关系等的原始数据形式，建立本数据库的用户视图。

Server端的实体-关系图涉及到四个实体：用户、群聊、标签和登录历史，主要用于服务器的数据组织、存储和管理。用户和群聊之间的关系为属于关系。

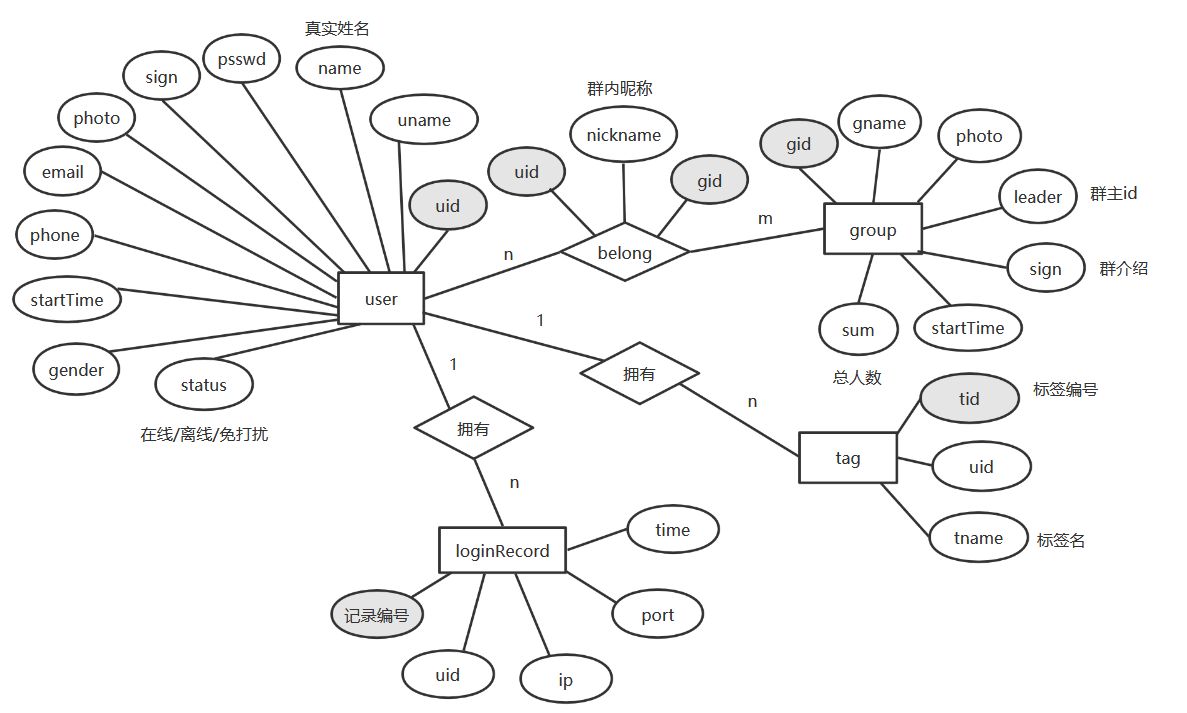


图 4-1 Server-ER图

用户之间的关系为朋友关系，用户A和用户B的关系包含两条记录，用于区别A对B的标签和分组以及B对A的标签和分组。

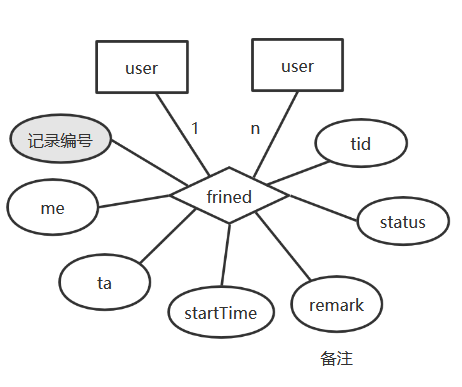


图 4-2 朋友表ER图

用户本地保存自己的聊天记录，分为单聊记录和群聊记录两类，二者都将以文件的形式存储在Client端。

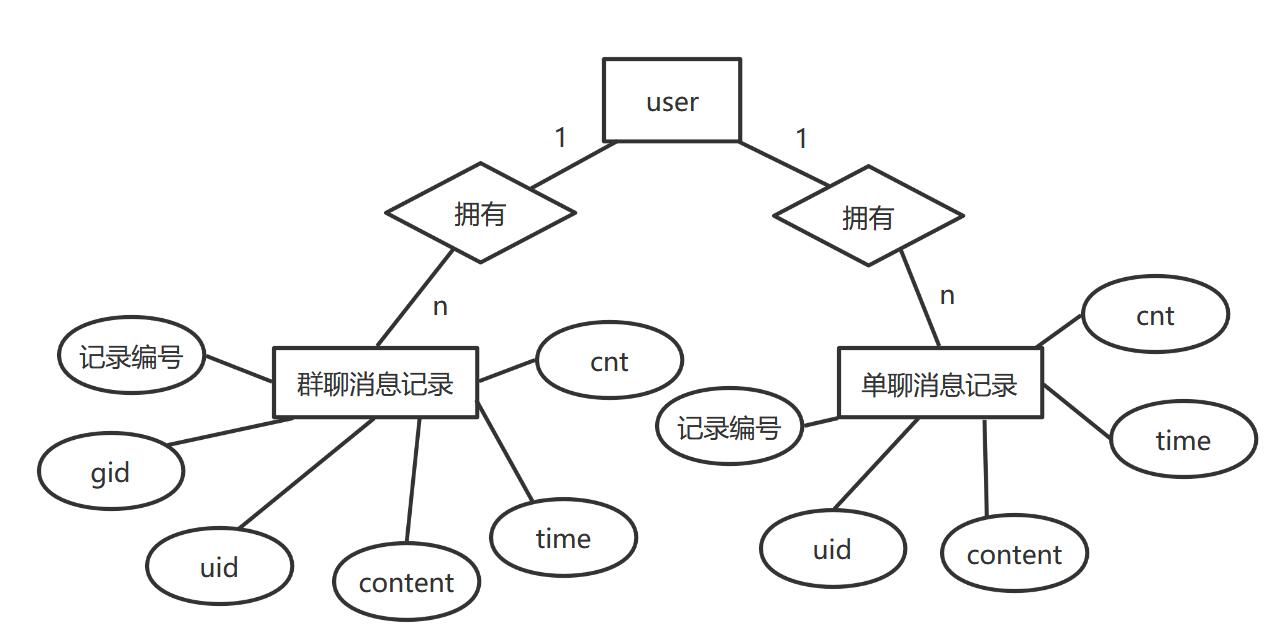


图 4-3 聊天ER图

### 表结构设计

把上述原始数据进行分解、合并后重新组织起来的数据库全局逻辑结构，包括所确定的关键字和属性、重新确定的记录结构和文卷结构、所建立的各个文卷之间的相互关系，形成本数据库的数据库管理员视图。

users表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| uid | 账号 | char(10) | 否 | 主键 |
| uname | 用户名 | nvarchar(20) | 否 |  |
| name | 真实姓名 | nvarchar(20) |  |  |
| psswd | 密码 | nvarchar(20) | 否 |  |
| sign | 个签 | nvarchar(50) |  |  |
| photo | 头像 | nvarchar(200) |  |  |
| email | 邮箱 | nvarchar(50) | 否 |  |
| phone | 电话 | varchar(20) |  |  |
| gender | 性别 | int |  |  |
| status | 状态 | int | 否 |  |
| startTime | 注册时间 | int | 否 |  |

friend表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| 记录编号 | 编号 | int | 否 | 主键 |
| me | 用户账号 | char(10) | 否 |  |
| ta | 用户账号 | char(10) | 否 |  |
| startTime | 时间 | int | 否 |  |
| remark | 备注 | nvarchar(20) |  |  |
| tid | 标签编号 | int |  |  |
| status | 状态 | int |  |  |

groups表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| gid | 群聊账号 | char(10) | 否 | 主键 |
| gname | 群聊名 | nvarchar(20) | 否 |  |
| photo | 头像 | nvarchar(200) |  |  |
| leader | 群主 | char(10) | 否 |  |
| sign | 群介绍 | nvarchar(50) |  |  |
| startTime | 创建时间 | int | 否 |  |
| sum | 人数 | int | 否 |  |

belong表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| gid | 群聊账号 | char(10) | 否 | 外键  主键 |
| uid | 用户账号 | char(10) | 否 |
| nickname | 群内昵称 | nvarchar(20) |  |  |

loginRecords表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| 记录编号 | 编号 | int | 否 | 主键 |
| uid | 用户账号 | char(10) | 否 |  |
| ip | IP地址 | nvarchar(50) | 否 |  |
| port | 端口 | int | 否 |  |
| time | 时间戳 | int | 否 |  |

tags表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| tid | 标签编号 | int | 否 | 主键 |
| uid | 用户账号 | char(10) | 否 |  |
| tname | 标签名 | nvarchar(20) | 否 |  |

群聊消息记录（本地文件）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| 记录编号 | 编号 | int | 否 | 主键 |
| gid | 群聊账号 | char(10) | 否 |  |
| uid | 用户账号 | char(10) | 否 |  |
| content | 内容 | nvarchar(MAX) | 否 |  |
| time | 时间戳 | int | 否 |  |

单聊消息记录（本地文件）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| 记录编号 | 编号 | int | 否 | 主键 |
| uid | 用户账号 | char(10) | 否 |  |
| content | 内容 | nvarchar(MAX) | 否 |  |
| time | 时间戳 | int | 否 |  |

证书表（AS数据库）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **是否可空** | **约束** |
| version | 版本号 | nvarchar(20) | 否 |  |
| serial | 序列号 | nvarchar(20) | 否 |  |
| deadline | 有效日期 | nvarchar(20) | 否 |  |
| name | 主体名 | nvarchar(20) | 否 | 主键 |
| pk | 公钥 | nvarchar(20) | 否 |  |

## 时序图

### 证书时序图

在Kerberos认证之前：证书发送，获取公钥。

User向AS发送自己的证书，即自己的公钥，AS通过该公钥加密一个随机数（Kc或Kv），作为Kerberos认证第一步的使用的Kc，以及User间，Client向Server转发使用的Kv。

Client与Server互发证书，使双方掌握自身公钥。



图 4-4 证书时序图

### Kerberos认证时序图

在应用程序之前：Kerberos认证，获取Session key

Client向AS发送请求，获取TGT，AS收到请求后，通过Kc加密生成票据；

Client凭票向TGS发送请求，获取Session key，TGS根据票据随机生成Session key，并用Kv（AS与Server间的Kc）加密，发送给Client。

Client将session转发给Server，Server回复，实现相互认证。

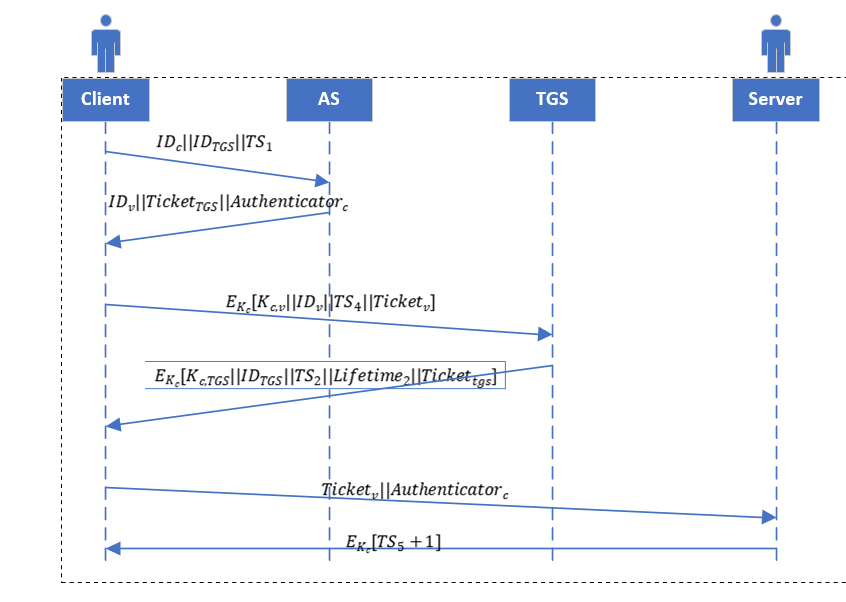


图 4-5 Kerberos认证时序图

### 应用时序图

应用程序：实现通讯

Client向Server发送注册请求，Server给予响应，Client体现在UI中。



图 4-6 请求时序图

Client 登录，先向Server发送信息，Server回复Client登录信息，若登陆成功，Client向Server发送问好信息，Server拉取消息队列中未读的信息给Client；然后Client向Server提出申请，请求自身好友列表等信息。



图 4-7 登陆时序图

Client与Server建立的为长连接，因此，每隔一段之间，Client都要向Server发送一段心跳包，若Server在限制时间内未收到心跳包，则说明Client处于离线状态，更新数据库。



图 4-8 心跳包时序图

Client A向B发信息（添加好友、组建群等），先向Server发送信息，Server先放入消息暂存队列并打上时间戳，根据信息转发给Client B，若B不在线，则消息暂存队列不出队，若B收到信息，Client B将回复传达给Server，Server根据回复内容更新数据库，并转达给Client A，Client A、B均更新UI。



图 4-9 信息发送时序图

## 数据报文

Json是Java和C#之间通信的利器，Java端将Java对象转变为Json串后发出，C#端接收到Json串后转换为C#对象；C#发出转变为Json串的对象，Java收到后解析成Java对象，Json串在不同语言之间起到一个桥梁的作用。对定义的Java或C#对象生成Json字串，以及从Json字串生成Java或C#对象，有很方便的方法。

同时发送方根据Json也可以将复杂的Socket报文序列化，而收方简单的反序列化就可以提取报文，因此本次通信数据结构采用Json模块。

|  |  |  |
| --- | --- | --- |
| Json数据字段 | 内容描述 | 细分 |
| msgId | 消息唯一标识符 | |
| Src | 发送者id | |
| Dst | 接收者id | |
| msgType | 消息类型 | 0001 发送证书报文 |
| 0002 AS向User发送Kc |
| 0003 Kerberos(1) |
| 0004 Kerberos(2) |
| 0005 Kerberos(3) |
| 0006 Kerberos(4) |
| 0007 Kerberos(5) |
| 0008 Kerberos(6) |
| 1001 注册信息 |
| 1002 登录信息 |
| 1003 请求好友界面 |
| 1004 好友界面信息 |
| 1005 问好信息（拉取离线信息），同时好友上线提醒 |
| 1006 心跳信息 |
| 1007 查找信息 |
| 1008 登出信息（Server端登出），同时好友离线提醒 |
| 1009 个人信息 |
| 1010 修改个人信息 |
| 2001 单聊信息 |
| 2002 群聊信息 |
| extend | 拓展字段，理论上可以继续通过Json序列化数据类型 | |
| statusReport | 状态报告（0，1)，msgType+statusReport等价于ack | |
| sTS | 发送方时间戳 | |
| rTS | 接收方时间戳 | |
| ContentType | 消息内容类型（一般为冗余，当消息类型为单聊或群聊时有用） | 101 文本消息 |
| 102 图片消息 |
| 103 语音消息 |
| 104 表情消息 |
| 105 动图消息 |
| 9001 添加好友信息 |
| 9002 加入群聊信息 |
| 9003 组建群聊信息 |
| 9004 退出群聊信息 |
| 9005 删除好友信息 |
| 9006 同意加好友 |
| 9007 拒绝加好友 |
| 9008同意加群 |
| 9009拒绝加群 |
| Xxx 其他消息类型 |

表 3 数据报文

# 详细设计

## 数据结构

报文数据结构，即Json数据结构：

|  |  |  |
| --- | --- | --- |
| 结构名 | 内容 | 功能 |
| Json | String msgId | 消息唯一标识符 |
| String Src | 发送者id |
| String Dst | 接收者id |
| String msgType | 消息类型 |
| String extend | 拓展字段，理论上可以继续通过Json序列化数据类型 |
| bool statusReport | 状态报告（0，1)，msgType+statusReport等价于ack |
| String sTS | 发送方时间戳 |
| String rTS | 接收方时间戳 |
| String ContentType | 消息内容类型（一般为冗余，当消息类型为单聊或群聊时有用） |

表 4 Json数据结构

程序内部数据结构：

|  |  |  |  |
| --- | --- | --- | --- |
| 报文 | 结构名 | 内容 | 功能 |
| 证书 | certificate | String version | 版本号 |
| String serial | 序列号 |
| String deadline | 有效日期 |
| String name | 主体名 |
| Long long pk | 公钥 |
| 密钥 | My\_k | String | 对称钥 |
| Client请求tgt | Message1 | String | Client端用户标识符 |
| String | 请求访问tgs |
| String | 验证时间同步 |
| 返回tgt | Message2 | String | Session key副本 |
| String | 确认ticket为TGS制作 |
| String | 签发时间 |
| String | 有效期 |
| Class | 访问tgs的ticket |
| 票据 |  | String | Session key副本 |
| String | Client端用户标识符 |
| String | Client地址 |
| String | 确认ticket为TGS制作 |
| String | 签发时间 |
| String | 有效期 |
| 请求Server ticket | Message3 | String | 用户访问服务器v的id |
| Class | 向TGS证实自己被认证 |
| Class | 验证ticket |
| 返回Server ticket | Message4 | String | TGS生成session key |
| String | 用户访问服务器v的id |
| String | 签发时间 |
| Class | Client访问server的ticket |
| 验证ticket | Class | String | Ticket的主人 |
| String | Ticket的主人地址 |
| String | 签发时间 |
| Client请求服务 | Message5 | Class | 向服务器证实被认证 |
| Class | 验证ticket有效 |
| 客户对服务器的可选认证 | Message6 | String +1 | 确认不是报文重放 |
| 注册信息、登录信息 | User | String uid | 账号 |
| String unmae | 用户名 |
| String name | 真实姓名 |
| String psswd | 密码 |
| String sign | 个签 |
| String photo | 头像 |
| String email | 邮箱 |
| String phone | 电话 |
| Int gender | 性别 |
| Int status | 状态 |
| Int startTime | 注册时间（以1970年标准时间为基准） |
| 好友信息 | Friend | Class User | 用户结构 |
| String remark | 备注 |
| Int startTime | 加好友时间 |
| String tid | 分组 |
| 聊天信息 | chat\_Message | Int head | 信息种类 |
| String content | 信息内容 |
| Class User | 发送方 |
| Int time | 时间戳 |
| 群 | Group | List map[User,nick] | 用户列表 |
| String gid | 群账号 |
| String photo | 群头像 |
| String leader | 群主账号 |
| String sign | 群介绍 |
| Int startTime | 创建时间 |
| 消息记录  record\_Message | | List[Class chat\_Message] | 接收到的信息 |

## 详细流程图

### AS流程

流程描述：AS只能接收数据报文首部为0001的证书信息，或者为0003的Kerberos认证信息：

当接收到0001的报文时，通过证书结构体反序列化Json，可以得到证书信息，从证书中提取公钥pk，并利用RSA加密将随机生成的Kc加密发送给Client，Kc存入AS数据库。

当接收到0003的报文时，通过Kerberos1结构体反序列化Json，得到，然后根据上文提到的对称钥Kc加密，并将Json报文发送给Client。



图 5-1 AS流程图

### TGS流程

流程描述：TGS只能接收数据报文首部为0005的Kerberos认证信息

当接收到0005的报文时，通过Kerberos3结构体反序列化Json，，AS与TGS共享数据库，TGS根据凭据，利用提前在AS与TGS提前约定好的Kc,v解密，验证身份，成功后，利用Kc加密并将Json报文转发给Client，若失败则发送ack信息。



图 5-2 TGS流程图

### Server流程

流程描述：Server有三种状态：等待连接状态、等待报文状态、处理报文状态。

最开始Server处于等待连接状态，当线程池中有线程接收到socket长连接后，改变状态进入等待报文状态，若此过程中，长期未收到心跳报文，则回到等待连接状态，若收到其他报文则进入报文处理状态，报文处理状态时，若收到了登出信息则跳转至等待连接状态。

当接收到0001的报文时，Server解析出证书信息，得到了证书主体对应的Pk，用于后续数字签名，同时将自己的证书发送给对应的Client，实现证书互信。

当接收到0007的报文时，Server根据认证模块中得到的Kv（即上述Kc），通过Des解密，得到。并根据中的时间戳，发送0008的报文，回复Client，完整Client与Server整个Kerberos认证模块。

当接收到1001，1010的注册，修改个人信息的报文，需要数据库的信息，会先验证数据库修改是否合法（如用户名冲、密码错误），然后返回ACK，若成功将状态为置为1，否则置为0，并返回给Client。

当接收到1005的问好信息，将数据库中用户的状态从离线转为上线状态，通知他的好友（即向好友转发问好信息），并将离线消息队列中的报文已2001单聊信息或者2002群里信息的格式发送给Client。

当长时间未接受心跳信息，或接收到登出信息时，修改数据库，并将等处信息通知他的好友，更新UI。

当接收报文为1002的登录信息时，Server根据内容验证，并返回ACK。

当接收报文为1003、1007、1009的搜索报文时，Server查询数据库，并将查寻到的列表通过Json序列号发送给Client。

当接受到群聊或单聊信息时，Server先将信息放入消息队列，由接收方是否上线的状态决定是否发送报文，然后根据消息内容类型判断报文为聊天信息，还是请求信息，执行不同逻辑。



图 5-3 Server流程图

### Client流程

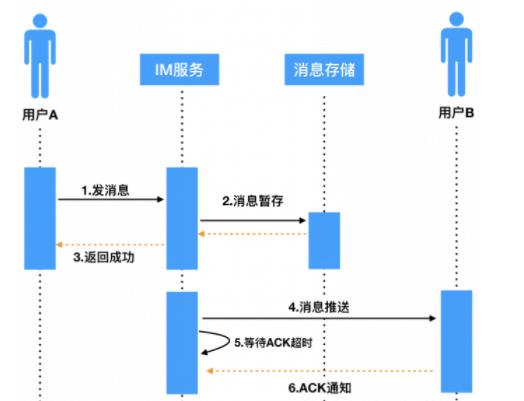
流程描述：

Client端Kerberos认证，证书发送，获取等与Server端代码无区别，因此没有重复体现。

当Client执行完认证等模块后，进入应用通讯程序，用户根据UI界面的状态选择功能。起始为登录界面，用户可以通过网页注册账号，或者在已有账号的状态下选择登录界面，与Server通讯，返回UI界面。

根据UI点击内容，与Server通讯不同的报文，执行聊天功能，如添加好友等。最后接收通过线程接收Server端报文，反馈UI。

可靠性考虑：



参考上面时序图，发消息大概整体上分为两部分：

1.用户A发送消息到IM服务器，服务器将消息暂存，然后返回成功的结果给发送方A（步骤1,2,3）

2.IM服务器接着再将短暂的用户A发出的消息，推送给接收方用户B（步骤4）

其中可能丢失消息的场景有下面这些：

在第一部分中，步骤1,2,3都可能存在失败的情况。

由于用户A发消息时一个请求和响应的过程，如果用户A在把消息发送到IM服务器的过程中，由于网络不通等原因失败了；或者IM服务器接收到消息进行服务端存储时失败了；或者用户A等待IM服务器一定的超时时间，但IM服务器一直没有返回结果，那么这些情况用户A都会被提示发送失败。

接下来，他可以通过重试等方式来弥补，注意这里可能会导致发送重复消息的问题。

比如：客户端在超时时间内没有收到响应然后重试，但实际上，请求可能已经在服务端成功处理了，只是响应慢了，因此这种情况需要服务端有去重逻辑，一般发送端针对同一条重试消息有一个唯一的ID，便于服务端去重。

第二部分中。消息在IM服务器存储完后，响应用户A告知消息发送成功了，然后IM服务器把消息推送给用户B的在线设备。

在推送的准备阶段或者把消息写入到内核缓冲区后，如果服务端出现掉电，也会导致消息不能成功推送给用户B。这种情况实际上由于连接的IM服务器可能已经无法正常运转，需要通过后期的补救措施来解决丢消息的问题，后续详细介绍。

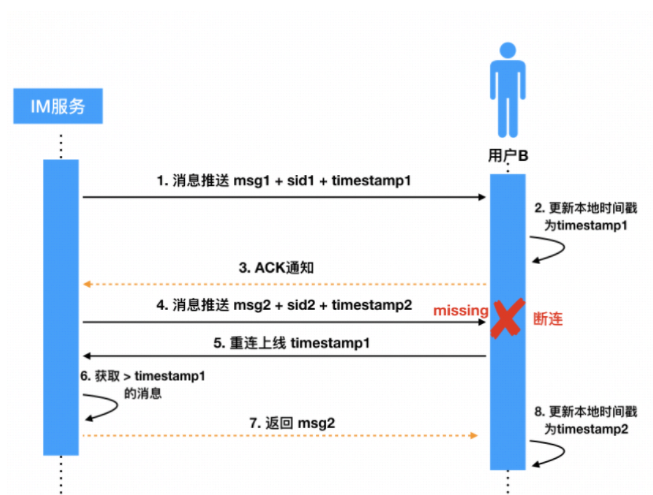
即使我们的消息成功通过TCP连接给到用户B的设备，但如果用户B的设备在接收后的处理过程出现问题，也会导致消息丢失。比如：用户B的设备在把消息写入本地DB时，出现异常导致没能成功入库，这种情况下，由于网络层面实际上已经成功投递了，但用户B却看不到消息。所以比较难处理。

解决方案：

1.针对第一部分，我们通过客户端A的超时重传和IM服务器的去重机制，基本就可以解决问题。

2.针对第二部分，业界一般参考TCP协议的ACK机制，实现一套业务层的ACK协议。

消息完整性检查



假设一台IM服务器在推送出消息后，由于硬件原因宕机了，这种情况下，如果这条消息真的丢了，由于负责的IM服务器宕机了无法触发重传，导致接收方B收不到这条消息。

问题在于：服务器机器宕机，重传这条路走不通了

那如果在用户B在重新上线时，让服务端有能力进行完整性检查，发现用户B有消息丢失的情况，就可以重新同步或者修复丢失的数据

比较常见的消息完整性检查的实现机制有时间戳比对

1、IM服务器给接收方B推送msg1，顺便带上一个最新的时间戳timestamp1，接收方B收到msg1后，更新本地最新消息的时间戳为timestamp1

2.IM服务器推送第二条消息msg2，带上一个当前最新的时间戳timestamp2，msg2在推送过程中由于某种原因接收B和IM服务器连接断开，导致msg2没有成功送达到接收方B。

3、用户B重新连上线，携带本地最新的时间戳timestamp1，IM服务器将用户B暂存的消息中时间戳大于timestamp1的所有消息返回给用户B，其中就包括之前没有成功的msg2.

4.用户b收到msg2后，更新本地最新消息的时间戳为timestamp2



图 5-4 Cilent流程图

# 模块详细设计

## Web服务器

Web-Server负责实现用户的注册和重置密码功能，其部署在Node.js平台下，借助Express（一种Web开发框架），使用JavaScript语言开发。网页使用HTML5/CSS/JavaScript编写。浏览器和Node之间使用HTTPS协议通信。

1. 注册模块

用户在填写注册信息表时，JS自动检查填写的合法性，唯有合法的输入才会使获取验证码和确认按钮可用；信息填写完成后，点击获取验证码，Browser使用Ajax向Node发送POST请求（url: /getvcode）；Node提取用户邮箱并发送6位随机数，同时向客户端发送POST响应提示发送成功。

用户填写验证码并确认；Browser向Node发送POST（url: /register）请求，请求体为注册信息（JSON数据）；Node提取请求体，验证合法性，更新数据库并响应。

1. 找回密码模块

用户填写信息获取验证码；Node获取对应的邮箱并向其发送6位随机数；用户输入验证码完成重置。

1. 函数设计

index.js

|  |  |  |  |
| --- | --- | --- | --- |
| **函数名** | **函数功能** | **参数** | **返回值** |
| toRegisterRegion | 进入注册窗口 | - | - |
| hideRegister | 隐藏注册区 | - | - |
| toLoginRegion | 进入登录窗口 | - | - |
| hideLogin | 隐藏登录区 | - | - |
| toResetRegion | 进入重置窗口 | - | - |
| hideReset | 隐藏重置区 | - | - |
| getvcode | 获取验证码 | number situ | - |
| sendRegisterInfo | 发送注册信息 | - | - |
| sendResetInfo | 发送重置信息 | - | - |
| disableBtn | 禁用按钮 | - | - |
| ableBtn | 启用按钮 | - | - |

app.js

|  |  |
| --- | --- |
| **URL** | **URL** |
| / | GET |
| /getRegisterVCode | POST |
| /register | POST |
| /getResetVCode | POST |
| /reset | POST |

|  |  |  |  |
| --- | --- | --- | --- |
| **函数名** | **函数功能** | **参数** | **返回值** |
| sendVCode | 进入发送验证码到指定邮箱窗口 | string email | - |
| addUser | 隐藏添加user到DB | object user | - |
| getUser | 获取某个user的信息 | id | object |
| changeUser | 修改某个user的信息 | object user | - |

## 服务器框架

### 详细设计

**1. 功能与业务**

服务器框架是AS、TGS和应用服务器公用的框架。后端的三种服务器都由服务器框架和挂载在框架上的业务逻辑组成。AS、TGS和应用服务器业务逻辑有区别，但服务器框架共用同一个。服务器框架的主要功能有：收发信息、线程池管理、服务器日志与管理三个部分。

**业务：**

服务器框架的业务有两类四种，第一类业务，为AS、TGS和应用服务器的业务逻辑提供通讯支持和算力支持。上述三类服务器的网络通信和线程调度都由服务器框架管理。第二类业务面向管理员，为管理员提供管服务器运行日志，便于对服务器进行维护。

1. **线程控制模块设计**

**①线程池的选择：**

这里使用的线程池的类型是CachedThreadPool初始状态下线程池中不存在线程，当有任务时立刻创建新线程，任务完成后线程进入线程池等待被调用，如果线程连续60S内线程没有被使用就会被停止资源被回收。

**②线程池的使用：**

框架main函数的进程为主进程，主进程的任务是运行日志与管理模块并持续监控并维护服务器的运行状态。线程池中的线程为工作线程，工作线程中有一个线程持续监听来自Client的请求，其他线程处于等待状态。当负责监听的线程接收到建立连接的请求，将从线程池中取一个线程与Client建立连接，监听线程维持监听状态不变。

1. **通信模块设计**

收信息时：一个线程监听端口接收请求但是不建立连接，当有Client请求时就从线程池中分配一个线程与Client建立连接接收信息，接收信息后调用对应业务逻辑的统一的入口函数对信息进行处理。

发送信息时：当业务逻辑有发送信息的需求是，需要调用框架中定义的发送信息函数sendMessage()。

1. **日志与管理模块设计**

**日志的记录：**

服务器每执行一个任务，就会有一条日志记录产生，但是这条日志记录不会立刻被写入到磁盘上的日志文件，而是被存入日志缓冲列表，每当列表中存在100条日志记录时，写进一次日志文件，并清空日志缓冲列表。值得注意的是，可能会存在同时有多个线程对日志缓冲列表进行写操作，同时还有可能在进行读和删除的操作，所以应该使用支持并发的集合类，常用的ArrayList和LinkList都不支持并发操作。解决方法是：使用Collections类成员函数synchronizedList函数给集合类实例上锁。

1. **框架的初始化过程：**

服务器框架启动时，首先初始化线程池，套接字初始化，挂载业务逻辑，从线程池中分配函数进入监听状态。应该先初始化套接字再挂载业务逻辑的原因是：挂载应用服务器业务的过程中需要想AS发送证书。

1. **工具类调用：**

调用Communication类。

### 数据成员

|  |  |  |  |
| --- | --- | --- | --- |
|  | **类型** | **变量名** | **功能** |
| Private | ExecutorService | cachedThreadPool | 线程池 |
| Private | Functionset | functionset | 挂载业务逻辑（AS、TGS、Sever） |
| Private | List<Log> | logbuf | 日志缓冲区 |
| Private | File | file | 日志文件的路径及文件名 |
| Private | Communication | communication | 套接字信息 |

**（3）函数设计**

**1. 函数列表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public |  | Main |  | 服务器程序入口 |
| Private | void | init | void | 框架初始化 |
| Private | void | initThreadPool | oid | 线程池初始化 |
| Private | void | initLogSystem | void | 初始化日志系统 |
| Public | Void | Send | String | IP地址 |
| int | 端口号 |
| String | 发送的信息 |

## 服务逻辑

### AS服务器

1. **详细设计**

**（1）功能与业务**

AS服务器可分为两个功能结构：服务器框架、AS业务逻辑。其中服务器框架已在（小节号）介绍，本节将重点介绍AS业务逻辑的部分。AS类是抽象类Functionset的派生类，AS服务器主要业务逻辑在类AS中实现。

**业务：**

AS服务器有两种业务：①与Client约定Kc并发送；②认证Client身份、生成票据并发送。其中Kc是身份认证和发送票据的基础，如果Client和AS服务器之间没有约定好的Kc，Client应该首先向服务器发送包含自己Pk的证书，AS服务器核实证书后使用Pk发送Kc，而Client发送给AS服务器的证书会被存入数据库直到证书失效。

**（2）模块设计**

根据AS服务器的两种业务可以将AS服务分为两个模块。

**①Kc模块：**当Client和AS事先没有约定好的Kc时，Client会向AS发送包含自身Pk的证书，AS会使用RSA非对称加密算法向Client端发送Kc之后进入Kerberos系统的流程。

**②票据模块：**Client向AS发送信息申请票据后，AS返回给Client的信息是：

**（3）工具类的调用：**

调用的外部类有：证书、数据库操作

**（4）接口与交互：**

AS服务器是一个完整、独立运行的程序，因此不提供对外函数接口，AS服务器通过网络使用报文和Client、TGS、应用服务器产生直接或间接的交互。

1. **数据成员**

|  |  |  |  |
| --- | --- | --- | --- |
| **权限** | **类型** | **变量名** | **功能** |
| Private | String | ktgs | 存储Ktgs，AS与TGS约定好的对称钥 |

1. **函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public | Void | entry | String message | 业务逻辑入口 |
| Private | String | getTicket | String idc | 生成Client要发送给TGS的票 |
| String adc |

### T**GS服务器（TGS类）**

1. **详细设计**

**（1）功能与业务**

TGS服务器可分为两个功能结构：服务器框架、TGS业务逻辑。其中服务器框架已在上节介绍，本节将重点介绍TGS业务逻辑的部分。TGS类是抽象类Functionset的派生类，TGS服务器主要业务逻辑在类TGS中实现。

**业务：**

TGS服务器有一种业务：接收Client的请求，核实Client身份并根据Client的请求给Client发送应用服务器的票据。

**（2）模块设计**

TGS服务器只有一种业务，只有一个模块。

**（3）工具类的调用：**

没有调用工具类。

**（4）接口与交互：**

TGS服务器是一个完整、独立运行的程序，因此不提供对外函数接口，TGS服务器通过网络使用报文和Client、AS、应用服务器产生直接或间接的交互。

1. **数据结构**

|  |  |  |  |
| --- | --- | --- | --- |
| **权限** | **类型** | **变量名** | **功能** |
| Private | String | ktgs | 存储Ktgs，AS与TGS约定好的对称钥 |

1. **函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public | void | entry | String message | 业务逻辑入口 |
| Private | Boolean | checkTicket | String ticket | 检查票据 |
| Private | String | getTicket | String idc | 生成Client发送给Sever的票 |
| String adc |

### **应用服务器**

1. **模块**
2. Server启动时首先调用加解密模块creataCertif函数生成证书，然后向AS发送证书；AS生成Kv并使用证书中的公钥进行RSA加密传输回复给Server。Kv用于Client的Kerberos认证，TGS生成的访问应用服务器即Server的票据使用Kv加密。Kv只有TGS和Server双方持有。之后Server与client互换证书，将client的证书保存在数据库，在数字签名阶段使用。
3. ServerFunction类继承框架模块的FunctionSet类。由Framework模块建立线程池，处理用户连接服务器的请求。Framework模块调用ServerFunction中的Integer函数，为每个上线用户建立一个单独的服务线程。上线用户需要定时向服务器发送心跳包，暂定为10s一次。当用户上线时，Server通知该用户的所有好友此上线信息。使用了观察者模式，被观测事件为用户状态，触发事件为通知该用户所有好友。
4. 状态机设计：

三个状态：

1. 等待连接State\_waitLink
2. 接收报文State\_waitInfo
3. 处理报文State\_handlerInfo

五个事件：

1. 连接成功事件Event\_linkSuccess，等待连接状态->接收报文状态，创建线程；
2. 收到信息事件Event\_infoReceive，接收报文状态->处理报文状态，处理报文；
3. 处理结束事件Event\_handlerSuccess，处理报文状态->接收报文状态，等待接收报文；
4. 断开连接事件Event\_linkOver，处理报文状态->等待连接状态，关闭线程；
5. 无心跳报文事件Event\_linkError，接收报文状态->等待连接状态，关闭线程。

|  |  |  |  |
| --- | --- | --- | --- |
| 当前状态 | 触发事件 | 下一状态 | 执行动作 |
| State\_waitLink | Event\_linkSuccess | State\_waitInfo | Action\_createThread |
| State\_waitInfo | Event\_infoRecive | State\_handlerInfo | Action\_handler |
| State\_waitInfo | Event\_linkError | State\_waitLink | Action\_closeThread |
| State\_handlerInfo | Event\_handlerSuccess | State\_waitInfo | Action\_waitReceive |
| State\_handlerInfo | Event\_linkError | State\_waitLink | Action\_closeThread |

1. Server主要包含六个模块：
2. 线程启动模块是（2）所描述的ServerFunction，主要作用为线程入口。
3. 加解密模块ServerSecurity包含DES加解密和RSA加解密函数、生成数字签名函数和生成证书函数。
4. 消息处理模块ServerHandler进行报文处理，每种报文都对应一个处理函数。
5. 消息回复模块ServerResponse进行报文发送，每种报文都对应一个函数进行消息生成及发送。
6. 数据库模块ServerDataBase进行数据库操作的封装，包括各种数据库查询和修改操作。
7. 界面显示模块，用于显示报文和log。



图 40 Server模块

1. **函数**

（1）消息处理模块

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 函数功能 | 参数 | 返回值 |
| Certif | 0001证书信息，存入数据库 | String message  报文 | boolean  是否成功处理 |
| Kv | 0002收到Kv，存入数据库 |
| regeister | 1001注册信息，存入数据库，返回ACK |
| login | 1002登录信息，返回登陆结果ACK |
| searchFriendList | 1003请求好友界面，查询数据库，返回好友列表 |
| hello | 1005问好信息，修改数据库，向好友转发 |
| heart | 1006心跳信息 |
| searchID | 1007查找信息，查询数据库，返回信息 |
| logout | 1008登出信息，修改数据库，向好友转发 |
| infomation | 1009查询个人信息，查找数据库，返回信息 |
| changeInfo | 1010修改个人信息，修改数据库，返回ACK |
| privateChat | 2001私聊信息，转发给目的Client |
| publiChat | 2002群聊信息，转发给群内其他成员 |

1. 消息回复模块

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 函数功能 | 参数 | 返回值 |
| sendCertif | 0001证书信息，发送证书 | String message  报文信息 | boolean  是否成功 |
| friendList | 1004回复好友界面 |

1. 数据库模块

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 函数功能 | 参数 | 返回值 |
| connectData | 连接数据库 | void | boolean  是否成功 |
| wCertif | 向数据库存入证书 | certificate证书 |
| wRegister | 向数据库存入注册信息 | User 用户信息 |
| wLogin | 向数据库写入登陆状态 | String status状态 |
| rFriendList | 查询好友列表 | String ID  用户ID | Friend 好友列表 |
| rSearchID | 查询用户 | User 用户信息 |
| rInfo | 查询个人信息 | User 个人信息 |
| wAddF | 修改好友列表，添加好友 | String IDA  String IDB  在A好友列表中添加B | boolean 是否成功 |
| wAddG | 修改群列表，添加群 | String IDA  String IDG  在A好友列表中添加群 |
| wDeleteF | 修改好友列表，删除好友 | String IDA  String IDB  在A好友列表中删除B |
| wDeleteG | 修改群列表，退出群聊 | String IDA  String IDG  在A好友列表中删除群 |
| wCreateG | 组建群聊 | String IDA  Group  A创建群 |

1. 加解密模块

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 函数功能 | 参数 | 返回值 |
| enDES | DES加密 | String message  报文信息  String key  所需密钥 | String message  加密结果 |
| deDES | DES解密 | String message  解密结果 |
| sigRSA | RSA数字签名 | String message  数字签名 |
| enRSA | RSA加密 | String message  加密结果 |
| createCertif | 生成证书 | void | String message  生成证书 |

1. **详细设计**

**（1）功能与业务**

AS服务器可分为两个功能结构：服务器框架、AS业务逻辑。其中服务器框架已在（小节号）介绍，本节将重点介绍AS业务逻辑的部分。AS类是抽象类Functionset的派生类，AS服务器主要业务逻辑在类AS中实现。

**业务：**

AS服务器有两种业务：①与Client约定Kc并发送；②认证Client身份、生成票据并发送。其中Kc是身份认证和发送票据的基础，如果Client和AS服务器之间没有约定好的Kc，Client应该首先向服务器发送包含自己Pk的证书，AS服务器核实证书后使用Pk发送Kc，而Client发送给AS服务器的证书会被存入数据库直到证书失效。

**（2）模块设计**

根据AS服务器的两种业务可以将AS服务分为两个模块。

**①Kc模块：**当Client和AS事先没有约定好的Kc时，Client会向AS发送包含自身Pk的证书，AS会使用RSA非对称加密算法向Client端发送Kc之后进入Kerberos系统的流程。

**②票据模块：**Client向AS发送信息申请票据后，AS返回给Client的信息是：

**（3）工具类的调用：**

调用的外部类有：证书、数据库操作

**（4）接口与交互：**

AS服务器是一个完整、独立运行的程序，因此不提供对外函数接口，AS服务器通过网络使用报文和Client、TGS、应用服务器产生直接或间接的交互。

1. **数据成员**

|  |  |  |  |
| --- | --- | --- | --- |
| **权限** | **类型** | **变量名** | **功能** |
| Private | String | ktgs | 存储Ktgs，AS与TGS约定好的对称钥 |

1. **函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public | Void | entry | String message | 业务逻辑入口 |
| Private | String | getTicket | String idc | 生成Client要发送给TGS的票 |
| String adc |

**工具类**

**（1）Communication类**

**1. 数据成员**

|  |  |  |  |
| --- | --- | --- | --- |
| **权限** | **类型** | **变量名** | **功能** |
| Private | String | ktgs | 存储Ktgs，AS与TGS约定好的对称钥 |

**2.函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public | void | listen | void | 监听端口 |
| Public | String | receive | void | 接收消息 |
| Public | void | send | String ip | 发送消息 |
| int port |
| String message |

1. **设计思路**

对socket套接字做的一次封装，使得建立连接的过程对外透明，降低了服务器框架代码的复杂度。socket通信过程在系统的多个地方都被用到，封装成类使得不用每使用一次socket就要重复一次socket通信过程的代码。

## Client界面

Client界面尚未详细明确内容，界面大致结构参照2.4 UI布局，需要通过程序实现，进一步美化，并完善基础功能（如UI体现时间、记录密码等细节功能），细节完善度参照个人意愿与能力。应用通讯主体功如下所示：

Client端由WPF编写的多个界面组成，通过用户对界面的各种操作实现整套数据发送的逻辑，如发送添加好友请求，发送群聊信息等。而对于数据接收，则通过线程监听实现，当线程收到信息后，通过解析、分析报文分类后抛出给界面线程，通过界面反馈给用户，并让用户进一步操作。

界面功能如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 界面名 | 功能 | 函数组成 | 参数列表 | 返回值 |
| 登录界面 | 封装并发送登录信息 | Login\_send | object sender, EventArgs e、账号、密码 | Bool，0为发送失败，1为登陆成功 |
| 被监听线程激活后，通知主界面登录是否成功，进入好友界面并发送界面请求 | Login\_recv | 报文信息 | Void |
| 用户点击后，弹出注册网页 | Hyperlink\_click | object sender, EventArgs e | Void |
| 用户点击后，弹出修改密码界面 | Hyperlink\_click | object sender, EventArgs e | Void |
| 进入好友界面时发送心跳报文 | Send\_Msg | 无 | Void |
| 启动程序后，从本地读取账号等缓存信息，进行初始化 | Init | 无 | Void |
| 用户界面 | 封装并发送聊天报文（封装单聊、群聊报文） | User\_click | object sender, EventArgs e、用户名、对象名 | Void |
| 弹出查找界面 | Search\_click | object sender, EventArgs e | Void |
| 被监听线程激活后，通知接收主界面信息，并对界面初始化 | Init（） | 数据报文 | Void |
| 封装并发送删除好友 | Del\_click | object sender, EventArgs e | Void |
| 弹出修改界面 | Update\_click | object sender, EventArgs e | Void |
| 修改界面 | 根据用户输入信息封装并发送修改信息 | Send\_up\_click | object sender, EventArgs e | Void |
| 查找界面 | 根据用户输入信息封装并发送查找信息 | Search\_click | 响应、输入信息 | Void |
| 封装并发送添加报文 | Add\_click | object sender, EventArgs e | Void |
| 被监听线程激活后，通知查找界面返回信息 | Search\_recv | 报文信息 | Void |
| 得到返回界面信息后，将搜索信息体现在界面中 | Init | 报文信息 | Void |
| 聊天界面 | 发送用户报文 | Send\_msg | 报文信息 | Bool |
| 被监听线程激活后，解析报文并体现在聊天界面 | Recv\_init | 报文信息 | Void |

由于Client主体都由WPF中线程组成，界面主体确定的情况下，主要流程都由界面中的函数响应完成，参数大多为界面中的全局变量。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 线程函数名 | 功能 | 参数 | | 返回值 |
| Init | 根据完成线程设置 | 配置文件路径 | | Void |
| createCertif | 生成证书 | 无 | | Class 生成的证书 |
| p\_Send | 发送数据报文 | 报文，目的地址 | | Bool |
| p\_recv | 接收数据报文 | 无 | | String 数据报文 |
| ThreadCallback | 回调函数，将报文通知主进程 | 数据报文 | | Void |
| enDES | DES加密 | String message  报文信息  String key  所需密钥 | String message  加密结果 | |
| deDES | DES解密 | String message  解密结果 | |
| sigRSA | RSA数字签名 | String message  数字签名 | |
| enRSA | RSA加密 | String message  加密结果 | |
| GetTimer | 作为事件，每隔一段时间发送心跳报文 | object sender, EventArgs e |  | |
| Save | 将登录界面信息保存到文件，用于记住密码等功能。 | Class 界面信息 | void | |

## Json报文

|  |  |  |  |
| --- | --- | --- | --- |
| Json函数名 | 功能 | 参数 | 返回值 |
| JsonHelper | 构造函数，初始化Json | 无 | Void |
| FromJson<T> | 反序列化 | Json序列化报文 | 反序列化数据结构 |
| ToJson<T> | 序列化 | 数据结构 | 序列化报文 |

## 工具类

### 证书生成

**Certificate类**

1. **数据成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 证书 | certificate | String version | 版本号 |
| String serial | 序列号 |
| String deadline | 有效日期 |
| String name | 主体名 |
| Long long pk | 公钥 |

1. **函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public | String | getCertificate | void | 获取证书 |
| Private | String | readCertificate | void | 从文件读取证书 |
| Private | String | createCertificate | void | 生成一个证书 |

1. **证书生成设计**

版本号默认为 EC1.0；

序列号默认为 0（不考虑证书盗用）

有效日期均为程序运行当日+1天

主体名为主机名称

公钥调用RSA的公私钥生成，并存入内存

### DES加密

**DES类**

1. **数据成员**

|  |  |  |
| --- | --- | --- |
| 默认密钥 | Keys | private static byte[] Keys = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF }; |

1. **函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **返回值** | **函数名** | **参数** | **函数功能** |
| Public | String | EncryptDES | 明文，密钥 | 加密字符串 |
| Public | String | DncryptDES | 密文，密钥 | 解密字符串 |

1. **代码设计**

在课堂实验中，我们已经封装好自己的DES加解密工具，不过代码结构上并不美观，效率也不如，C#和JAVA语言自带DES加解密模块的函数，因为，我们选择调用系统内部函数，将函数封装为上述两个函数。

### RSA解密

**RSA类**

1. **数据成员**

无成员。

1. **函数设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 返回值 | 函数名 | 参数 | 函数功能 |
| Public | Void | RSAKey | 私钥，公钥 | 随机生成公私钥 |
| Public | String | RSAEncrypt | 公钥，明文 | 用公钥加密明文 |
| Public | String | RSADecrypt | 私钥，密文 | 用私钥解密 |
| Public | Bool | GetHash | 待签名的字符串，Hash描述符 | 获取Hash描述符 |
| Public | bool | SignatureFormatter | 私钥，待签名的Hash描述，签名后的结果 | 通过私钥，对MD5加密后的数据进行数字签名 |
| Public | String | SignatureDeformatter | 公钥，Hash描述，签名后的结果 | 数字签名验证 |

1. **代码设计**

同DES加密，也进行封装，提供上述接口。

# 任务分工

# 调试设计