1.电子宠物

2.类QQ聊天工具

3.游戏（坦克大战，魔兽，CS）

# 类QQ软件系统设计书

## 需求分析

### 1.1功能需求分析

QQ系统最核心的功能就是即时聊天，聊天又不是任意两个人就可以彼此聊天，只有好友之间才可以互相通信，所以这需要一个好友管理。所以本系统包括两个主要功能：

1. 在线即时聊天：发送文本、发送图片、发送文件、魔幻表情（可以输入文本，进行修改播放文本）、视频和语音。
2. 好友管理：添加好友、删除好友、黑名单、好友分组、最近联系人和查找好友。

本系统总体用例图如图1.1：



图1.1系统总体用例图

### 1.2主要功能流程分析

聊天的流程图如图1.2所示：



图1.2系统聊天流程图

好友管理流程图如图1.3所示：



图1.3好友管理流程图

登录流程图如图1.4所示：



图1.4登录流程图

### 1.3性能需求分析

可用性、可修改性、性能、安全性、可测试性和易用性

## 系统总体分析

### 2.1网络拓扑结构

本系统采用的体系结构是C/S，服务器是本系统的中心，所有的客户端全部跟中心服务器通信，然后服务器再转发给相应的客户端。其网络拓扑结构图如图2.1所示：



图2.1系统网络拓扑结构图

本系统使用分层结构，可以用MACV表示，其中M是数据持久层，即数据库；A表示数据访问层，屏蔽存储数据方式的不同性，用一个统一的接口供给业务层使用；C表示业务层，它来完成系统的绝大部分功能，包括网络中的信息传递，使得网络传输对表现层透明化；V表示表现层，主要用来和用户交互，展示数据。系统的分层结构图如图2.2所示：



~~图2.2系统分层结构图~~

### 2.2数据库设计

数据库概念设计，本系统最主要的实体是用户，它的属性包括用户编号、用户名字、密码、性别、状态、注册日期、生日和地址。如表2.1所示：

附加状态表，属性包括：状态编号、状态名称。如表2.2所示：

因为有好友管理所以再有一个好友表，它属性包括本人编号、好友编号、分组编号。如表2.3所示：

附加权限分配表，属性包括：本人编号、好友编号、权限编号。如表2.4所示：

附加权限表，属性包括：权限编号、权限名称。如表2.5所示：

这里又多了一个分组表，它的属性包括用户编号、分组编号和分组名称。如表2.6所示：

可能系统还要保留所有的聊天记录，所以需要一个聊天记录表，它的属性包括发送者、接受者、内容、时间、是否转发（起到留言作用）。如表2.7所示：

1. 用户（户编号、用户名字、个性签名、密码、性别、状态、注册日期、生日、地址）
2. 状态表（状态编号、状态名称）
3. 好友表（本人编号、好友编号、备注、分组名称）
4. 权限分配表（本人编号、好友编号、权限编号）若权限放到上一个表中，一对好友有两条记录，现在只有特殊设置才插入一条记录
5. 权限表（权限编号、权限名称）
6. 分组表（用户编号、分组名称）
7. 聊天记录（记录编号、发送者、接收者、内容、时间、图片个数）
8. 留言记录（聊天记录编号、接收者）若留言放到上表中，那么每条记录要多一个字段，现在动态处理留言
9. 加友请求（请求编号、发送者、接收者、请求消息、时间）动态表

表2.1用户信息表

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 是否主键 | 解释 |
| uID | int | yes | 用户编号 |
| uname | int | no | 用户姓名 |
| upswd | string | no | 密码 |
| usex | int | no | 性别 |
| ustate | int | no | 状态 |
| sdate | datatime | no | 注册日期 |
| bdate | datetime | no | 生日 |
| add | string | no | 地址 |

表2.2状态表

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 是否主键 | 解释 |
| sID | int | yes | 状态编号 |
| sname | string | no | 状态名称 |

表2.3好友表

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 是否主键 | 解释 |
| me | int | yes | 本人 |
| you | int | yes | 好友 |
| team | int | no | 分组编号 |
| priority | int | no | 好友权限 |

以上只是简单的设计，最后定形还要看最后的实现，因为可能涉及到其他必需的动态属性，用来辅助系统的实现。

## 系统模块设计和实现

### 3.1业务层设计和实现

首先既然是C/S体系结构，那么肯定要分为客户端和服务器两个模块。连个模块通过网络通信，所以一定要涉及一种通信协议。上面已经提到了要用业务层屏蔽网络，使得表现层就像在本地读取数据一样。所以业务层的模块划分如图3.1所示：



图3.1业务层模块划分图

现在讨论通信模型，有几种客户端解决方案：

1. 每个客户端与服务器只用一个套接字链接，全部命令和消息都通过这个套接字链接传输，然后被两端的解析器解析，如果是命令执行相应的操作，如果是消息则转发给相应的用户。其结构如图3.2所示：



图3.2方案1结构图

1. 每个客户端与服务器有两个套接字链接，一个负责传输消息，一个负责解析并执行命令。负责传输消息的要可以分辨发送者和接受者。其结果如图3.3所示：



图3.3方案2结构图

1. 每个客户端与服务器有多个套接字链接，一个负责命令解析并执行，其余的负责传输消息，其数量等于正在聊天的人数，一个正在聊天好友独占一个套接字链接。这样负责发送消息的链接可以不用具备分辨发送者和接受者的能力了。
2. 每个客户端与服务器有多个套接字链接，一个负责命令解析并执行，其余的负责传输消息，其数量是由配置文件设置的，也就是一个进程池（每个进程都有一个套接字链接）池，当需要时候就从其中找到清闲的链接发送出去。
3. 不使用套接字连接，使用无连接的UDP，这样在服务器端只需两个监听器，一个转发消息，一个用于解析和执行命令。这样客户端就不需要套接字链接管理了，只当需要时候发送数据报就可以了。如果怕数据报丢失，可以在UDP上在创建一层，用于容错。
4. 服务器端拥有两个监视器，一个用来转发消息，使用UDP协议，一个用来分配服务器线程，与客户端建立命令链接，使用TCP协议。

在这里不管哪种方案都要考虑虚拟技术，为每个链接字链接配置缓冲区，以避免阻塞。方案1是最节省套接字资源的方案，解析器工作很繁重，适合配置低的计算机；方案2较方案1有点改善，分开了消息和命令；方案3的吞吐量不受限制，但是一旦用户量过大，易于消耗光资源系统崩溃；方案4的吞吐量是固定的，用户量超额时候，只是反应延迟大了点，但是不会系统崩溃。

综上考虑，可以优先考虑使用的方案是方案2、3、5和6。本系统使用机制和策略分开的编程思想，把各个模型做成模块，系统运行时可以动态确定，所以在程序运行开始，需要一个初始化套接字链接，通过它传输参数确定使用的通信模型，当建立模型结束后，这个初始化套接字链接变成命令套接字链接。

接着讨论服务器端的服务器线程分配：

1. 一个客户分配一个线程，当客户端请求来临时，为其创建专属服务线程；
2. 使用线程池，当客户端请求来临时，从线程池中取一个闲线程为其服务，当全部线程都不闲时，阻塞用户请求。

### 3.2模块接口分析

上面提到了机制和策略分离的编程思想，这就要定义一个统一的接口，该接口必须是完备的，而且是清晰的。下面来设计通信模块的接口：

1. 客户端通信模块的接口如表3.1所示：

|  |  |  |
| --- | --- | --- |
| **接口声明** | **函数功能** | **参数解释** |
| mcreate(int afrd) | 创建或获取通信模块 | afrd是好友的编号 |
| mcreateCmd() | 创建命令通信模块 | 无 |
| msentCmd(string acmd) | 发送命令 | acmd是发送到命令 |
| msentTxt(string atxt) | 发送文本 | atxt是发送的文本 |
| msetTxtOper(Oper aoper) | 设置接收到文本后的处理器 | aoper就是处理器 |
| msetCmdOper(Oper aoper) | 设置接收到命令后的处理器 | aoper就是处理器 |

1. 服务器端通信模型的接口如表3.2所示：

|  |  |  |
| --- | --- | --- |
| **接口声明** | **函数功能** | **参数解释** |
| mcreate(Socket askt) | 创建或获取通信模块 | askt初始化套接字连接 |
| msentCmd(string acmd) | 发送命令 | acmd是发送到命令 |
| msentTxt(string atxt) | 发送文本 | atxt是发送的文本 |
| msetTxtOper(Oper aoper) | 设置接收到文本后的处理器 | aoper就是处理器 |
| msetCmdOper(Oper aoper) | 设置接收到命令后的处理器 | aoper就是处理器 |
| run() | 运行通信模块开始监听，运行服务线程分配线程 | 无 |

interface Oper{

public void mdealWith(string atxt){

}

}

~~为了更加优良的可修改性，本系统采用回调函数（把通信模块再分成两层），一层负责传输数据，（屏蔽通信模型做不到）负责传输字符串；然后再调用~~

？？本系统最终决定把好友管理和聊天模块做成工具类，即没有状态。

在此处，对系统有了更进一步了解，可以对照一章的流程图分析，在用户的每一步之间，系统完成的工作了。

学习总结：在对系统划分模块或者分层时，要先对接口进行分析，然后再进行实现的考虑，此时考虑就会事半功倍。顺着功能流程图进行接口调用的考虑，并把接口完成的操作尽量写明，再对系统进一步了解后再补充。

登录中点击确定后，系统验证用户密码，如果成功，有两种初始方案：

1. 初始化好友列表，系统打开好友列表界面，初始化链接变成命令链接。
2. 打开好友列表界面，然后初始化好友列表，初始化链接变成命令链接。

所以好友管理模块有一个登录接口，login(int aid,string apswd)，这个函数的流程为：把命令翻译成命令字符串，然后调用msentCmd发送给服务器。

在这里决定客户端每一窗口拥有一个业务模块；服务器端每一个线程有一个业务模块，每个业务模块有一个好友管理模块或者聊天模块。所以业务上层负责把命令翻译成字符串用业务下层发送。

学习总结：这种简单模块划分方法对单线程有效，对于多线程就显得有点乏力了，所以先创建好线程模型，然后针对每个线程进行模块划分，在其中有一个编程技巧就是共享资源使用factory类或者静态字段控制存储。

~~服务器端聊天线程接到文本后，与接收人创建链接，接着创建服务器聊天模块。~~

1. ~~接到文本~~
2. ~~创建与接收人的a链接~~
3. ~~用接收人编号和链接a创建当前链接的聊天模块。~~
4. ~~用当前链接和本人编号创建a链接的聊天模块。~~

本系统决定全部链接都是使用服务器的主监听器创建，然后根据最初传送的初始化消息确定其性质，即命令链接或者聊天链接。

这里有两个业务上层的设计方案：

1. 无状态的工具类，这时它要包括全部的链接，工作时候用参数决定用哪个链接。
2. 此方案要包括状态，一个聊天会话一个该类，链接如果没有辨别好友的能力，那么这些类将会全部存储在一个map里面，有通信模块选择合适的该类转发。

上面的方案1与客户端解决方案1和2可以很好协作。上面的方案2与客户端解决方案3、4、5和6很合适。

### 3.3数据访问层模块设计和实现

每一个表或者视图分配一个类，类似面向对象的数据库。

这里面固定的操作可以做成数据库存储过程或函数，也可以用高级语言实现。

## 实现系统

本系统使用迭代式开发方式，现在实现客户端解决方案2和业务上层方案1的组合，类图如图4.1所示：



图4.1客户端类图

本系统使用迭代式开发方法，第一阶段完成通信模块，然后是分别向上下。

每个客户端与服务器有两个套接字链接，一个负责传输消息，一个负责解析并执行命令。负责传输消息的要可以分辨发送者和接受者。一个客户链接分配一个线程，当客户端请求来临时，为其创建专属服务线程。

服务器端的类设计如图4.2所示：

图4.2链接创建时序图



~~图4.3服务器类图~~

图文同时传输问题，使用传输协议。

~~本系统决定：每个客户端使用三个链接，一个用来传输好友管理命令，一个用来传输文件，最后一个用来传输文本消息。现在讨论图文同传到解决方案。使用协议方式，链接的建立和释放全部是由通信模型控制，消息的处理是由业务层处理。~~



~~图4.4服务端类图~~

主要代码的实现：

1. 主监控器创建：

WHILE运行中

获得SOCKET

创建专属线程（SOCKET）

运行专属线程

1. 专属服务线程创建连接：

本系统使用静态的属性管理共享的套接字链接。

学习总结：本系统从已知的最明确的功能（在线聊天）入手，然后罗列出系统功能，接着开始系统功能各个流程分析，再然后是体系结构的选择和数据库的设计，接着又划分模块，然后是接口设计，套接字等资源的配置方法，系统线程模型。

从这次系统分析设计过程中，可以看到系统的分析应该从已知的最明确的点出发，然后一步一步的推出未知，一次不能得到全部的内容，记住过程是迭代式的。在这次过程中，认为一般过程可以是这样的功能需求分析，功能流程，体系结构，线程分配工作模型，其他一些资源（套接字）分配方式，每个进程的模块划分（这里面对共享资源可以用一个factory类或者静态属性管理），接口分析和类协作图，这个过程并不是一走到底的，它存在反复性，任意一步可能退回前面步骤补充。

类分为有状态的类和没有状态的类。

## 五、重新设计

模块名称—>责任—>接口。

### 5.1通信模块

1.通信模块ComModuleClt和ComModuleSer分别是客户端和服务端的通信模块。假设网络上不会丢包，不会出现传输错误的包，现在设计简单的通信协议。

2.通信模块的责任是，建立网络链接，发送接收命令，发送接收消息，关闭链接。

Clt：initCom、sentCmd、sentInfo、getCmd、getInfo、closeCom。

Ser：initCom(Socket)、sentCmd、sentInfo、getCmd、getInfo、closeCom、getComModule(id)。

3.演绎验证。

客户端：

|  |
| --- |
| 初始化：  com=new ComMudule();  com.initCom();//初始化命令和消息链接  登录：  crateLogString(str);  sentCmd(str);  if(getCmd()==ok){  }  关闭：  closeCom();//简单关闭socket |

服务端：

|  |
| --- |
| 初始化：  socket=serSocket.accept();  new Thread(new ComRunnable(socket){  public run(){  com=new SerComMule();  com.initCom(socket);//初始化命令和消息链接  work(com);//启动监听线程  }  }).start;  转发：  com=ComModuleSer.getComModule(id);  com.sentInfo(strInfo);  关闭：  ComModuleSer.delComModule(id);  com.closeCom();//简单关闭socket，关闭协议由上层完成 |

4.修改设计

现在通信模块包括三个socket，一个同步命令使用，一个消息转发和接收服务器通知使用、一个创建新链接使用（可能被意外中断）。

### 5.2业务模块

1.业务模块WorkClt和WorkSer分别是客户端和服务端的业务模块。

2.业务模块功能较多，系统的功能全部是该部分实现的，包括：登录、读取好友列表、读取留言、查找好友、增删修改好友、好友组管理、聊天、关闭。

WorkClt：setCom、login、getFrdList、getNotes、startupListener 、findFrd、~~showFrd~~、addFrd、delFrd、renameFrd、addGroup、delGroup、renameGroup、moveToGroup(id,group)、sentInfo、getInfo、requestClose、closeWork。

WorkSer：（反射机制）setCom、startupListener、login、getFrdList、getNotes、findFrd、~~showFrd~~、addFrd、delFrd、renameFrd、addGroup、delGroup、renameGroup、moveToGroup、sentInfo、getInfo、closeWork。

3.演绎

客户端：

|  |
| --- |
| 初始化：  clt=new WorkClt();  clt.setCom(com); // finish the work before authentication  if(true==clt.login(id,pswd)){ //authentication  //here,all function are available  getFrdList(frdMapgroup);  getNotes(noteMapfrd);  //maps’ elements are LinkList, one is class User, other is class Info  startupListener();  }  搜索好友：  frd=new User();  frd.setid(id);  frd.setname(name);  frd.setaddress(string);  findFrd(frd);//fill class user, and viewer get Info from frd after called  好友管理：~~//local friend list update for each 10 days~~  addFrd(id,request);//asynchronous，not wait for response  //what fllowing are synchronized  delFrd(id);  renameFrd(id,newname);  好友组管理：  addGroup(groupname);//forbid the same group name  delGroup(groupname);//deleting close friend group forbidden  renameGroup(old,new);  moveToGroup(Frdid,groupname);  聊天：  sentInfo(id, info);  sentInfo(id,infolist);  /\*\*  infolist=new LinkList<InfoNode>();  class InfoNode{  public int iType;//1.string2.photo  public Object iInfo;  }  \*\*/  info=new Info();  getInfo(info);  /\*\*  class Info{  private int fromid;  private long date;  private LinkList<InfoNode> info;  }  \*\*/  关闭：  requestClose(){  com.sentCmd(“close request”);  com.sentInfo(“close request”);  }  //received agree close request  closeWork();//简单关闭Com |

服务器端：

|  |
| --- |
| 初始化：  com.initCom(socket);  ser=new WorkSer();  ser.setCom(com);  ser.startupListener();//start up two listeners  监听：  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  string=com.getCmd();  scanner=new Scanner(string);  cmd=scanner.next();  WorkSer.getClass().getMethod(cmd,new Class[]{Scanner.class})  .invoke(WorkSer.this, new Object[]{scanner});  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  string=com.getInfo();  scanner=new Scanner(string);  fun=scanner.next();  WorkSer.getClass().getMethod(fun,new Class[]{Scanner.class})  .invoke(WorkSer.this, new Object[]{scanner});  登录：  //after login success  ComModuleSer.addComModule(id,com); |

## 六、系统分析与设计心得体会

### 6.1划分模块



图6.1模块划分工作



图6.2高层系统和低层系统关系

黑盒化低层系统，主要集中在高层系统的类型，完成的工作和怎么完成。在确定某层系统完成工作时候，可以考虑它透明化什么，使得上层不用考虑什么，从而知道它的职责所在，不出现越权现象。两层之间往往需要传递数据，好的数据载体就是一个设计优良的数据结构，可以尽量使得类的功能单一化。

### 6.2数据库设计

我认为数据库设计应该分为两步走：

第一步：客观实体字段设计，保证满足功能需求

第二步：系统辅助字段设计，帮助系统完成功能

### 6.3系统设计和分析过程

可以考虑到设计分析过程是：

系统类型(who)——>

功能需求分析（what）——>

用例流程设计（how）——>

性能需求分析——>

数据库第一步设计——>

模块划分——>

线程分析建模——>

模块和线程合并，资源分配——>

自顶向下分析接口和实现，注意越权现象——>

同步和互斥分析，使用同一个资源，多线程引起问题，分析时候假设多个

线程同时运行，检查是否出现问题——>

数据库第二步设计——>

### 6.4以ChatOnline系统为例子分析

任何系统分为初始期、工作期和关闭期，其中最重要的是工作期，同时它也是最稳定的一个状态。在分析时候，先从工作期开始，然后确定初始期需要创建哪里变量，申请哪些资源等等。



图6.3系统一般状态图

系统的功能其实就是View层的功能，然后用例流程就是该层的how。注意View只负责传输数据到下一层和更新GUI，其他不做。下面以ChatOnline的发送消息为例说明自顶向下设计过程：

1功能用例：



图6.4发送消息用例图

2功能流程：



图6.5用例流程图

|  |  |
| --- | --- |
| 动作对象 | 动作 |
| 用户 | 输入文本或图片，点击发送 |
| 系统 | 从输入框收集数据，封装到数据结构里 |
| 系统 | 发送数据 |
| 系统 | 显示刚刚发送的数据 |

3推演下一层的接口：

逻辑层有两个接口，负责发送消息，需要对方帐号和消息为参数。函数声明如下：

void sendInfo(int aid, String ainfo);

void sendInfoWithPhoto(int aid, InfoWithPhoto ainfo);

决定使用封装带有图片消息的数据结构是：

|  |
| --- |
| class InfoWithPhoto{  public List<String> iTxt;  //第一个图片插入第一个字符串后面，第一字符串是空，则以图片开始  public List<Photo> iPhoto;  } |

现在开始新的一轮分析，分析刚刚推演演绎出来的接口sendWithPhoto。

1功能：负责把数据发送到服务器端，需要响应以建立图片文件传输通道，异步函数，因为会用到cmdSocket，所以不能让服务器接收到其他命令，否则服务器会对命令的响应和当前响应会冲突，涉及到互斥操作。

2流程：



图6.6发送带有图片的消息时序图

这接口的伪代码实现如下：

|  |
| --- |
| iCom.sendInfo(“sendInfoWithPhoto ”+ainfo.toLine());  iCom.getCmd();  socket=iCom.getLink();  new Thread(socket,ainfo){  void run(socket){  socket.sendString(ainfo.getFileLine());  }  };  class InfoWithPhoto{  public List<String> iTxt;  public List<Photo> iPhoto;  String toLine(){  StringBuilder bbld=new StringBuilder();  for (String term : iTxt)  bbld.append(term+(char)0);  return bbld.toString();  }  } |

