软工设计文档

钱桥，许建林，谭志鹏，黄鑫，张道维，任勇

November 16, 2013

Abstract

本文是清华大学软件工程大作业的设计文档，主要从: 概述，服务器，客户端三个题目来分别叙述.

Contents

1 概述

简介:

该部分从功能角度描述了 QQ 糖这款游戏的需求。功能需求共有五大部分，分别是用户身份验证、玩家数据、游戏大厅、玩游戏、好友系统。前四部分已基本确定，并将需求细化至可检测的几小条。第五部分好友系统并不影响整体框架，且是附加功能，所以具体的功能、需求以及呈现方式将在工程后期再酌情添加。

此外，在前四部分中有一些附加功能，它们在作业中并没有要求（但可提高用户体验）。所以，这些部分的实现优先级较低。

1.1 项目结构

1. 项目分为服务器端和客户端两部分。服务器端运行在服务器上，负责用户数据存储，游戏大厅管理以及游戏数据的运算。客户端运行在用户的手机上，负责与用户的交互。

2. 在本项目中，客户端需要与服务器进行通讯。考虑到游戏的即时性，我们并没有使用 Http 协议进行通讯，而是使用 Socket 进行通讯。通讯的所有数据均为 JSON 格式。在开发过程中，所有通讯数据的格式都详细写在 wiki 中，客户端与服务器必须按照该标准进行数据传输与解析。

3. 用户可以在客户端中任意界面（非注册和登陆）进行登出操作，并返回登陆界面。

1.2 服务器端结构

1. 服务器端可拆分为四个工程，它们的分工为:

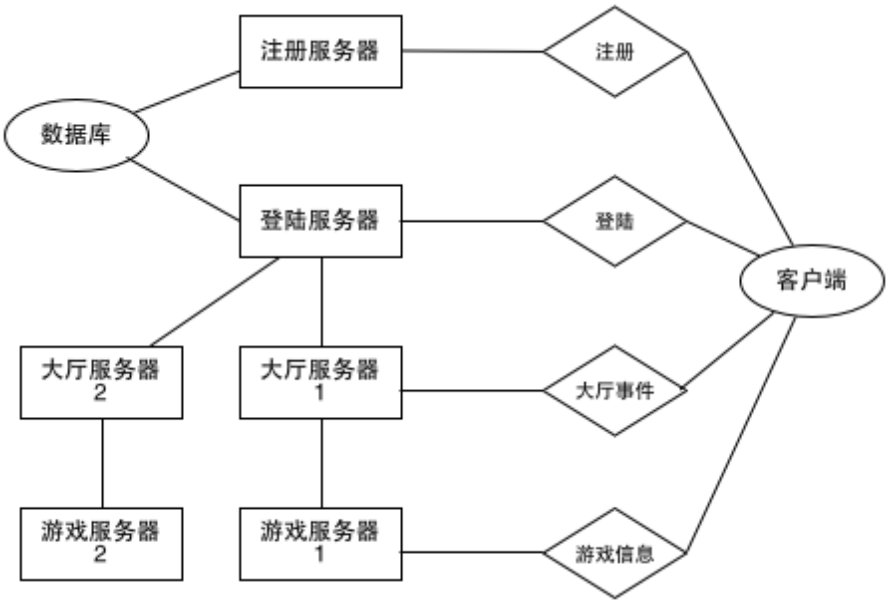

Figure 1: 服务器端结构

   (a) 登陆服务器—- 负责验证用户信息
   (b) 大厅服务器—- 负责维护大厅、房间数据
   (c) 游戏服务器—- 负责维护游戏数据
   (d) 注册服务器—- 负责进行注册操作

2. 这样设计的优点是:

(a) 可由一个人单独负责一个工程，减少代码冲突。

(b) 四个工程可以运行在四台服务器上，均衡负载。

(c) 当用户量增大时，可以新增大厅服务器和游戏服务器来缓解压力。

1.3 客户端结构

(a) 客户端采用了经典的 MVC(Model-View-Controller)设计模式。重要的逻辑控制全部在 Controller 类中完成，具体的通信、数据处理等均在相应的 Model 里面完成，显示全部在 Viewer 类中完成。而 Controller 则实现了典型的 Singletom 设计模式。

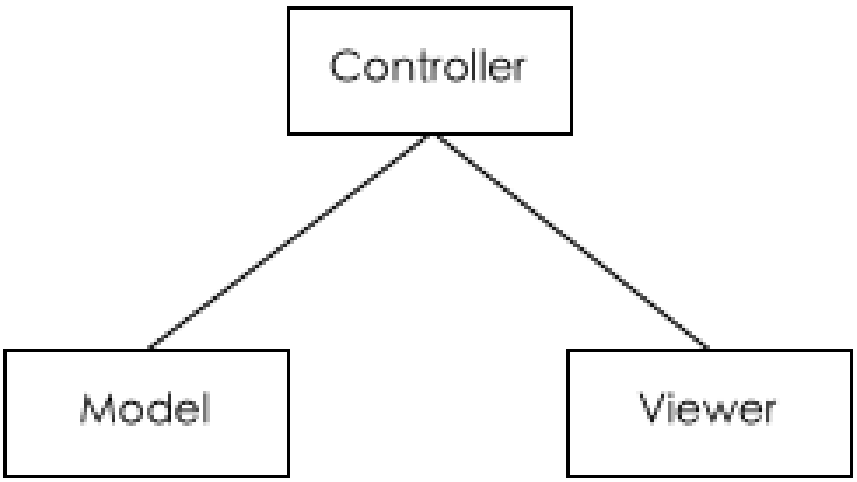

Figure 2: 客户端结构

i. 以登陆模块举例，SigninModel 负责与登陆服务器通信，并将通讯结果告知 Controller，由 Controller 控制 Viewer 显示。再以游戏模块举例，游戏画面的显示全部在 Viewer 内完成，Viewer 不处理任何逻辑，它只需要向 Controller 索要当前的游戏画面(静态)，然后绘制即可。

ii. 此外，Model、Viewer、Controller 都拥有自己独立的线程，不会互相阻塞。

2 服务器端

2.1 数据库结构

3. 数据库中共有两张表。分别是用户注册信息以及用户游戏信息:

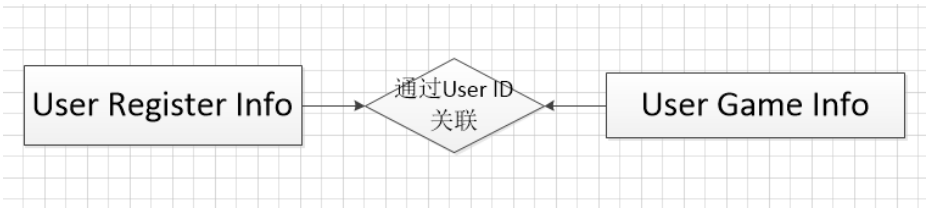

Figure 3: 注册信息

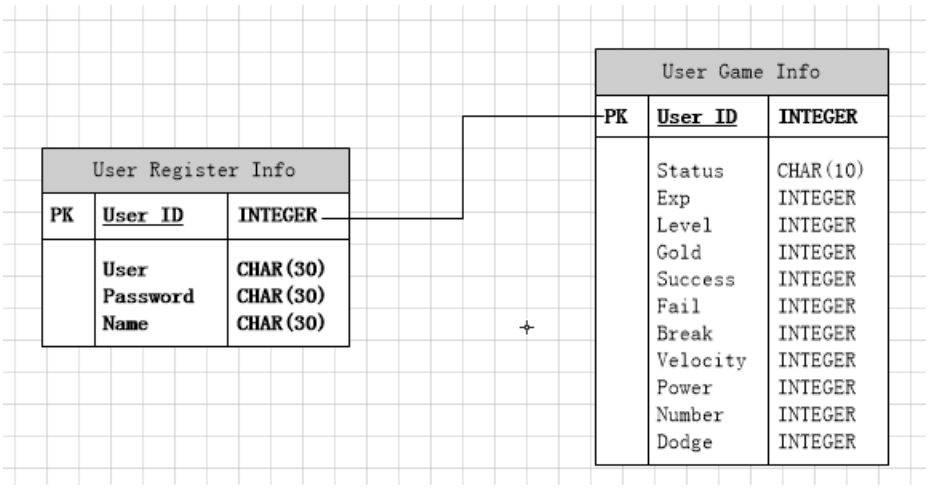

Figure 4: 用户游戏信息

(a) **用户注册信息:**UserID、User(用户名)、Password(登录密码)、name(昵称)。其中 User 为主键。其余键对应的值均为必须，即不能为空。

(b) **用户游戏信息:**UserID、Status(登录状态)、Exp(经验值)、Level(等级)、Gold(金币)、Success(胜利)、Fail(失败)、Break(逃跑)、Velocity(移动速度)、Power(炸弹威力)、Number(炸弹数量)、Dodge(闪避率)。其中**UserID 为主键**。

(c) 两张表通过 UserID 相互关联。

2.2 注册系统

4. **功能介绍**

(a) 注册系统负责注册功能。首先接收客户端发过来的用户注册信息，并检验是否合法。若合法则存入数据库中，返回成功信息；若不合法，则返回失败信息。

5. **对外通信**

(a) 注册的流程如下:

(b) 客户端向注册服务器发送登录请求，JSON 格式如下:

user，password，name 分别表示用户的账号，密码和昵称。

(c) 服务器会返回登录结果，成功和失败的 JSON 格式分别如下:

i. 成功:

ii. 失败:

6. **内部结构**

(a) **Main 类**为程序入口，建立注册 Server。内部有连接端口信息 port。

(b) **Server 类**是注册服务器，是线程类，用来建立 ServerSocket 等待客户端连接。

i. public void start(): 注册服务器开始运行。

ii. public void run(): 服务器持续监听客户端的连接。

(c) **Connect 类**是线程类，用于与客户端建立连接。

i. public void add(Socket socket): 每来一个客户端就新建一个 Connect 与之连接。

ii. public void start(Socket socket): 开始与客户端通信。

iii. public void stop(): 与客户端断开连接

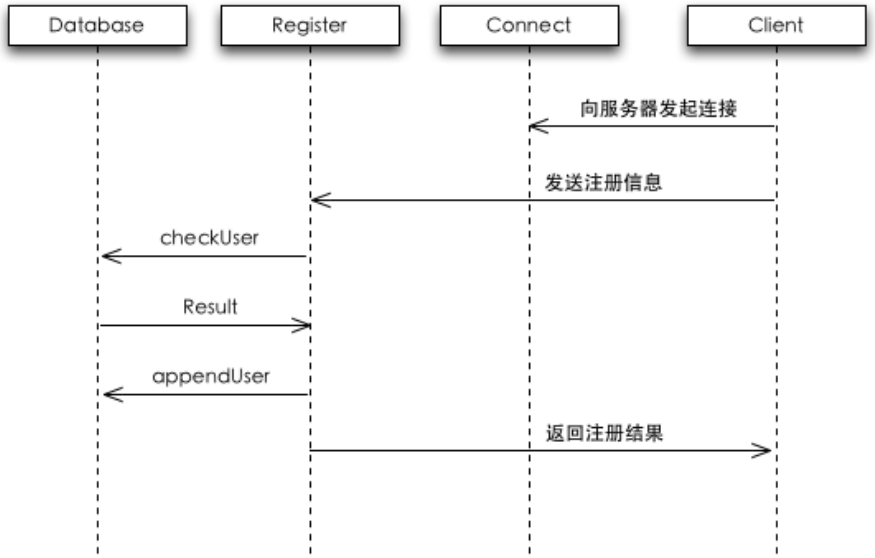

Figure 5: 注册流程

```
{
    "user": "user",
    "password": "password",
    "name": "name"
}
```

Figure 6: 登录 JSON 格式

```
{
    "result": "success"
}
```

Figure 7: 成功 JSON 格式

```
{
    "result": "fail",
    "reason": "reason"
}
```

Figure 8: 失败 JSON 格式

iv. public void run(): 接受客户端的注册请求，处理后返回注册结果。具体格式见“对外通信”。

v. port: 存储端口信息。

vi. timer: 用于记录连接时间。

(d) **Register 类**负责与数据库通信。

i. JSONObject checkUser(JSONObject data): 检查注册信息是否合法(是否符合 JSON 格式，用户名是否已存在，用户名、密码、昵称是否为空)。若合法则转入 appendUser；若不合法，则返回失败信息。

ii. JSONObject appendUser(JSONObject data): 将用户注册信息加入数据库，返回成功信息。

iii. 内部存储数据库信息。

iv. dbName: 数据库地址。

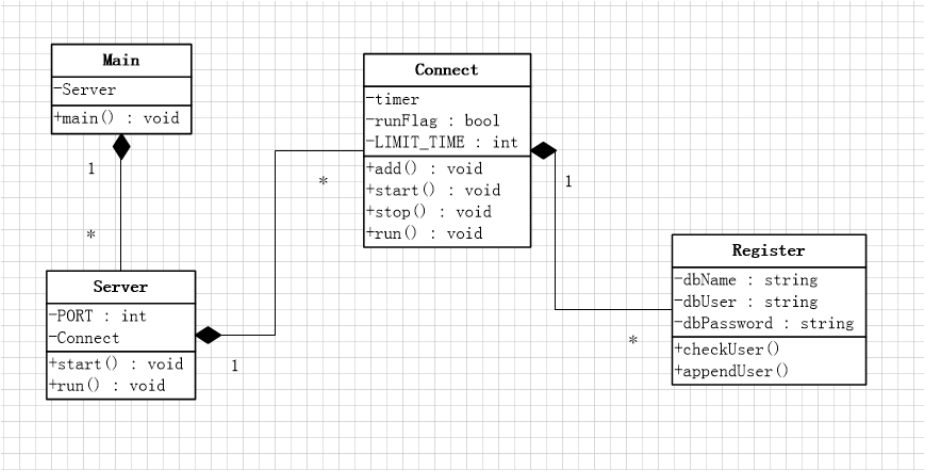

Figure 9: 失败 JSON 格式

v. dbUser: 数据库用户。

vi. dbPassword: 数据库密码。

(e) Server 及 Connect 均为线程类，Server 为主线程，Connect 为副线程。Server 每接收到一个客户端，则会新开一个线程 Connect 用来与客户端通信。客户端发送 JSON 格式的用户注册信息，Connect 线程通过调用 Register 类的 checkUser 来判断用户注册信息是否合法，合法则添将其加至数据库中，并返回成功信息；否则返回失败信息。无论注册是否成功，15s 后一律断开与客户端的连接。

2.3 登录系统

(a) **功能介绍**

i. 登陆系统负责验证用户信息，将通过验证的用户引向大厅服务器。

(b) **对外通信**

i. 大厅服务器会向登陆服务器发起连接，发送数据格式如下:

```
{
    "user": "hall-server1",
    "password": "i am password",
    "ip": "166.111.134.210",
    "port": 10001
}
```

Figure 10: 发送数据格式

ii. user 和 password 是大厅的名称与密码，ip 和 port 是大厅服务器为客户端准备的地址。

iii. 用户登陆，流程如下:

iv. 客户端发给登陆服务器的登陆请求格式如下:

v. user，password 分别表示用户的账号和密码，target 表示用户希望登陆的大厅。

vi. 登陆服务器发给大厅服务器的登陆通知格式如下:

vii. 其中 user 和 key 表示用户的账号和一次性口令，details 为该用户的详细资料。

viii. 登陆服务器发给客户端的数据格式如下:

ix. 其中 key 表示该用户的口令，ip 和 port 表示客户端需要希望连接的大厅服务器的地址。

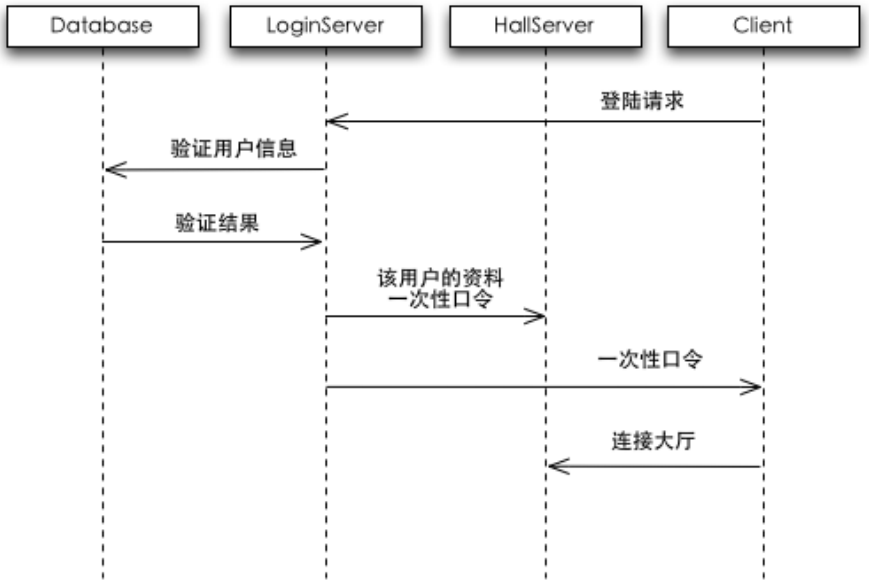

Figure 11: 用户登录流程

```
{
    "user": "user",
    "password": "password",
    "target": "name"
}
```

Figure 12: 登录请求格式

```
{
    "type": "query",
    "user": "test",
    "key": "1234567890",
    "details": {…}
}
```

Figure 13: 登录通知格式

```
{
    "user": "test",
    "key": "1234567890",
    "details": {…},
    "ip": "166.111.134.210",
    "port": 10000
}
```

Figure 14: 数据格式

x. 用户在大厅中会进行游戏，个人资料也会有所改变。大厅服务器会发送如下格式数据来更新用户数据:

xi. limit 表示该大厅还可以容纳多少用户，userData 用来更新用户数据，其中每一个字典表示一个用户，user 表示这个用户的名称，status 可以是”online”或”offline”表示该用户还在线或是已下线，details 是这个用户最新的详细资料。

(c) **内部结构**

i. **Main 类**为程序入口。建立大厅服务器和用户客户端的登录服务器。

```
{
    "type": "hallStatus",
    "limit": 48,
    "userData": [
        {
            "user": "test1",
            "status": "online",
            "details": {…}
        },
        {
            "user": "test2",
            "status": "offline",
            "details": {…}
        }
    ]
}
```


Figure 15: 更新用户数据格式

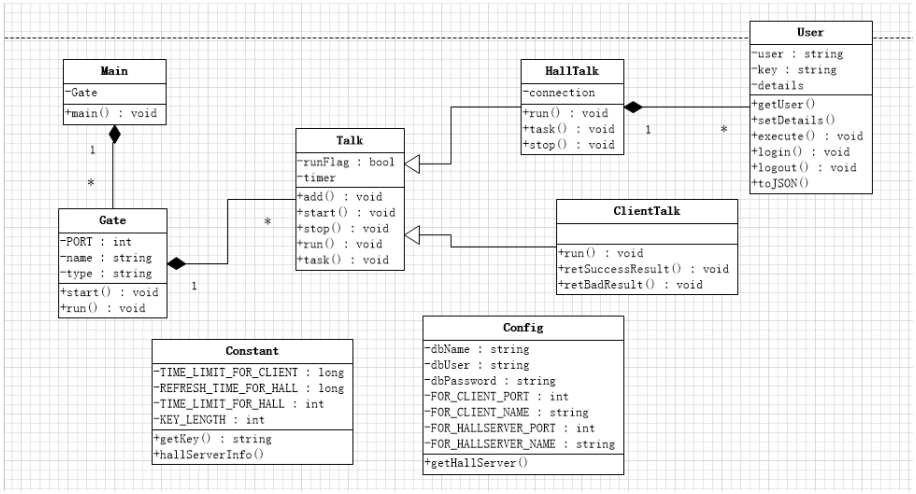

Figure 16: 内部结构

ii. **Gate 类**是线程类，是通信入口。负责监听客户端，并建立新线程 Talk 用于连接。
   A. public void start(String type、String name、int port): 开始运行登录服务器。其中 type 为登录服务器的类型(大厅及用户)、port 是端口信息。
   B. public void run(): 持续监听大厅与客户端的连接请求。
   C. Talk 类是线程类，用于与大厅或用户通信。
   D. public void add(String type、Socket socket): 通过判断 type 的类型分别新建对应的 Talk(HallTalk、ClientTalk)与之通信。
   E. public void start(Socket socket): 开始通信。
   F. public void stop(): 断开连接，结束通信。
   G. runFlag: 表示线程是否运行。
   H. timer: 记录连接时间。

iii. **HallTalk 类**继承 Talk 类，也是线程类，用于与大厅服务器进行通信。
   A. public void run(): 解析大厅服务器发送的消息。若大厅服务器的 database 中用户名 user 已存在、password 一致且原登录状态是“offline”，则大厅登录成功。且若 runFlag 为 true，则持续接受刷新大厅发过来的数据。
   B. void task(): 在与大厅服务器连接断开之前，持续发送用户数据至大厅服务器。
   C. protected void stop(): 与大厅服务器断开。

iv. **ClientTalk 类**继承 Talk 类，也是线程类，用于与用户客户端进行通信。
   A. public void run(): 首先调用 User 类的 getUser 判断用户登录信息是否合法，并判断大厅是否可登录，若都符合，则连入大厅，并传送用户数据至 HallTalk。
   B. void retSuccessResult(JSONObject result, String ip, int port): 返回客户端 ip 及 port 信息。
   C. void retBadResult(String reason): 返回失败信息。
   D. void task(): 结束通信。

v. **User 类**包含用户客户端对数据库信息的操作。
   A. String reason: 用于记录登录出错信息。
   B. User getUser(JSONObject data): 判断用户登录信息是否合法。
   C. JSONObject setDetails(ResultSet input): 记录 input 中的用户游戏信息。
   D. public static void execute(String Hall, JSONObject data): 更新数据库中用户的游戏信息。
   E. public static void logout(String hall): 用户登出。
   F. public static void login(String hall, String User): 用户登录。
   G. Config 类存储所有与环境相关的常量。
   H. Constant 类存储所有与环境无关的常量以及一些常用函数。
   I. void log(): 输出调试信息。
   J. String getKey(): 随机生成 64 位的 key，作为之后用户连接大厅与游戏的密码。
   K. JSONObject hallServerInfo(): 返回大厅服务器信息。

vi. 共有两个主线程，gateHallServer 和 gateClient，分别用于接收大厅服务器和用户客户端的登录，两者相对独立。这两个登录服务器每 receive 一个登录请求，就另开一个新线程 HallTalk 和 ClientTalk 分别处理。ClientTalk 用于与客户端进行通信。它会调用 User 类的 getUser 去检查用户登录信息的合法性，然后判断大厅是否可连接，之后向 HallTalk 发送用户信息。HallTalk 用于与大厅服务器通信，接收用户数据，并调用 User 类更新数据库中的信息，并返回最新的用户信息。

(d) **大厅系统**

i. **功能介绍:** 大厅系统是一个枢纽，它的主要功能有:
   A. 处理登陆服务器发来的用户登录通知
   B. 接受客户端的连接，若该用户通过了登陆服务器的验证则将其加入大厅
   C. 处理客户端的事件，如进入房间、准备等
   D. 检测游戏是否可以开始，将开始游戏的信息发送给游戏服务器以及对应的客户端
   E. 处理游戏服务器发来的游戏结果

ii. **对外通信:**
   A. 1) 最初，游戏服务器会向大厅服务器发起连接，发送数据格式如下: key 是用来证

```
{
    "key": "I am password"
}
```

Figure 17: 数据格式

明游戏服务器身份的口令
   B. 2) 客户端向大厅服务器发起连接，发送的数据格式如下: 其中，user 表示该用户的名称，key 为从登陆服务器获取的一次性密码。
   C. 大厅服务器返回给客户端的数据格式如下:
      - users 为一个字典，以表示大厅内所有用户的信息，其索引是用户名，对应的值是这个用户的信息；rooms 为一个数组，表示大厅内所有房间的状态。
      - room 和 pos 表示该用户当前所在房间和座位(不在房间时二者均为-1)

```
{
    "user": "test",
    "key": "1234567890"
}
```


Figure 18: 数据格式


```
{
    "type": "success",
    "users": {
        "test": {
            "room": 0,
            "pos": 0,
            "ready": "ready",
            "details": {…}
        }
    },
    "rooms": [
        0,
        0,
        0,
        0,
        0
    ]
}
```


Figure 19: 数据格式

- ready 表示该用户当前是否准备(”ready”或”unready”)
- details 为该用户的详细信息，格式详见最后补充
- rooms 为一个 int 的数组，的长度为大厅内房间数量，其中 rooms[i]表示 i 号房间是否开始游戏(0 表示未开始，1 表示正在游戏中)

D. 3) 客户端的每次动作都需要向大厅服务器发送请求，格式如下:


```
{
    "type": "enter",
    "room": 1,
    "pos": 2
}
```


Figure 20: 发送请求格式

- room 和 pos 表示申请进入的房间号和座位号，特别地，999 表示任意。
- type 还可以是 leave 表示申请离开房间，ready 表示申请准备，unready 表示取消准备。

E. 服务器每隔 T 毫秒向客户端发送一次数据以更新大厅状态，格式如下:

F. events 表示近 T 秒发生过的所有事件，事件分为以下几种:

- 某个用户登入
- 某个用户进入某个房间，并在某个座位坐下
- 某个用户准备
- 某个用户取消准备
- 某个用户离开房间
- 某个用户的详细信息被更新
- 某个用户登出

G. rooms 仍然表示所有房间的状态。

H. 补充:客户端可通过上述数据分析自己的请求是否成功。

Figure 21: 更新状态数据

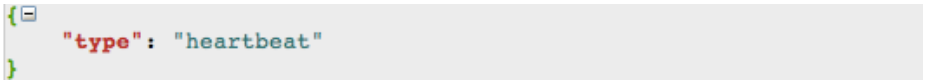

Figure 22: 心跳包

I. 客户端需要每隔 10 秒向登录服务器发送一次心跳包以证明自己在线:

J. 每当大厅服务器捕捉到一局游戏开始，会按如下流程进行处理:

K. 大厅服务器向游戏服务器发送请求的格式如下:

L. id 表示游戏编号，users 用来表示该局游戏的玩家信息，其中 user 表示玩家账号，key 表示玩家连向游戏服务器的一次性口令，details 为该玩家的详细资料。

M. 游戏服务器返回给大厅服务器的数据格式如下:

N. id 表示游戏编号，users 用来表示该局游戏的玩家信息，其中 user 表示玩家账号，ip 和 port 表示该玩家需要连向的游戏服务器的地址。

O. 大厅服务器发给客户端的数据格式如下:

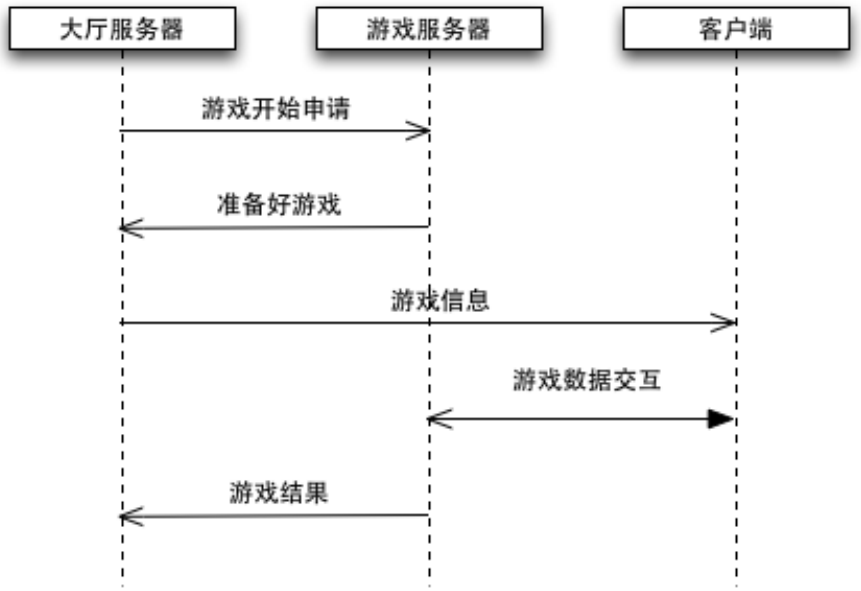

Figure 23: 开始流程


```
{
    "type": "ready",
    "id": 1,
    "users": [
        {
            "user": "test1",
            "key": "1234567890",
            "details": {…}
        },
        {
            "user": "test2",
            "key": "1234567812",
            "details": {…}
        }
    ]
}
```


Figure 24: 请求格式

P. ip 和 port 表示该客户端需要连接的游戏服务器的地址，key 为连接时将用到的一次性密码。

Q. 游戏结束后，游戏服务器发给大厅服务器的游戏结果如下:

R. 其中 id 表示游戏编号，result 用来表示游戏结果，分别有”win”表示胜利，”lose”表示失败，”break”表示断线。

iii. **内部结构**

A. 类结构

- **Main 类:** 程序的入口，负责初始化各类
- **Gate 类:** 通讯入口，负责监听端口
- **Talk 类:** 通信类，负责维护一个通信。它是一个纯虚类。
- **LoginTalk 类:** 负责处理与登陆服务器的通讯
- **GameTalk 类:** 负责处理与大厅服务器的通讯
- **ClientTalk 类:** 负责处理与一个客户端的通讯

```
{
    "type": "start",
    "id": 1,
    "users": [
        {
            "user": "test1",
            "ip": "166.111.134.210",
            "port": 8001
        },
        {
            "user": "test2",
            "ip": "166.111.134.210",
            "port": 8002
        }
    ]
}
```


Figure 25: 返回数据格式


```
{
    "type": "game",
    "ip": "166.111.134.70",
    "port": 8001,
    "key": "1234567890"
}
```


Figure 26: 数据格式


```
{
    "type": "finished",
    "id": 1,
    "result": [
        {
            "user": "test1",
            "result": "win"
        },
        {
            "user": "test2",
            "result": "lose"
        }
    ]
}
```


Figure 27: 游戏结果数据

- **HallInfo 类:** 负责维护大厅内的状态
- **Room 类:** 负责维护房间内的状态
- **Constant 类:** 负责记录与运行环境无关的常量，以及一些常用函数
- **Config 类:**负责记录与环境相关的常量

B. 主要接口

- Main:
  - public void main(String args) 程序入口
  - void manual() 通过标准输入控制程序行为

- Gate:
  - public void start(String type, String name, int port) 新启一个线程监听 port 端口，其中 type 表示连向这个端口的是哪一类群体(游戏服务器、客户端)
- Talk:

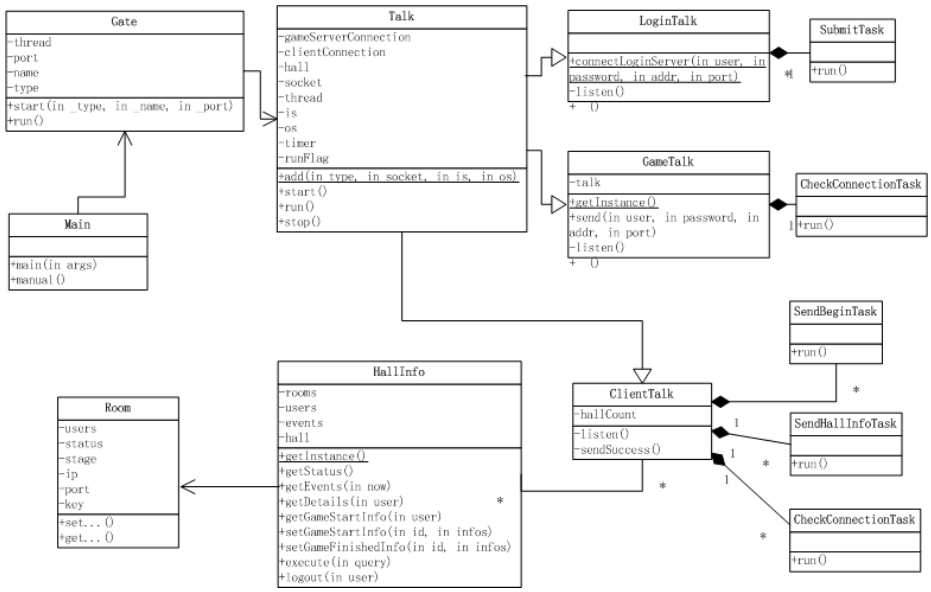

Figure 28: 类结构

  - public static void add(String type, Socket socket, ReadBuffered is, Printwriter os) 新建一个 Talk 类用来维护一个通信
  - public void start() 为这个通信新建一个线程
  - void run()开始运行这个线程，即开始监听对方发来的消息
  - stop() 杀掉这个线程，即停止这个通信
- LoginTalk:
  - public static void connectLoginServer(String user, String password, String addr, int port) 连接登陆服务器并创建一个 LoginTalk 的实例以维护二者的通讯
  - void listen() 监听登陆服务器发来的消息并处理
  - SubmitTask :: void run() 该函数每隔 10 秒被调用一次，将事件队列里的信息发给登陆服务器(这里的事件队列用来存储别的线程需要发给登陆服务器的消息，这样设计是为了控制发送频率)
- GameTalk:
  - GameTalk getInstance() 获取 GameTalk 的实例，若当前没有游戏服务器连接则返回空
  - void listen() 监听游戏服务器发来的消息并处理
  - void send(String msg) 给游戏服务器发送消息
  - CheckConnectionTask :: run() 每 20 秒运行一次，用来检测游戏服务器是否断开连接
- ClientTalk:
  - void listen() 监听客户端发来的消息并处理
  - void sendSueecss() 向客户端发送连接成功的消息
  - SendHallInfoTask :: run() 每 1 秒运行一次，向客户端发送大厅内最新的变化
  - CheckConnectionTask :: run() 每 20 秒运行一次，检查客户端是否断开连接
  - SendBeginTask :: run() 每 200 毫秒运行一次，检查该客户端是否开始游戏，若开始则发送开始游戏的信息
- HallInfo:
  - HallInfo getInstance() 大厅为单件模式，该函数可获取大厅的实例

- JSONObject getStatus() 获取大厅当前状态
- JSONObject getEvents(int now) 获取大厅最新变化
- JSONObject getDetails(String user) 获取一个玩家信息
- JSONObject getGameStartInfo(String user) 获取一个玩家是否开始游戏的信息
- JSONObject setGameStartInfo(int id, JSONObject infos) 设置一局游戏开始的信息
- JSONObject setGameFinishedInfo(int id, JSONObject infos) 填写一局游戏的结果
- void execute(JSONObject query) 执行一个用户的一个操作(进入房间，准备等)
- void logout(String user) 执行一个用户的登出操作

• Room: 该类的对外接口全部是成员变量的 set 和 get 函数。这里不再赘述。

C. **游戏系统**

• **功能介绍** 这一系统负责处理客户端之间的游戏逻辑，同时将游戏结果传回大厅服务器。

• **对外通信**

– 对外通信的流程如下: 与大厅服务器的通信格式前面已经描述过，这里不再

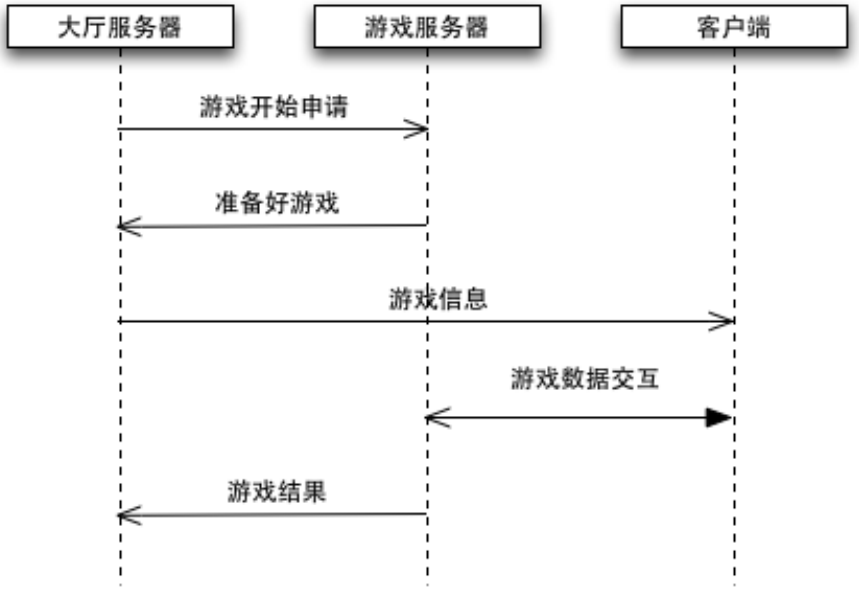

Figure 29: 用户登录流程

赘述。游戏过程中与客户端的通信格式将在后面客户端的部分详细介绍。

• **系统结构**

– **类结构**

Figure 30: 用户登录流程

* Main: 负责新开一个或者多个游戏处理器（GameServer），每个 GameServer 占用一个线程。

* GameServer:每个此类的实例是一个游戏处理器，每个游戏处理器会连接到某个大厅服务器（HallServer）的某个端口，然后和此 HallServer 通讯，负责接收游戏的开始。
* Game:每次 HallServer 向 GameServer 发送一局新的游戏开始的消息之后都会生成一个新的 Game 的实例，用来完全负责一局游戏的内部逻辑，游戏结束后会通知 GameServer。
* GameController:这是一个抽象类，不同的子类根据不同的算法更新游戏的状态。每个 Game 类的实例都有一个此类的某个子类的实例作为成员变量。
* SimpleGameController:GameController 的一个子类，按照某种方式来更新游戏数据。
* User:用来纪录一个用户的全部信息，包括当前的状态，IP 等等。
* Map:用来纪录一局游戏的地图信息。
* Config: 纪录一些关于游戏设定的常量，例如游戏地图大小等等。

– **各个类主要的对外接口**
* GameServer: 提供向 HallServer 发送信息的函数 send()
* 提供一个可以返回一个 Socket 的函数 getAvailableSocket()
* Game:作为一局游戏的控制中枢，保存着关于游戏的所有信息，其他类可以通过此类获得关于游戏的信息，例如通过 getMap()，可以获得游戏地图的引用。
* GameController(以及其子类 SimpleGameController）:
* 提供处理数据接收到的数据的接口 dealInfo(ReceivedInfo info)
* 提供检查当前游戏是否结束的接口 checkGameOver()
* 提供产生游戏结果的方法 generateResult（）和 generateResultForHall（）
* 提供游戏结束后处理游戏遗留信息的接口 takeCareOfGame（）
* JSONObjectMaker:
* 提供制作游戏结果 JSONObject 的接口 makeResultForClient（）和 makeResult（）
* 提供制作游戏玩家信息 JSONObject 的接口 makePlayerInfo（）
* 提供制作当前炸弹信息 JSONObject 的接口 makeBombInfo（）
* 提供制作给玩家发送整合的同步信息 JSONObject 的接口 makeConcurrentInfoForClient（）
* Map:
* 提供打印当前地图信息的接口 print（）
* 提供询问位置是否是箱子的接口 isBox（int i，int j）其中 i，j 是要判断的坐标。
* 提供更新炸弹的接口 updateBubbles（ArrayList<Bubble> bubbles, long time）其中 bubbles 代表要更新的 bubble，而 time 是要更新到的最新时间。
* 提供更新游戏人物信息的接口 updateUsers（Game game, User user, long time）其中 game 是当前的游戏，user 是要更新的人物，而 time 是要更新到的最新的时间。
* 提供更新游戏箱子状态的接口 updateBox（int i, int j, long deltaTime）其中 i，j 是箱子的坐标，而 deltaTime 是要更新的时间长度。

3 客户端

3.1 通信与逻辑

A. **功能介绍**
客户端逻辑部分负责和服务器通信，交互及处理数据，响应 Viewer 请求数据以及相关操作的请求，处理游戏进行中的几乎所有逻辑。

B. **接口及内部实现**

C. 客户端逻辑部分主要由 Controller 类以及 SigninModel，SignupModel，HallModel，GameModel 四个类，以及 Room，User，Flag 三个类，Config 类组成。

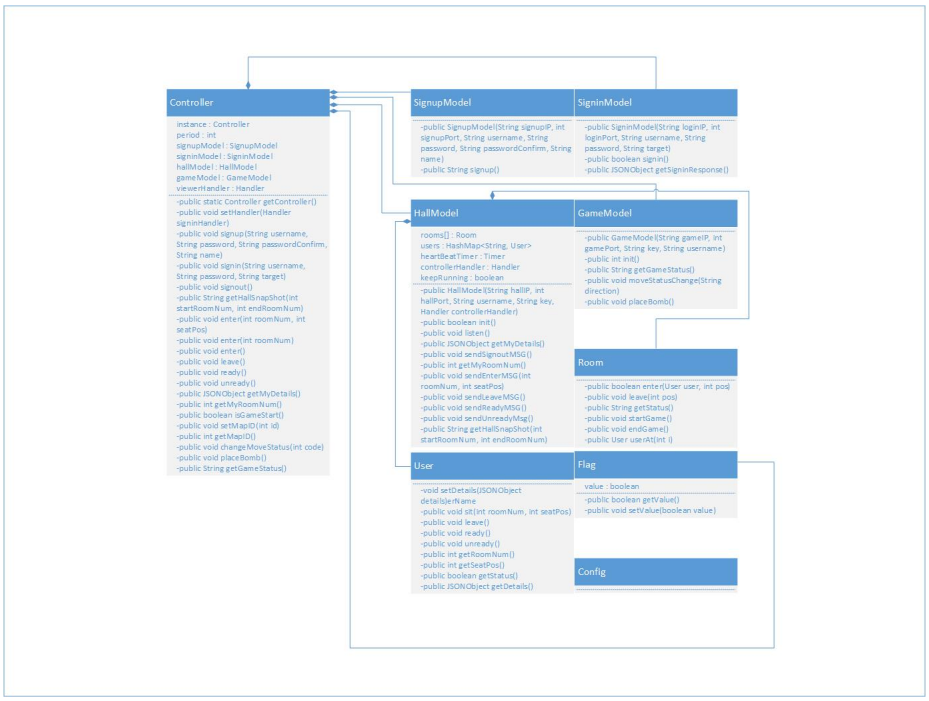

Figure 31: 组成类

D. **Controller 类:**主要负责逻辑控制，起到核心控制的功能，拥有自己的线程管理，自己的工作基本都在自己独立的线程内完成，采取分时间片的方式进行，其中通过一个状态标志变量记录当前所处状态（登录、大厅、游戏）。

E. **接口描述:**

F. public static Controller getController() :获取 Controller 实例

G. public void setHandler(Handler signinHandler) :设置界面的 Handler，用于异步调用 Controller 接口时结果信息的返回。

H. public void signup(String username, String password, String passwordConfirm, String name):注册接口，参数依次为用户名、密码、密码确认、昵称，将通过 SignupModel 与 Server 进行通信，细节将在 SignupModel 类中描述，结果信息通过 Handler 异步返回，返回格式如下: 其中“success”表示登录成功，但是也可

```
{
    "type": "signup",
    "status": "success"
}
```

Figure 32: 返回格式

能登录不成功，则返回的将是”EmptyUsernameError”，”IllegalPasswordError”，”InternetError”，或者其他失败原因。

I. public void signin(String username, String password, String target):登录接口，参数依次为用户名、密码、登录的目的大厅代号，将通过 SigninpModel 与 Server 进行通信，细节将在 SigninpModel 类中描述，结果信息通过 Handler 异步返回，返回格式如下: 其中“Success”表示登录成功，但是也可能登录不成功，则返回的将是”InternetError”或者具体的原因。

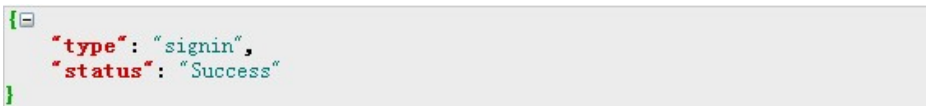

Figure 33: 返回格式

J. public void signout():登出接口，在这里 Controller 将把逻辑切换到登录之前的状态。

K. public String getHallSnapShot(int startRoomNum, int endRoomNum):获取大厅快照接口，参数为起始房间号、结束房间号，将通过 HallModel 与 Server 进行通信，细节将在 HallModel 类中描述，结果通过 Handler 异步返回，返回数据格式如下: 其中”start”和”end”依次是传入的两个参数，rooms 键对应的是

Figure 34: 返回格式

一个 JSONArray，表示每个房间的信息，”Playing”表示该房间正在进行游戏，其他可能取指为:”Free”和”Full”，users 键对应的是一个是一个 JSONArray，”UserName”为该用户对应的用户名，”ready”表示该用户已经准备，也可能为”unready”，表示没有准备，details 为用户的详细信息，其格式见 wiki 页面 User Details。pos 为该玩家在房间中的位置。

L. public void enter(int roomNum, int seatPos):进入房间接口，参数:房间号，座位号，进入指定房间指定座位。

M. public void enter(int roomNum):进入房间接口，参数:房间号，进入指定房间任意座位。

N. public void enter():进入房间接口，参数:房间号，进入任意房间任意座位。

O. 将通过 HallModel 与 Server 进行通信，细节将在 HallModel 类中描述，结果通过 Handler 异步返回，返回数据格式如下: 其中“Success”表示进入成功，但是也

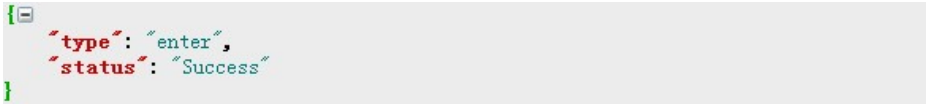

Figure 35: 返回格式

可能不成功，则返回的将是具体的原因。

P. public void leave():离开房间，将通过 HallModel 与 Server 进行通信，细节将在 HallModel 类中描述，结果通过 Handler 异步返回，返回数据格式如下: 其中“Success”表示进入成功，但是也可能不成功，则返回的将是具体的原因。

Figure 36: 用户登录流程

Q. public void ready():准备，将通过 HallModel 与 Server 进行通信，细节将在 HallModel 类中描述，结果通过 Handler 异步返回，返回数据格式如下: 其中

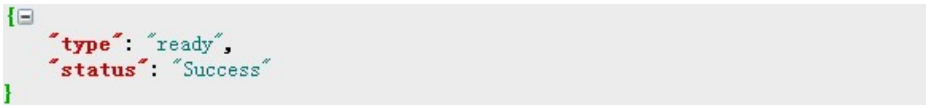

Figure 37: 返回格式

“Success”表示进入成功，但是也可能不成功，则返回的将是具体的原因。

R. public void unready():取消准备，将通过 HallModel 与 Server 进行通信，细节将在 HallModel 类中描述，结果通过 Handler 异步返回，返回数据格式如下: 其中

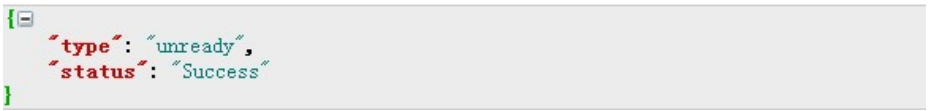

Figure 38: 返回格式

“Success”表示进入成功，但是也可能不成功，则返回的将是具体的原因。

S. public JSONObject getMyDetails():获取玩家自己的详细信息，返回格式见 wiki 页面 User Details。

T. public int getMyRoomNum():获取玩家自己当前的房间号，返回结果为玩家当前所处的房间号，如果没在任何房间，则返回值为-1。

U. public boolean isGameStart():查询游戏是否开始。void setMapID(int id):设置游戏地图编号，参数:地图编号。

V. public int getMapID():查询游戏地图编号。

W. public void changeMoveStatus(int code):游戏过程中，改变玩家的运动状态。参数:运动状态代码，0 为向上运动，1 为向下运动，2 为向左运动，3 为向右运动，0 为停止，将通过 GameallModel 与 Server 进行通信，细节将在 GameallModel 类中描述。

X. public void placeBomb():游戏过程中，释放炸弹，将通过 GameallModel 与 Server 进行通信，细节将在 GameallModel 类中描述。

Y. public String getGameStatus():获取当前游戏的界面详细信息，以供 Viewer 绘制游戏界面，通过 GameModel 实现，细节将在 GameallModel 类中描述。

Z. **SignupModel 类:**负责和 Server 进行通信，执行注册的逻辑。**接口描述:**

. public SignupModel(String signupIP, int signupPort, String username, String password, String passwordConfirm, String name):构造函数，设置注册服务器 IP，端口号，用户注册信息。

. public String signup():注册接口，和服务器通信。返回值为注册结果信息，可能取值为:”InternetError”，”success”或者具体失败信息。

SigninModel 类:负责和 Server 进行通信，执行登录逻辑。**接口描述:**

. public SigninModel(String loginIP, int loginPort, String username, String password, String target):构造函数，设置登录服务器 IP，端口号，用户登录信息。

. public boolean signin():登录接口，和服务器通信。返回值通信结果，成功返回 true，否则返回 false。

. public JSONObject getSigninResponse():获取具体返回信息，其中包含下一步 Controller 控制 Hall 模块逻辑所需的信息。

. **HallModel 类:**负责和 Server 进行通信，持续监听 Server 发送的信息，并且定时向 Server 发送心跳包，同时维护当前大厅内的信息。

. **接口描述:**

. public HallModel(String hallIP, int hallPort, String username, String key, Handler controllerHandler):构造函数，设置大厅服务器 IP，端口号，通信时的身份凭证信息，Controller 的 Handler。

. public boolean init():初始化 socket 等变量，与 HallServer 通信，获取大厅初始信息。

. public void listen():监听 HallServer，处理其发送的消息，维护大厅信息。

. public JSONObject getMyDetails():获取玩家自己的详细信息。

. public int getMyRoomNum():获取玩家当前所处的房间号，如果玩家没有在任何房间，则返回-1。

. public void sendSignoutMSG():登出接口，停止对 HallServer 的监听，以及心跳包的发送。

. public void sendEnterMSG(int roomNum, int seatPos):向 HallServer 发送玩家进入房间请求的信息，HallServer 返回的结果将通过 Handler 异步返回给 Controller。

. public void sendLeaveMSG():向 HallServer 发送玩家离开房间请求的信息，HallServer 返回的结果将通过 Handler 异步返回给 Controller。

. public void sendReadyMSG():向 HallServer 发送玩家准备游戏请求的信息，HallServer 返回的结果将通过 Handler 异步返回给 Controller。

. public void sendUnreadyMsg():向 HallServer 发送玩家取消准备游戏请求的信息，HallServer 返回的结果将通过 Handler 异步返回给 Controller。

. public String getHallSnapShot(int startRoomNum, int endRoomNum):参数:起始房间号和结束房间号，返回当前大厅快照，以便 Controller 提供给 Viewer 绘制大厅界面。

. **GameModel 类:**负责游戏时和 GameServer 通信，数据处理，为 Controller 提供需要的游戏信息。

. **接口描述:**

. public GameModel(String gameIP, int gamePort, String key, String username):构造函数，设置游戏服务器 IP，端口号，通信时的身份凭证信息。

. public int init():初始化 socket 等变量，与 GameServer 通信，完成握手，初始化游戏地图信息。

. 首先向 GameServer 发送如下格式信息: 其中 UserName 为用户账号，key 为构造

Figure 39: 发送格式

函数中设置的身份认证 key。GameServer 收到客户端发送的信息之后，会返回如下格式数据:

. 当所有客户端均已完成上述和 GameServer 的通信之后，GameServer 会向所有客户端发送如下数据: 收到 GameServer 的 start 消息之后，握手完成。

. public String getGameStatus():获取当前的游戏状态信息，以供 Viewer 绘制游戏界面。返回值为如下格式的数据:

Figure 40: 发送格式

Figure 41: 发送格式

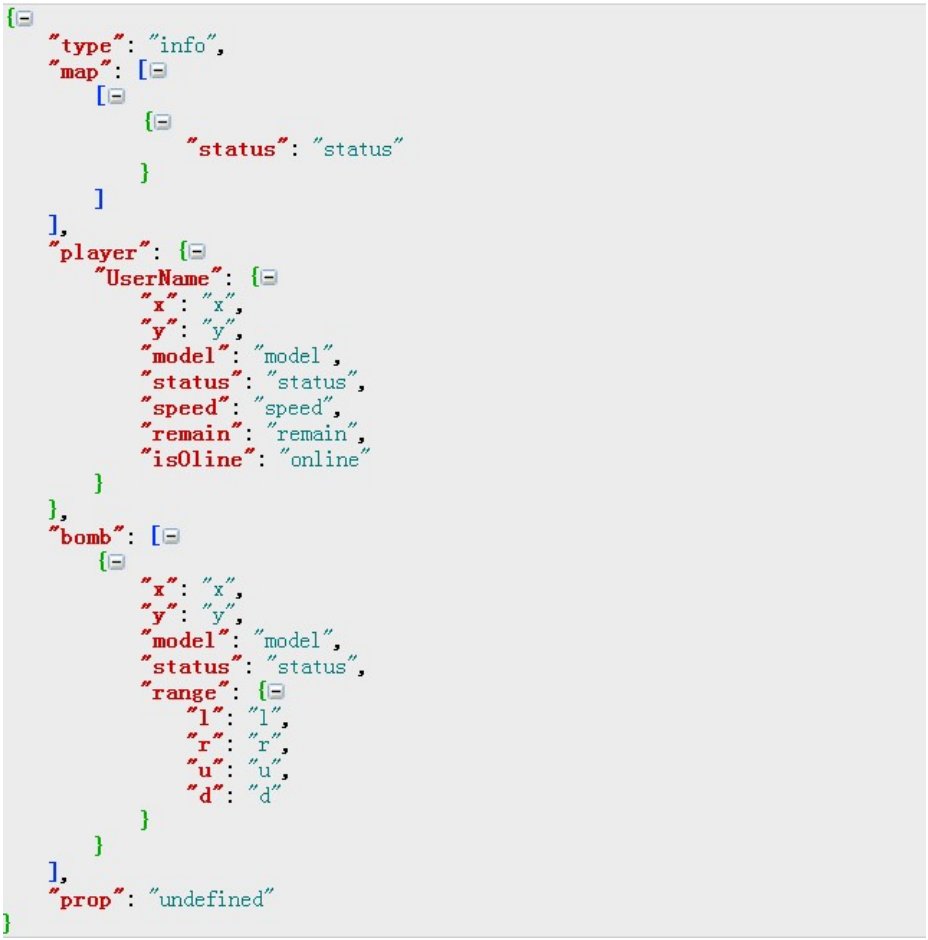

Figure 42: 发送格式

. 各个参数的含义详见 wiki 页面 Game Server And Users 和 Client Controller And ViewerInterface In Game。特别地，因为 GameServer 发送信息的速度无法达到每秒 30 帧，因此 GameServer 发送的速度是慢于界面刷新的速度，这些空白的帧需要 GameModel 根据前一帧中玩家的位置、速度和方向信息计算得出。

. public void moveStatusChange(String direction):向 GameServer 发送玩家运动状态改变的信息，参数为表示方向的字符串。发送的数据格式如下: 各个参数的含义详见 wiki 页面 Game Server And Users。

. public void placeBomb():向 GameServer 发送用户放置炸弹的信息。格式如下: 各个参数的含义详见 wiki 页面 Game Server And Users。

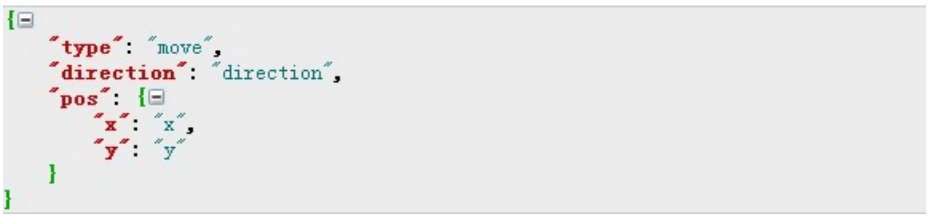

Figure 43: 发送格式

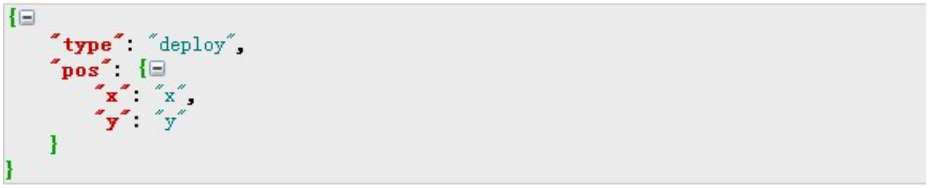

Figure 44: 发送格式

. 特别地，用户是否吃到道具，不需要 Controller 调用任何接口，也不需要 Viewer 进行任何判断，全部由 GameModel 进行计算，如果计算得出本玩家吃到了道具，则向 GameServer 发送如下格式信息: 各个参数的含义详见 wiki 页面 Game

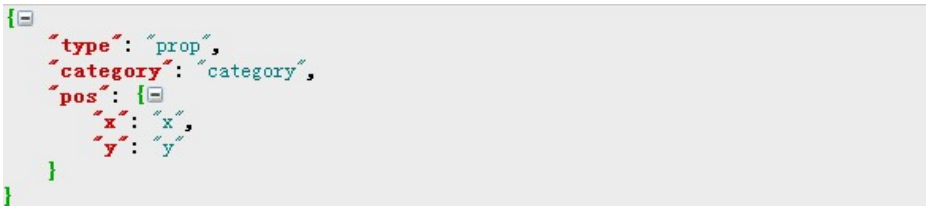

Figure 45: 发送格式

Server And Users。

. GameServer 会每隔一段时间向客户端发送游戏状态信息，格式如下: 各个参数的含义详见 wiki 页面 Game Server And Users。
. 游戏结束后，GameServer 会向客户端发送如下格式信息:
. 各个参数的含义详见 wiki 页面 Game Server And Users。
. **Room 类;**封装了房间的信息及操作。 **接口描述:**
. public boolean enter(User user, int pos):进入房间，参数为玩家、座位号，返回为进入成功与否。public void leave(int pos):离开房间，参数为座位号。public String getStatus():获取当前房间状态，返回值可能为”Playing”，”Free”和”Full”。public void startGame():设置房间为游戏中状态。public void endGame():设置房间退出游戏中状态。public User userAt(int i):获取座位号为 i 的玩家，如果该位置没有玩家，将返回 null。
. **User 类:**封装了玩家的信息及操作。**接口描述:**
. public void setDetails(JSONObject details):设置玩家的 details 信息。
. public void sit(int roomNum, int seatPos):设置玩家进入某个房间的某个位置。
. public void leave():设置玩家离开房间。
. public void ready():设置玩家为准备游戏状态。
. public void unready():设置玩家为未准备游戏状态。
. public int getRoomNum():获取玩家当前所在房间号，如果玩家没有在房间中，则返回-1。

```
{
    "type": "info",
    "map": [
        [
            {
                "status": "status"
            }
        ]
    ],
    "users": {
        "UserName": {
            "x": "x",
            "y": "y",
            "model": "model",
            "status": "status",
            "speed": "speed",
            "remain": "remain",
            "isOline": "online"
        }
    },
    "bubbles": [
        {
            "x": "x",
            "y": "y",
            "model": "model",
            "status": "status",
            "range": {
                "l": "l",
                "r": "r",
                "u": "u",
                "d": "d"
            }
        }
    ],
    "prop": "undefined"
}
```


Figure 46: 发送格式


```
{
    "type": "result",
    "UserName1": "result1",
    "UserName2": "result2",
    "UserName3": "result3",
    "UserName4": "result4"
}
```


Figure 47: 发送格式

. public int getSeatPos():获取玩家当前所在座位号，如果玩家没有在房间中，则返回-1。

. public boolean getStatus():获取玩家当前状态，可能为”ready”，“unready”。

. public JSONObject getDetails():获取玩家当前详细信息。

. **Flag 类:**封装了对 boolean 基本类型的操作，以作为互斥锁的对象。

. **Config 类:**包含一些常数。

3.2 显示

. **模块介绍**

界面显示(View)部分负责接收 Controller 发来的数据，并将其友好的显示出来。同时 View 还负责捕捉用户的操作事件，将其发送给 Controller。界面显示部分主

要界面包括:登陆界面，注册界面，游戏大厅界面，游戏房间界面，游戏界面。另外还包括好友信息界面等有待扩展的子界面。

. **框架**

. 游戏界面部分主要采用的设计框架为，由继承自 Activity 的类来控制界面的跳转，控制，而用继承于 SurfaceView 的类来控制显示，捕捉用户操作。每个 Activity 子类可以通过切换 SurfaceView 子类在几个子界面间切换，同时 Activity 子类也可以控制进入其他 Activity 子类。

. 游戏中界面切换方式有两种:

. activity to activity: 通过 intent 进行跳转，从一个显示模块转入另一个显示模块，也是界面控制权的切换，切换速度较慢

. activity 切换 surfaceView: 通过 setContentView(View view)方法切换界面，适合在几个子界面间快捷切换采用这种模式有利于界面部分工作的分工，不同人可以负责不同的 activity 控制几个 surfaceView 子界面来实现一个功能模块，通过 activity 切换进入其他模块。不同 activity 之间没有依赖性。

. 游戏中的 Activity 子类包括 MainActivit，HallActivity，RoomActivity，GameAciv-ity:1、MainActivity 类主要负责用户登陆，注册等操作。

. HallActivity 主要负责大厅信息显示，同时提供访问好友界面，访问商店等其他服务功能。

. RoomActivity 主要负责房间信息的显示，主要包括在房间内的玩家信息，地图信息以及聊天信息显示。

. GameActivity 主要负责游戏过程，游戏结果的显示

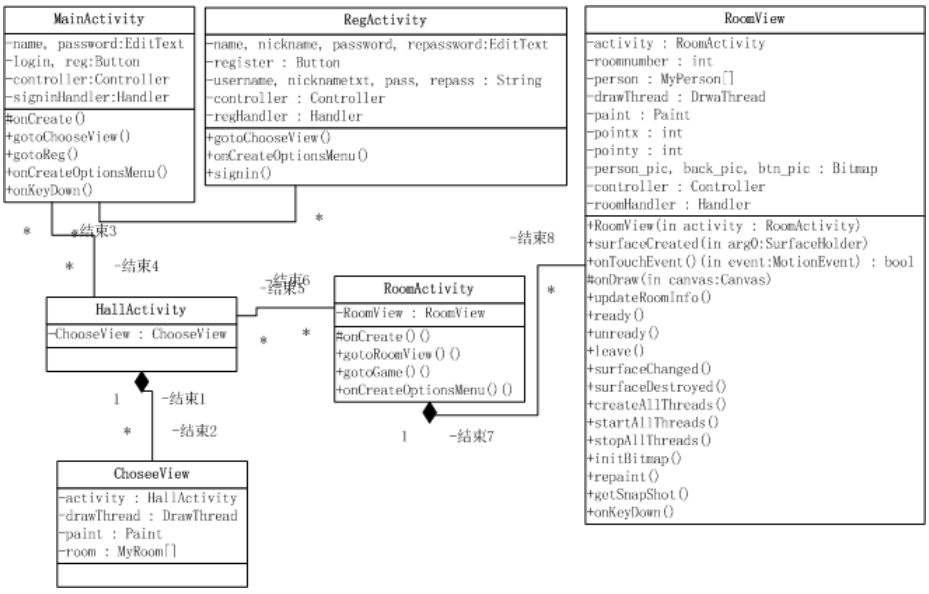

Figure 48: 框架

. **登陆注册界面**

. 类:MainActivity

. 用户操作接口:

. 登陆界面:提供用户名，密码输入框和登陆按钮。同时提供注册按钮跳。

. 注册界面:提供用户名，密码，确认密码输入框和确定注册按钮和返回按钮。

. 界面跳转接口:

. public void gotoHall():登陆成功后跳转至大厅 HallActivity

. publc void gotoSginup():点击注册按钮跳转至注册界面

. public void gotoLogin():注册成功后跳回登陆界面
. 调用 Controller 的接口:
. 登陆:public void signin(String username, String password, String target)
. 注册:public void signup(String username, String password, String passwordConfirm)

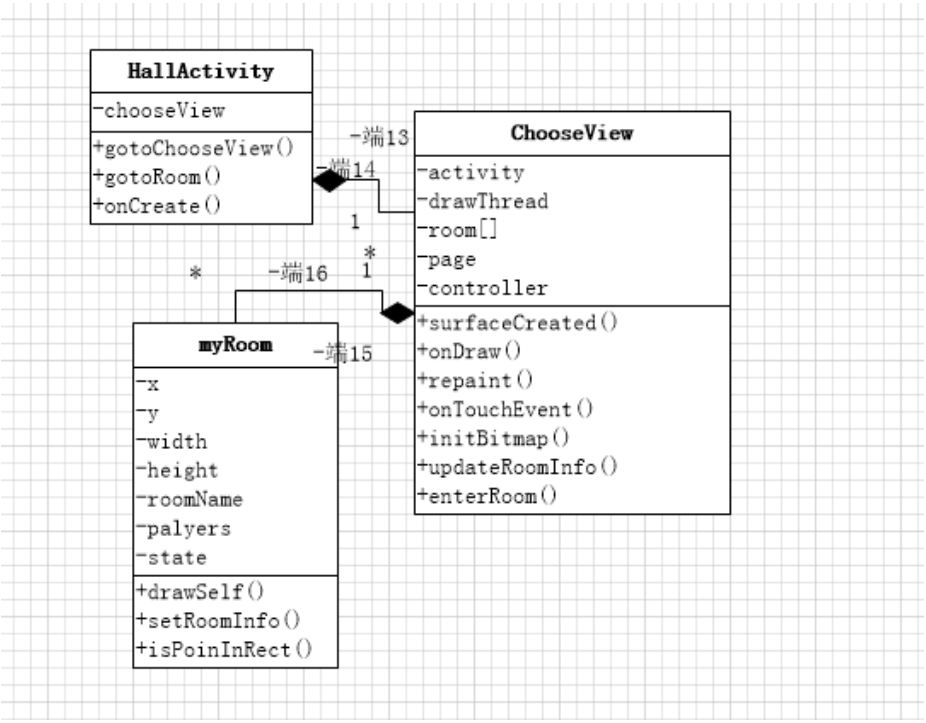

Figure 49: 大厅界面

大厅界面

. 类: HallActivity，ChooseView, FriendView, myRoom:
. HallActivity: 控制大厅界面跳转
. ChooseView: 定时通过 Controller 获取大厅快照，显示各个房间信息显示，用户点击某个房间后告知 Controller, 进入成功则传递消息给 HallActivity 跳转至 RoomActivity。同时提供按钮进入好友列表。提供按钮登出放回登陆注册界面。
. FriendView: 通过 Controller 获取好友列表，显示好友列表信息。提供返回按钮回到跳回 ChooseView。
. myRoom: 负责存储显示每个房间信息部件
. 用户操作接口:
. 大厅选择界面:每个房间被点击表示进入该房间，提供查看好友列表按钮，登出按钮。
. 好友列表界面:点击每个好友可以查看该好友详细信息。提供返回按钮。
. 界面跳转接口: beginenumerate
. Public void gotoChooseView():进入大厅选择界面
. Public void gotoFriendView(): 进入好友类表界面
. Publc void gotoRoom(): 进入房间
. 调用 Controller 的接口:
. Public String getHallSnapShot(int startNum, int endNum):获取大厅快照，取得从 startNum – endNum 的房间信息。

. Public signout() : 登出操作

. **房间界面:**

. 类:　RoomActivity，RoomView, myPerson:

. RoomActivity: 控制房间界面跳转

. RoomView: 定时通过 Controller 的 GetSnapShot 函数获取房间内快照，显示各个座位信息显示，用户点击某个房间后告知 Controller, 进入成功则传递消息给 RoomActivity 跳转至 RoomActivity。同时提供按钮进入好友列表。提供按钮登出放回登陆注册界面。

. myPerson: 负责存储显示每个人的存在信息

. 用户操作接口:

. 房间界面:每个人物被点击表示该座位是否已经坐人，点击该座位，显示座位上该玩家的个人信息；设置准备（取消准备），以及登出按钮。

. 界面跳转接口:

. Public void gotoChooseView():进入大厅

. Publc void gotoGame(): 进入游戏

. 调用 Controller 的接口:

. Public String getSnapShot(int roomNumber, int roomNumber)获取房间快照

. Public ready() 在房间里进行准备

. Public unready() 在房间里取消准备

. Public leave() 退出某一房间

. 利用 Handler 获得 controller 发来的准备，取消准备，退出房间的确认信息。如果与 controller 确认成功（收到”Success”），则进行相应的切换操作，否则提示客户端存在的网络问题。

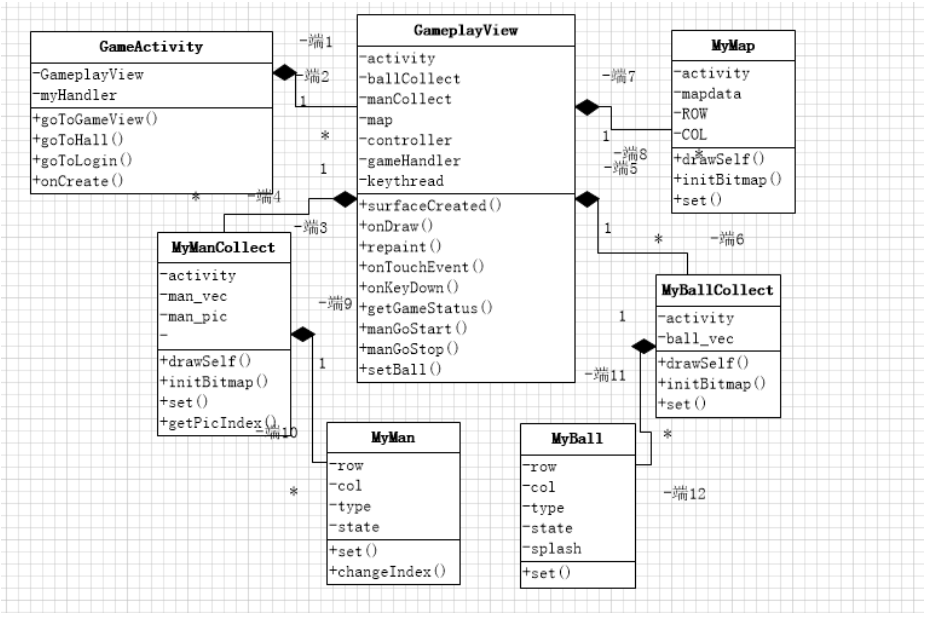

Figure 50: 游戏界面

. **游戏界面**

. 类:GameActivity, GameView, resultView, myMap, myBall, myBallCollect, myMan, myManCollect

. GameActivity:控制游戏部分界面跳转