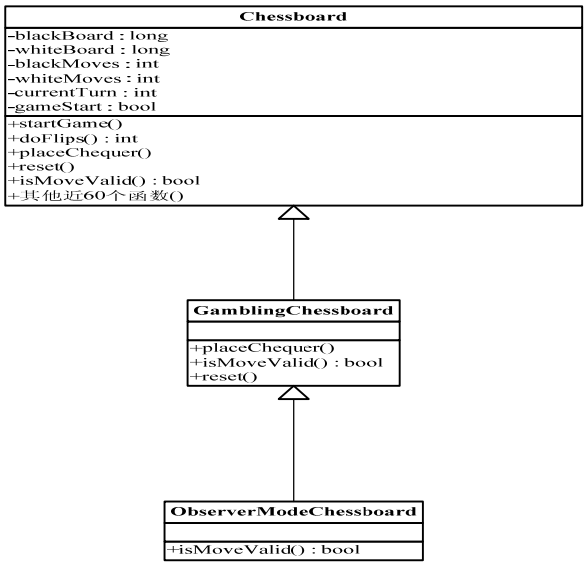
黑白棋程序设计文档

1. 棋盘设计

我们的棋盘采用了为棋盘表示，用两个 long 型数分别表示黑子和白子的位置。

棋盘走步用一个 ArrayList 保存以便悔棋和前进。另外还有两个 int 型变量记录两方走的步数。变量 currentTurn 用来保存当前轮到谁走。UML 图例如下：



上面 UML 图中的 GamblingChessboard 是提供对战用的而 ObserverModeChessboard 是用来观战用的，之所以这么设计是因为棋子绘制要通过向棋盘模型发消息以检测某个位置的状态（黑，白，空)来绘图。棋盘中的函数事先多涉及位操作，不具体叙述，这里只说明几点实现棋盘翻转和判断可走位置的方法：

方向增量数组：

```
public final static int NORTH=-8;
public final static int SOUTH=8;
public final static int WEST=-1;
public final static int EAST=1;
public final static int NORTH_WEST=-9;
public final static int NORTH_EAST=-7;
public final static int SOUTH_WEST=7;
public final static int SOUTH_EAST=9;
//下面的数组描述了往某个方向搜索的增量
public final static int[] directionInc=
{NORTH,SOUTH,WEST,EAST,NORTH_WEST
    ,NORTH_EAST,SOUTH_WEST,SOUTH_EAST};
```

```
    //下面的数组描述了棋盘上每个棋子的可搜索方向，因为并非每个棋子都可以向八个
//方向搜索的
    public final static int[] dirMask=
    {
        0x51, 0x51, 0x73, 0x73, 0x73, 0x73, 0x62, 0x62,
        0x51, 0x51, 0x73, 0x73, 0x73, 0x73, 0x62, 0x62,
        0xD5, 0xD5, 0xFF, 0xFF, 0xFF, 0xFF, 0xEA, 0xEA,
        0xD5, 0xD5, 0xFF, 0xFF, 0xFF, 0xFF, 0xEA, 0xEA,
        0xD5, 0xD5, 0xFF, 0xFF, 0xFF, 0xFF, 0xEA, 0xEA,
        0xD5, 0xD5, 0xFF, 0xFF, 0xFF, 0xFF, 0xEA, 0xEA,
        0x94, 0x94, 0xBC, 0xBC, 0xBC, 0xBC, 0xA8, 0xA8,
        0x94, 0x94, 0xBC, 0xBC, 0xBC, 0xBC, 0xA8, 0xA8,
    };
```

这样在实现翻转的时候可以根据方向掩码来决定搜索方向，函数如下：

```
    public void doFlips(int pos,ChequerState turn)
    {
        int directions=dirMask[pos];
        if((directions & 0x80) !=0)
            directionFlips(directionInc[0],pos,turn);

        if( (directions & 0x40) !=0)
            directionFlips(directionInc[1],pos,turn);

        if( (directions & 0x20) !=0)
            directionFlips(directionInc[2],pos,turn);

        if( (directions & 0x10) !=0)
            directionFlips(directionInc[3],pos,turn);

        if( (directions & 0x08) !=0)
            directionFlips(directionInc[4],pos,turn);

        if( (directions & 0x04) !=0 )
            directionFlips(directionInc[5],pos,turn);

        if( (directions & 0x02 ) !=0 )
            directionFlips(directionInc[6],pos,turn);

        if( (directions & 0x01 ) !=0 )
            directionFlips(directionInc[7],pos,turn);
    }
```

其中 directionFlips 实现了在某个特定方向的翻转。

下面叙述一下判断可走位置的方法，此方法和前面的翻转很相似：

```
//获得可走位置的函数，结果以 long 返回，其中置一的位表示可走
public long getValidMoves(ChequerState turn)
{
    long v=0;
    long emptyLoc=getEmptyLocations();
    int shift=Long.numberOfLeadingZeros(emptyLoc);
     //得到开头 0 的个数，因为这些位置不用判断可直接跳过去
    long mask=0x8000000000000000L;
    mask>>>=shift;
    while( (shift&0xFFC0)==0 && (emptyLoc<<shift)!=0) // shift&0xFFC0 判断 shift<64
    {
        if( (emptyLoc & mask )!=0 )
        {
            if(canMove(shift,turn))
                v|=mask;
        }
        mask>>>=1;
        shift+=1;
    }
    return v;
}
```

其中函数的 canMove 的实现和 doFlips 很相似，如下：

```
public boolean canMove(int pos,ChequerState turn)
{
    boolean ret=false;
    int directions=dirMask[pos];
    if((directions & 0x80) !=0)
        ret|=directionFlipable(directionInc[0],pos,turn);
    if(ret)
        return true;
    if( (directions & 0x40) !=0)
        ret|=directionFlipable(directionInc[1],pos,turn);
    if(ret)
        return true;
    if( (directions & 0x20) !=0)
        ret|=directionFlipable(directionInc[2],pos,turn);
    if(ret)
        return true;
    if( (directions & 0x10) !=0)
        ret|=directionFlipable(directionInc[3],pos,turn);
    if(ret)
        return true;
    if( (directions & 0x08) !=0)
        ret|=directionFlipable(directionInc[4],pos,turn);
```

```
        if(ret)
            return true;
        if( (directions & 0x04) !=0 )
            ret|=directionFlipable(directionInc[5],pos,turn);
        if(ret)
            return true;
        if( (directions & 0x02 ) !=0 )
            ret|=directionFlipable(directionInc[6],pos,turn);
        if(ret)
            return true;
        if( (directions & 0x01 ) !=0 )
            ret|=directionFlipable(directionInc[7],pos,turn);
        return ret;
    }
```

其中 directionFlipable 是判断在某个方向是否可翻转，与 directionFlips 类似。

2. 网络通信

网络通信这部分从概念上讲比较简单，从实现上来说就不得不考虑一些细节，所以难免变得复杂。这里先说明在网络通信中用到的方法或者说技术：

- 线程池

线程池（有人称为连接池，意思差不多）的作用是在服务器端处理频繁的网络连接，因为可想而知服务器的负担很重，而服务线程开启的消耗很大，所以可以将那些服务完的线程睡眠并放入“池”中，当有请求的时候唤醒某个线程去执行，执行完了再进入睡眠状态，这样可以减少线程开启的消耗和加快处理速度。

- JDBC 数据库连接

为了使用数据库管理玩家信息，我们的程序使用了 JDBC，具体的数据库程序我们使用了 Access，这是为了方便起见，当然你也可以选择使用其他的数据库，只要更改服务器配置文件即可，服务器程序每次运行都会读取配置文件以便进行数据库连接，配置文件的例子如下：

```
#Othello Game Server Database Properties
driver=sun.jdbc.odbc.JdbcOdbcDriver
url=jdbc:odbc:OthelloGameServer
usr=Elegate
password=******
usersInfoTableName=UsersInfo
usersInfoTableCreateCommand=CREATE TABLE UsersInfo(Nickname VARCHAR(20) PRIMARY KEY,Sex VARCHAR(10),Email VARCHAR(50),Password VARCHAR(50),Marks INTEGER WITH DEFAULT 0,Win INTEGER WITH DEFAULT 0,Lose INTEGER WITH DEFAULT 0,Draw INTEGER WITH DEFAULT 0)
```

配置文件中要给出驱动程序，数据库地址，用户名和密码（可以没有），表名和创建表的 sql 命令（如果表不存在的话，程序会自创建一个表）。

- 通信模式

在需要通信时，服务端或客户短向要通信的地址端口打开 Socket 连接，然后从套结字中创建流，这样就可以通信，当一次交互完成即将套结字关闭。

我们设计的通信是以消息对象的方式发送，所有的消息继承自父类 Msg 或其子类。每个消息都关联一个消息类型，这样就可以根据消息类型作不同的处理：
基类 Msg 如下：

```
public class Msg implements Serializable
{

    private static final long serialVersionUID = 1L;
    //消息类型
    public enum MsgType
    {
        REG_MSG,LOGIN_MSG,LOGOUT_MSG,QUERY_ROOM_MEMBERS_MSG
        ,JOIN_ROOM_MSG,QUIT_GAME_MSG,ANS_MSG,MOVE_MSG
        ,JOIN_GAME_MSG,HAND_UP_MSG,START_GAME_MSG
        ,PLAIN_MSG,OPPONENT_QUIT_GAME_MSG,GAME_OVER_MSG
        ,UPDATE_USERINFO_MSG,UPDATE_PARTNER_USERINFO_MSG
        ,UPDATE_ROOM_MSG,JOIN_WATCHER_MSG,LEAVE_WATCHER_MSG
        ,OBERSERVED_PLAYER_QUIT_GAME_MSG,JOIN_WATCHER_ANS_MSG
    ,CHECK_ONLINE,QUERY_TOP_USERS_MSG,QUERY_TOP_USERS_ANS_MSG
    };

    String msg=null;
    private MsgType type=null;

    public Msg(MsgType type)
    {
        this.type=type;
        msg=null;
    }
    public Msg(MsgType type,String msg)
    {
        this.type=type;
        this.msg=msg;
    }
    public MsgType getMsgType()
    {
        return this.type;
    }

    protected void setMsgType(MsgType type)
    {
        this.type=type;
    }

    public String getMsg()
```

```
    {
        return this.msg;
    }

    public void setMsg(String msg)
    {
        this.msg=msg;
    }

    public String toString()
    {
        return this.getClass()+"[type="+type+",msg="+msg+"]";
    }
}
```

当回答消息中 msg 属性不为 null 的话说明有异常发生，msg 中包含了对异常的说明。下面就是我们具体的通信的消息类型：

- 登陆消息，LOGIN_MSG:

  当用户登录的时候向服务器发送此消息，服务器会返回 LoginAnsMsg 以示用户登录成功或失败。

- 注册消息，REG_MSG:

  当用户注册的时候发送此消息，服务器简单的用消息 Msg 回应，如果成功返回消息的 msg 属性将为 null，否则为失败原因。

- 退出消息，LOGOUT_MSG:

  这个消息当玩家推出程序或点击 Logout 按钮的时候发出，服务端以消息 Msg 回应。

- 查询房间信息消息，QUERY_ROOM_MEMBERS_MSG

  客户端向服务器发送此消息查询房间信息，服务端用 QueryRoomMembersAnsMsg 回应。

- 加入房间消息，JOIN_ROOM_MSG:

  当用户点击房间中的空桌子的时候向服务器发出此消息，服务器验证桌子是否可用或用户是否已经加入，然后向用户返回回答消息 JoinRoomAnsMsg

- 加入游戏消息，JION_GAME_MSG:

  当用户选择和某个用户对弈的时候除了向服务器发送消息外还向对手发送此消息，然后，双方可以举手开始游戏。

- 举手消息，HAND_UP_MSG：

  当双方都进入房间并占用了同一张桌子的时候可以向服务器发送此消息，当双方都已经举手后才可以开始游戏。

- 开始游戏消息，START_GAME_MSG：

  当双方都已经向服务器发送了举手消息后，服务器向比赛双发发送此消息以表示游戏可以开始，并且在此消息中指明了双方的颜色（随机）。

- 聊天消息，PLAIN_MSG：

  当双方聊天的时候发送此消息，如果是私聊，则只发给对方，否则发给服务器让其转发至所有在线玩家。

- 游戏结束消息，GAME_OVER_MSG：

  当游戏结束的时候由最后落子的一方向服务器发送次消息，表明赢家或者平局，服

务器根据结果更新用户的胜场数，输场数，平均场数和得分。

- 更新房间信息消息，UPDATE_ROOM_MSG

当用用户加入房间或退出房间，则服务器相所有在此房间的用户发送更新房间信息消息，以使客户端更新房间玩家的列表。

- 进入旁观模式消息，JOIN_WATCHER_MSG

当某个用户想观战的时候向被观看的双方发送此消息，收到此消息的用户将其加入观察者列表并回应 JOIN_WATCHER_ANS_MSG 消息告诉当前棋面。然后就可以在走棋的时候向所有旁观者发送走棋消息。

- 退出旁观模式消息，LEAVE_WATCHER_MSG：

当退出旁观模式是想被观看的双方发送次消息。

- 被观察者异常退出消息，OBERSERVED_PLAYER_QUIT_GAME_MSG：

当被观察者退出时向所有观看其棋局的玩家发送此消息，那些旁观者受到消息后被迫退出观察者模式。

- 查询 Top 10 玩家的消息，QUERY_TOP_USERS_MSG

当用户想获知 Top 10 玩家的信息时，向服务器发送此消息。服务器返回 Top10 的列表消息 QUERY_TOP_USERS_ANS_MSG。

- 查询用户是否在线消息，CHECK_ONLINE：

此消息由服务器定时发出，每隔一段时间会向在线用户发送此消息以检查用户是否因为不明原因掉线，如果掉线将其从在线用户列表中删除。客户段只需简单的以 Msg（Msg.MsgType.ANS_MSG)回答即可。

- 走子消息，MOVE_MSG：

当博弈双方走子的时候向对方和所有观战方发送此消息。

- 其他 7 条消息不具体叙述，如 QUERY_TOP_USERS_ANS_MSG，JOIN_WATCHER_ANS_MSG，因为这些消息和上面提到的一些消息相关已经被间接提到。

消息介绍完了，下面给出我们的消息处理模型：

```
//读取消息并回写回答消息
ObjectInputStream in=new ObjectInputStream
(skt.getInputStream());
ObjectOutputStream out=new ObjectOutputStream
(skt.getOutputStream());          //从 Socket 创建对象流
Msg msg=(Msg)in.readObject(); //读取消息
skt.shutdownInput();
Msg ansMsg=dispatchMsg(msg); //处理消息
out.writeObject(ansMsg);          //回写回答消息
out.flush();
out.close();
skt.close();                      //关闭套结字
```

消息处理函数 dispatchMsg，它处理了各种可能的消息，具体的函数调用如 userLogin 在此不赘述：

```
private Msg dispatchMsg(Msg msg)
    {
        Msg.MsgType type=msg.getMsgType();
        if(type==Msg.MsgType.REG_MSG)
```

```
        {
            return registerUser((RegMsg)msg);
        }
        else if(type==Msg.MsgType.LOGIN_MSG)
        {
            return userLogin((LoginMsg)msg);
        }
        else if(type==Msg.MsgType.LOGOUT_MSG)
        {
            return userLogout((LogoutMsg)msg);
        }
        else if(type==Msg.MsgType.QUERY_ROOM_MEMBERS_MSG)
        {
            return queryRoomMembers((QueryRoomMembersMsg)msg);
        }
        else if(type==Msg.MsgType.JOIN_ROOM_MSG)
        {
            return joinRoomMsg((JoinRoomMsg)msg);
        }
        else if(type==Msg.MsgType.QUIT_GAME_MSG)
        {
            return quitGameMsg((QuitGameMsg)msg);
        }
        else if(type==Msg.MsgType.HAND_UP_MSG)
        {
            return handUpMsg((HandUpMsg)msg);
        }
        else if(type==Msg.MsgType.PLAIN_MSG)
        {
            return transmitPalinMsg((PlainTextMsg)msg);
        }
        else if(type==Msg.MsgType.GAME_OVER_MSG)
        {
            return gameOverMsg((GameOverMsg)msg);
        }
        else if(type==Msg.MsgType.QUERY_TOP_USERS_MSG)
        {
            return queryTopUsersMsg((QueryTopUsersMsg)msg);
        }
        return null;
    }
```

3. 用户信息及房间信息的表示
   - 用户信息

     基类 UserInfo 记录用户的注册信息属性包括：

private final String nickname;　　//用户名用来唯一标示一个用户
private String sex;
private String email;
子类 ExtendedUserInfo 添加了属性：
private int marks;
private int win;
private int lose;
private int draw;
OnlineUserInfo 继承了 ExtendedUserInfo，添加了属性：
private String ip;　　//记录用户 IP
private int port;　　//记录用户监听端口
记录了在线用户的信息。
OrderedUserInfo 继承自 ExtendedUserInfo，添加了属性：
private int rank;　　//记录排名
用来记录排名的用户，在查询 Top 10 Players 时使用。

- 房间信息

RoomInfo 类负责管理房间，属性包括:
private String name;　　//房间名
private ArrayList<Partner> roomMembers;　　//房间桌子列表
Partner 类负责管理每张桌子，成员属性包括：
private OnlineUserInfo firstPlayer;　　//成员一信息
private OnlineUserInfo secondPlayer;　　//成员二信息
private int handUpCount=0;　　//举手计数
此类中对 handUpCount 的操作都是同步的。