2048游戏 程序设计实践

2018211302班 2018211068 王景扬

2018211302班 2018210074 熊宇

【摘要】本2048游戏设计采用Qt编程，采用C/S架构，其中Client端对主页、注册、登录、游戏四个主要模块进行设计，Server端自主实现多线程，利用ODBC与数据库进行交互。本游戏具备良好的人机交互界面，操作方式符合用户直觉，与主流程序的操作方式一致，图形界面满足简洁、元素反差对照、空白分割、平衡恰当、元素对齐等要求。所实现的程序使用多进程/多线程等并发计算技术完成某项实际功能（例如：多用户同时在线操作或数据异步更新等）；程序在用户输入错误数据等异常情况下不出现异常退出、挂起、输出内部错误信息等情况，能够提供用户易于理解和接受的反应；本程序还正确地应用了观察者模式，合理地使用了并发、持久化、线程池等技术，架构合理，为未来功能和性能的扩展做了相应准备。

【关键词】Qt编程　信号与槽　观察者模式　C/S架构　数据库　QtSql　线程池

1.主要功能

本2048游戏采用C/S架构，Server端使用数据库存储用户信息，Client使用Qt编写的图形界面与用户进行人机交互。

本次编程代码采用驼峰法命名。

本次界面设计所有资源图片来自weibo@鱼鱼气泡水，字体及字号见界面功能备注。

1.1主界面

用户启动Client端应用程序后，进入主界面。

1.1.1界面组成

- 标题
- 欢迎语
- 注册按钮
- 登录按钮

1.1.2界面功能

- 标题：Start Up
- 欢迎语：Welcome To 2048（Times New Roman 36pt）
- 注册按钮：点击即可跳转至注册界面 （Times New Roman 14pt）
- 登录按钮：点击即可跳转至登录界面 （Times New Roman 14pt）

1.2注册界面

1.2.1界面组成

- 标题
- 用户名输入框
- 密码输入框
- 异常信息显示框
- 注册按钮
- 返回按钮

1.2.2界面功能

- 标题：Register
- 用户名输入框：待输入注册用户名 （Times New Roman 14pt）
- 密码输入框：待输入注册密码 （PASSWORD mode）
- 异常信息显示框：显示异常信息 （Times New Roman 9pt）
- 注册按钮：用户点击注册按钮后 （Times New Roman 14pt）
- 若注册用户名或注册密码为空，异常信息显示框显示用户名或密码不能为空，不建立连接；
- 若注册用户名和注册密码都不为空，建立连接，以“Register 用户名 密码”格式发送到服务器，等待服务器应答。
- 若服务器应答“success”，则清空用户名、密码输入框，断开连接释放资源，跳转至登录界面；否则异常信息显示框显示该用户名已被注册。
- 返回按钮：用户点击返回按钮后，返回主界面。（Times New Roman 14pt）

1.3登录界面

1.3.1界面组成

- 标题
- 用户名输入框
- 密码输入框
- 异常信息显示框
- 登录按钮
- 返回按钮

1.3.2界面功能

- 标题：Login
- 用户名输入框：待输入登录用户名 （Times New Roman 14pt）
- 密码输入框：待输入登录密码 （PASSWORD mode）
- 异常信息显示框：显示异常信息 （Times New Roman 9pt）
- 登录按钮：用户点击登录按钮后 （Times New Roman 14pt）
- 若登录用户名或登录密码为空，异常信息显示框显示用户名或密码不能为空，不建立连接；
- 若登录用户名和登录密码都不为空，建立连接，以“Login 用户名 密码”格式发送到服务器，等待服务器应答。
- 若服务器应答首字段为“success”，则清空用户名、密码输入框，断开连接释放资源，跳转至游戏界面；否则异常信息显示框显示用户名或密码错误。
- 返回按钮：用户点击返回按钮后，返回主界面。（Times New Roman 14pt）

1.4游戏界面

1.4.1界面组成

- 用户名显示框
- 用户当前分数显示框
- 用户最高分数显示框
- 该游戏最高分数显示框
- 2048游戏主显示板
- 游戏操作说明框
- 加载云端数据按钮
- 重置数据按钮
- 存储数据到云端按钮
- 结束游戏按钮
- 上移按钮
- 下移按钮
- 左移按钮
- 右移按钮

1.4.2界面功能

- 用户名显示框：显示当前游戏用户名 （Times New Roman 9pt）
- 用户当前分数显示框：显示当前用户的得分 （Times New Roman 9pt）
- 用户最高分数显示框：显示当前用户自注册以来的最高分 （Times New Roman 9pt）
- 该游戏最高分数显示框：显示该2048游戏玩家所得历史最高分 （Times New Roman 9pt）
- 2048游戏主显示板：2048游戏面板 （Pristina 10pt）
- 游戏操作说明框：操作说明 （Segoe Print 9pt）
- 加载云端数据按钮：从云端恢复上次退出时存储的游戏状态 （Segoe Print 9pt）
- 重置数据按钮：将当前游戏状态重置 （Segoe Print 9pt）
- 存储数据到云端按钮：保存当前游戏状态到云端 （Segoe Print 9pt）
- 结束游戏按钮：结束游戏 （Segoe Print 9pt）
- 上移按钮：向上移动 （Segoe Print 9pt）
- 下移按钮：向下移动 （Segoe Print 9pt）
- 左移按钮：向左移动 （Segoe Print 9pt）
- 右移按钮：向右移动 （Segoe Print 9pt）

1.5 服务端功能

利用线程池、套接字编程和数据库操作等，实现了以下服务端功能：

- 能够接受并处理来自客户端的注册、登录、存档请求。
- 能够在接收到注册请求时，向MySQL数据库中插入用户的用户名和密码信息，并在用户名重复的情况下返回注册失败的消息。
- 能够在接收到登录请求时，查询数据库以检查密码和用户名是否匹配。若用户名不存在或密码不正确，则返回登陆失败信息；若登录成功，则返回该用户的棋盘

状况、用户最高分、用户当前分数和游戏最高分。

- 能够在接收到存档请求时，向MySQL数据库中更新用户的棋盘状况、用户最高分和用户当前分数。
- 编程实现了一个线程池，可以自由设置线程池类型为固定线程数或固定每个线程处理的最大连接数，默认为固定线程数。
- 每当监听到一个TCP连接请求，在线程池中为这个TCP连接分配一个线程进行处理，能够同时接收并处理多个用户的请求。
- 在接收到格式错误的请求，或数据库操作失败时，会打印并向客户端返回错误信息，客户端根据错误信息进行适当处理。
- 能够在接收到客户端请求时，将当前日期时间、客户端IP地址、客户端端口号以及请求消息打印到控制台。对请求进行相应处理后，会将处理结果（成功或失败的信息，从数据库中查询得到的数据等）打印到控制台，方便观察和调试。

2.模块划分

2.1客户端

- main

  初始化所有窗体，基于观察者模式确立信号与槽机制。

- functions

  包含各模块共同需求的头文件，宏定义各模块共同需求的变量和常数。

- widget

  声明及定义主界面，实现主界面组成及功能。

- register

  声明及定义注册界面，实现注册界面组成及功能。

- login

  声明及定义登录界面，实现登录界面组成及功能。

- game

  声明及定义游戏界面，实现游戏界面组成及功能。

2.2服务端

- main

  创建一个TcpServer并开始监听。

- ThreadHandler类

  线程池类，采用单例模式，动态创建和管理服务端用于请求处理的线程，有两种模式：固定线程数、固定线程持有连接数。

- TcpServer类

  服务端的监听套接字，用于监听来自任意客户端的TCP连接请求，在成功建立TCP连接后创建TcpSocket与之进行通信。

- TcpScket类

  服务端的通信套接字，用于与特定客户端进行通信。一个TcpServer会为每一个TCP连接创建一个TcpSocket。

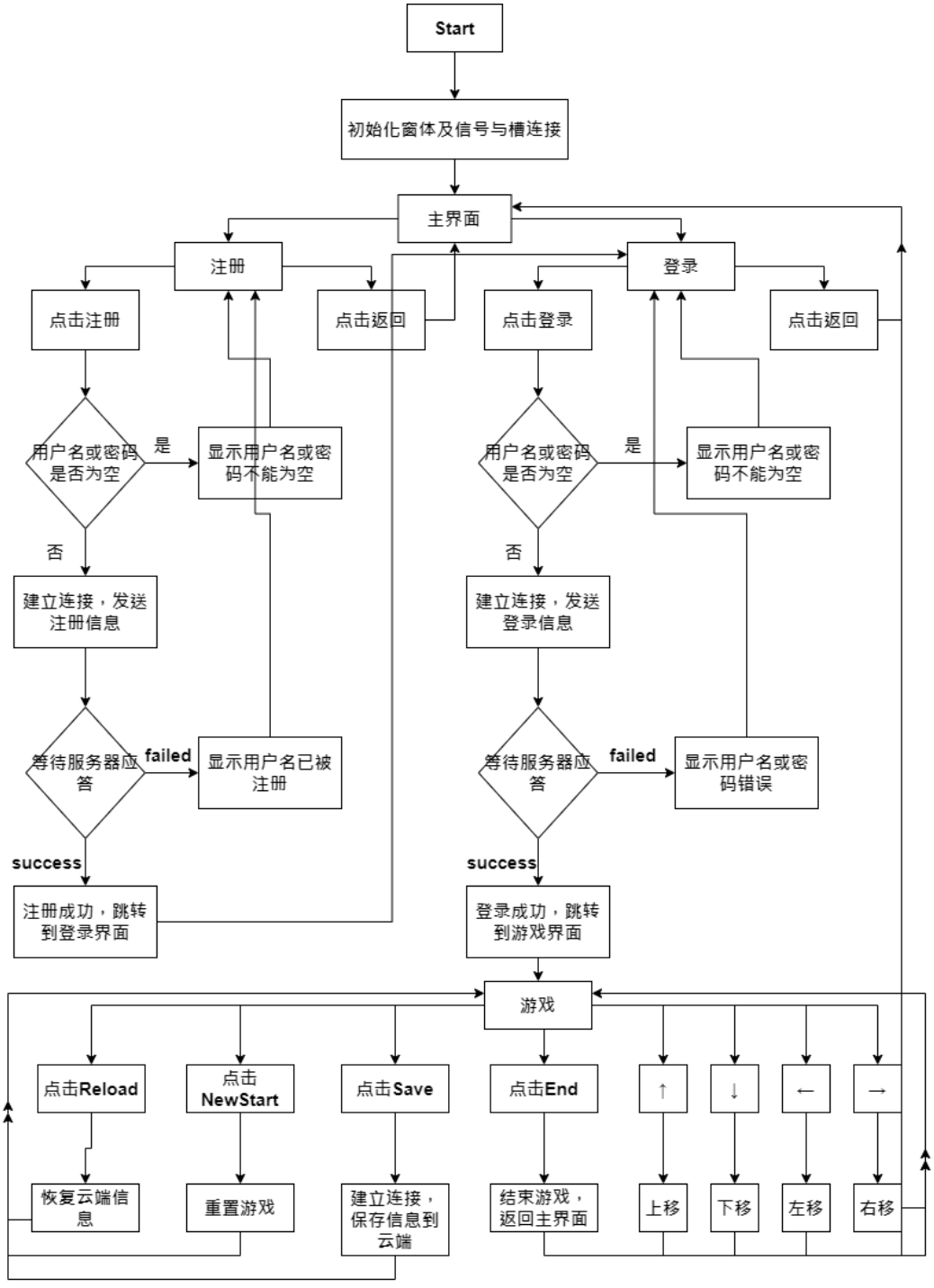
- userinfo表

  MySQL数据库中的表，有id、username、password、matrix、curScore、topScore字段，用于持久化存储用户信息。

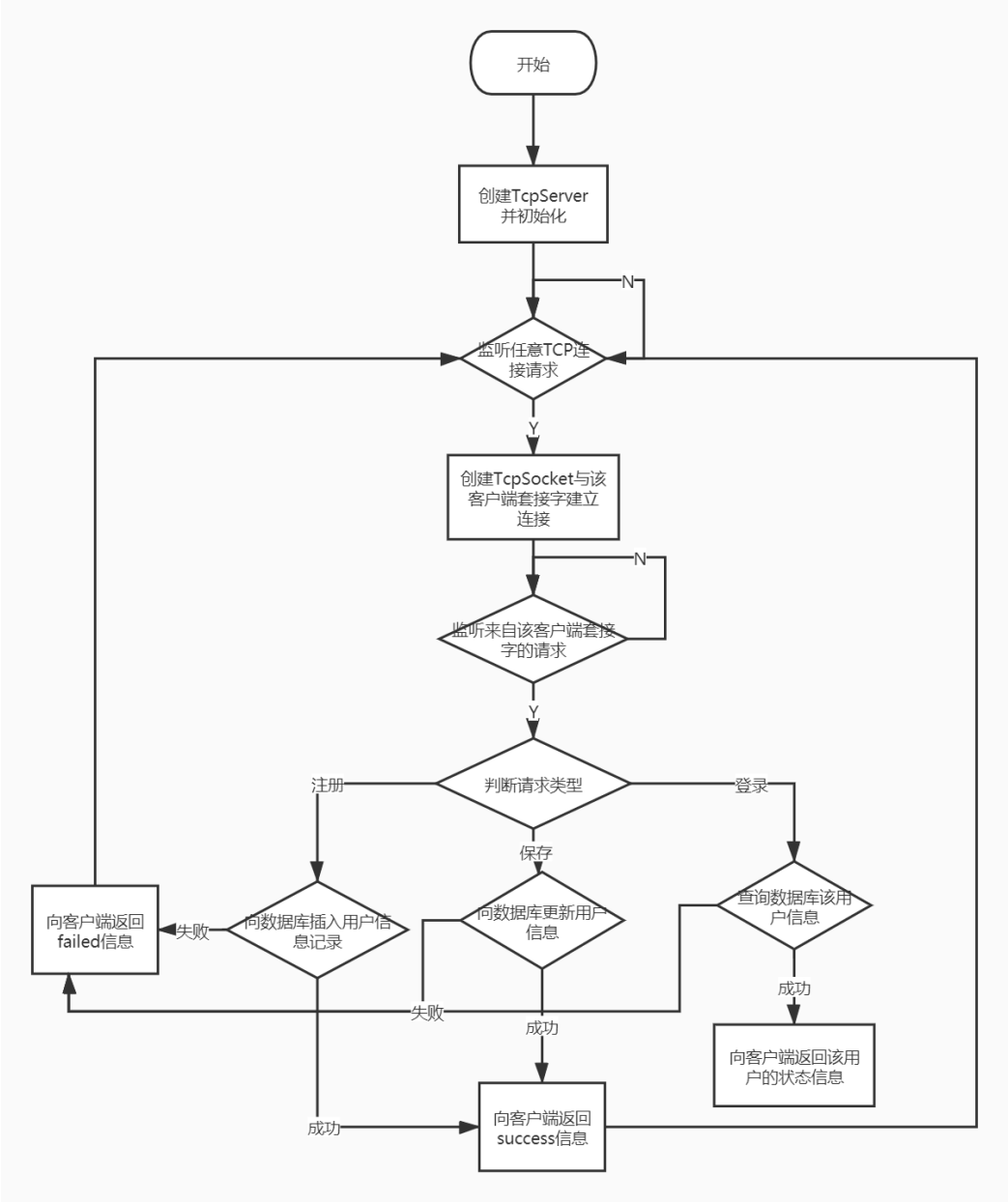
3.程序流程

3.1客户端流程图

使用的绘图软件



3.2服务端流程图


开始
创建TcpServer并初始化
N
监听任意TCP连接请求
Y
创建TcpSocket与该客户端套接字建立连接
N
监听来自该客户端套接字的请求
Y
判断请求类型
注册
登录
保存
向客户端返回failed信息
失败
向数据库插入用户信息记录
向数据库更新用户信息
查询数据库该用户信息
成功
失败
成功
向客户端返回该用户的状态信息
成功
向客户端返回success信息

4.概要设计

4.1编写目的

概要设计为2048开发提供基本的设计基础。

背景：2048是一款比较流行的益智休闲游戏，最早于2014年3月20日发行。原版2048首先在GitHub上发布，原作者是Gabriele Cirulli，后被移植到各个平台。它操作简单、容易上手，曾经在手游市场风靡一时。本次开发在基本游戏功能的基础上加入了图形界面和账户信息云存储，开发这款软件的主要目的在于供人娱乐和消遣。

4.2基本设计概念

同 3.程序流程

4.3功能需求与程序设计关系

客户端：

功能模块	子功能模块	功能描述	与其他模块关系
main	QObject::connect	信号与槽的观察者模式	设计模块间的相互调用与切换
Widget	on_needRegis_clicked	前往注册界面	激活ShowRegister信号，通过main模块中定义的信号和槽，跳转至注册界面
Widget	on_needLogin_clicked	前往登录界面	激活ShowLogin信号，通过main模块中定义的信号和槽，跳转至登录界面
Widget	ReceiveShowMain	展示自身	作为接收其他模块的ShowMain信号的槽函数
Register	on_ready_clicked	进行注册行为	如果输入不为空，就向服务器发送注册信号，根据服务器的响应选择异常显示或跳转至登录界面
register	on_back_clicked	返回主界面	激活ShowMain信号，通过main模块中定义的信号和槽，跳转至主界面
Register	ReceiveShowRegis	展示自身	作为接收其他模块的ShowRegister信号的槽函数
Register	ReceiveData	接收服务器响应	接收服务器响应
Login	on_ready_clicked	进行登录行为	如果输入不为空，就向服务器发送登录信号，根据服务器的响应选择异常显示或跳转至游戏界面

功能模块	子功能模块	功能描述	与其他模块关系
Login	on_back_clicked	返回主界面	激活ShowMain信号，通过main模块中定义的信号和槽，跳转至主界面
Login	ReceiveShowLogin	展示自身	作为接收其他模块的ShowLogin信号的槽函数
Login	ReceiveData	接收服务器响应	接收服务器响应
Game	ResetGame	重置游戏	无显式关系
Game	AddRandNum	生成随机数	无显式关系
Game	GetNullCount	获取空位置数量	无显式关系
Game	CheckGameOver	检查游戏是否结束	无显示关系
Game	MoveLeft	左移	无显式关系
Game	MoveRight	右移	无显式关系
Game	MoveUp	上移	无显示关系
Game	MoveDown	下移	无显式关系
Game	ClearScreen	清屏	无显式关系
Game	RefreshShow	更新界面展示	无显式关系
Game	on_newStart_clicked	启动重置游戏	启动ResetGame
Game	on_reload_clicked	加载云端数据	无显示关系
Game	on_save_clicked	保存数据到云端	向服务器发送保存信号
Game	on_end_clicked	返回主界面	激活ShowMain信号，通过main模块中定义的信号和槽，跳转至主界面
Game	on_up_clicked	上移	启动MoveUp
Game	on_down_clicked	下移	启动MoveDown
Game	on_left_clicked	左移	启动MoveLeft
Game	on_right_clicked	右移	启动MoveRight
Game	ReceiveShowGame	展示自身	作为接收其他模块的ShowGame信号的槽函数
Game	ReceiveData	接收服务器响应	接收服务器响应

服务端：

功能模块	子功能模块	功能描述	调用关系或时机
ThreadHandler	getThread	从线程池获得一个线程	TcpServer
ThreadHandler	removeThread	从线程池释放一个进程	TcpServer

功能模块	子功能模块	功能描述	调用关系或时机
ThreadHandler	getClass	获取该类一个实例（单例模式）	TcpServer
TcpServer	setMaxPendingConnections	setMaxPendingConnections	构造时
TcpServer	clear	断开所有连接，线程计数器请0	析构时
TcpServer	sockDisConnectSlot	断开指定连接	TcpSocket发送sockDisConnect信号时
TcpServer	incomingConnection	重写以获取多线程	监听到连接请求时调用
TcpSocket	sendData	发送信号的槽	发送消息时
TcpSocket	disConTcp	断开TCP连接	TcpServer发送sentDisConnect信号时
TcpSocket	readData	接收数据并处理	本模块发送readyRead信号时

5.思考总结

5.1 熊宇

主要负责图形化界面以及游戏代码逻辑。

在本次程序设计实践中，我最开始实现了无图形界面的命令行游戏。后期转入Qt时，Winsock在Qt中可以连接但无法正常发送消息，最后才发觉必须使用Qt自身监听套接字库QTcpServer和通讯套接字库QTcpSocket，改动后才可使用。好在实现游戏代码逻辑中没有什么麻烦出现，让我得以对UI界面进行精细设计。在Game界面的Save中，最开始我只将当前数据保存至云端，却忽视了更新Reload的直接来源——temp数据，在测试中发现了异常，最终改正。

经过本次程序设计实践，我进一步提升了自己的编程能力，培养了自己优秀的编程习惯，进一步形成了自己的编程风格。在排难的过程中，我培养了自己结合所有材料自学的能力。在代码优化的过程中，提升了自己工程角度精益求精、从细微处入手的专业态度。在界面设计中，提升了我从用户角度出发、从大众使用习惯出发的换位思考、人性化设计的能力，使我受益匪浅。

5.2王景扬

主要负责服务端代码逻辑。

总的来说还是遇到了不少困难。虽然能够较熟练地运用C语言和Java的线程、套接字、数据库操作，但QT有自己的线程库QThread、监听套接字库QTcpServer、通讯套接字库QTcpSocket和数据库操作库QSqlQuery，之前对这些库并不熟悉，只好边做边学。好在过程还算顺利，QT的库感觉还挺好用的，尤其是数据库操作要比C和Java的简单一些。

线程池的实现也事先查阅了不少资料，了解了一些实现方法，最后简单地实现了一个。

总的来说，这次服务端的开发，让我对线程池技术、套接字编程以及数据库操作更加熟练了，很有收获。

6.部分核心代码

1.客户端

(1)上下左右移动

```
//////////////////////////////////////////////////////////////////////
//
// Function:         MoveLeft
// Description:
//  左移
// Args:
//  无参
// Return Values:
//      void
//      无描述
//////////////////////////////////////////////////////////////////////
//
void Game::MoveLeft()
{
    /* 变量i用来遍历行项的下标，并且在移动时所有行相互独立，互不影响 */
    int i;
    for (i = 0; i < NUMSIZE; ++i)
    {
        /* 变量j为列下标，变量k为待比较（合并）项的下标，循环进入时k<j */
        int j, k;
        for (j = 1, k = 0; j < NUMSIZE; ++j)
        {
            if (board[i][j] > 0) /* 找出k后面第一个不为空的项，下标为j，
之后分三种情况 */
            {
                if (board[i][k] == board[i][j])
                {
                    /* 情况1：k项和j项相等，此时合并方块并计分 */
                    score += board[i][k++] *= 2;
                    board[i][j] = 0;
                    ifNeedAddNum = 1; /* 需要生成随机数和刷新界面 */
                }

                else if (board[i][k] == 0)
                {
                    /* 情况2：k项为空，则把j项赋值给k项，相当于j方块移动到k
方块 */
                    board[i][k] = board[i][j];
                    board[i][j] = 0;
                    ifNeedAddNum = 1;
                }
```

```
                else
                {
                    /* 情况3：k项不为空，且和j项不相等，此时把j项赋值给k+1
项，相当于移动到k+1的位置 */
                    board[i][++k] = board[i][j];
                    if (j != k)
                    {
                        /* 判断j项和k项是否原先就挨在一起，若不是则把j项赋
值为空（值为0） */
                        board[i][j] = 0;
                        ifNeedAddNum = 1;
                    }
                }
            }
        }
    }
}
```

(2)存档到云端

```
//////////////////////////////////////////////////////////////////////
//
// Function:         on_save_clicked
// Description:
//  按下Save按键，将当前数据存档到云端
// Args:
//  无参
// Return Values:
//      void
//      无描述
//////////////////////////////////////////////////////////////////////
//
void Game::on_save_clicked()
{
    /* 将临时信息也更改 */
    this->tempScore=this->score;
    for(int i=0;i<NUMSIZE;i++)
    {
        for(int j=0;j<NUMSIZE;j++)
        {
            temp[i][j]=board[i][j];
        }
    }

    this->tcpClient = new QTcpSocket(this);
    this->tcpClient->abort();

    connect(this-
>tcpClient,&QTcpSocket::readyRead,this,&Game::ReceiveData);
```

```
    connect(this->tcpClient,&QTcpSocket::disconnected,[](){qDebug()
<< "Disconnected." ;});

    QString ipAdd(QString::fromStdString(SERVER_IP));
//10.128.221.5
    QString portd(QString::fromStdString(SERVER_PORT));
    qDebug()<<"No error";

    this->tcpClient->connectToHost(ipAdd,portd.toInt());

    if(this->tcpClient->waitForConnected())
    {
        qDebug()<<"System Ready!";
    }
    else
    { /* 连接失败则尝试重连 */
        qDebug()<<"Failed."<<this->tcpClient->errorString();

        while(!this->tcpClient->waitForConnected())
        {
            this->tcpClient->connectToHost(ipAdd,portd.toInt());
            qDebug()<<"Retry connectToHost.";
        }

        qDebug()<<"System Ready!";
    }



    /* 刷新buffer */
    ZeroMemory(buf, BUF_SIZE);

    /* 保存：Save 用户名 数组[4][4]以逗号为间隔 用户当前分数 用户最高分 游戏
最高分 */
    strcat(buf,"Save ");
    strcat(buf,this->userName);
    strcat(buf," ");

    for(int i=0;i<NUMSIZE-1;i++)
    {
        for(int j=0;j<NUMSIZE;j++)
        {
            strcat(buf,QString::number(this->board[i]
[j],10).toLatin1().data());
            strcat(buf,",");
        }
    }

    strcat(buf,QString::number(this->board[3]
[0],10).toLatin1().data());
    strcat(buf,",");
    strcat(buf,QString::number(this->board[3]
[1],10).toLatin1().data());
```

```
    strcat(buf,",");
    strcat(buf,QString::number(this->board[3]
[2],10).toLatin1().data());
    strcat(buf,",");
    strcat(buf,QString::number(this->board[3]
[3],10).toLatin1().data());
    strcat(buf," ");
    strcat(buf,QString::number(this->score,10).toLatin1().data());
    qDebug()<<buf;


    /* 向服务器发送数据 */
    this->tcpClient->write(QString::fromStdString(buf).toUtf8());

    if (SOCKET_ERROR == retVal)
    {
        qDebug() << "send failed !";
    }

    /* 刷新buffer */
    ZeroMemory(buf, BUF_SIZE);

    /* 断开连接 */
    this->tcpClient->disconnectFromHost();
    this->tcpClient->close();

}
```

(3)重置游戏

```
//////////////////////////////////////////////////////////////////////
//
// Function:         ResetGame
// Description:
//  重置游戏
// Args:
//  无参
// Return Values:
//      void
//      无描述
//////////////////////////////////////////////////////////////////////
//
void Game::ResetGame()
{
    score = 0;
    ifNeedAddNum = 1;
    ifGameOver = 0;
    if_prepare_exit = 0;

    /* 了解到游戏初始化时出现的两个数一定会有个2，所以先随机生成一个2，其他均为
0 */
```

```
    int n = rand() % (NUMSIZE* NUMSIZE);

    int i;
    for (i = 0; i < NUMSIZE; ++i)
    {
        int j;
        for (j = 0; j < NUMSIZE; ++j)
        {
            board[i][j] = (n-- == 0 ? 2 : 0);
        }
    }

    /* 前面已经生成了一个2，这里再生成一个随机的2或者4，概率之比9:1 */
    AddRandNum();

    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
            qDebug()<<board[i][j]<<" ";
        qDebug()<<endl;
    }

    /* 在这里刷新界面并显示的时候，界面上已经默认出现了两个数字，其他的都为空
（值为0） */
    RefreshShow();
}

////////////////////////////////////////////////////////////////////
//
// Function:         AddRandNum
// Description:
//  生成随机数
// Args:
//  无参
// Return Values:
//      void
//      无描述
////////////////////////////////////////////////////////////////////
//
void Game::AddRandNum()
{
    srand((unsigned int)time(0));
    int n = rand() % GetNullCount(); /* 确定在何处空位置生成随机数 */

    int i;
    for (i = 0; i < NUMSIZE; ++i)
    {
        int j;
        for (j = 0; j < NUMSIZE; ++j)
        {
            /* 定位待生成的位置 */
            if (board[i][j] == 0 && n-- == 0)
```

```
            {
                board[i][j] = (rand() % 10 ? 2 : 4); /* 生成数字2或
4，生成概率为9:1 */
                return;
            }
        }
    }
}

//////////////////////////////////////////////////////////////////////
//
// Function:         AddRandNum
// Description:
//  获取空位置数量
// Args:
//  无参
// Return Values:
//      int
//      空位置数量
//////////////////////////////////////////////////////////////////////
//
int Game::GetNullCount()
{
    int n = 0;

    int i;
    for (i = 0; i < NUMSIZE; ++i)
    {
        int j;
        for (j = 0; j < NUMSIZE; ++j)
        {
            board[i][j] == 0 ? ++n : 1;
        }
    }

    return n;
}
```

2.服务端

(1)线程池

单例实现

```
ThreadHandle & ThreadHandle::getClass()
{
    static ThreadHandle th;
    return th;
}
```

从池中获取一个线程

```
QThread *ThreadHandle::getThread()//分配一个线程
{
    if (!initFirst)
    {
        initThreadType(THREADSIZE,10);
    }
    if (type == THREADSIZE)
        return findThreadSize();
    else
        return findHandleSize();
}

QThread * ThreadHandle::findHandleSize() //查找到线程里的连接数小于最大值就返回查找到的，找不到就新建一个线程
{
    for (auto it  = threadSize.begin();it != threadSize.end()
;++it)
    {
        if (it.value() < size)
        {
            it.value() ++;
            return it.key();
        }
    }

    //新建线程插入线程池
    QThread * tmp = new QThread;
    threadSize.insert(tmp,1);
    tmp->start();
    return tmp;
}

QThread *ThreadHandle::findThreadSize() //遍历查找所有线程中连接数最小的那个，返回。否则返回第一个
```

```
{
    auto it = threadSize.begin();
    auto ite = threadSize.begin();
    for (++it ; it != threadSize.end(); ++it)
    {
        if (it.value() < ite.value())
        {
            ite = it;
        }
    }
    ite.value() ++;
    return ite.key();
}
```

从池中删除一个线程

```
void ThreadHandle::removeThread(QThread * thread)//删除一个线程
{
    auto t = threadSize.find(thread);
    if (t != threadSize.end())
    {
        t.value() --;
        if (type == HANDLESIZE && t.value() == 0 &&
threadSize.size() > 1)
        {
            threadSize.remove(thread);
            thread->exit();
            thread->wait(3000);
            delete thread;
        }
    }
}
```

(2)数据库

```
//连接MySQL数据库
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("127.0.0.1");
db.setUserName("root");
db.setPassword("root");
db.setDatabaseName("game2048");

if(!db.open()){
qDebug()<<"数据库连接失败:"<<db.lastError().text()<<endl;
exit(0);
}else{
qDebug()<<"数据库连接成功"<<endl;
}
```

(3)多线程捕获新的TCP连接

```
void TcpServer::incomingConnection(qintptr socketDescriptor) //多线程必须在此函数里捕获新连接
{
    if (tcpClient->size() > maxPendingConnections())//继承重写此函数后，QTcpServer默认的判断最大连接数失效，自己实现
    {
        QTcpSocket tcp;
        tcp.setSocketDescriptor(socketDescriptor);
        tcp.disconnectFromHost();
        return;
    }

    //创建一个新的TcpSocket和线程处理该连接
    auto th = ThreadHandle::getClass().getThread();
    auto tcpTemp = new TcpSocket(socketDescriptor);
    QString ip =  tcpTemp->peerAddress().toString();
    qint16 port = tcpTemp->peerPort();

    //断开连接的处理，从列表移除，并释放断开的Tcpsocket，此槽必须实现，线程管理计数也是靠的他

 connect(tcpTemp,&TcpSocket::sockDisConnect,this,&TcpServer::sockDisConnectSlot);

    //断开信号

 connect(this,&TcpServer::sentDisConnect,tcpTemp,&TcpSocket::disConTcp);

    tcpTemp->moveToThread(th);//把tcp类移动到新的线程，从线程管理类中获取
    tcpClient->insert(socketDescriptor,tcpTemp);//插入到连接信息中
    emit connectClient(socketDescriptor,ip,port);
}
```

(4)TCP连接断开方法

```
void TcpServer::sockDisConnectSlot(int handle,const QString & ip,quint16 prot,QThread * th)
{
    tcpClient->remove(handle);      //连接管理中移除断开连接的socket
    ThreadHandle::getClass().removeThread(th);  //告诉线程管理类哪个线程里的连接断开了
    emit sockDisConnect(handle,ip,prot);
```

```
}

void TcpSocket::disConTcp(int i)
{
    if (i == socketID)
    {
        this->disconnectFromHost();//与客户端断开连接
    }
    else if (i == -1) //-1为全部断开
    {
        disconnect(dis); //先断开连接的信号槽，防止二次析构
        this->disconnectFromHost();
        this->deleteLater();
    }
}
```