

基础实验

实验1

使用new申请空间

传递引用

实验2.1

实验2.2

使用new申请内存

实验3

Virtual function

实验5.1

综合实验

设计说明

数据库中用户信息格式

server端发送消息类型和格式

client端发送消息类型和格式

详细说明

pet_list

user_list

random_pet_list

Server端架构

Client 端架构

TcpClient窗体

1.登录/注册界面

2.查看信息界面

3. 比赛准备界面

4.送出精灵界面

FightManager窗体

问题与解决

1

2

3

4

5

附录A:基础部分程序源代码

实验1

实验2.1

实验2.2

实验3

实验4

实验5.1

实验5.2

基础实验

实验1

使用new申请空间

`new` 语句比 `malloc` 语句对程序员更加友好。写法上，更加简洁，也更加好懂。

以矩阵运算为例。当我们需要创建一个矩阵的时候，可以使用二级指针，在内存中申请一块空间。

```
1  mat = new int*[row_len] ;
2
3  for(int i=0 ; i<row_len ; i++){
4
5      mat[i] = new int [col_len] ;
6
7  }
```

观察上面的代码：

首先给二级指针`mat`申请内存。申请了长度为`row_len`的内存，类型是一级`int`指针。

然后给每个一级指针申请动态内存，长度为`col_len`，类型为`int`。

可以将 `[]` 中的内容看做长度，之前的 `int*`, `int` 等看做类型，这样申请空间的语句就变得好懂。多维数组申请空间需要从高级向低级申请。

传递引用

```
1  int **&mat
```

的写法很让人在意。从字面上看（从右向左看），这应该是“引用的二级指针”。但是一般的解释说这是二级指针的引用。

从意义上来说，引用只是一个别名，创建指向引用的指针是不合理的。这点容易想明白。但是写法上却有些让人迷惑。

相对的，这样的写法就是错误的：

```
1  int &**mat
```

不允许使用指向引用的指针

速览问题 (Alt+F8) 没有可用的快速修复

实验2.1

实验中创建了两个版本：在函数中传递形参，在函数中传递引用。发现传递引用的版本减少了很多对象的构造和析构，最终保留了传递引用的版本。

对于不带引用的版本，在报告中保留源代码和输出信息（包含对于构造对象和析构对象的分析）。

源代码

```
1  #include<iostream>
2  #include<cstdio>
3  #include<cmath>
```

```

4
5 class Point
6 {
7 private:
8     /* data */
9     double x, y ;
10 public:
11     Point(double x=0, double y=0)
12     : x(x), y(y)
13     {
14         std::cout<<"Creating a Point...\n" ;
15     }
16     Point (const Point& obj){
17         *this = obj ;
18         std::cout<<"Copying a Point...\n" ;
19     }
20     ~Point(){
21         std::cout<<"Deleting a Point...\n" ;
22     }
23
24     double get_x() const {return x;}
25     double get_y() const {return y;}
26 };
27 double dist(Point A, Point B) {
28     // 返回A和B的euclid距离
29     double dist_x = A.get_x() - B.get_x() ;
30     double dist_y = A.get_y() - B.get_y() ;
31     return sqrt(dist_x*dist_x + dist_y*dist_y) ;
32 }
33
34 class Circle
35 {
36 private:
37     /* data */
38     Point c ;
39     double r ;
40 public:
41     Circle(Point c, double r=0)
42     : c(c), r(r)
43     {
44         std::cout<<"Creating a Circle...\n" ;
45     }
46     Circle(const Circle& obj){
47         *this = obj ;
48         std::cout<<"Copying a Circle...\n" ;
49     }
50     ~Circle(){
51         std::cout<<"Deleting a Circle...\n" ;
52     }
53     Point get_c() const {return c ; }
54     double get_r() const {return r; }
55 };
56
57 bool is_intersect(Circle o1,Circle o2){
58     std::cout<<"\n" ;
59     Point c1, c2 ;
60     c1 = o1.get_c() ;
61     c2 = o2.get_c() ;

```

```

62     return o1.get_r() + o2.get_r() > dist(c1, c2) ;
63 }
64
65 int main(){
66     Point A(1,0), B(0,1) ;
67     std::cout<<"\n" ;
68     std::cout<<dist(A,B)<<"\n";
69     std::cout<<"\n" ;
70     Circle C1(Point(1,0), 2), C2(Point(4,0), 1.2);
71     std::cout<<"\n" ;
72     std::cout<<is_intersect(C1, C2)<<"\n" ;
73     std::cout<<"\n" ;
74     C1.get_c() ;
75     std::cout<<"\n" ;
76     return 0 ;
77 }
78

```

输出信息

```

1  创建A和B
2
3  Creating a Point...
4
5  Creating a Point...
6
7  dist创建形参并且销毁
8
9  Copying a Point...
10
11 Copying a Point...
12
13 1.41421
14
15 Deleting a Point...
16
17 Deleting a Point...
18
19 创建Circle的时候，首先创建临时的Point，然后通过copy复制创建Circle的c，创建Circle，最后删掉临时的Point
20
21 Creating a Point...
22
23 Copying a Point...
24
25 Creating a Circle...
26
27 Deleting a Point...
28
29 Creating a Point...
30
31 Copying a Point...
32
33 Creating a Circle...
34
35 Deleting a Point...
36

```

```
37 is_intersect先创建形参Circle的c, 然后通过copy复制创建形参
38
39 Creating a Point...
40
41 Copying a Circle...
42
43 Creating a Point...
44
45 Copying a Circle...
46
47 创建c1,c2, 使用函数get_c 会copy一个实例出来, 然后返回, 赋值之后删掉。然后给dist传参,
    这时候要创建形参, 用完删掉。删掉c1,c2. 最后删掉o1, o2(当然删掉Circle之后要把c也删掉)。
48
49 Creating a Point...
50
51 Creating a Point...
52
53 Copying a Point...
54
55 Deleting a Point...
56
57 Copying a Point...
58
59 Deleting a Point...
60
61 Copying a Point...
62
63 Copying a Point...
64
65 Deleting a Point...
66
67 Deleting a Point...
68
69 Deleting a Point...
70
71 Deleting a Point...
72
73 1
74
75 Deleting a Circle...
76
77 Deleting a Point...
78
79 Deleting a Circle...
80
81 Deleting a Point...
82
83 可以看到, 调用get_c也会使用复制构造函数
84
85 Copying a Point...
86
87 Deleting a Point...
88
89 最后删掉main中创建的实例(按照创建时间倒序删除, 使用堆栈来保存创建的对象)
90
91 Deleting a Circle...
92
93 Deleting a Point...
```

```
94
95 Deleting a Circle...
96
97 Deleting a Point...
98
99 Deleting a Point...
100
101 Deleting a Point...
```

实验2.2

使用new申请内存

```
1 void fun()
2
3 {
4
5     A *a = new A();
6
7 }
8
9 int main()
10
11 {
12
13     while(1)
14     {
15
16         fun();
17
18     }
19
20
21     return 0;
22
23 }
```

当离开fun时，虽然离开了作用域，但用new动态开辟空间的对象是不会析构的，你可以观察任务管理器，看到内存一直在上升。但你在其他地方却无法使用a所开辟的空间，因为a这个指针是保存在栈上的，当离开作用域后就自动析构(或者说自动消失了)，但它所在分配空间是分配在堆上的，只有主动析构或程序结束，才会释放空间，也就是丢失了这块空间的地址，无法操作这块空间了。

本程序析构函数中，一定要释放矩阵的内存。不然就会只将指针指向的地址释放掉，但是堆中相应的空间还是被占用。

Delete 和 delete [] 的区别

<https://blog.csdn.net/cbNotes/article/details/38900799>

delete 的参数只有一个。之前使用了这样的语句：

```
1 delete pA1, pA2, pA3 ;
```

结果从输出信息中发现只调用了—一个析构函数。 pA2, pA3其实都没有被delete掉。

实验3

Virtual function

当基类中有一种功能，需要在所有的子类中实现，但是不同的子类实现方法又很不相同，这时候可以使用**virtual function**的方式声明该函数，表示这个函数是有的，但是我并没有明确的定义出来，你作为子类应该自己想办法定义这个函数。严格来说，上面说的是**pure virtual function**，是父类没有对函数进行定义。

普通的**virtual function**，父类是可以进行定义的，但是子类可以根据 ([具体情况 ([进行更改。 ([

当然一般的函数就是**non-virtual function**，子类是不可以更改的。

```
1 virtual double getArea() const=0 ; // pure virtual function. No definition
```

Shape 类里面就使用**virtual function** 定义了**getArea**函数，因为只知道这是一个形状，对于获取面积不能提供任何信息，也就没什么可写的，全部留给子类去完成。

```
1 virtual double getArea() const
2
3 {
4
5     return a * b;
6
7 }
```

而在**Rectangle**里面，就可以对**getArea**进行定义了。

后面在**Square**里面，就没有再对**getArea**进行定义，而是直接调用了**Rectangle**里面定义好的函数。

构造函数的调用顺序

Creating a Shape...

Creating a Rectangle...

Creating a Square...

9

Creating a Shape...

Creating a Rectangle...

12

Creating a Shape...

Creating a Circle...

12.5664

Deleting a Circle...

Deleting a Shape...

Deleting a Rectangle...

Deleting a Shape...

Deleting a Square...

Deleting a Rectangle...

Deleting a Shape...

这里获取了产生各个对象并且输出面积的时候，构造函数的调用顺序。可以看到，构造函数是由内而外进行调用的。父类先于子类。因为子类包含了父类的一部分，要先构造内部，才能在外面进行进一步的构造。

实验5.1

如果类中至少有一个函数被声明为纯虚函数，则这个类就是抽象类。纯虚函数是通过在声明中使用 `= 0` 来指定的。

设计**抽象类**（通常称为 **ABC**）的目的，是为了给其他类提供一个可以继承的适当的基类。抽象类不能被用于实例化对象，它只能作为**接口**使用。如果试图实例化一个抽象类的对象，会导致编译错误。

抽象类中会定义好一个能够对外界开放的函数，并且将其定义为虚函数，要由子类实现具体的内容。这样规范了统一的接口形式，而且便于灵活开发。

之前在实验3中，已经使用了**pure virtual function** 去定义Shape中的getArea.

```
1 Shape *s2 = &r1;
2
3 std::cout << s2->getArea() << "\n\n";
```

这里使用一个Shape类型的指针访问一个Rectangle类型的对象。我们想要使用Rectangle类型的getArea计算面积并且返回值。在没有加上virtual的时候，由于指针是Shape类型，程序调用了Shape类型的getArea。

如果使用了virtual关键字，程序将根据引用或指针指向的**对象类型**来选择方法，否则使用**引用类型或指针类型**来选择方法。

使用了virtual 之后，程序就能够正常调用Rectangle中重新定义的getArea了

综合实验

设计说明

数据库中用户信息格式

name(key, unique)	password	pet_num	pet_list	total_fight	win_fight
QString	QString	int	QString	int	int

server端发送消息类型和格式

list[0]	list[1]	list[2]	list[3]	list[4]	list[5]
z	true/false				
d	true/reload/false	pet_num	pet_list	total_fight	win_fight
a	all_user_list				
o	online_user_list				
detail	pet_num	pet_list	total_fight	win_fight	
rand_p	random_pet_list				

client端发送消息类型和格式

list[0]	list[1]	list[2]	list[3]	list[4]
z	name	password	pet_list	
d	name	password		
a				
o				
detail	name			
rand_p				
result	pet_num	pet_list	total_fight	win_fight

详细说明

pet_list

字符串类型。每个精灵的信息最后，用 | 分隔。

每个精灵的信息包括 编号，名字，等级，经验。信息之间用 * 分隔。

user_list

字符串类型。包含一串用户的name. 信息之间用 * 分隔。

random_pet_list

字符串类型。包含一串随机精灵的编号。信息之间用 * 分隔。

Server端架构

程序开始运行后，实例化TcpServer类(继承自QMainWindow)的对象win. 也就是Server的主界面。

在win的构造函数中，实例化一个server_m类(继承自QTcpServer)的对象server.

窗口只有一个按钮“开启服务器”，按下之后，server开始监听。

在server的构造函数中，启动数据库User(如果不存在则创建)，并试图在数据库中创建table user(如果已经存在则不执行操作)。

如果有client试图建立连接，则会触发server的incomingConnection函数，分配一个socket描述符descriptor，server会实例化一个新的socketthread类(继承自QThread)的对象thread，然后thread开始运行。

在thread的构造函数中，实例化一个socket_m类(继承自QTcpSocket)的对象socket，其socket描述符即为传递过来的descriptor；同时该线程中的对象sql打开数据库User。多个线程共享同一个数据库。

每个thread有一个socket。每当有一个用户登陆的时候，server就会多打开一个thread。程序使用多线程异步通信方式。

之后每个线程进入等待消息的状态。如果接收到来自client的消息，则进行处理。共有7种消息：注册，登录，请求所有用户的列表，请求在线用户列表，请求某用户的详细数据，让服务器发送15个随机的精灵序号，更新数据库。

Client 端架构

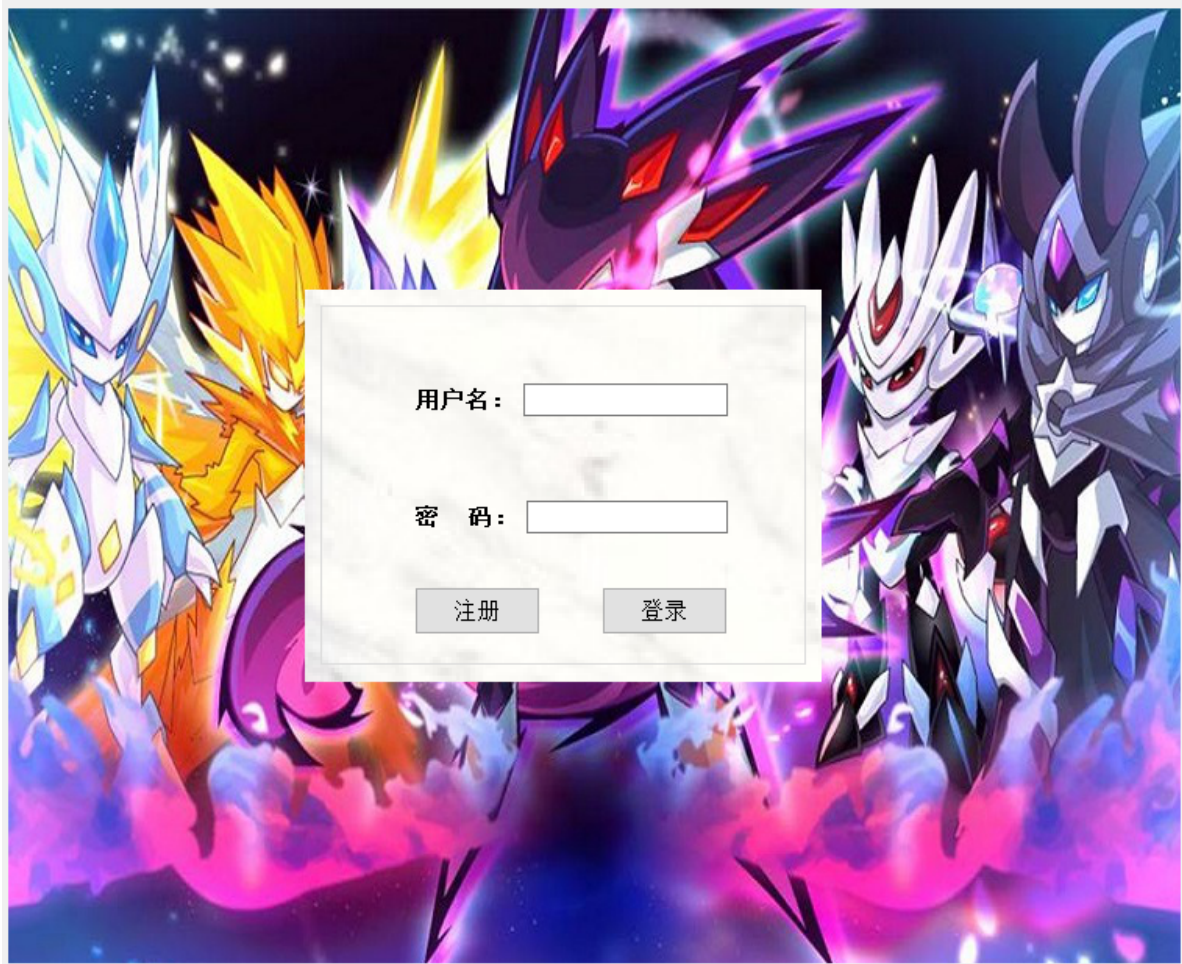
Client端共有两个窗体类：TcpClient 和 FightManager (都是继承自QMainWindow)。而TcpClient是Client端的主窗口。

TcpClient窗体

TcpClient窗体共有四个页面：

1. 登录/注册界面
2. 查看信息界面
3. 比赛准备界面
4. 送出精灵界面

1.登录/注册界面



用户选择登录或者注册。如果注册已经存在的用户名，则会提示用户名已被注册。登录时，如果用户名不存在，或者密码错误，则会提示用户名或密码错误。如果尝试在多客户端上登录同一个账号，则会提示该用户已登录。

用户第一次注册账号时，系统会提示注册成功。这时点击登录，则可以进入游戏。登陆成功之后，socket会试图和server端建立连接。连接使用固定的ip(127.0.0.1)和port(8000)。注册，登陆不成功，或者主动退出账号，server端都会断开连接。

2.查看信息界面

用户: abc		宠物个数勋章: 铁牌	
胜率: 66%		满级宠物勋章: 铁牌	

	名称	等级
1	里奥斯	6



所有用户

abc

胜率: 66%

刷新

升级对战

决斗

退出

1	名称	里奥斯
2	等级	6
3	类号	8
4	属性	火
5	血量	650
6	攻击力	150
7	防御力	75
8	敏捷度	22
9	暴击率	5
10	闪避率	5
11	经验值	600
12	技能一	normalAttack
13	技能二	密集火网
14	技能三	烈火燎原

在这里，用户可以查看自己的用户名，比赛胜率和获得的荣誉。

左上方的表格显示的是用户自己的精灵列表。单击表格中的某一行可以在右侧表格中显示该精灵的详细信息。

左下方的表格显示其他用户的详细信息。用户可以通过上方的两个下拉列表选择。左边的下拉列表可以选择请求全部用户的名单或者在线用户的名单；右边的下拉列表可以选择 用户名单中的某一位用户，显示他的详细信息。

点击刷新，则可以刷新当前页面上的信息(可能有新用户登录游戏，在线用户列表变化；精灵升级赛获胜，获得经验，等等)。

点击升级对战和决斗，都会进入页面3.

升级对战获胜后可以获得经验，失败则无事发生。

决斗赛胜利后可以获得对战的敌方精灵(1级)，失败后则挑选用户列表中的三只精灵，由用户选择一只送出(若是不够3只则全部列出)。

点击退出返回到界面1，关闭正在通信的socket.

3. 比赛准备界面

	名称	等级
1	布布花	1
2	圣光之子	2
3	圣光之子	3
4	谱尼	4
5	圣光之子	5
6	谱尼	6
7	巴拉龟	7
8	布布花	8
9	里奥斯	9
10	圣光之子	10
11	巴鲁斯	11



1	名称	圣光之子
2	等级	2
3	类号	4
4	属性	圣灵
5	血量	530
6	攻击力	140
7	防御力	55
8	敏捷度	11
9	暴击率	1
10	闪避率	1
11	经验值	0
12	技能一	normalAttack
13	技能二	宇宙微茫
14	技能三	光扬宇宙

返回

开始对战

刷新

选定自己的某个精灵，并选择升级对战或者决斗之后，会进入比赛准备界面。左边的列表会显示服务器随机发送过来的15个精灵，等级从1到15级递增。单击某一行，可以在右边的列表中显示该精灵的详细信息。

单击刷新，服务器会送来新的15个精灵的列表，并显示在屏幕上。

单击开始对战，将会跳转到FightManager窗口，进行战斗演示和结算。

单击返回，则返回到界面2.

4.送出精灵界面

尊敬的用户：

非常遗憾！由于您输了决斗赛，请您选择一个精灵送出。但是请不要灰心，您可以通过升级赛不断磨砺您的精灵提升战力，欢迎再战！



	名称	等级
1	蘑菇怪	1
2	谱尼	1
3	布布花	1

确认送出

在决斗赛中失败之后，会跳转到送出精灵界面。列表中显示系统从用户的精灵列表中随机抽取处的三个精灵(如果不足三个则全部抽取)的名称和等级。单击某一行可以在左边的方框中显示该精灵的图片。单击确认送出，则会将目前选择的精灵从用户的精灵列表中移除，并回到界面2。

如果用户送出精灵之后，目前已经没有精灵，系统则会随机分发给用户一个初始状态的精灵(1级)。

FightManager窗体



两只精灵会按照自身的敏捷度确定攻击间隔，然后自动进行攻击。有可能两只精灵同时发动攻击，系统会在下方的信息栏中进行提示。

下方的列表显示了战斗的详细信息。基本格式为

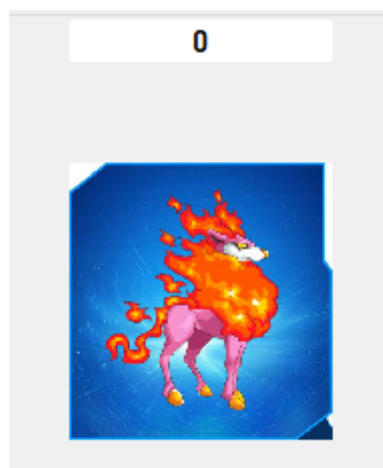
pet_name 发动攻击 skill_name 攻击类型 是否暴击

pet_name 受到攻击 skill_name 伤害值 是否属性克制 是否闪避

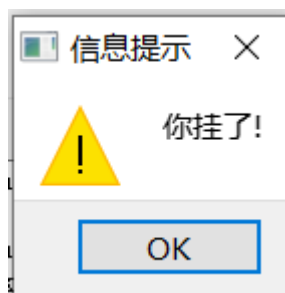
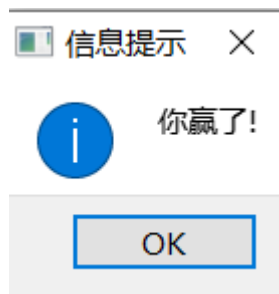
当精灵受到伤害的时候，上方的精灵图片会发生抖动，血量条会减少。

其中一只精灵死亡后，战斗结束

谱尼	发动攻击	落芳天华	闪避增益	暴击:YES	
谱尼	受到攻击	normalAttack	伤害:50	属性克制:NO	躲避:NO
里奥斯	受到攻击	落芳天华	伤害:40	属性克制:NO	躲避:NO
里奥斯	死亡				



战斗胜利或者失败之后，会弹出相应的提示框。



如果战斗失败，则会返回到主窗口的页面3；如果战斗胜利，则会留在战斗窗口，玩家可以单击重新开始，开始一局新的战斗。

问题与解决

1

编写自定义类，使用自定义slot函数的时候，系统报错。如果要定义自己的槽函数，必须要定义类的时候，加上Q_OBJECT宏定义，问题不再出现。

```
1 class XXX
2 {
3     Q_OBJECT
```

2

server的online_user是被所有thread共享的。线程在访问online_user的时候，必须要实现线程同步。

- 当有thread正在读取online_user的时候，不能有thread对online_user进行写操作。
- 当有thread正在写入online_user的时候，不能有其他thread写入online_user，同时不能有thread读取online_user。

程序中使用QSemaphore 实现此功能。

```
1 int readcount=0;
2 QSemaphore mutex(1),wrt(1);
```

```
1 wrt.acquire(); // 同一时间只能有一个thread修改user (server_m 中的QStringList
online_user , 被所有的thread共享)
2 user->append(myname);
3 wrt.release();
```


wrt的权限仅有一个。

当有一个thread使用了wrt的权限之后，其他想写入online_user的thread就只能挂起，等待wrt的权限被释放。保证了同一时间只有一个thread写入online_user。

```
1 //读取在线用户列表
2     mutex.acquire();           //互斥访问
3     readcount++;
4     if(readcount == 1)
5         wrt.acquire();
6     mutex.release();
7
8     QStringList ulist=*user;    //读操作
9     sendData+="#";
10    for(int i=0;i<ulist.size();i++)
11    {
12        sendData+=ulist[i];
13        sendData+="*";
14    }
15    mutex.acquire();           //读者减一
16    readcount--;
17    if(readcount==0)           //若此为最后一个读者，则释放写信号量
18        wrt.release();
19    mutex.release();
```

readcount 记录了正在读取online_user的thread的个数。

mutex的作用不是保证同一时间只有一个thread读取online_user, 而是保证第一个读取online_user的用户能够拿到wrt权限，同时最后一个reader释放掉wrt的权限。也就是，当有reader的时候，不能写入online_user；所有的reader都退出的时候，可以有thread写入online_user。

同时，如果此时有writer正在写入online_user, 那么想要读取online_user的thread就会在 wrt.acquire()处被挂起，直到wrt的权限被释放。保证了当有thread正在写入online_user的时候不能有thread读取online_user。

注：数据库User也是被所有的thread以及server共享的，但是在需要更新User的时候，没有使用互斥锁，因为每个thread都代表着不同的用户，操作的肯定是数据库User中的不同部分，不存在资源访问冲突的情况。

3

阻塞模式下，在I/O操作完成前，执行的操作函数一直等候而不会立即返回，该函数所在的线程会阻塞在这里。相反，在非阻塞模式下，套接字函数会立即返回，而不管I/O是否完成，该函数所在的线程会继续运行。

程序中实现的是多线程并行运行，每个线程单socket通信，所以适合阻塞式socket通信。而Qt中的QTcpSocket实现的是非阻塞式socket通信。为此，使用waitForBytesWritten函数实现阻塞式socket通信。该函数会等待socket将信息完全写入缓冲区之后，再向下继续运行。函数的默认等待时间是30s, 也可以手动设置。

测试程序如下

```

1  QTcpSocket tmpSock;
2  char* buf = "hello";
3  tcpSock.connectToHost("59.64.159.87", 7716);
4  tcpSock.write(buf, strlen(buf)+1);
5  sleep(3000);
6  tcpSock.disconnect();

```

在执行write函数之后，进程立即进行休眠，导致线程未能向缓冲区写入数据。

在write函数之后加入waitForBytesWritten函数，线程会向缓冲区写完数据之后再进行休眠。

4

在应用中插入图片的方式：

```

1  QPixmap pix_big(":/pictures/img/bg.jpeg");
2  int w = ui->label_pic_big->width();
3  int h = ui->label_pic_big->height();
4  ui->label_pic_big->setPixmap(pix_big.scaled(w,h));
5  QPixmap pix_small(":/pictures/img/white.jpeg");
6  w = ui->label_pic_big->width();
7  h = ui->label_pic_big->height();
8  ui->label_pic_small->setPixmap(pix_small.scaled(w,h));

```

界面上设置固定大小的QLabel, 然后向QLabel中插入图片。

5

Qt中的QString类型字符串提供了非常方便的字符串操作方式。

QString类型可以和QString类型的字符串或者const char * 拼接。

可以使用.split() 方法将字符串按照关键字进行分割，返回QStringList. split中可以设置分割的关键字，程序中使用 | 和 * 作为分割的关键字。

附录A:基础部分程序源代码

实验1

```

1  #include <stdio>
2  #include <iostream>
3
4  // 矩阵的行数和列数
5  const int row_len = 4;
6  const int col_len = 5;
7
8  void get_memory(int **&mat){    // 申请内存空间
9      mat = new int*[row_len] ;
10     for(int i=0 ; i<row_len ; i++){
11         mat[i] = new int [col_len] ;

```

```

12     }
13 }
14
15 void release_memory(int **&mat){    // 释放内存空间
16     for(int i=0 ; i<row_len ; i++){
17         delete mat[i] ;
18     }
19     delete mat ;
20 }
21
22 void read_in_mat(int **&mat)
23 {
24
25     for (int i = 0; i < row_len; i++)
26     {
27         for (int j = 0; j < col_len; j++)
28         {
29             std::cin >> mat[i][j];
30         }
31     }
32     std::cout<<"\n" ;
33     return;
34 }
35
36 void print_mat(int **&mat)
37 {
38     for (int i = 0; i < row_len; i++)
39     {
40         for (int j = 0; j < col_len; j++)
41         {
42             std::cout << mat[i][j] << " ";
43         }
44         std::cout << "\n";
45     }
46     std::cout<<"\n" ;
47     return;
48 }
49
50 void mat_add(int **&mat1, int **&mat2, int **&mat3)
51 {
52     for (int i = 0; i < row_len; i++)
53     {
54         for (int j = 0; j < col_len; j++)
55         {
56             mat1[i][j] = mat2[i][j] + mat3[i][j];
57         }
58     }
59     return;
60 }
61
62 void mat_sub(int **&mat1, int **&mat2, int **&mat3)
63 {
64     for (int i = 0; i < row_len; i++)
65     {
66         for (int j = 0; j < col_len; j++)
67         {
68             mat1[i][j] = mat2[i][j] - mat3[i][j];
69         }

```

```

70     }
71     return;
72 }
73
74
75
76 int main()
77 {
78     int **A1, **A2, **A3 ;
79     get_memory(A1) ;
80     get_memory(A2) ;
81     get_memory(A3) ;
82
83     read_in_mat(A1) ;
84     read_in_mat(A2) ;
85
86     mat_add(A3, A1, A2) ;
87     print_mat(A3) ;
88
89     mat_sub(A3, A1, A2) ;
90     print_mat(A3) ;
91
92     release_memory(A1) ;
93     release_memory(A2) ;
94     release_memory(A3) ;
95
96     print_mat(A1) ;
97     return 0;
98 }

```

实验2.1

```

1  #include<iostream>
2  #include<cstdio>
3  #include<cmath>
4
5  class Point
6  {
7  private:
8      /* data */
9      double x, y ;
10
11  public:
12      Point(const double& x=0, const double& y=0)
13          : x(x), y(y)
14      {
15          std::cout<<"Creating a Point...\n" ;
16      }
17      Point (const Point& obj){
18          *this = obj ;
19          std::cout<<"Copying a Point...\n" ;
20      }
21      ~Point(){
22          std::cout<<"Deleting a Point...\n" ;
23      }
24

```

```

25     double get_x() const {return x;}
26     double get_y() const {return y;}
27
28     Point& operator = (const Point& p) {
29         x = p.x ;
30         y = p.y ;
31         return *this ;
32     }
33 };
34
35 double dist(const Point& A, const Point& B) {
36     // 返回A和B的euclid距离
37     double dist_x = A.get_x() - B.get_x() ;
38     double dist_y = A.get_y() - B.get_y() ;
39     return sqrt(dist_x*dist_x + dist_y*dist_y) ;
40 }
41
42 class Circle
43 {
44 private:
45     /* data */
46     Point c ;
47     double r ;
48 public:
49     Circle(const Point& c, const double& r=0)
50     : c(c), r(r)
51     {
52         std::cout<<"Creating a Circle...\n" ;
53     }
54     Circle(const Circle& obj){
55         *this = obj ;
56         std::cout<<"Copying a Circle...\n" ;
57     }
58     ~Circle(){
59         std::cout<<"Deleting a Circle...\n" ;
60     }
61     Point get_c() const {return c ; }
62     double get_r() const {return r ; }
63 };
64
65 bool is_intersect(const Circle& o1, const Circle& o2){
66     std::cout<<"\n" ;
67     return o1.get_r() + o2.get_r() > dist(o1.get_c(), o2.get_c()) ;
68 }
69
70 int main(){
71     Point A(1,0), B(0,1) ;
72     std::cout<<"\n" ;
73     std::cout<<dist(A,B)<<"\n";
74     std::cout<<"\n" ;
75     Circle C1(Point(1,0), 2), C2(Point(4,0), 1.2);
76     std::cout<<"\n" ;
77     std::cout<<is_intersect(C1, C2)<<"\n" ;
78     std::cout<<"\n" ;
79     C1.get_c() ;
80     std::cout<<"\n" ;
81     return 0 ;
82 }

```

实验2.2

```
1  #include<iostream>
2  #include<cstdio>
3  #include<iomanip>
4
5  class Mat
6  {
7  private:
8      /* data */
9      int lines, rows ;
10     int **mat ;
11 public:
12     Mat(const int& lines, const int& rows)
13     : lines(lines), rows(rows)
14     {
15         mat = new int* [lines] ;
16         for(int i=0 ; i<lines ; i++){
17             mat[i] = new int [rows] ;
18         }
19         std::cout<<"Creating a Mat. Apply for memory...\n" ;
20     }
21     Mat (const Mat& obj){
22         lines = obj.lines ;
23         rows = obj.rows ;
24         mat = new int* [lines] ;
25         for(int i=0 ; i<lines ; i++){
26             mat[i] = new int [rows] ;
27         }
28         for(int i=0 ; i<lines ; i++ ){
29             for(int j=0 ; j<rows ; j++ )
30                 mat[i][j] = obj.mat[i][j] ;
31         }
32         std::cout<<"Copying a Mat. Apply for memory...\n" ;
33     }
34     ~Mat() {
35         for(int i=0 ; i<lines ; i++){
36             delete mat[i] ;
37         }
38         delete mat ;
39         std::cout<<"Deleting a Mat. Release memory...\n" ;
40     }
41
42     int get_lines() const {return lines; }
43     int get_rows() const {return rows; }
44
45     void read_in(){
46         for(int i=0 ; i<lines ; i++ ){
47             for(int j=0 ; j<rows ; j++ )
48                 std::cin>>mat[i][j] ;
49         }
50         std::cout<<"\n";
51     }
52
53     void print_out() const {
```

```

54         for(int i=0 ; i<lines ; i++ ){
55             for(int j=0 ; j<rows ; j++ ){
56                 std::cout<<std::setw(10)<<mat[i][j]<<" " ;
57             }
58             std::cout<<"\n" ;
59         }
60         std::cout<<"\n" ;
61     }
62
63     Mat
64     operator + (const Mat& A) const {
65         Mat re(lines, rows) ;
66         for(int i=0 ; i<lines ; i++ ) {
67             for(int j=0 ; j<rows ; j++ ) {
68                 re.mat[i][j] = mat[i][j] + A.mat[i][j] ;
69             }
70         }
71         return re ;
72     }
73
74     Mat
75     operator - (const Mat& A) const {
76         Mat re(lines, rows) ;
77         for(int i=0 ; i<lines ; i++ ) {
78             for(int j=0 ; j<rows ; j++ ) {
79                 re.mat[i][j] = mat[i][j] - A.mat[i][j] ;
80             }
81         }
82         return re ;
83     }
84
85     Mat&
86     operator = (const Mat& A){
87         for(int i=0 ; i<lines ; i++ ) {
88             for(int j=0 ; j<rows ; j++ ) {
89                 mat[i][j] = A.mat[i][j] ;
90             }
91         }
92     }
93 };
94
95 const int lines=3, rows=3 ;
96
97 int main(){
98
99     Mat A1(lines,rows), A2(lines,rows), A3(lines,rows) ;
100     A1.read_in() ;
101     A2.read_in() ;
102
103     A3 = A1 + A2 ;
104     A3.print_out() ;
105     A3 = A1 - A2 ;
106     A3.print_out() ;
107
108     Mat *pA1, *pA2, *pA3 ;
109     pA1 = new Mat(lines,rows) ;
110     pA2 = new Mat(lines,rows) ;
111     pA3 = new Mat(lines,rows) ;

```

```

112
113     pA1->read_in() ;
114     pA2->read_in() ;
115
116     *pA3 = *pA1 + *pA2 ;
117     pA3->print_out() ;
118     *pA3 = *pA1 - *pA2 ;
119     pA3->print_out() ;
120
121     delete pA1 ;
122     delete pA2 ;
123     delete pA3 ;
124     return 0 ;
125 }

```

实验3

```

1  #include <iostream>
2  #include <cmath>
3
4  class Shape
5  {
6  public:
7      Shape()
8      {
9          std::cout << "Creating a Shape...\n";
10     }
11     Shape(const Shape &obj)
12     {
13         *this = obj;
14         std::cout << "Copying a Shape...\n";
15     }
16     ~Shape()
17     {
18         std::cout << "Deleting a Shape...\n";
19     }
20     virtual double getArea() const
21     {
22         std::cout << "Calling shape getarea...\n";
23         return 0 ;
24     }
25 };
26
27 class Rectangle
28     : public Shape
29 {
30     protected: // 可以被子类继承
31         double a, b;
32
33     public:
34         Rectangle(const double &a, const double &b)
35             : a(a), b(b)
36         {
37             std::cout << "Creating a Rectangle...\n";
38         }
39         Rectangle(const Rectangle &obj)

```



```

40     {
41         *this = obj;
42         std::cout << "Copying a Rectangle...\n";
43     }
44     ~Rectangle()
45     {
46         std::cout << "Deleting a Rectangle...\n";
47     }
48
49     virtual double getArea() const
50     {
51         std::cout << "Calling rectangle getarea...\n";
52         return a * b;
53     }
54 };
55
56 const double Pi = 3.1415926;
57
58 class Circle
59     : public Shape
60 {
61 private:
62     double r;
63
64 public:
65     Circle(const double &r)
66         : r(r)
67     {
68         std::cout << "Creating a Circle...\n";
69     }
70     Circle(const Circle &obj)
71     {
72         *this = obj;
73         std::cout << "Copying a Circle...\n";
74     }
75     ~Circle()
76     {
77         std::cout << "Deleting a Circle...\n";
78     }
79
80     virtual double getArea() const
81     {
82         std::cout << "Calling circle getarea...\n";
83         return Pi * r * r;
84     }
85 };
86
87 class Square
88     : public Rectangle
89 {
90 private:
91     double x;
92
93 public:
94     Square(const double &x)
95         : Rectangle(x, x), x(x)
96     {
97         std::cout << "Creating a Square...\n";

```

```

98     }
99     Square(const Square &obj)
100         : Rectangle(x, x)
101     {
102         *this = obj;
103         std::cout << "Copying a Sqaure...\n";
104     }
105     ~Square()
106     {
107         std::cout << "Deleting a Square...\n";
108     }
109 };
110
111 int main()
112 {
113     Square s1(3);
114     std::cout << s1.getArea() << "\n\n";
115     Rectangle r1(3, 4);
116     std::cout << r1.getArea() << "\n\n";
117     Circle c1(2);
118     std::cout << c1.getArea() << "\n\n";
119     Shape *s2 = &r1;
120     std::cout << s2->getArea() << "\n\n";
121     return 0;
122 }
123

```

实验4

```

1  #include<iostream>
2  #include<cstdlib>
3  #include<ctime>
4  int main(){
5      srand((unsigned) time(0)) ;
6      int price = 1+rand()%1000 ;
7      std::string input ;
8      int price_input ;
9      int guess_cnt=10 ;
10     // 总共10次有效的猜数的机会
11     // 一个理智的player 应该能在10次的时候猜出答案
12     while(guess_cnt--){
13         std::cout<<"Please input an integer within [1,1000]: \n" ;
14         std::cin>>input ;
15         price_input = atoi(input.c_str()) ;
16         if( ! (price_input>=1 && price_input<=1000)){
17             std::cout<<"Wrong input!!\n" ;
18             guess_cnt++ ;
19         }
20         else {
21             if(price_input > price){
22                 std::cout<<"Too big.\n" ;
23             }
24             else if (price_input < price){
25                 std::cout<<"Too small.\n" ;
26             }
27             else {

```

```

28         std::cout<<"Bingo! You get the right answer.\n" ;
29         return 0 ;
30     }
31 }
32 }
33 std::cout<<"You've used up all your chances!\n" ;
34 return 0 ;
35 }

```

实验5.1

```

1  #include <iostream>
2  #include <cmath>
3
4  class Shape
5  {
6  public:
7      Shape()
8      {
9          std::cout << "Creating a Shape...\n";
10     }
11     Shape(const Shape &obj)
12     {
13         *this = obj;
14         std::cout << "Copying a Shape...\n";
15     }
16     ~Shape()
17     {
18         std::cout << "Deleting a Shape...\n";
19     }
20     virtual double getArea() const =0 ;
21     // virtual double getArea() const
22     // {
23     //     std::cout << "Calling shape getarea...\n";
24     //     return 0 ;
25     // }
26 };
27
28 class Rectangle
29     : public Shape
30 {
31     protected: // 可以被子类继承
32         double a, b;
33
34     public:
35         Rectangle(const double &a, const double &b)
36             : a(a), b(b)
37         {
38             std::cout << "Creating a Rectangle...\n";
39         }
40         Rectangle(const Rectangle &obj)
41         {
42             *this = obj;
43             std::cout << "Copying a Rectangle...\n";
44         }
45         ~Rectangle()

```

```

46     {
47         std::cout << "Deleting a Rectangle...\n";
48     }
49
50     virtual double getArea() const
51     {
52         std::cout << "Calling rectangle getarea...\n";
53         return a * b;
54     }
55 };
56
57 const double Pi = 3.1415926;
58
59 class Circle
60     : public Shape
61 {
62 private:
63     double r;
64
65 public:
66     Circle(const double &r)
67         : r(r)
68     {
69         std::cout << "Creating a Circle...\n";
70     }
71     Circle(const Circle &obj)
72     {
73         *this = obj;
74         std::cout << "Copying a Circle...\n";
75     }
76     ~Circle()
77     {
78         std::cout << "Deleting a Circle...\n";
79     }
80
81     virtual double getArea() const
82     {
83         std::cout << "Calling circle getarea...\n";
84         return Pi * r * r;
85     }
86 };
87
88 class Square
89     : public Rectangle
90 {
91 private:
92     double x;
93
94 public:
95     Square(const double &x)
96         : Rectangle(x, x), x(x)
97     {
98         std::cout << "Creating a Square...\n";
99     }
100     Square(const Square &obj)
101         : Rectangle(x, x)
102     {
103         *this = obj;

```

```

104         std::cout << "Copying a Sqaure...\n";
105     }
106     ~Square()
107     {
108         std::cout << "Deleting a Square...\n";
109     }
110 };
111
112 int main()
113 {
114     Square s1(3);
115     std::cout << s1.getArea() << "\n\n";
116     Rectangle r1(3, 4);
117     std::cout << r1.getArea() << "\n\n";
118     Circle c1(2);
119     std::cout << c1.getArea() << "\n\n";
120     Shape *s2 = &r1;
121     std::cout << s2->getArea() << "\n\n";
122     return 0;
123 }
124
125

```

实验5.2

```

1  #include<iostream>
2
3  class Point
4  {
5  private:
6      /* data */
7      double x, y ;
8  public:
9      Point(const double& x, const double& y )
10         : x(x), y(y)
11     {
12         std::cout<<"Creating a Point...\n" ;
13     }
14     Point(const Point& obj){
15         *this = obj ;
16         std::cout<<"Copying a Point...\n" ;
17     }
18     ~Point()
19     {
20         std::cout<<"Deleting a Point...\n" ;
21     }
22     void print_Point() {
23         std::cout<<"("<<x<<","<<y<<")\n" ;
24     }
25     Point& operator ++ () {
26         x+=1 ;
27         y+=1 ;
28         return *this ;
29     }
30
31     Point& operator -- () {

```

```

32         x-=1 ;
33         y-=1 ;
34         return *this ;
35     }
36     // 重载后置++, -- 需要额外的int占位参数
37     // 为了绕过语法限制
38     Point operator ++ (int) {
39         Point re = *this ;
40         x+=1 ;
41         y+=1 ;
42         return re ;
43     }
44
45     Point operator -- (int) {
46         Point re = *this ;
47         x-=1 ;
48         y-=1 ;
49         return re ;
50     }
51 };
52
53 int main(){
54     Point p(0,0) ;
55     // Point *ptr ;
56     // ptr = &p ;
57     // std::cout<<ptr<<std::endl ;
58     // ptr = &(++p) ;
59     // std::cout<<ptr<<std::endl ;
60     // ptr = &(p++) ;
61     // std::cout<<ptr<<std::endl ;
62
63     (p++).print_Point() ;
64     (p--).print_Point() ;
65     (++p).print_Point() ;
66     (--p).print_Point() ;
67
68     return 0 ;
69 }

```