# 面向对象程序设计实践（C++）实验报告

## C++ 基础知识实验

### 题目要求

编写C++程序完成“矩阵”以下功能：假定矩阵大小为4×5（整型）

1. 矩阵空间采用new动态申请，保存在指针中
2. 定义矩阵初始化函数，可以从cin中输入矩阵元素
3. 定义矩阵输出函数，将矩阵格式化输出到cout
4. 定义矩阵相加的函数，实现两个矩阵相加的功能，结果保存在另一个矩阵中
5. 定义矩阵相减的函数，实现两个矩阵相减的功能，结果保存在另一个矩阵中
6. 动态申请三个矩阵：A1、A2、A3
7. 初始化A1、A2
8. 计算并输出A3 = A1加A2，A3 = A1减A2
9. 释放矩阵空间

代码

Matrix.hpp

/\*

Author: Sheldon Zhang

Description: A simple support for matrix

\*/

#include <vector>

#include <iostream>

#include "cmath"

#include "complex.h"

using namespace std;

#define ROW\_MAX 1000

#define COL\_MAX 1000

template <typename T>

class matrix;

template <typename T>

istream & operator >> (istream &is, matrix<T> & obj);

template <typename T>

ostream & operator << (ostream &os, matrix<T> const & obj);

/\*

Class Description: a class provides fundamental matrix storage and calculation

supported by vector in STL

\*/

template <typename T>

class matrix

{

private:

friend istream & operator >> <>(istream &is, matrix<T> & obj);

friend ostream & operator << <>(ostream &os, matrix<T> const & obj);

private:

int row, col;

vector<vector<T>> data;

private:

void zero();

public:

matrix();

matrix(int inputRow, int inputCol);

matrix(const matrix<T> &obj);

matrix(const matrix<T> \*obj);

~matrix();

matrix<T> operator + (const matrix<T> &rhs);

matrix<T> operator - (const matrix<T> &rhs);

matrix<T> operator \* (const matrix<T> &rhs);

matrix<T> & operator = (const matrix<T> &rhs);

void operator \*= (const matrix<T> &rhs);

void operator += (const matrix<T> &rhs);

void operator -= (const matrix<T> &rhs);

void resize(int row, int col);

};

/\*----------------------------------------------------\*/

//IMPLEMENT

/\*----------------------------------------------------\*/

template <typename T>

/\*

function name: istream & operator >>

description: overload operator >> (stream input)

arguments:

istream &is: input stream object from which characters are extracted

matrix<T> &obj: a referance of an object of class matrix

return value: the istream object(is)

\*/

istream & operator >> (istream &is, matrix<T> &obj)

{

for(int i = 0; i < obj.row; i++)

for(int j = 0; j < obj.col; j++)

is >> obj.data[i][j];

return is;

}

template <typename T>

/\*

function name: ostream & operator <<

description: overload operator <<

arguments:

ostream &os: output stream object where characters are inserted

matrix<T> const &obj: a referance of an object of class matrix

return value: the ostream object(os)

output format:

[

[ , ... , ]

[ , ... , ]

]

\*/

ostream & operator << (ostream &os, matrix<T> const &obj)

{

os << "[" << endl;

for(int i = 0; i < obj.row; i++)

{

os << "[ ";

for(int j = 0; j < obj.col; j++)

os << obj.data[i][j] << " ";

os << " ]" <<endl;

}

os << "]";

return os;

}

template <typename T>

/\*

function name: matrix

description: a constructor of class matrix

1.set varible row and col to 0

2.resize data to a 0 by 0 matrix

arguments: None

return value: None

\*/

matrix<T>::matrix()

{

row = 4; col = 5;

data.resize(4, vector<T>(5));

}

template <typename T>

/\*

function name: matrix

description: a constructor of class matrix

1. check if the arguments are legal

2. record the values of arguments

3. allocate space for the matrix of the wanted dimention

4. set all elements to 0

arguments:

inputRow: the wanted row of the matrix

inputCol: the wanted column of the matrix

return value: None

\*/

matrix<T>::matrix(int inputRow, int inputCol)

{

if(inputRow <= 0 || inputCol <= 0)

std::cerr << "Invalid Parameter" << '\n';

if(inputRow > ROW\_MAX || inputCol > COL\_MAX)

std::cerr << "matrix is too large" << '\n';

row = inputRow;

col = inputCol;

data.resize(row, vector<T>(col));

zero();

}

template <typename T>

/\*

function name: matrix

description: a copy constructor of class matrix

arguments: a referance of an object of class matrix

return value: None

\*/

matrix<T>::matrix(const matrix<T> &obj)

{

col = obj.col;

row = obj.row;

data.resize(row, vector<T>(col));

for(int i = 0; i < row; i++)

for(int j = 0; j < col; j++)

data[i][j] = obj.data[i][j];

}

template <typename T>

/\*

function name: matrix

description: to create a copy of the matrix which the pointer points at

arguments: a pointer of class matrix

return value: None

\*/

matrix<T>::matrix(const matrix<T> \*obj)

{

col = obj->col;

row = obj->row;

data.resize(row, vector<T>(col));

for(int i = 0; i < row; i++)

for(int j = 0; j < col; j++)

data[i][j] = obj->data[i][j];

}

template <typename T>

/\*

function name: ~matrix

description: a destructor of class matrix

1. set varible row and col to 0

2. delete all elements in data

arguments: None

return value: None

\*/

matrix<T>::~matrix()

{

row = 0;

col = 0;

data.clear();

}

template <typename T>

/\*

function name: zero

description: set all elements in data to 0

arguments: None

return value: None

\*/

void matrix<T>::zero()

{

for(int i = 0; i < row; i++)

for(int j = 0; j < col ;j++)

data[i][j] = 0;

}

template <typename T>

/\*

function name: matrix<T>::operator +

description: overload operator +

1. check if the dimension of two matrices matches

2. add the elements in the same localtion

(add this->data[i][j] and rhs.data[i][j])

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: an object of class matrix

\*/

matrix<T> matrix<T>::operator + (const matrix<T> &rhs)

{

if(row != rhs.row || col != rhs.col)

std::cerr << "the dimension is not the same" << '\n';

matrix<T> matrixTemp(rhs.data.size(), rhs.data[0].size());

for(int i = 0; i < rhs.data.size(); i++)

for(int j = 0; j < rhs.data[0].size(); j++)

matrixTemp.data[i][j] = this->data[i][j] + rhs.data[i][j];

return matrixTemp;

}

template <typename T>

/\*

function name: matrix<T>::operator -

description: overload operator -

1. check if the dimension of two matrices matches

2. subtruct this->data[i][j] by rhs.data[i][j]

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: an object of class matrix

\*/

matrix<T> matrix<T>::operator - (const matrix<T> &rhs)

{

if(row != rhs.row || col != rhs.col)

std::cerr << "the dimension is not the same" << '\n';

matrix<T> matrixTemp(rhs);

for(int i = 0; i < rhs.data.size(); i++)

for(int j = 0; j < rhs.data[0].size(); j++)

matrixTemp.data[i][j] = this->data[i][j] - rhs.data[i][j];

return matrixTemp;

}

template <typename T>

/\*

function name: matrix<T>::operator \*

description: overload operator \*

1. check if the dimension of two matrices matches

2. do the multiple operation

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: an object of class matrix

\*/

matrix<T> matrix<T>::operator \* (const matrix<T> &rhs)

{

if(this->data[0].size() != rhs.data.size())

std::cerr << "matrix A can not be multiplied by matrix B" << '\n';

matrix<T> matrixTemp(this->row, rhs.col);

for(int i = 0; i < this->row; i++)

for(int j = 0; j < this->col; j++)

for(int k = 0; k < rhs.col; k++)

matrixTemp.data[i][k] += this->data[i][j] \* rhs.data[j][k];

return matrixTemp;

}

template <typename T>

/\*

function name: matrix<T>::operator =

description: overload operator =

change all elements of current object to those in rhs

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: an object which pointer this points at of class matrix

\*/

matrix<T>& matrix<T>::operator = (const matrix<T> &rhs)

{

this->row = rhs.row;

this->col = rhs.col;

this->data.resize(row, vector<T>(col));

this->data = rhs.data;

return \*this;

}

template <typename T>

/\*

function name: matrix<T>::operator +=

description: overload operator +=

1. check if the dimension of two matrices matches

2. change this->data[i][j] to this->data[i][j] + rhs.data[i][j]

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: None

\*/

void matrix<T>::operator += (const matrix<T> &rhs)

{

if(row != rhs.row || col != rhs.col)

std::cerr << "the dimension is not the same" << '\n';

for(int i = 0; i < this->row; i++)

for(int j = 0; j < this->col; j++)

this->data[i][j] = this->data[i][j] + rhs.data[i][j];

}

template <typename T>

/\*

function name: matrix<T>::operator -=

description: overload operator -=

1. check if the dimension of two matrices matches

2. change this->data[i][j] to this->data[i][j] - rhs.data[i][j]

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: None

\*/

void matrix<T>::operator -= (const matrix<T> &rhs)

{

if(row != rhs.row || col != rhs.col)

std::cerr << "the dimension is not the same" << '\n';

for(int i = 0; i < this->row; i++)

for(int j = 0; j < this->col; j++)

this->data[i][j] = this->data[i][j] - rhs.data[i][j];

}

template <typename T>

/\*

function name: matrix<T>::operator \*=

description: overload operator \*=

1. check if the dimension of two matrices matches

2. create a temporary object the same as the current one

3. change the dimension of current object to the answer's demention and set

all elements in data to 0

4. do the multiple operation

arguments: matrix<T> &rhs: a referance of an object of class matrix

return value: None

\*/

void matrix<T>::operator \*= (const matrix<T> &rhs)

{

if(this->data[0].size() != rhs.data.size())

std::cerr << "matrix A can not be multiplied by matrix B" << '\n';

matrix<T> matrixTemp(this);

this->resize(this->row, rhs.col);

for(int i = 0; i < matrixTemp.row; i++)

for(int j = 0; j < matrixTemp.col; j++)

for(int k = 0; k < rhs.col; k++)

this->data[i][k] += matrixTemp.data[i][j] \* rhs.data[j][k];

}

template <typename T>

/\*

function name: matrix<T>::resize

description:

1. change the dimension of current object

2. set all elements in data to 0

arguments:

inputRow: the wanted row of the matrix

inputCol: the wanted column of the matrix

return value: None

\*/

void matrix<T>::resize(int inputRow, int inputCol)

{

row = inputRow; col = inputCol;

data.resize(row, vector<T>(col));

zero();

}

matrix.cpp

#include "matrix.hpp"

#include <iostream>

#include <algorithm>

using namespace std;

int main(int argc, char \*argv[])

{

matrix<int> \*mat1 = new matrix<int>(2, 3); // 创建矩阵 mat1

matrix<int> \*mat2 = new matrix<int>(2, 3); // 创建矩阵 mat2

cin >> \*mat1 >> \*mat2; // 通过 cin 输入元素

cout << \*mat1 << \*mat2; // 通过 cout 输出

// cout << \*mat1 << endl << \*mat2 << endl;

cout << \*mat1 + \*mat2 << endl; // 支持加减运算

cout << \*mat1 - \*mat2 << endl;

matrix<int> \*mat3 = new matrix<int>(\*mat1 + \*mat2); // 通过矩阵的和构造新矩阵

cout << \*mat3 << endl;

delete mat1;

delete mat2;

delete mat3;

return 0;

}

感悟与反思

这次试验我实现了一个功能较为完备的矩阵运算库，包括基础的加减运算，元素乘法和点乘，虽然功能较为简单，但是我在其中学到了模板的使用方法并进行了大量的练习。同时对含有class的代码有了debug的经验。通过这次实验我窥见了面向对象编程的方便和先进之处，主要在于对数据和逻辑的封装。面向对象，则在基于对象的基础上增加了多态性。所谓多态，就是可以用统一的方法对不同的对象进行同样的操作。当然，这些对象不能完全不同，而需要有一些共性，只有存在了这些共性才可能用同样的方法去操作它们。

## 类与对象实验

### 圆形

#### 题目要求

编写C++程序完成“圆形”以下功能：

1. 定义一个 Point 类，其属性包括点的坐标，提供计算两点之间距离的方法；（要求当用户不输入坐标数据时，能够默认为坐标原点（0，0））
2. 定义一个圆形类，其属性包括圆心和半径；
3. 创建两个圆形对象，提示用户输入圆心坐标和半径，判断两个圆是否相交，并输出结果；
4. 观察圆形对象以及Point类成员的构造函数与析构函数的调用。（提示及要求：1，可通过在构造与析构函数中加入输出提示信息的方式观察相关调用；可以使用system("pause")进行程序的暂停；2，能够理解并说明每一次构造与析构函数调用是哪个对象的调用，并观察和解释相关调用顺序及其原因）

#### 代码

#include <iostream>

#include <algorithm>

#include <cstdio>

#include <cmath>

#include <cstdlib>

using namespace std;

class point {

private:

double x;

double y;

public:

// friend class rectangle;

point() {}

point(double \_x = 0, double \_y = 0) { // 默认的输入为0

x = \_x;

y = \_y;

}

~point() {}

double getDis(point rhs);

};

double point::getDis(point rhs) {

return sqrt((x - rhs.x) \* (x - rhs.x) + (y - rhs.y) \* (y - rhs.y));

}

class circle {

public:

point\* s;

double r;

circle(int \_x, int \_y, double \_r) {

s = new point(\_x, \_y);

r = \_r;

}

~circle() {}

};

bool foo(circle& lhs, circle& rhs) { // 判断两个圆是否相交

// cout << lhs.s->getDis(\*rhs.s) << endl << lhs.r + rhs.r << endl;

return lhs.s->getDis(\*rhs.s) < lhs.r + rhs.r;

}

int main(int argc, char \*argv[])

{

double r1, r2;

int x1, y1, x2, y2;

cin >> x1 >> y1 >> r1 >> x2 >> y2 >> r2; // 输入每个圆的圆心和半径

circle c1(x1, y1, r1);

circle c2(x2, y2, r2);

cout << foo(c1, c2) << endl;

return 0;

}

#### 感悟与反思

尝试了函数中参数的默认值和两个自定类型的交互。同时了解了有继承对象的构造和析构顺序，其中构造总是从基类开始，逐层构造到子类，析构时像出栈一样，先析构子类再析构基类。

在面向对象编程中，对象自身是内聚的，是保管好自己的数据，完成好自己的操作的，而对外界呈现出自己的状态和行为。但是没有绝对的自力更生，需要向外部开放接口同时和其他对象交互，既包括获知其他对象的状态，也包括仰赖其他对象的行为。明白了面向对象的思想的核心内容其实就是封装变化，利用技术手段比如接口和抽象类来分离系统中容易变化的部分和基本不变的部分。

### 矩阵

#### 题目要求

编写C++程序完成“矩阵”类以下功能：

1. 用类来实现矩阵，定义一个矩阵的类，属性包括：
   * 矩阵大小，用 lines, rows（行、列来表示）；
   * 存贮矩阵的数组指针，根据矩阵大小动态申请（new）。
2. 矩阵类的方法包括：
   * 构造函数，参数是矩阵大小，需要动态申请存贮矩阵的数组；
   * 析构函数，需要释放矩阵的数组指针；
   * 拷贝构造函数，需要申请和复制数组；
   * 输入，可以从cin中输入矩阵元素；
   * 输出，将矩阵格式化输出到cout；
   * 矩阵相加的函数，实现两个矩阵相加的功能，结果保存在另一个矩阵类，但必须矩阵大小相同；
   * 矩阵相减的函数，实现两个矩阵相减的功能，结果保存在另一个矩阵类，但必须矩阵大小相同。
3. 定义三个矩阵：A1、A2、A3；
4. 初始化A1、A2；
5. 计算并输出A3 = A1加A2，A3=A1减A2；（要求及提示：最好能实现对赋值操作符“=”的重载；注意检查“自赋值”、释放“旧元素”）
6. 用new动态创建三个矩阵类的对象：pA1、pA1、pA3；
7. 初始化pA1、pA2；
8. 计算并输出pA3=pA1加pA2，pA3=pA1减pA2；
9. 释放pA1、pA2、pA3。

#### 代码

matrix.h

#pragma once

#include <cstdio>

#include <algorithm>

#include <iostream>

template <class T>

class Matrix {

T\* data;

int row;

int col;

public:

Matrix();

Matrix(int \_row, int \_col);

Matrix(Matrix<T> &mat);

~Matrix();

T& operator() (int \_row, int \_col);

Matrix<T>& operator = (const Matrix& mat);

template <class elementType>

friend std::ostream& operator << (std::ostream& os, const Matrix<elementType>& mat) {

os << '[' << '\n';

for(int i = 0; i < mat.row; i++) {

os << '[';

for(int j = 0; j < mat.col; j++) {

os << mat.data[i \* mat.col + j];

if(j != mat.col - 1)

os << ',';

}

os << "]," << '\n';

}

os << ']' << '\n';

return os;

}

template <class elementType>

friend std::istream& operator >> (std::istream& is, const Matrix<elementType>& mat){

for(int i = 0; i < mat.row; i++)

for(int j = 0; j < mat.col; j++)

is >> mat.data[i \* mat.col + j];

}

template <class elementType>

friend Matrix<elementType> operator+ (Matrix<elementType>& matl, Matrix<elementType>& matr) {

if(matl.col == matr.col && matl.row == matr.row) {

Matrix<elementType> ans(matl);

for(int i = 0; i < ans.row \* ans.col; i++)

ans.data[i] += matr.data[i];

return ans;

}

else {

std::cerr << "error dimension [operator +]" << '\n';

}

}

template <class elementType>

friend Matrix<elementType> operator- (Matrix<elementType>& matl, Matrix<elementType>matr) {

if(matl.col == matr.col && matl.row == matr.row) {

Matrix<elementType> ans(matl);

for(int i = 0; i < ans.row \* ans.col; i++)

ans.data[i] -= matr.data[i];

return ans;

}

else {

std::cerr << "error dimension [operator +]" << '\n';

}

}

template <class elementType>

friend Matrix<elementType> operator\* (Matrix<elementType>& mat, const elementType tar) {

Matrix<elementType> ans(mat);

for(int i = 0; i < mat.row \* mat.col; i++)

ans.data[i] \*= tar;

return ans;

}

};

matrix.hpp

#include "matrix.h"

template <class T>

Matrix<T>::Matrix() {

row = 0;

col = 0;

data = nullptr;

}

template <class T>

Matrix<T>::Matrix(int \_row, int \_col) {

this -> row = \_row;

this -> col = \_col;

data = new T [row \* col];

}

template <class T>

Matrix<T>::~Matrix() {

this -> row = 0;

this -> col = 0;

delete []data;

}

template <class T>

Matrix<T>::Matrix(Matrix& mat) {

this -> col = mat.col;

this -> row = mat.row;

this -> data = new T[col \* row];

for(int i = 0; i < row \* col; i++)

data[i] = mat.data[i];

}

template <class T>

T& Matrix<T>::operator() (int \_row, int \_col) {

return data[\_row \* col + \_col];

}

template <class T>

Matrix<T>& Matrix<T>::operator = (const Matrix& mat) {

if(\*this == mat)

return \*this;

col = mat.col;

row = mat.row;

delete []data;

data = new T[col \* row];

for(int i = 0; i < col \* row; i++)

data[i] = mat.data[i];

return \*this;

}

unit\_test.cpp

#include "matrix.h"

#include "matrix.hpp"

#include <iostream>

using namespace std;

int main(int argc, char \*argv[])

{

Matrix<int> \*mat1 = new Matrix<int>(3, 4);

Matrix<int> \*mat2 = new Matrix<int>(3, 4);

cin >> \*mat1;

cin >> \*mat2;

cout << "load" << endl;

cout << \*mat1 << \*mat2 << endl;

cout << \*mat1 + \*mat2 << endl;

cout << \*mat1 - \*mat2 << endl;

cout << \*mat1 \* 2 << endl;

delete mat1;

delete mat2;

return 0;

}

#### 感悟与反思

题目要求和第一题的差别不大，我再上一次代码的基础上进行了精简。尝试了库+单元测试的开发形式，除去了之前冗余的部分，对代码进行了精简。同时了解到了 “using namespace std;” 带来的灾难性结果（使用 std 明明空间时会造成流的重载报错），让我对命名空间有了新的认识。这次试验也是对之前template的再一次实践，模板是C++中泛型编程的基础。一个模板就是一个创建类或函数的蓝图或公式，在使用模板时，我们提供足够的信息将模板转换为特定的类或函数。这种转换发生在编译时，即在编译时由编译器确认具体的类型信息，这样省去了大量的函数重载，加快了开发效率。

## 继承与派生实验

### 题目要求

编写C++程序完成“形状”的以下功能：

1. 声明一个基类Shape（形状），其中包含一个方法来计算面积；
2. 从Shape派生两个类：矩形类和圆形类；
3. 从矩形类派生正方形类；
4. 分别实现派生类构造函数、析构函数和其他方法；
5. 创建派生类的对象，观察构造函数、析构函数调用次序；（提示及要求：1，可通过在构造与析构函数中加入输出提示信息的方式观察相关调用；可以使用system("pause")进行程序的暂停；2，能够理解并说明每一次构造与析构函数调用是哪个对象的调用，并观察和解释相关调用顺序及其原因）
6. 对不同对象计算面积。

### 代码

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <algorithm>

#include <cstdio>

#include <cstring>

#include <cmath>

using namespace std;

class Shape { // 构造 Shape 类

public:

Shape() {std::cout << "class shape init" << std::endl;}

~Shape() {std::cout << "class shape destroyed" << std::endl;}

double getArea() { // 计算面积的方法

return 0;

}

};

class Circle : public Shape { // 从 Shape 派生 Circle 类

int r;

public:

Circle():Shape() {std::cout << "class circle init" << std::endl;} // 构造函数

Circle(int \_r):Shape() {r = \_r;std::cout << "class circle init" << std::endl;} // 构造函数

~Circle() {std::cout << "class circle destroyed" << std::endl;}

double getArea() { // 析构函数

return M\_PI \* r \* r;

}

};

class Rectangle: public Shape { // 从 Shape 类派生 Rectangle 类

int length, height;

public:

Rectangle():Shape() {std::cout << "class rectangle init" << std::endl;} // 构造函数

Rectangle(int \_length, int \_height):Shape() { // 构造函数

length = \_length;

height = \_height;

std::cout << "class rectangle init" << std::endl;

}

~Rectangle() {std::cout << "class rectangle destroyed" << std::endl;} // 析构函数

double getArea() {

return height \* length;

}

};

class Square :public Rectangle{ // 从 Rectangle 类派生 Square 类

public:

Square():Rectangle() {std::cout << "class square init" << std::endl;} // 构造函数

Square(int \_length): Rectangle(\_length, \_length) { // 构造函数

std::cout << "class square init" << std::endl;

}

~Square() {std::cout << "class square destroyed" << std::endl;} // 析构函数

};

int main(int argc, char \*argv[])

{

Circle circle(1);

Rectangle rec(1, 2);

Square square(4);

std::cout << circle.getArea() << " " << rec.getArea() << " " << square.getArea() << std::endl;

return 0;

}

### 感悟与反思

这道题目主要考察了子类的派生。其中构造和析构顺序的顺序是，构造总是从基类开始，逐层构造到子类，析构时像出栈一样，先析构子类再析构基类。静态对象的构造和析构顺序与编译器的链接顺序有关，可选的做法是通过一个全局的静态函数进行调用，保证使用前已经初始化，例子如下：

**static** Foo**&** getStaticFooObject(**void**) {

**static** Foo theStaticObject;

**return** theStaticObject;

}

## I/O 流实验

### 题目要求

编写C++程序完成猜价格游戏的以下功能：

1. 假定有一件商品，程序用随机数指定该商品的价格（1-1000的整数）；
2. 提示用户猜价格，并输入：若用户猜的价格比商品价格高或低，对用户作出相应的提示；
3. 直到猜对为止，并给出提示。（提示及要求：1，要求使用C++的输入输出方式（cin, cout），不能使用C语言的printf等；2，注意检查输入的合法性）

### 代码

#include <iostream>

#include <algorithm>

#include <cstdlib>

#include <cstdio>

#include <ctime>

using namespace std;

int main(int argc, char \*argv[])

{

srand((unsigned)time(NULL));

int ans = 1 + rand() % 999; // 生成随机数答案

int guess = -1; // 储存猜的数字

while (guess != ans)

{

cin >> guess; // 读入猜的值

//cout << guess << endl;

if (guess == 1001) // 直接看答案

{

cout << "the answer " << ans << endl;

return 0;

}

if (guess > 1000 || guess <= 0) // 判断非法输入

{

cout << "invalid input" << endl;

continue;

}

if (guess > ans)

cout << "it is larger than the answer" << endl;

else

cout << "it is smaller than the answer" << endl;

}

cout << "you got the answer" << ans << endl;

return 0;

}

### 感悟与反思

本题较为简单，相当于不断读取用户输入完成交互，可以读入int类型并判断范围进行非法输入判断。同时判断输入值和真实值的大小关系进行提示，保证在判断之前cin的变量被初始化过。

## 重载实验

### 虚函数

#### 题目要求

针对题目3的“形状”类，编写C++程序完成以下功能：

1. 将【形状】 中的基类计算面积的方法定义为虚函数，比较与【形状（A）】程序的差异；
2. 将【形状】中的基类定义为抽象类，比较与【形状（A）】程序的差异。

#### 代码

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <algorithm>

#include <cstdio>

#include <cmath>

using namespace std;

class Shape { // 含有 virtual f()=0 的类是抽象类

public:

Shape() {std::cout << "class shape init" << std::endl;}

~Shape() {std::cout << "class shape destroyed" << std::endl;}

virtual double getArea() = 0; // 使用 virtual 定义为虚函数

};

class Rectangle: public Shape {

private:

int x, y;

public:

Rectangle(int \_x, int \_y): Shape() {

x = \_x;

y = \_y;

std::cout << "class rectangle init" << std::endl;

}

~Rectangle() {std::cout << "class rectangle destroyed" << std::endl;}

double getArea() {

return x \* y;

}

};

class Circle: public Shape {

private:

int r;

public:

Circle(int \_r) {

r = \_r;

std::cout << "class circle init" << std::endl;

}

~Circle() {std::cout << "class circle destroyed" << std::endl;}

double getArea() {

return M\_PI \* r \* r;

}

};

int main() {

Circle\* circle = new Circle(1);

Rectangle \*rec = new Rectangle(2, 3);

std::cout << circle->getArea() << " " << rec->getArea() << std::endl;

return 0;

}

#### 感悟与反思

明确概念定义一个函数为虚函数，不代表函数为不被实现的函数。定义他为虚函数是为了允许用基类的指针来调用子类的这个函数。一个函数为纯虚函数，才代表函数没有被实现，为了实现一个接口，起到一个规范的作用，规范继承这个类的程序员必须实现这个函数。定义基类Shape的getArea方法为纯虚函数，所以基类Shape时抽象类。若直接实例化Shape会出现以下报错：

error C2259: 'CShape' : cannot instantiate abstract class due to following members:

warning C4259: 'void \_\_thiscall CShape::Show(void)' : pure virtual function was not defined

抽象类是一种特殊的类，它是为了抽象和设计的目的为建立的，它处于继承层次结构的较上层。抽象类只能作为基类来使用，其纯虚函数的实现由派生类给出。如果派生类中没有重新定义纯虚函数，而只是继承基类的纯虚函数，则这个派生类仍然还是一个抽象类。如果派生类中给出了基类纯虚函数的实现，则该派生类就不再是抽象类了，它是一个可以建立对象的具体的类。实现了纯虚函数的子类，该纯虚函数在子类中就编程了虚函数，子类的子类即孙子类可以覆盖该虚函数，由多态方式调用的时候动态绑定。

通过这次试验我了解到了虚函数和纯虚函数的区别和抽象类的概念和使用方法。

### 对 Point 类重载 ++ 和 -- 运算符

#### 题目要求

编写C++程序完成以下功能：

1. Point类的属性包括点的坐标（x，y）；
2. 实现 Point类重载＋＋和――运算符：
   * ++p，--p，p++，p--；
   * ＋＋和――分别表示x，y增加或减少1。

#### 代码

#include <cstdio>

#include <iostream>

class Point { // 定义 Point 类

int x, y; // 包括坐标 x, y

public:

Point() {}

Point(int \_x, int \_y) {

x = \_x;

y = \_y;

}

~Point() {}

Point operator++ () { // ++p

this -> x ++;

this -> y ++;

return \*this;

}

Point operator ++(int) { // p++

Point point(\*this);

this -> x ++;

this -> y ++;

// this -> z ++;

return point;

}

Point operator -- () { --p

this -> x --;

this -> y --;

return \*this;

}

Point operator -- (int) { // p--

Point point(\*this);

//std::cout << "\_\_4\_\_" << std::endl;

this -> x --;

this -> y --;

return point;

}

friend std::ostream& operator << (std::ostream& os, const Point point) {

os << point.x << " " << point.y << std::endl;

return os;

}

};

int main(int argc, char \*argv[])

{

Point point(1 , 2);

// int x = 1;

std::cout << point;

std::cout << point++;

std::cout << point--;

return 0;

}

输出

1 2

1 2

2 3

#### 感悟与反思

++i和i++有很大区别，其中++i表示先进行自加再进行运算，i++表示先进性运算再进行自加。在实现上也有些差别，后自加需要在参数中传入 ”(int)” 作为占位符。从实现可以看出，对于某个复杂类型的自加，前后自加有效率上的差异，其中前自加只是改变自身的值，但是后自加需要创建一个新的对象，造成了多余的时间消耗。

## 大作业Word Game（版本一）

### 作业要求

单词消除游戏由两类参与者组成：闯关者（即游戏玩家），出题者（为游戏增加游戏中使用单词）。游戏规则为，游戏每一轮，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词，如果闯关者输入正确（即闯关者输入的单词与刚刚显示的单词完全一致，包含大小写）则为通过。一关可以由一轮或者多轮组成。

出题者属性要求必须含有：出题者姓名、出题者出题数目、等级，若有需要可以自行添加其余属性。

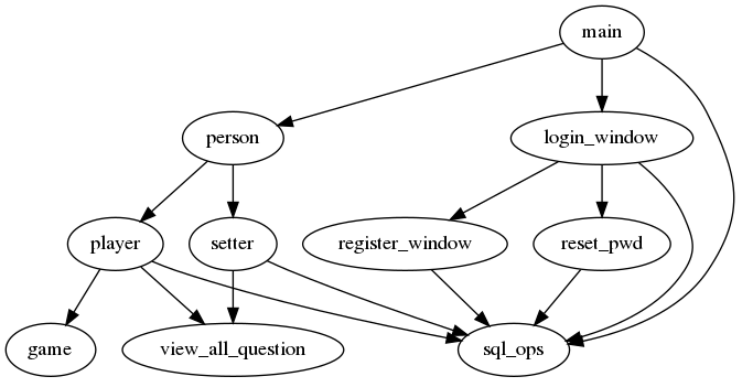
具体功能包括：

* 实现闯关者，出题者本地的注册、登录。
* 程序支持多人注册，同一时间只有一人可以登录。
* 实现游戏规则：出题者增加游戏中使用单词。游戏每一关，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词，如果闯关者输入正确则为通过。

 任何角色均可查询所有闯关者、出题者，按照属性查找相应闯关者、出题者。

* 可以根据闯关者闯过关卡数、经验、等级等对闯关者排名，根据出题者出题数目、等级对出题者排名。

### 模块划分



（此图为最终的模块设计，不完全在版本一中实现）

设计思想：将每个交互动作封装成一个对象，通过其中的信号量和其他的对象进行交互。其中person是player和setter的基类，每个具体操作的结果反映为和SQL的交互。

### 详细设计

1. 设计原理

单词消除游戏是一个面向多个对象的少状态程序，可以注意到其中的状态转移只有从未登录状态转移至player（做题者）或setter（出题者），那么这样可以在这三个状态的基础上进行扩展并通过main函数进行转换。此外本程序使用sqlite数据库进行数据管理，保证了数据的有效性和有序性，做到了有序的数据管理。在进行游戏时，为了达到定时消失的效果，使QT的定时器进行操作，一旦定时器超时就执行它的回调函数取消对应Label的显示。这样就使用户操作更加连贯。

1. 关键代码描述

Main.cpp

int main( int argc, char \*argv[] ) {

player\_ins = nullptr; // 将对应指针进行初始化

setter\_ins = nullptr;

// style::set\_style();

QApplication a( argc, argv );

QFont font; // 字体设置

font.setPointSize( 10 );

font.setFamily( "Microsoft YaHei" );

a.setFont( font );

database = new sql\_ops(); // 初始化数据库

login\_window \*login = new login\_window();

if ( login->exec() == QDialog::Accepted ) { // 若 login 正常返回

if ( player\_ins ) { // 若 player 被初始化

qDebug() << "start player";

player\_ins->exec();

// return 0;

}

if ( setter\_ins ) { // 若 setter 被初始化

qDebug() << "start setter";

setter\_ins->exec();

}

// if(!player\_ins && !setter\_ins)

}

// QMessageBox::warning(a, "login failed", "wrong username or password");

return 0;

}

Person.cpp （描述基类Person）

class Person {

public:

QString username;

QString pwd;

QString nickname;

int level;

int exp;

int count;

public:

Person() {

level = 0;

exp = 0;

}

Person(QString \_username,

QString \_pwd,

QString \_nickname,

int \_level,

int \_exp,

int \_count) {

this->username = \_username;

this->pwd = \_pwd;

this->nickname = \_nickname;

this->level = \_level;

this->exp = \_exp;

this->count = \_count;

}

Person(const Person &rhs) {

username = rhs.username;

pwd = rhs.pwd;

nickname = rhs.nickname;

level = rhs.level;

exp = rhs.exp;

}

int get\_level(QString s) {

if(s.length() <= 3)

return 1;

else if(s.length() <= 6)

return 2;

else if(s.length() <= 9)

return 3;

else if(s.length() <= 12)

return 4;

else

return 5;

}

};

sql\_ops.h

class sql\_ops {

private:

QSqlDatabase info; /\*\*< TODO: describe \*/

/\*\*

\* @brief

\*

\* @return bool

\*/

bool connect();

/\*\*

\* @brief

\*

\* @param username

\* @return bool

\*/

bool can\_use\_this\_name( QString username );

// bool comp( QPair<int, QString> lhs, QPair<int, QString> rhs );

public:

/\*\*

\* @brief

\*

\*/

sql\_ops();

/\*\*

\* @brief

\*

\* @param username

\* @param pwd

\* @return bool

\*/

bool login( QString username, QString pwd );

/\*\*

\* @brief

\*

\* @param username

\* @param pwd

\* @param type

\* @return bool

\*/

bool user\_register( QString username, QString pwd, int type );

/\*\*

\* @brief

\*

\* @param question

\* @param level

\* @param username

\* @return bool

\*/

bool add\_question( QString question, int level, QString username );

/\*\*

\* @brief

\*

\* @param username

\* @param pwd

\* @return Player

\*/

Player \*login\_player( QString username, QString pwd );

/\*\*

\* @brief

\*

\* @param username

\* @param pwd

\* @return Setter

\*/

Setter \*login\_setter( QString username, QString pwd );

/\*\*

\* @brief

\*

\* @param username

\* @param pwd

\* @return int

\*/

int get\_type( QString username, QString pwd );

/\*\*

\* @brief

\*

\* @param level

\* @return QString

\*/

QString get\_question( int level );

/\*\*

\* @brief

\*

\* @param username

\* @param pwd

\* @param level

\* @param exp

\* @param count

\* @param nickname

\*/

void save\_person( QString username, QString pwd, int level, int exp,

int count, QString nickname );

/\*\*

\* @brief

\*

\* @param username

\* @param level

\* @param exp

\* @param count

\* @param new\_pwd

\* @return bool

\*/

bool reset\_pwd( QString username, int level, int exp, int count,

QString new\_pwd );

/\*\*

\* @brief

\*

\* @param item

\*/

void load\_question\_item( QListWidget \*item );

/\*\*

\* @brief

\*

\* @param player

\* @param setter

\*/

void get\_rank( QListWidget \*player, QListWidget \*setter );

/\*\*

\* @brief

\*

\* @param username

\* @return QString

\*/

QString get\_avatar( QString username );

/\*\*

\* @brief

\*

\* @param username

\* @param path

\*/

void save\_avatar( QString username, QString path );

};

game.h

class game : public QDialog {

Q\_OBJECT

public:

/\*\*

\* @brief constructing the obejct

\*

\* @param parent

\*/

explicit game( QWidget \*parent = 0 );

// void get\_content(QString \_s);

/\*\*

\* @brief

\*

\*/

~game();

/\*\*

\* @brief

\* just start the game

\* @param \_s the quesiton string

\* @param level currect quesion level

\* @param p pointer to the caller

\*/

void start\_game( const QString \_s, int level, Player \*p );

private slots:

/\*\*

\* @brief

\* exit

\*/

void on\_cancelBtn\_clicked();

/\*\*

\* @brief

\* check whether the answer is right

\*/

void on\_OKBtn\_clicked();

/\*\*

\* @brief

\* set string \_s to label

\* @param \_s

\* the question string

\*/

void set\_text( const QString \_s );

/\*\*

\* @brief

\* make the label empty when time is up

\*/

void cancel\_context();

/\*\*

\* @brief

\* update the window shown

\*/

void update();

private:

Ui::game \*ui; /\*\*< UI \*/

QString s; /\*\*< store the question \*/

int left\_time; /\*\*< left time \*/

};

player.h

class Player : public QDialog, public Person {

Q\_OBJECT

public:

explicit Player(QWidget \*parent = 0);

~Player();

void update();

private slots:

void on\_game\_clicked();

void on\_questions\_clicked();

void on\_change\_clicked();

private:

Ui::Player \*ui;

void check\_levelup();

//int get\_level(QString s);

};

login\_window.h

class login\_window : public QDialog {

Q\_OBJECT

public:

/\*\*

\* @brief

\* constructing the object

\* @param parent

\*/

explicit login\_window( QWidget \*parent = 0 );

/\*\*

\* @brief

\*

\*/

~login\_window();

private slots:

/\*\*

\* @brief

\* check whether the username and pwd match

\*/

void on\_OKBtn\_clicked();

/\*\*

\* @brief

\* show register window

\*/

void on\_register\_2\_clicked();

/\*\*

\* @brief

\* show forget\_pwd window

\*/

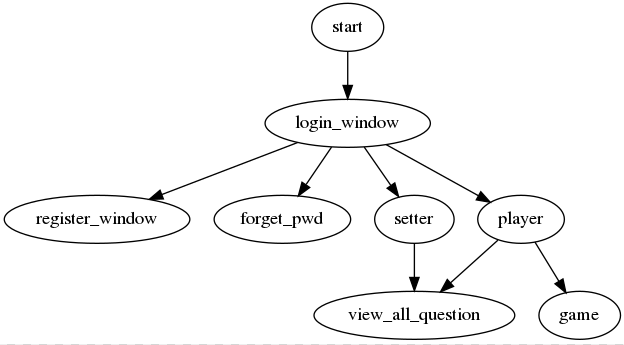
void on\_forget\_pwd\_clicked();

private:

Ui::login\_window \*ui; /\*\*< UI \*/

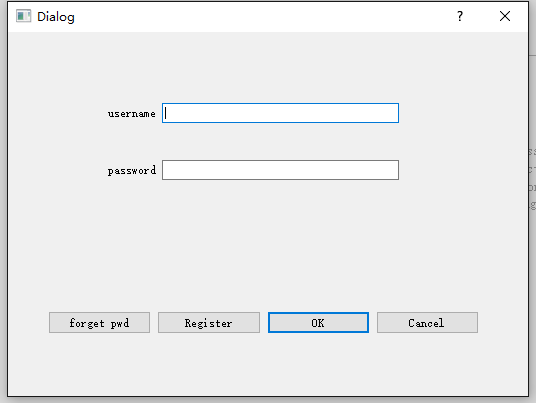
};

1. 系统流程图

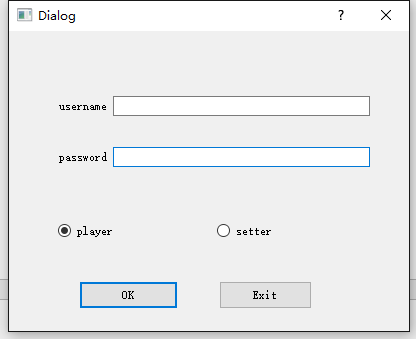


1. 系统界面

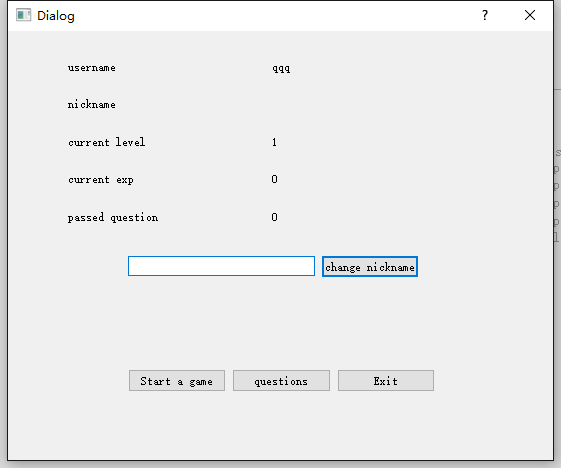
Login\_window



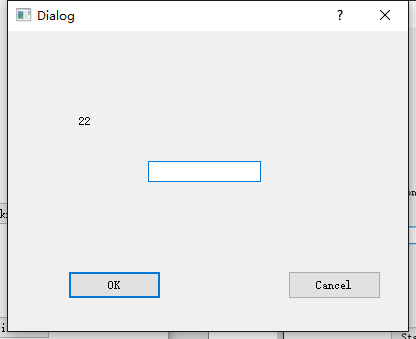
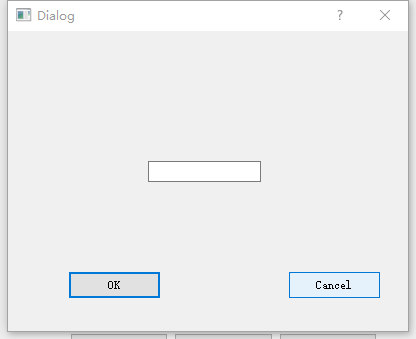
register\_window



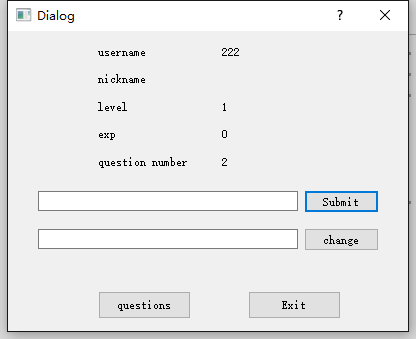
player



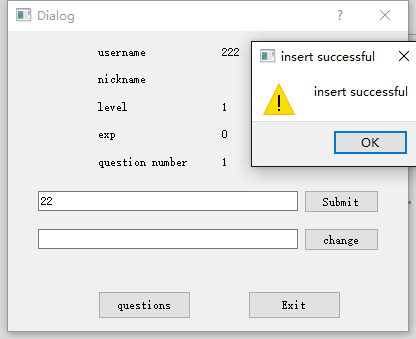
Game（单词消失前/单词消失后）

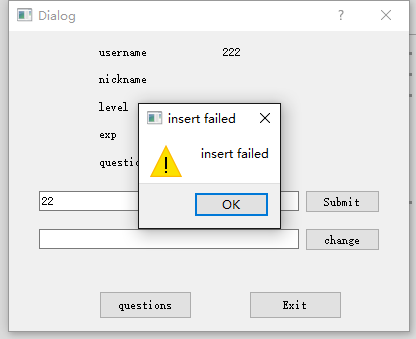
Setter



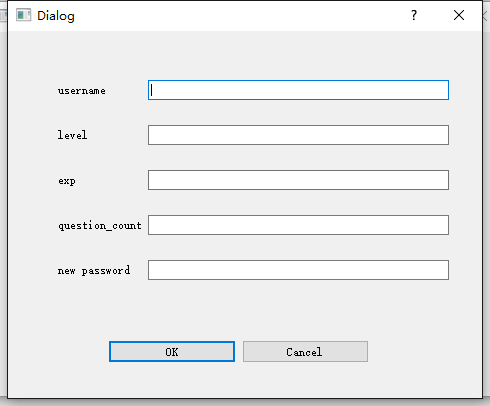
Setter添加题目



Setter添加重复题目



Forget\_pwd



1. 调试与说明
   1. 实现了主要功能：

1. 闯关者，出题者本地的注册、登录

2. 出题者增加游戏中使用单词。游戏每一关，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词，如果闯关者输入正确则为通过。

3. 任何角色均可查询所有闯关者、出题者，按照属性查找相应闯关者、出题者。

4. 使用文件或者数据库作为存储对象，自行设计文件格式，或者数据库表结构。

* 1. 附加功能：
     1. 添加了找回密码的功能，通过每个用户的用户名，答题数等信息重置密码。
     2. 添加了昵称系统，用户可以在其主页面进行昵称的更改。

## 大作业Word Game（版本二）

### 作业要求

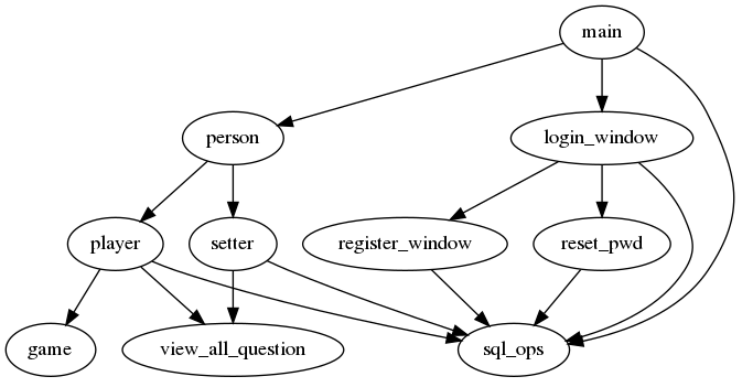
单词消除游戏由两类参与者组成：闯关者（即游戏玩家），出题者（为游戏增加游戏中使用单词）。游戏规则为，游戏每一轮，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词，如果闯关者输入正确（即闯关者输入的单词与刚刚显示的单词完全一致，包含大小写）则为通过。一关可以由一轮或者多轮组成。

闯关者每闯过一关，增加一定经验值。经验值会根据闯过的该关卡的关卡号、该关的闯关耗费时间共同决定。当经验值累计到一定程度闯关者等级增加。闯关失败需要重新闯该关。

游戏自带词库，而且已经注册的出题者可以为系统出题，即增加词库的新词，已经存在的单词不能再次添加（词库中的单词构成一个单词池，但建议根据单词的长度来组织存储。每次出题时，系统从该单词池中按照关卡难度随机的选择相应长度的单词）。每成功出题一次，更新该出题者的出题数目。出题者等级根据出题人成功出题数目来升级。

* 必须在题目一的基础上进行修改。
* 请根据要求设计每一关的出题方式，注意随着关卡数增加，题目难度增加。请合理处理出题人新添加新词的使用方式，并且新加词组不会影响游戏难度。
* 设计闯关者经验值，等级增加策略。设计出题者等级升级策略。

### 模块划分



### 详细设计

1. 设计原理

延续了上个版本的基础设计，在玩家和出题者升级的策略上做出了小幅的修改，由固定经验值变为了当前等级n的2\*n+1，是一个动态的晋级指标，闯关者和出题者共用晋级策略。增加了玩家每次做题的个数，一旦失败本次闯关无效，在后台有一个闯关定时器，为玩家提供额外的时间奖励。题目依据长度进行分级，按等级进行管理，每次给闯关者分配的题目是和他的等级相匹配的，如果之前闯关失败，会保存之前错误的题目，当再次闯关时优先使用之前错误的题目。随着关卡数的增加，每个题目的显示时间变短（有最短显示时间），相当于加大了难度。

1. 关键代码

Player（控制升级）

void Player::check\_levelup( int temp\_level ) {

exp += bonus;

exp += temp\_level;

int temp = level;

level += exp / ( 2 \* level + 1 );

exp %= 2 \* temp + 1;

count += 1;

qDebug() << "[player]" << exp;

}

Player（进行游戏）

bool Player::play\_game() {

for ( int i = 0; i < level; i++ ) {

game G;

if ( last\_failed == "" ) {

QString question = database->get\_question( level );

qDebug() << "[get question]" << question;

if ( question == "-1" ) {

QMessageBox::warning( this, "no question",

"there is no question for this level" );

return false;

} else {

last\_failed = question;

G.start\_game( question, get\_level( question ), this );

}

} else {

qDebug() << "[last failed] " << last\_failed;

G.start\_game( last\_failed, get\_level( last\_failed ), this );

}

if ( G.exec() == QDialog::Accepted )

last\_failed = "";

else

return false;

}

return true;

}

Game.cpp控制加分的定时器

void game::start\_game( const QString \_s, int level, Player \*p ) {

this->set\_text( \_s );

QTimer::singleShot( 1000 \* ( 6 - level ), this, SLOT( cancel\_context() ) );

QTimer \*update\_left\_time = new QTimer( this );

connect( update\_left\_time, SIGNAL( timeout() ), this, SLOT( update() ) );

update\_left\_time->start( 1000 );

QTimer \*bonus\_timer = new QTimer( this ); // 加分定时器

connect( bonus\_timer, SIGNAL( timeout() ), p, SLOT( get\_bonus() ) );

bonus\_timer->start( 500 \* ( 6 - level ) );

ui->left\_time->display( 6 - level );

left\_time = 6 - level;

s = \_s;

qDebug() << "[left time]" << left\_time;

}

数据库操作 sql\_ops.cpp

sql\_ops::sql\_ops() {

info = QSqlDatabase::addDatabase( "QSQLITE" );

info.setDatabaseName( "info.db" );

if ( !info.open() )

qDebug() << "failed to open database" << info.lastError().text();

else

qDebug() << "connected to database";

QSqlQuery query;

if ( !query.exec( "create table if not exists user\_info("

"username varchar primary key,"

"pwd varchar, "

"type int,"

"level int,"

"exp int,"

"count int,"

"nickname varchar)" ) )

qDebug() << "[create user\_info]" << query.lastError().text();

if ( !query.exec( "create table if not exists question\_info("

"question varchar primary key, "

"level int, "

"from\_who varchar)" ) )

qDebug() << "[create question\_info]" << query.lastError().text();

if ( !query.exec( "create table if not exists avatar\_info("

"username varchar primary key, "

"address varchar)" ) )

qDebug() << "[create avatar\_info]" << query.lastError().text();

}

bool sql\_ops::connect() {

if ( !info.open() ) {

qDebug() << "[database error]" << info.lastError().text();

return false;

}

return true;

}

bool sql\_ops::can\_use\_this\_name( QString username ) {

if ( connect() ) {

QSqlQuery query;

query.exec(

QString( "select username from user\_info where username ='%1'" )

.arg( username ) );

if ( !query.next() )

return true;

else

return false;

} else

return false;

}

bool sql\_ops::user\_register( QString username, QString pwd, int type ) {

username = encrypt( username );

pwd = encrypt( pwd );

if ( !can\_use\_this\_name( username ) ) return false;

if ( connect() ) {

QSqlQuery query;

if ( !query.exec(

"insert into user\_info(username, pwd, type, level, exp, "

"count, nickname)" +

QString( "values('%1', '%2', %3, 1, 0, 0, '')" )

.arg( username )

.arg( pwd )

.arg( type ) ) ) {

qDebug() << "[insert failed] " << query.lastError().text();

qDebug() << QString( "values('%1', '%2', %3, 1, 0, 0, '')" )

.arg( username )

.arg( pwd )

.arg( type );

return false;

} else

qDebug() << "[insert successful] "

<< QString( "values('%1', '%2', %3, 0, 0, 0)" )

.arg( username )

.arg( pwd )

.arg( type );

}

return true;

}

bool sql\_ops::add\_question( QString question, int level, QString username ) {

if ( connect() ) {

QSqlQuery query;

if ( query.exec(

"insert into question\_info(question, level, from\_who)" +

QString( "values('%1', %2, '%3')" )

.arg( question )

.arg( level )

.arg( username ) ) ) {

qDebug() << "[add question successful]";

return true;

} else {

qDebug() << "[add question failed]" << query.lastError().text();

return false;

}

}

return false;

}

bool sql\_ops::login( QString username, QString pwd ) {

username = encrypt( username );

pwd = encrypt( pwd );

if ( connect() ) {

QSqlQuery query;

if ( !query.exec(

QString( "select \* from user\_info where username='%1'" )

.arg( username ) ) )

qDebug() << "[log in failed] " << query.lastError().text();

if ( query.next() ) {

QString temp\_pwd = query.value( 1 ).toString();

qDebug() << "[from database temp pwd]: " << temp\_pwd << "[pwd]"

<< pwd;

if ( temp\_pwd == pwd ) {

qDebug() << "[login]" << username;

return true;

// qDebug() << "test";

} else

return false;

} else

return false;

}

return false;

}

Player \*sql\_ops::login\_player( QString username, QString pwd ) {

if ( !login( username, pwd ) ) return nullptr;

Player \*temp = new Player();

if ( connect() ) {

QSqlQuery query;

if ( query.exec( "select \* from user\_info where username = " +

QString( "'%1'" ).arg( username ) ) ) {

qDebug() << "[login successful]"

<< QString( "%1 " ).arg( username ) + "login";

while ( query.next() ) {

temp->username = query.value( 0 ).toString();

temp->pwd = query.value( 1 ).toString();

temp->level = query.value( 3 ).toInt();

temp->exp = query.value( 4 ).toInt();

temp->count = query.value( 5 ).toInt();

temp->nickname = query.value( 6 ).toString();

qDebug() << "[from data base load player]" << temp->username

<< " " << temp->nickname;

qDebug() << "[player address]" << temp;

}

} else {

qDebug() << "[login failed]";

delete temp;

return nullptr;

}

}

return temp;

}

Setter \*sql\_ops::login\_setter( QString username, QString pwd ) {

if ( !login( username, pwd ) ) return nullptr;

Setter \*temp = new Setter;

if ( connect() ) {

QSqlQuery query;

if ( query.exec( "select \* from user\_info where username = " +

QString( "'%1'" ).arg( username ) ) ) {

qDebug() << QString( "%1 " ).arg( username ) + "login";

while ( query.next() ) {

temp->username = query.value( 0 ).toString();

temp->pwd = query.value( 1 ).toString();

temp->level = query.value( 3 ).toInt();

temp->exp = query.value( 4 ).toInt();

temp->count = query.value( 5 ).toInt();

temp->nickname = query.value( 6 ).toString();

}

} else {

qDebug() << "login failed";

delete temp;

return nullptr;

}

}

return temp;

}

int sql\_ops::get\_type( QString username, QString pwd ) {

if ( !login( username, pwd ) ) return -1;

if ( connect() ) {

QSqlQuery query;

query.exec( QString( "select \* from user\_info where username='%1'" )

.arg( username ) );

if ( query.next() ) { // player -> 0 & setter -> 1

qDebug() << "[get type]" << query.value( 2 ).toInt();

return query.value( 2 ).toInt();

} else

return -1;

}

return -1;

}

QString sql\_ops::get\_question( int level ) {

QString question;

qDebug() << "[get\_question] get " << level;

if ( connect() ) {

QSqlQuery query;

if ( query.exec(

QString( "select \* from question\_info where level = %1 order "

"by random() limit 1" )

.arg( level ) ) ) {

while ( query.next() ) {

question = query.value( 0 ).toString();

return question;

}

qDebug() << "[database failed] return '-1' ";

return -1;

} else {

qDebug() << "[database failed] " << query.lastError().text();

}

}

qDebug() << "[database failed] return '-1' ";

return "-1";

}

void sql\_ops::save\_person( QString username, QString pwd, int level, int exp,

int count, QString nickname ) {

if ( connect() ) {

QSqlQuery query;

qDebug() << "[save person] to run: "

<< QString( "update user\_info set pwd='%1', level=%2, exp=%3, "

"count=%4, nickname='%5' where username='%6'" )

.arg( pwd )

.arg( level )

.arg( exp )

.arg( count )

.arg( nickname )

.arg( username );

if ( !query.exec(

QString( "update user\_info set pwd='%1', level=%2, exp=%3, "

"count=%4, nickname='%5' where username='%6'" )

.arg( pwd )

.arg( level )

.arg( exp )

.arg( count )

.arg( nickname )

.arg( username ) ) ) {

qDebug() << "[save person error]" << query.lastError().text();

} else

qDebug() << "[database update] success";

}

}

bool sql\_ops::reset\_pwd( QString username, int level, int exp, int count,

QString new\_pwd ) {

if ( connect() ) {

QSqlQuery query;

query.exec( QString( "select \* from user\_info where username='%1'" )

.arg( username ) );

if ( query.next() ) {

if ( query.value( 3 ).toInt() == level &&

query.value( 4 ).toInt() == exp &&

query.value( 5 ).toInt() == count ) {

if ( !query.exec( QString( "update user\_info set pwd='%1' "

"where username='%2'" )

.arg( new\_pwd )

.arg( username ) ) ) {

qDebug() << "[reset\_pwd]" << query.lastError().text();

return false;

} else {

qDebug() << "[reset\_pwd]"

<< QString( "%1 changed pwd to %2" )

.arg( username )

.arg( new\_pwd );

return true;

}

}

}

}

return false;

}

void sql\_ops::load\_question\_item( QListWidget \*item ) {

if ( connect() ) {

QSqlQuery query;

if ( query.exec( "select \* from question\_info" ) ) {

while ( query.next() ) {

item->addItem( new QListWidgetItem(

QString( "question: %1, level: %2, from: %3" )

.arg( query.value( 0 ).toString() )

.arg( query.value( 1 ).toInt() )

.arg( query.value( 2 ).toString() ) ) );

qDebug() << "[load\_question]"

<< QString( "question: %1, level: %2, from: %3" )

.arg( query.value( 0 ).toString() )

.arg( query.value( 1 ).toInt() )

.arg( query.value( 2 ).toString() );

}

}

}

}

bool comp( QPair<int, QString> lhs, QPair<int, QString> rhs ) {

if ( lhs.first == rhs.first ) return lhs.second < rhs.second;

return lhs.first > rhs.first;

}

void sql\_ops::get\_rank( QListWidget \*player, QListWidget \*setter ) {

QQueue<QPair<int, QString>> player\_list;

QQueue<QPair<int, QString>> setter\_list;

if ( connect() ) {

QSqlQuery query;

if ( query.exec( "select \* from user\_info" ) ) {

while ( query.next() ) {

if ( query.value( 2 ).toInt() == 0 ) {

player\_list.append(

qMakePair( query.value( 5 ).toInt(),

query.value( 0 ).toString() ) );

} else {

setter\_list.append(

qMakePair( query.value( 5 ).toInt(),

query.value( 0 ).toString() ) );

}

}

} else

qDebug() << "[get rank] " << query.lastError().text();

if ( !player\_list.empty() )

qSort( player\_list.begin(), player\_list.end(), comp );

if ( !setter\_list.empty() )

qSort( setter\_list.begin(), setter\_list.end(), comp );

int player\_index = 1, setter\_index = 1;

while ( !player\_list.empty() ) {

player->addItem( new QListWidgetItem(

QString( "rank: %1, username: %2, solved\_problems: %3" )

.arg( player\_index )

.arg( player\_list.front().second )

.arg( player\_list.front().first ) ) );

player\_list.pop\_front();

player\_index += 1;

}

while ( !setter\_list.empty() ) {

setter->addItem( new QListWidgetItem(

QString( "rank: %1, username: %2, set\_problems: %3" )

.arg( setter\_index )

.arg( setter\_list.front().second )

.arg( setter\_list.front().first ) ) );

setter\_list.pop\_front();

setter\_index += 1;

}

}

}

QString sql\_ops::get\_avatar( QString username ) {

if ( connect() ) {

QSqlQuery query;

if ( !query.exec( QString( "select \* from avatar\_info "

"where username='%1'" )

.arg( username ) ) )

qDebug() << "[get\_avatar]" << query.lastError().text();

qDebug() << "[get\_avatar] try:"

<< QString( "select \* from avatar\_info "

"where username='%1'" )

.arg( username )

<< "[username]" << username;

if ( query.next() ) {

qDebug() << "[get\_avatar\_1]" << query.value( 1 ).toString();

return query.value( 1 ).toString();

}

qDebug() << "get\_avatar\_2\_ no username";

return "";

}

return "";

}

void sql\_ops::save\_avatar( QString username, QString path ) {

if ( connect() ) {

QSqlQuery query;

query.exec( QString( "select \* from avatar\_info where username='%1'" )

.arg( username ) );

if ( query.next() ) {

if ( query.exec( QString( "update avatar\_info set address='%1' "

"where username='%2'" )

.arg( path )

.arg( username ) ) ) {

qDebug() << "[update\_avatar] success"

<< QString( "update avatar\_info set address='%1' "

"where username='%2'" )

.arg( path )

.arg( username );

} else

qDebug() << "[save avatar] failed" << query.lastError().text();

} else {

if ( query.exec(

QString( "insert into avatar\_info(username, address) "

"values('%1', '%2')" )

.arg( username )

.arg( path ) ) ) {

qDebug() << "[insert avatar] success"

<< QString(

"insert into avatar\_info(username, address) "

"values('%1', '%2')" )

.arg( username )

.arg( path );

} else

qDebug() << "[save avatar] failed" << query.lastError().text();

}

}

}

// QQueue<QPair<int, QString>> \*sql\_ops::get\_rank() {

// QQueue<QPair<int, QString>> \*ans = new QQueue<QPair<int, QString>>;

// if ( connect() ) {

// QSqlQuery query;

// if ( query.exec( "select \* from user\_info" ) ) {

// while ( query.next() ) {

// int count = query.value( 5 ).toInt();

// QString username = query.value( 0 ).toString();

// ans->append( qMakePair( count, username ) );

// }

// } else

// qDebug() << "[get rank] " << query.lastError().text();

// }

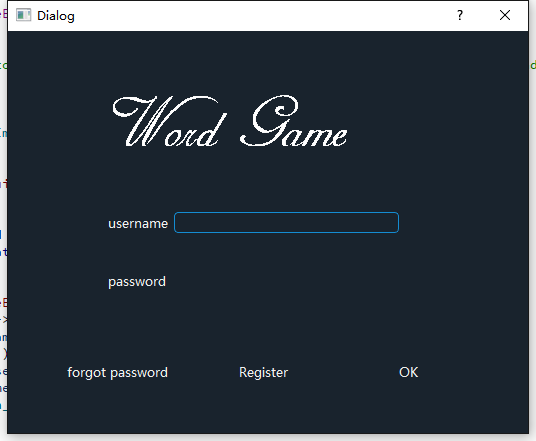
// if ( !ans->empty() ) qSort( ans->begin(), ans->end(), comp );

// return ans;

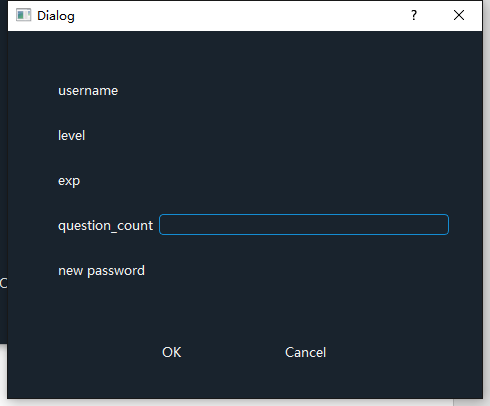
//}

1. 页面设计

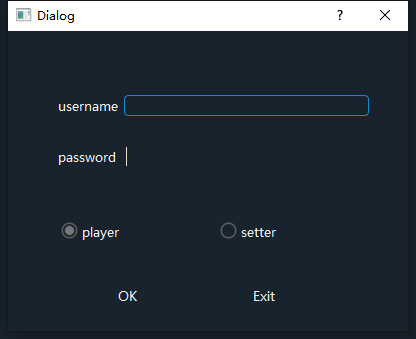
Login\_window



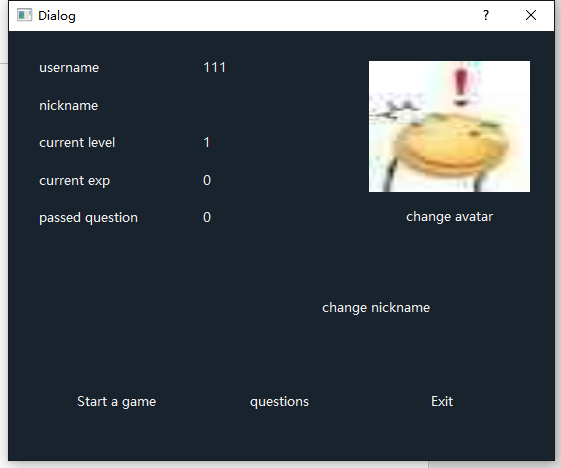
Forgot\_pwd



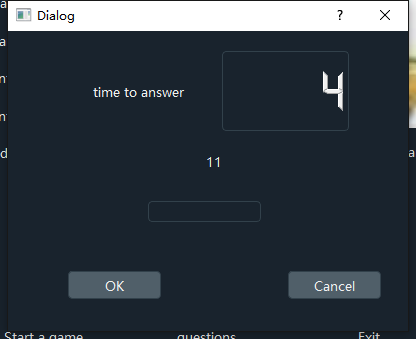
Register\_window



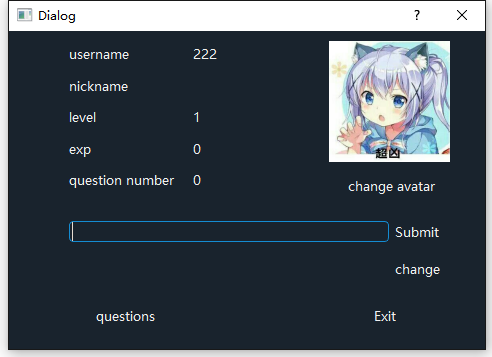
Player（头像显示）



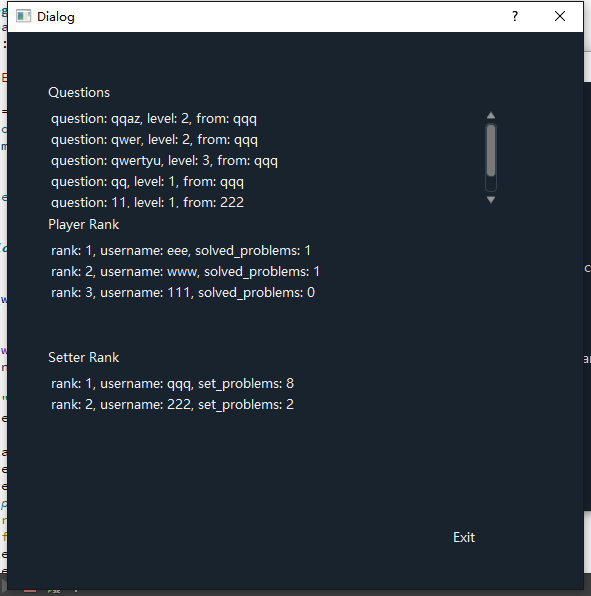
Game



Setter（头像显示）



1. 调试与说明
2. 完成课主要功能
3. 请根据要求设计每一关的出题方式，注意随着关卡数增加，题目难度增加。请合理处理出题人新添加新词的使用方式，并且新加词组不会影响游戏难度。
4. 闯关者每闯过一关，增加一定经验值。经验值会根据闯过的该关卡的关卡号、该关的闯关耗费时间共同决定。当经验值累计到一定程度闯关者等级增加。闯关失败需要重新闯该关。
5. 设计闯关者经验值，等级增加策略。设计出题者等级升级策略。
6. 添加的新功能
7. 头像选择与显示（同步到数据库）
8. 对出题者和闯关者分别进行排名
9. 页面装饰及美化
10. 对数据库添加加密解密功能，保证用户数据安全



1. 实验笔记

单机版的单词消除游戏开发完毕，进行一下简单的总结。

这是我第一次使用QT进行开发，前期踩了很多的坑，了解到在QT内部不能使用std标准库还有学习了SQL的基本语法和相关的debug方法。为了进行数据库内的debug我专门安装了sqlite数据库并在debug时实时查看其中的元素。我进行了大量的测试，确保了针对每一个可能出现的错误都有相应的处理代码，保证在数据不正确的情况下对数据库没有影响。

前端方面使用原生QT进行设计和开发，在部分页面上使用代码进行元素添加。在第二个版本中引入了QSS（类似CSS2.1）进行页面的装饰和美化。这个部分中我自学了相关的QSS的技术。在这个过程中我基本看完了QT上有关的所有库的文档，学习了如何快速找到需要组件的能力。

在用户状态的转以上，我使用了两个不同类型的指针，在登陆后检查哪个不为空。这样的方式并不算优雅，但是等我想到使用信号量进行构造时别的部分中已经这么使用了，造成了代码重构的困难。从这里我明白了在开始实现之前应该进行具体而微的构思，而不是进行简单的抽象之后直接上手实现，再考虑代码间的信息传递。

## 大作业Word Game（版本三）

### 作业要求

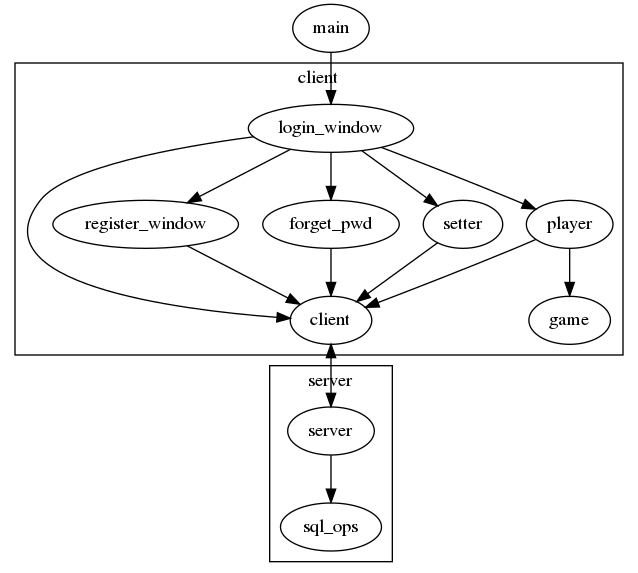
在题目二的基础上，将游戏由本地单机，扩展为服务器多人游戏平台，使用客户端/服务器的方式，同一时间可以多人登录系统。将所有闯关者、出题者信息保存在服务器。

要求：

* 必须在题目二基础上进行修改。
* 使用socket进行通信。
* 需要完成服务器端程序，以及客户端程序。客户端可以启动多个同时与服务器交互，要求服务器具有并发处理能力。

### 详细设计

1. 组件划分



对代码进行了逻辑拆分，分离了数据库部分的逻辑，状态转移部分仍由客户端部分完成，实现了远程的数据库。其他的部分逻辑上和之前的保持一致。通过Json RPC完成通信，其中由client和server组件进行数据传输，解析和序列化。在两边的程序中添加了消息队列的设计，通过检查消息头即可检查是否是自己请求的数据到来。使用了QT中管理信号的线程完成消息队列的管理。调用函数时每次进行请求，通过Json进行数据封装。并通过UDP进行数据传输。

在服务器上通过Json RPC完成数据通信。通信协议见下面。使用多线程完成请求响应，其中使用bool类型的locked变量实现全局锁，保证数据库的线性操作，不会造成冲突。

1. 功能说明
2. Client和server是完成信息传递的重要组件。完成信息的序列化和反序列化，并构造报文给UDP进行传输，同时里面加上了自己的端口（address域）以便于让server正确返回信息。
3. Login\_window：构造欢迎页面和相应的组件逻辑
4. Register\_window：注册页面
5. Forget\_pwd：忘记密码页面
6. Setter：出题者界面
7. Player：闯关者界面
8. Game：游戏界面
9. Sql\_ops：数据库操作封装
10. 通信协议

JSON-RPC，是一个无状态且轻量级的远程过程调用（RPC）传送协议，其传递内容透过 JSON 为主。相较于一般的 REST 透过网址（如 GET /user）调用远程服务器，JSON-RPC 直接在内容中定义了欲调用的函数名称（如 {"method": "getUser"}），这也令开发者不会陷于该使用 PUT 或者 PATCH 的问题之中。 本规范主要定义了一些数据结构及其相关的处理规则。它允许运行在基于 Socket、HTTP 等诸多不同消息传输环境的同一进程中。其使用 JSON（RFC 4627）作为数据格式。

由于 JSON-RPC 使用 JSON，它具有与其相同的类型系统。JSON 可以表示四个基本类型（String、Numbers、Boolean 和 Null）和两个结构化类型（Objects 和 Array）。任何时候文档涉及 JSON 数据类型，第一个字母都必须大写：Object、Array、String、Number、Boolean、Null。包括 True 和 False 也要大写。在客户端与任何被匹配到的服务端之间交换的所有成员名字应是区分大小写的。 函数、方法、过程都可以认为是可以互换的。客户端被定义为请求对象的来源及响应对象的处理程序。服务端被定义为响应对象的起源和请求对象的处理程序。该规范的一种实现为可以轻而易举的填补这两个角色，即使是在同一时间，同一客户端或其他不相同的客户端。 该规范不涉及复杂层。

示例

在以下的例子里，-->表示数据发送至服务器，<--表示数据发送至客户端。

带索引数组参数的rpc调用：

--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}

<-- {"jsonrpc": "2.0", "result": 19, "id": 1}

--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}

<-- {"jsonrpc": "2.0", "result": -19, "id": 2}

实际报文



使用 “method” 段表明调用的函数，使用 “params” 数组完成传参。通过重构调用函数完成信息的发送

使用的报文格式：

Client：

“jsonrpc”: “2.0” // 协议标准

“address”：自己收端口的端口号

“method”：调用的函数

“id”：消息序号（序列化及反序列化需要）

“params”：参数列表

Server：

“jsonrpc”: “2.0” // 协议标准

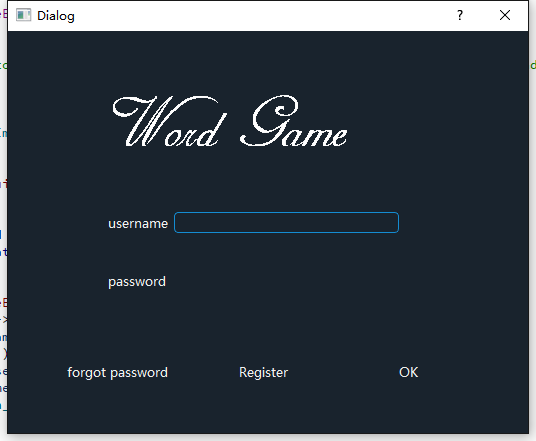
“method”：和收到的保持一致（确定client的回调函数）

“id”：和收到的一致

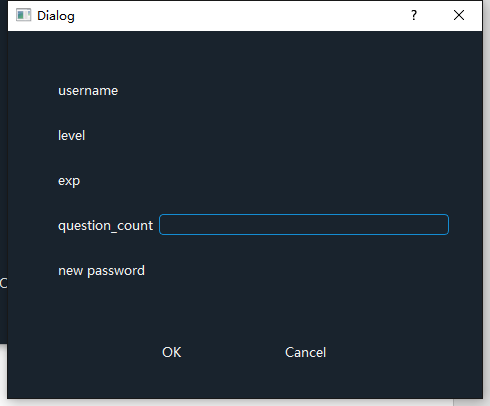
“params”：返回值

界面设计：

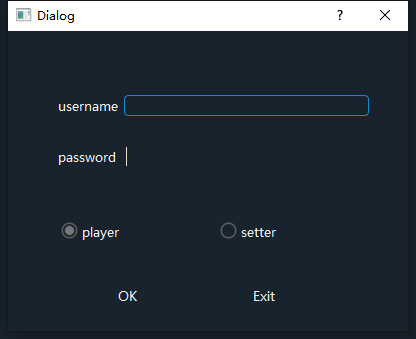
Login\_window



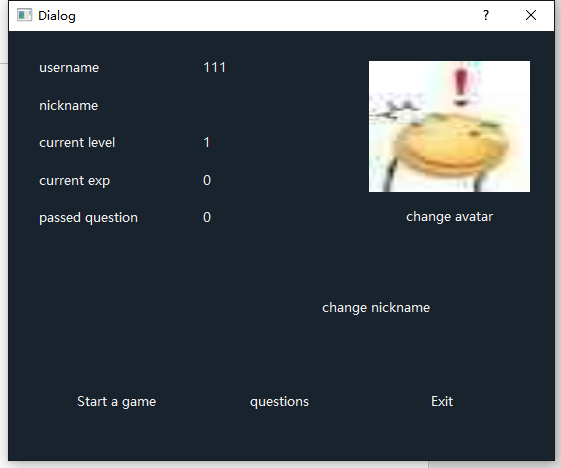
Forgot\_pwd



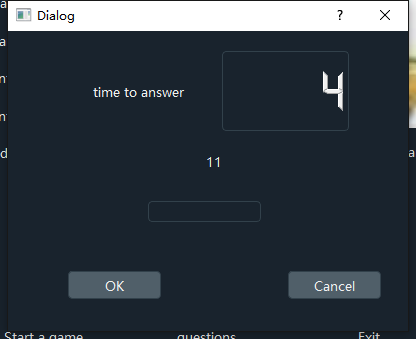
Register\_window



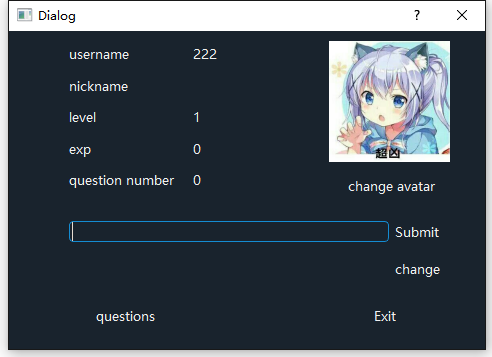
Player（头像显示）



Game



Setter（头像显示）



1. 关键代码解析

构造json报文

void client::construct\_json( QString method, QJsonArray array ) {

QJsonObject json;

json.insert( "jsonrpc", "2.0" );

json.insert( "address", QString( "%1" ).arg( receive\_port ) );

json.insert( "method", method );

json.insert( "id", QString( "%1" ).arg( index ) );

index++;

json.insert( "params", QJsonValue( array ) );

QJsonDocument doc;

doc.setObject( json );

QByteArray byte\_array = doc.toJson( QJsonDocument::Compact );

QString str\_json( byte\_array );

qDebug() << "[client] send json " << str\_json;

udp\_send( str\_json );

}

在解析报文后根据之前调用的参数进行回调

void client::call\_func( QString func\_call, QJsonValue params ) {

qDebug() << "[client call\_func] to call " << func\_call;

locked = false;

QStringList func\_list;

func\_list << "login"

<< "user\_register"

<< "add\_question"

<< "get\_question"

<< "save\_person"

<< "reset\_pwd"

<< "load\_question\_item"

<< "load\_player\_rank"

<< "load\_setter\_rank"

<< "get\_avatar"

<< "save\_avatar";

QString username, pwd, question, nickname;

int level, exp, count, type;

QJsonArray array;

switch ( func\_list.indexOf( func\_call ) ) {

case 0:

qDebug() << "unlock for login";

if ( params.isArray() ) {

QJsonArray array = params.toArray();

if ( array.size() == 0 ) {

msg\_queue.push\_back( -1 );

break;

}

username = array.at( 0 ).toString();

pwd = array.at( 1 ).toString();

type = array.at( 2 ).toString().toInt();

level = array.at( 3 ).toString().toInt();

exp = array.at( 4 ).toString().toInt();

count = array.at( 5 ).toString().toInt();

nickname = array.at( 6 ).toString();

msg\_queue.push\_back( 1 );

if ( type == 0 ) {

player\_ins = new Player;

player\_ins->username = username;

player\_ins->pwd = pwd;

player\_ins->level = level;

player\_ins->exp = exp;

player\_ins->count = count;

// player\_ins->type = type;

break;

} else {

setter\_ins = new Setter;

setter\_ins->username = username;

setter\_ins->pwd = pwd;

setter\_ins->level = level;

setter\_ins->exp = exp;

setter\_ins->count = count;

break;

}

} else

msg\_queue.push\_back( -1 );

break;

case 1:

qDebug() << "unlock for user\_register";

if ( params.isBool() ) {

if ( params.toBool() == true ) {

msg\_queue.push\_back( 1 );

break;

}

} else {

qDebug() << "[client call func] bad json";

}

msg\_queue.push\_back( -1 );

break;

case 2:

qDebug() << "unlock for add\_question";

if ( params.isBool() ) {

if ( params.toBool() == true ) {

msg\_queue.push\_back( 1 );

break;

}

} else {

qDebug() << "[client call func] bad json";

}

msg\_queue.push\_back( -1 );

break;

case 3:

if ( params.isString() ) {

QString temp = params.toString();

if ( temp != "-1" ) {

exec\_result = temp;

msg\_queue.push\_back( 1 );

break;

}

}

msg\_queue.push\_back( -1 );

break;

case 4: // void save\_person

break;

case 5:

if ( params.isBool() ) {

if ( params.toBool() == true ) {

msg\_queue.push\_back( 1 );

break;

}

}

msg\_queue.push\_back( -1 );

break;

case 6:

if ( params.isArray() ) {

array = params.toArray();

for ( int i = 0; i < array.size(); i++ ) {

QJsonObject obj = array.at( i ).toObject();

question\_item.push\_back(

"question: " + obj.value( "question" ).toString() +

" level: " + obj.value( "level" ).toString() +

" from: " + obj.value( "from" ).toString() );

}

// qSort( question\_item.begin(), question\_item.end(), comp );

msg\_queue.push\_back( 1 );

break;

}

msg\_queue.push\_back( -1 );

break;

case 7:

if ( params.isArray() ) {

array = params.toArray();

for ( int i = 0; i < array.size(); i++ ) {

QJsonObject obj = array.at( i ).toObject();

qDebug() << obj;

player\_rank.push\_back(

qMakePair( obj.value( "count" ).toString().toInt(),

obj.value( "username" ).toString() ) );

qDebug() << "[push to player\_rank]" << player\_rank.last();

}

msg\_queue.push\_back( 1 );

// qDebug() << "[!!!player\_rank]" << player\_rank.size();

break;

}

msg\_queue.push\_back( -1 );

break;

case 8:

if ( params.isArray() ) {

array = params.toArray();

for ( int i = 0; i < array.size(); i++ ) {

QJsonObject obj = array.at( i ).toObject();

qDebug() << obj;

setter\_rank.push\_back(

qMakePair( obj.value( "count" ).toString().toInt(),

obj.value( "username" ).toString() ) );

qDebug() << "[push to setter\_rank]" << setter\_rank.last();

}

// qDebug() << "[!!!setter\_rank]" << setter\_rank.size();

msg\_queue.push\_back( 1 );

break;

}

msg\_queue.push\_back( -1 );

break;

case 9:

if ( params.isString() ) {

exec\_result = params.toString();

msg\_queue.push\_back( 1 );

qDebug() << "[client get avatar]: " << exec\_result;

break;

}

msg\_queue.push\_back( -1 );

break;

case 10: // void save\_avatar

break;

default:

break;

}

}

阻塞方式

locked = true;

client\_sender->construct\_json( "add\_question", array ); // send request

while ( locked ) { // block

QTime dieTime = QTime::currentTime().addMSecs( 100 );

while ( QTime::currentTime() < dieTime ) // check whether state changes when blocked

QCoreApplication::processEvents( QEventLoop::AllEvents, 100 );

}

1. 实验笔记

通过这次实验，理解了 C++ 语言面向对象编程的基本思想，掌握了类的声明以及对象的创建, 了解了类的成员变量和成员方法的特性以及类的构造方法的使用。学会了QT的信号机制，同时对其做了扎实的应用，这次三个版本共两千余行代码对我的工程能力有很大的提升。在这次大作业中我学会了socket编程，吸取上次的教训，在开始分割逻辑之前就定好了Json的报文内容，让我在后面少走了很多弯路。虽然Json的解析过程繁琐但是胜在稳定和良好的封装，在发送时我就能看到发送的内容是否正确。同时填补了C++的任务队列的空缺，我先写了一个小demo进行原理验证：

class Producer : public QThread {

private:

int index;

public:

Producer() { index = 0; }

void run() override {

for ( int i = 0;; i++ ) {

if ( index >= 10 ) break;

if ( buffer.size() != 0 ) continue;

index += 1;

freeBytes.acquire();

QJsonObject obj;

obj.insert( "index", QString( "%1" ).arg( index ) );

QJsonDocument document;

document.setObject( obj );

QByteArray byteArray = document.toJson( QJsonDocument::Compact );

QString strJson( byteArray );

qDebug() << "[prodecuer]" << strJson;

buffer = byteArray;

usedBytes.release();

}

// for ( int i = 0; i < DataSize; ++i ) {

// freeBytes.acquire();

// buffer[ i % BufferSize ] =

// "ABCD"[ QRandomGenerator::global()->bounded( 4 ) ];

// usedBytes.release();

// }

}

};

class Consumer : public QThread {

Q\_OBJECT

public:

void run() override {

for ( int i = 0; i < 10; i++ ) {

usedBytes.acquire();

QJsonParseError jsonError;

QJsonDocument doucment =

QJsonDocument::fromJson( buffer, &jsonError );

if ( !doucment.isNull() &&

( jsonError.error == QJsonParseError::NoError ) ) {

if ( doucment.isObject() ) {

QJsonObject object = doucment.object();

if ( object.contains( "index" ) ) {

QJsonValue value = object.value( "index" );

if ( value.isString() ) {

QString strName = value.toString();

qDebug() << "[consumer] index: " << strName.toInt();

buffer.clear();

}

}

}

}

freeBytes.release();

}

// for ( int i = 0; i < DataSize; ++i ) {

// usedBytes.acquire();

// fprintf( stderr, "%c", buffer[ i % BufferSize ] );

// freeBytes.release();

// }

// fprintf( stderr, "\n" );

}

};

int main( int argc, char \*argv[] ) {

QCoreApplication app( argc, argv );

Producer producer;

Consumer consumer;

producer.start();

consumer.start();

producer.wait();

consumer.wait();

return 0;

}

得到的log信息如下

[prodecuer] "{\"index\":\"1\"}"

[consumer] index: 1

[prodecuer] "{\"index\":\"2\"}"

[consumer] index: 2

[prodecuer] "{\"index\":\"3\"}"

[consumer] index: 3

[prodecuer] "{\"index\":\"4\"}"

[consumer] index: 4

[prodecuer] "{\"index\":\"5\"}"

[consumer] index: 5

[prodecuer] "{\"index\":\"6\"}"

[consumer] index: 6

[prodecuer] "{\"index\":\"7\"}"

[consumer] index: 7

[prodecuer] "{\"index\":\"8\"}"

[consumer] index: 8

[prodecuer] "{\"index\":\"9\"}"

[consumer] index: 9

[prodecuer] "{\"index\":\"10\"}"

[consumer] index: 10

可以看到其中每次producer向缓冲区内写入内容并阻塞producer至内容被consumer取出，这样我们可以制造出一个有block的任务队列，并且这个任务队列拥有独立的线程。每次向队列中放进具体的参数，直至被调用函数取出。这样就完成了同步的调用。

这样的单元测试通过后才进行逻辑上的合并，通过任务队列我顺利实现了在block之后的参数传递。这样的实验方法应该是我本次大作业的收获了。

从需求出发搭建小的demo，通过原理验证后再与已有的项目合并。虽然路途上遇到了很多的bug但是我清楚每个操作的底层没有原理上的问题，不会在某一步卡住后突然反应过来需要推倒重来。这样的架构上的挑战是我觉得最有意思的地方。