# 详细设计

目录

[一、数据库模块 2](#_Toc171814617)

[DB类： 2](#_Toc171814618)

[二、前端模块 4](#_Toc171814619)

[学生界面： 5](#_Toc171814620)

[教师界面： 5](#_Toc171814621)

[管理员界面： 6](#_Toc171814622)

[中转界面： 6](#_Toc171814623)

[三、数据可视化模块 8](#_Toc171814624)

[一分一段表： 8](#_Toc171814625)

[直方图： 9](#_Toc171814626)

[线性相关度分析： 10](#_Toc171814627)

[绘制散点图： 11](#_Toc171814628)

[四、UML类图 12](#_Toc171814629)

[UML类图： 12](#_Toc171814630)

## 数据库模块

**该模块负责实现数据库相关操作，包括登录，对成绩的增删查改和排序，包含文件mysql\_connect.h mysql\_connect.cpp**

### DB类：

该模块包含一个类DB(DataBase)，具体定义如下：

class DB{

private:

//Qsqldatabase的私有成员变量db，负责连接数据库

QSqlDatabase db;

//初始化连接函数，在构造函数中调用

QSqlDatabase Init();

public:

DB();

~DB();

//检查是否有重复的数据库连接,如有则删

void connection\_check();

//登录检查

int login\_check(QString username, QString password);//返回登陆账户的identity(int)：0表示管理员 1表示学生 2表示老师 -1表示错误 3表示新用户

//密码修改 输入用户名和新密码

bool revise\_password(QString username,QString new\_password);

//返回某学生的单科成绩

int get\_single\_score(QString student\_id,QString subject);

//将一个学生的科目和成绩保存在一个pair的vector数组中返回

VP get\_all\_score(QString student\_id);

//修改某个学生某科目的成绩

bool update\_score(QString student\_id,QString subject,int new\_score);

//将excel文件上传到数据库，批量上传可用for循环

bool upload\_score(QString file\_path);

//批量获取学生的单科成绩，保存在pair类型的vector中返回

void get\_students\_scores(QString start\_student\_id,QString end\_student\_id,QString subject,std::vector<int>& scores);

//删除某个学生

bool delete\_student(QString student\_id);

//添加学生信息

bool add\_score(QString student\_id, QString student\_name, VP subject\_scores);

//获取各科排名 总分用SUM替代

int get\_rank(QString student\_id,QString subject,int& delta);

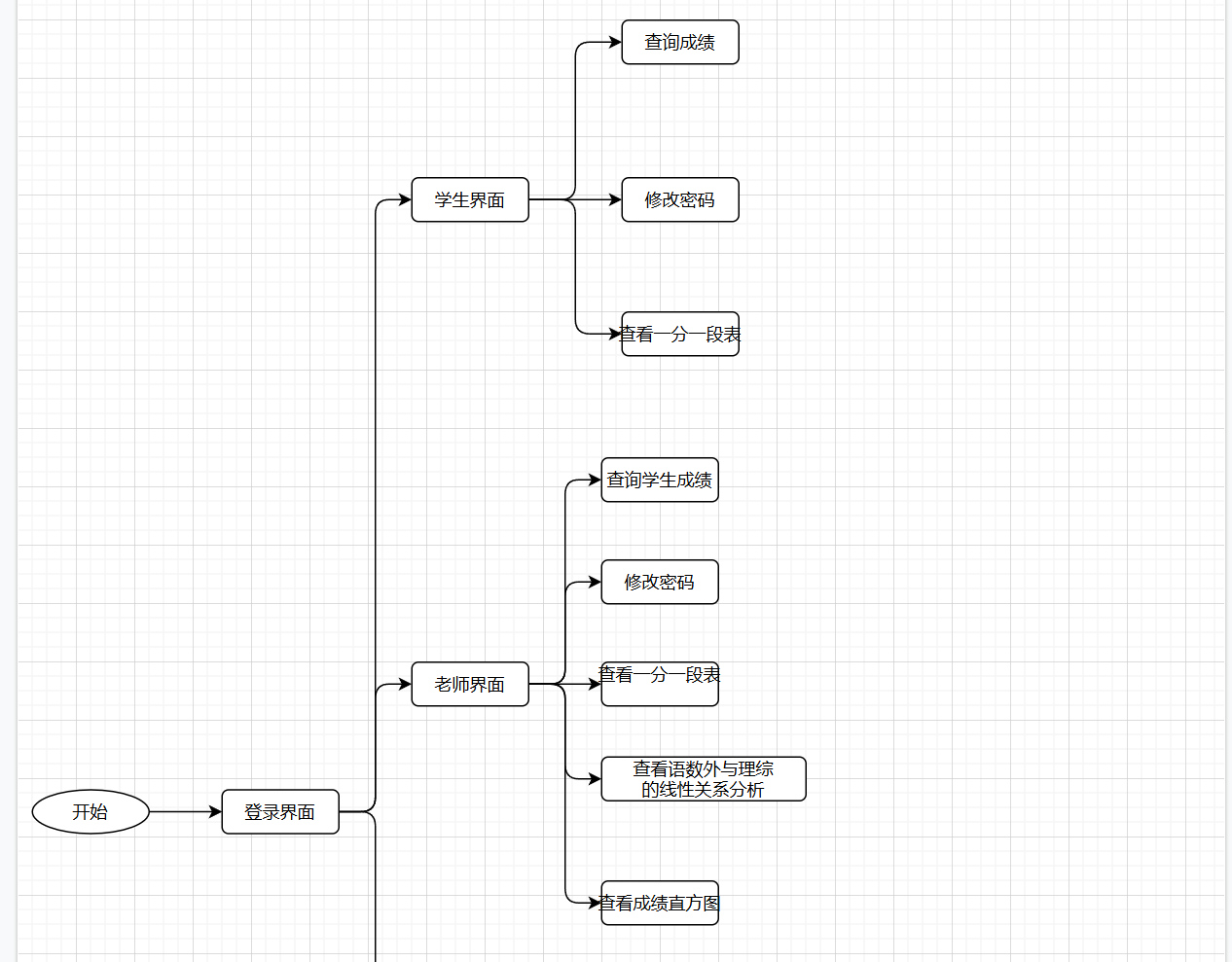
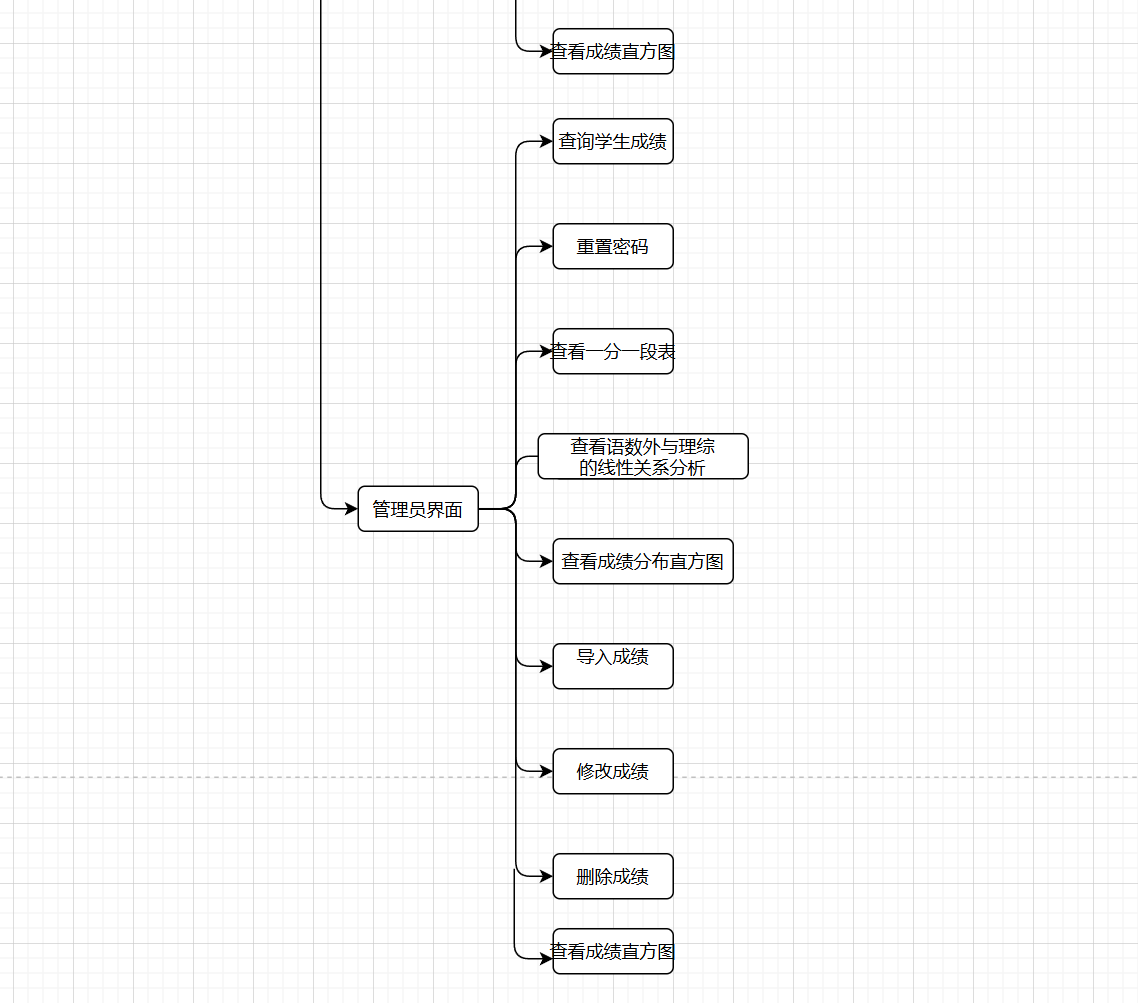
//获取所有学生的一分一段数据 使用应用返回结果 类型QMap<int,int> 总分用SUM表示

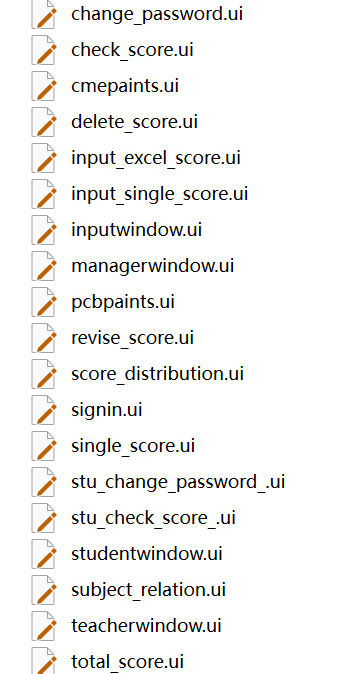
void get\_all\_ranks(QString subject,QMap<int,int> &scoreDistribution);

void get\_ranks(QString subject,std::vector<int>& rank);

};

# 前端模块

该模块主要负责与用户直接进行交互的可视化窗口界面实现。该模块包含的窗口界面如下

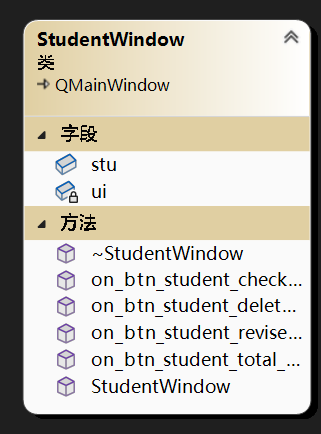
**该模块包含文件如下：**

**从上到下依次对应：**

**修改密码界面\查询分数界面\线性关系分析界面\成绩删除界面\上传excel界面\上传分数界面\管理员界面\一分一段表界面\登录界面\学生界面\中间跳转界面**

由于此处界面相关类过多，故从中挑选部分用户界面举例说明：

### 学生界面：



class StudentWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit StudentWindow(QString s,QWidget \*parent = nullptr);

void on\_btn\_student\_check\_clicked(); //查询成绩

void on\_btn\_student\_delete\_clicked(); //删除成绩

void on\_btn\_student\_revise\_password\_clicked();//重置密码

void on\_btn\_student\_total\_clicked();//一分一段表

~StudentWindow();

private:

Ui::StudentWindow \*ui;

student stu;

};

### 教师界面：

class TeacherWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit TeacherWindow(QWidget \*parent = nullptr);

void on\_btn\_teacher\_check\_clicked(); //查询成绩

void on\_btn\_teacher\_revise\_password\_clicked();//重置密码

void on\_btn\_teacher\_total\_clicked();//一分一段表

void on\_btn\_teacher\_single\_clicked();//单科成绩直方图

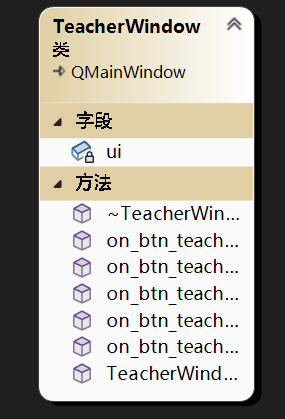
void on\_btn\_teacher\_subject\_clicked(); //展示科目间的线性相关关系

~TeacherWindow();

private slots:

//void on\_btn\_teacher\_subject\_relation\_clicked();

private:

 Ui::TeacherWindow \*ui}；

### 管理员界面：

class managerwindow : public QMainWindow

{

Q\_OBJECT

public:

explicit managerwindow(QWidget \*parent = nullptr);

void on\_btn\_manager\_input\_clicked();//录入成绩

void on\_btn\_manager\_revise\_clicked();//修改成绩

void on\_btn\_manager\_check\_clicked(); //查询成绩

void on\_btn\_manager\_delete\_clicked(); //删除成绩

void on\_btn\_manager\_revise\_password\_clicked();//重置密码

void on\_btn\_manager\_total\_score\_clicked();//一分一段表

void on\_btn\_manager\_subject\_clicked();//学科间线性关系

void on\_btn\_manager\_single\_clicked();//单科成绩直方图

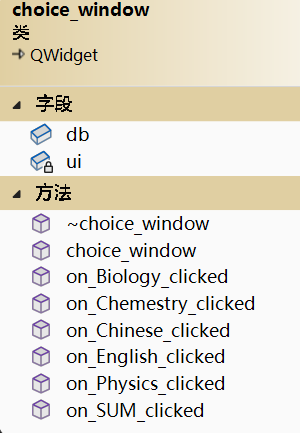
~managerwindow();

private:

Ui::managerwindow \*ui;

};

### 中转界面：



class choice\_window;

}

class choice\_window : public QMainWindow

{

Q\_OBJECT

public:

explicit choice\_window(QString newstudent\_id,bool student\_call=1,QWidget \*parent = nullptr);

QString student\_id;

bool student=1;

~totalscore();

private:

Ui::totalscore \*ui;

private slots:

void on\_Chinese\_clicked();

void on\_Math\_clicked();

void on\_English\_clicked();

void on\_Physics\_clicked();

void on\_Chemestry\_clicked();

void on\_Biology\_clicked();

void on\_SUM\_clicked(); //按点击的学科按钮跳转到不同界面

};

## 数据可视化模块

该模块主要负责将数据库中的数据转化为直方图，分析数据间的相关性，生成一分一段表。包含的文件有:

**Scoredistribution.h/.cpp/.ui 负责绘制一分一段表**

**Paint.h/.cpp/.ui 负责绘制直方图**

**Relationship.h/.cpp/.ui 负责分析线性相关度**

**pcbpaints.h/.cpp/.ui 负责绘制散点图**

**cmepaints.h/.cpp/.ui 负责绘制散点图**

### 一分一段表：

class score\_distribution : public QWidget

{

Q\_OBJECT

public:

explicit score\_distribution(QString new\_subject,QString new\_student\_id,bool student\_call=1,QWidget \*parent = nullptr);

void initchart(QString subject); //初始化函数

void create\_distribution\_chart(QString subject,QString student\_id);//绘图函数

QString student\_id;//学生id

QString subject; //科目

bool student=1; //判断是否为学生窗口唤起

void on\_score\_returnPressed(); //检测输入框回车

~score\_distribution();

private:

Ui::score\_distribution \*ui; //ui组件

};

### 直方图：

class Paint{

public:

Paint(std::vector<int> data,QString subject):data(data),subject(subject){

data1=new int[7];

data2=new int[5];

}

QChart \*createBarChart1(); //创建直方图

QChart \*createBarChart2(); //创建直方图

void getnums1(int\*& Data);//获取成绩数据

void getnums2(int\*& Data);//获取人数数据

private:

std::vector<int> data; //数据

QString subject;//学科

int \*data1; //成绩数据

int \*data2; //学科数据

};

### 线性相关度分析：

class Relationship{

public:

Relationship(std::vector<int>a,std::vector<int>b):a(a),b(b){

Series=nullptr;

}

QChart \* getrelationshipmap(); //绘图函数

QString getrelationship(); //返回线性相关度

private:

//用 a存储语数外成绩之和，b存储物化生成绩之和

std::vector<int>a;

std::vector<int>b;

QLineSeries \*Series; //图线

};

### 绘制散点图：

class PCBpaints : public QWidget

{

Q\_OBJECT

public:

explicit PCBpaints(QString subject,QWidget \*parent = nullptr);

~PCBpaints();

private slots:

void on\_pushButton\_clicked(); //返回按钮

private:

Ui::PCBpaints \*ui;

QString subject; //学科

};

class CMEpaints : public QWidget

{

Q\_OBJECT

public:

explicit CMEpaints(QString subject,QWidget \*parent = nullptr);

~CMEpaints();

private slots:

void on\_pushButton\_clicked(); //返回按钮

private:

Ui::CMEpaints \*ui;

QString subject; //学科

};

## UML类图

## UML类图：

如图，由于采用窗口化的设计方法，各个类之间独立性较强，所有窗口类都是Qmainwindow的子类，其他类之间不存在继承和派生的关系。每个窗口类负责处理自己独立的事件，由其他函数调用。DB类是数据库及其操作函数的封装类，负责从数据库拉取数据，因而被相当多的类调用。

具体成员变量和函数在上文已经给出 考虑到作图简洁性问题，此图仅给出类名和相互关系