



《面向对象程序设计》课程设计

题目：考勤管理系统

年级：23 级计算机

开发组序号：第 15 组

开发组情况：

姓名	学号	班级	分工	自评分
汪钰轩	202311040071	3 班	底层逻辑代码的编写，项目报告书的编写，调试代码，代码后期的完善和维护	50%
宋冠庭	202311040081	3 班	图形化代码的实现， 调试代码，代码后期的完善和维护	50%

指导教师：郜山权

开发日期：2024.5.27

目录

1. 概述.....	3
2. 总体设计.....	4
3. 详细设计.....	5
3.1 登录系统设计.....	5
3.1.1 登录功能.....	5
3.1.2 注册功能.....	12
3.2 程序主界面设计.....	19
3.3 录入考勤信息.....	20
3.3.1 信息 info 类设计.....	20
3.3.2 保存按钮实现.....	23
3.4 修改考勤信息.....	24
3.4.1 删除信息.....	24
3.4.2 修改信息.....	28
3.5 查看出勤信息.....	32
3.5.1 全览功能.....	32
3.5.2 查找功能.....	36
3.6 统计出勤信息.....	40
3.6.1 学生统计.....	40
3.6.2 统计课程.....	45
3.7 学生端页面.....	49
4. 编码设计.....	50
5. 测试时出现过的问题及其解决方法.....	66
6. 总结.....	66

考勤管理系统

汪钰轩、宋冠庭

(燕山大学 信息科学与工程学院)

1. 概述

1.1 目的与意义

本项目的目的是基于 C++ 开发设计一款功能完备的考勤管理系统，通过信息化手段对学生的出勤情况进行高效、准确的记录和管理，提升管理效率、数据准确性和透明度，便于查询统计，增强学生纪律性，并为学校管理层提供科学的决策依据。

1.2 主要完成的任务及解决的主要问题

本项目主要实现了用户密码加密功能、考勤记录管理功能、数据持久化功能和用户友好交互式界面，解决了用户身份验证、数据存储读取、考勤记录统计分析及图形化的交互式界面等问题。

1.3 人员分工

在人员分工上，我组汪钰轩同学负责底层逻辑代码的编写、项目报告书的编写、调试代码、代码后期的完善和维护，宋冠庭同学负责图形化代码的实现、调试代码、代码后期的完善和维护。

1.4 开发计划与开发环境

开发计划主要涵盖了需求分析、系统总体设计、模块内容开发、测试和优化等内容。开发环境为 Visual Studio 2022 和 qtcreator 4.7.0。该系统的实施将显著提升学校考勤管理的效率和准确性，为管理决策提供支持。

2. 总体设计

2.1 总体结构及模块功能

系统总体结构采用模块化设计，主要分为用户管理模块、考勤记录管理模块、数据持久化模块和用户交互模块。

用户管理模块负责用户的注册、登录和权限管理；考勤记录管理模块负责学生出勤数据的录入、修改、查询和统计分析；数据持久化模块负责将用户信息和考勤记录存储到文件中，以确保数据的长期保存和安全；用户交互模块则提供友好的图形化界面，简单易上手，方便用户进行操作和数据查看。

2.2 总体流程

系统的总体流程从用户登录开始，根据用户角色的不同，进入不同的功能界面。老师可以进行考勤记录管理操作，而学生则主要查看个人的考勤记录。各模块之间通过统一的接口进行数据传递和调用，确保系统的高内聚和低耦合，从而提高系统的可维护性和扩展性。

3. 详细设计

3.1 登录系统设计

登录界面如下，用户可以输入其用户名和密码，并确定用户类型是否为老师。

图 3-1 登录页面

3.1.1 登录功能

其中“登录”按钮执行了登录相关操作，其槽函数如下：

```
1. void page_login::on_login_clicked()  
2. {  
3.     // 创建用户对象，用于存储登录信息
```

```

4.         User user(username,password,isteacher);
5.
6.         // 对密码进行SHA-256 加密
7.         password = QString::fromStdString(Ly::Sha256::getInstance().getHexMessageDigest(password.toStdString()));
8.
9.         // 比较文件中存储的用户名和密码与用户输入的用户名和密码是否匹配
10.        if(compareFileContents(USERFILE,username,password,isteacher)){
11.            // 登录成功
12.            QMessageBox::information(nullptr, "Success", "登录成功! ");
13.            // 如果是老师用户, 则显示管理页面, 否则显示学生页面
14.            if(isteacher){
15.                this->hide();
16.                page_manage.show();
17.            }else{
18.                this->hide();
19.                page_stu.show();
20.            }
21.            // 清空用户名和密码
22.            username.clear();
23.            password.clear();
24.        }else{
25.            // 登录失败, 显示警告信息
26.            QMessageBox::warning(nullptr, "登录失败", "用户名或密码不正确!");
27.            // 清空用户名和密码
28.            username.clear();
29.            password.clear();
30.        }
31.    }

```

流程图如下：

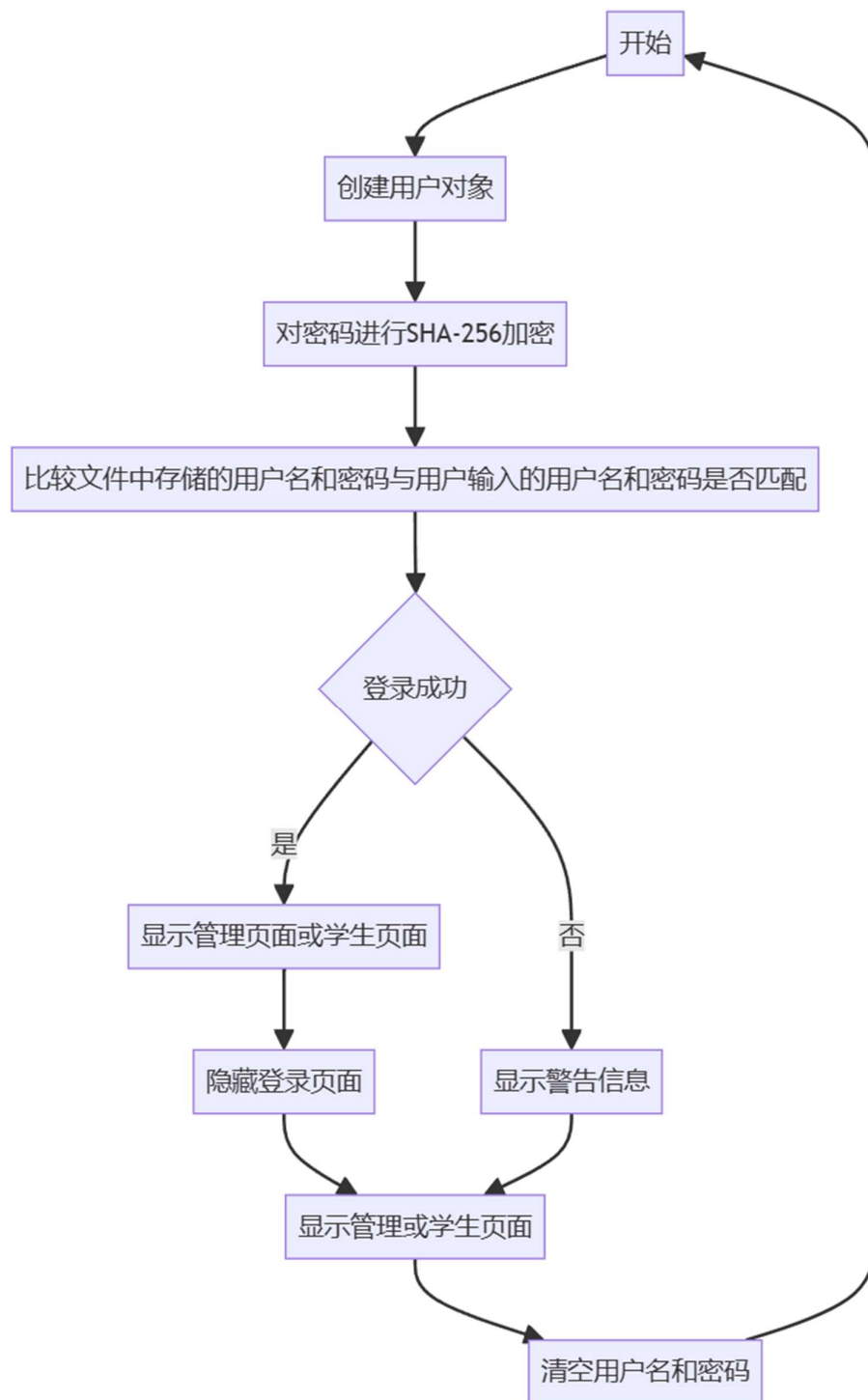


图 3-2 登录流程

登录过程中执行了比较文件中存储的用户名和密码与用户输入的用户名和密码是否匹配的操作，即函数：

```
1.    bool compareFileContents(const QString &filePath, const QString &username, const QString &password, bool isteacher)
```

其中：

- filepath 代表要比较的文件路径
- username 代表要检查的用户名
- password 代表要检查的密码
- isteacher 代表用户是否为老师的布尔值
- 函数如果找到匹配项则返回 true，否则返回 false

下面是该函数的具体实现：

```
1.    /**
2.     * @brief 比较文件内容与提供的用户名、密码以及用户类型是否匹配。
3.     * @param filePath 要比较的文件路径。
4.     * @param username 要检查的用户名。
5.     * @param password 要检查的密码。
6.     * @param isteacher 表示用户是否为教师的布尔值。
7.     * @return 如果找到匹配项则返回 true，否则返回 false。
8.     */
9.    bool compareFileContents(const QString &filePath, const QString &username, const QString &password, bool isteacher) {
10.        QFile file(filePath); // 创建一个 QFile 对象，用于操作文件
11.        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) { // 打开
                                文件进行只读文本模式的读取
12.            qDebug() << "无法打开文件进行读取：
                                " << file.errorString(); // 如果文件打开失败，输出错误信息
```



```

13.         return false; // 返回失败
14.     }
15.     QTextStream in(&file); // 创建 QTextStream 对象, 用于读取文件内容
16.     while (!in.atEnd()) { // 循环读取文件直到结束
17.         QString line = in.readLine(); // 读取文件中的一行内容
18.         QStringList parts = line.split(' '); // 以空格分割一行内容为多个部分
19.         if (parts.size() != 3) { // 如果分割后的部分数量不等于3
20.             qDebug() << "文件中的行格式无效: " << line; // 输出无效行的信息
21.             continue; // 继续下一次循环
22.         }
23.
24.         QString fileUsername = parts[0]; // 获取分割后的第一个部分, 即用户名
25.         QString filePassword = parts[1]; // 获取分割后的第二个部分, 即密码
26.         bool fileIsTeacher = parts[2].toInt(); // 获取分割后的第三个部分, 将其转换为布尔型
27.
28.         if (fileUsername == username && filePassword == password && fileIsTeacher == isteacher) { // 如果用户名、密码和教师状态都匹配
29.             qDebug() << "文件中找到匹配项! "; // 输出匹配项信息
30.             return true; // 返回成功

```

```
31.         }
32.     }
33.
34.     qDebug() << "文件中未找到匹配项。"; // 输出未找到匹配项的信息
35.     return false; // 返回失败
36.
37. }
```

该函数的流程图如下：

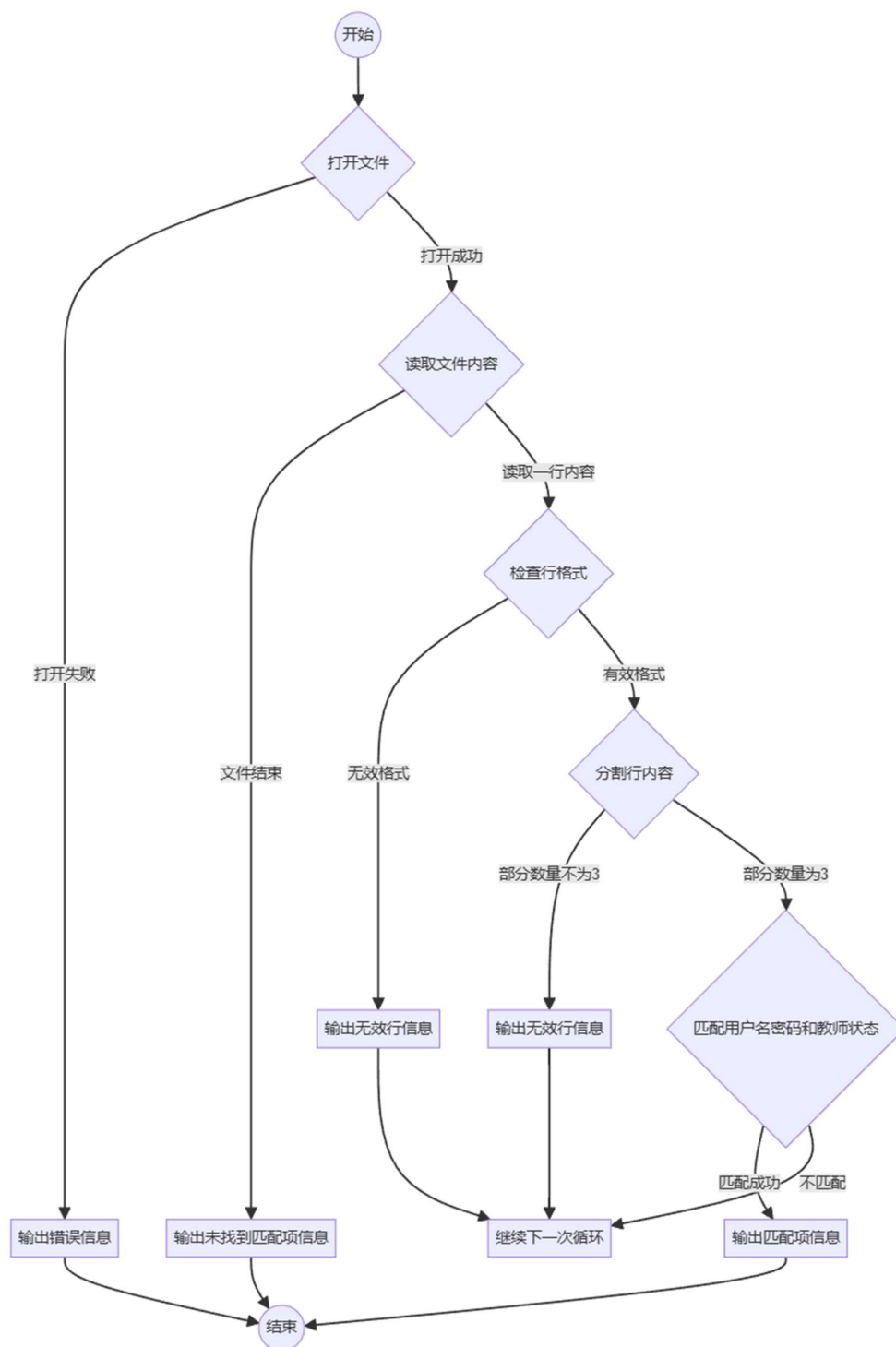


图 3-3 账户信息比对函数

3.1.2 注册功能

注册按钮执行了注册相关操作，其槽函数如下：

```
1. void page_login::on_Register_clicked()
2. {
3.     // 创建用户对象，用于存储注册信息
4.     User user(username,password,isteacher);
5.
6.     // 检查用户名是否已存在
7.     if(!isUsernameExists(USERFILE,username)){
8.         // 打开用户文件以追加方式写入新用户信息
9.         QFile file(USERFILE);
10.        if (!file.open(QIODevice::Append | QIODevice::Text)) {
11.            // 打开文件失败，输出错误信息并返回
12.            qDebug() << "Failed to open file for appending:" << file
13.                .errorString();
14.            return;
15.        }
16.        // 使用 QTextStream 写入新用户信息到文件中
17.        QTextStream out(&file);
18.        out << username << ' ' << QString::fromStdString(Ly::Sha256:
19.            :getInstance().getHexMessageDigest(password.toStdString())) << ' ' << is
20.            teacher << Qt::endl;
21.
22.        // 关闭文件
23.        file.close();
24.
25.        // 注册成功，显示成功信息提示框，并根据用户类型显示相应的页面
26.        QMessageBox::information(nullptr, "Success", "注册成功! ");
27.        if(isteacher){
28.            this->hide();
```

```
27.         page_manage.show();
28.     }else{
29.         this->hide();
30.         page_stu.show();
31.     }
32.
33.         // 清空用户名和密码
34.         username.clear();
35.         password.clear();
36.     }else{
37.         // 用户名已存在, 显示注册失败的警告信息
38.         QMessageBox::warning(nullptr, "注册失败", "用户名已经存在! ");
39.
40.         // 清空用户名和密码
41.         username.clear();
42.         password.clear();
43.     }
44. }
```

流程图如下：

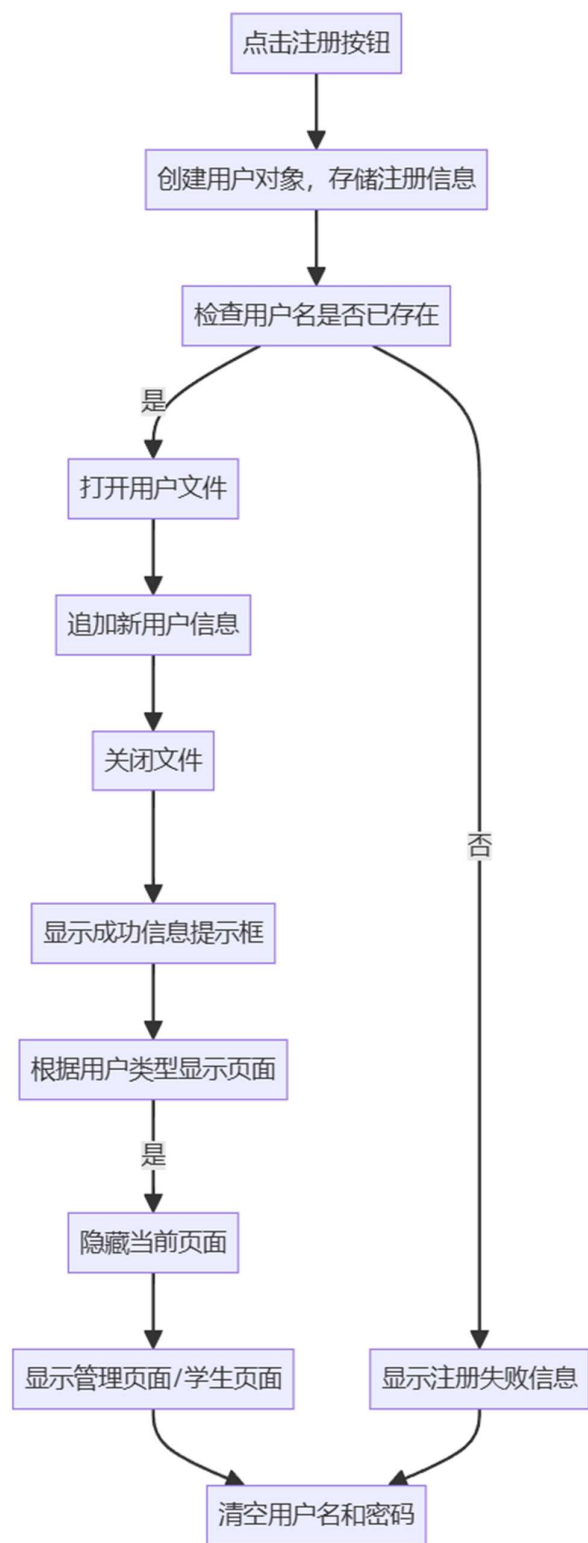


图 3-4 注册流程

在注册操作中，执行了检查给定用户名是否已经存在于文件中的操作，即函数：

```
1.     bool isUsernameExists(const QString &filePath, const QString &username)
```

其中：

- filePath 代表要检查的文件路径
- username 代表要检查的用户名
- 如果用户名存在于文件中则返回 true，否则返回 false

下面是该函数的具体实现：

```
1.     /**
2.      * @brief 检查给定用户名是否存在于文件中。
3.      *
4.      * @param filePath 要检查的文件路径。
5.      * @param username 要检查的用户名。
6.      * @return 如果用户名存在于文件中则返回 true，否则返回 false。
7.      */
8.     bool isUsernameExists(const QString &filePath, const QString &username) {
9.         QFile file(filePath); // 创建一个 QFile 对象，用于操作文件
10.        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) { // 打开
11.            // 文件进行只读文本模式的读取
12.            qDebug() << "无法打开文件进行读取：
13.            " << file.errorString(); // 如果文件打开失败，输出错误信息
14.            return false; // 返回失败
15.        }
16.        QTextStream in(&file); // 创建 QTextStream 对象，用于读取文件内容
```

```

16.         while (!in.atEnd()) { // 循环读取文件直到结束
17.             QString line = in.readLine(); // 读取文件中的一行内容
18.             QStringList parts = line.split(' '); // 以空格分割一行内容为多个部分
19.             if (parts.size() != 3) { // 如果分割后的部分数量不等于3
20.                 qDebug() << "文件中的行格式无效: " << line; // 输出无效行的信息
21.                 continue; // 继续下一次循环
22.             }
23.
24.             QString fileUsername = parts[0]; // 获取分割后的第一个部分, 即用户名
25.             if (fileUsername == username) { // 如果文件中的用户名与传入的用户名相同
26.                 qDebug() << "用户名存在于文件中! "; // 输出用户名存在的信息
27.                 file.close(); // 关闭文件
28.                 return true; // 返回成功
29.             }
30.         }
31.         qDebug() << "用户名不存在于文件中。"; // 输出用户名不存在的信息
32.         file.close(); // 关闭文件
33.         return false; // 返回失败
34.     }

```

流程图如下:

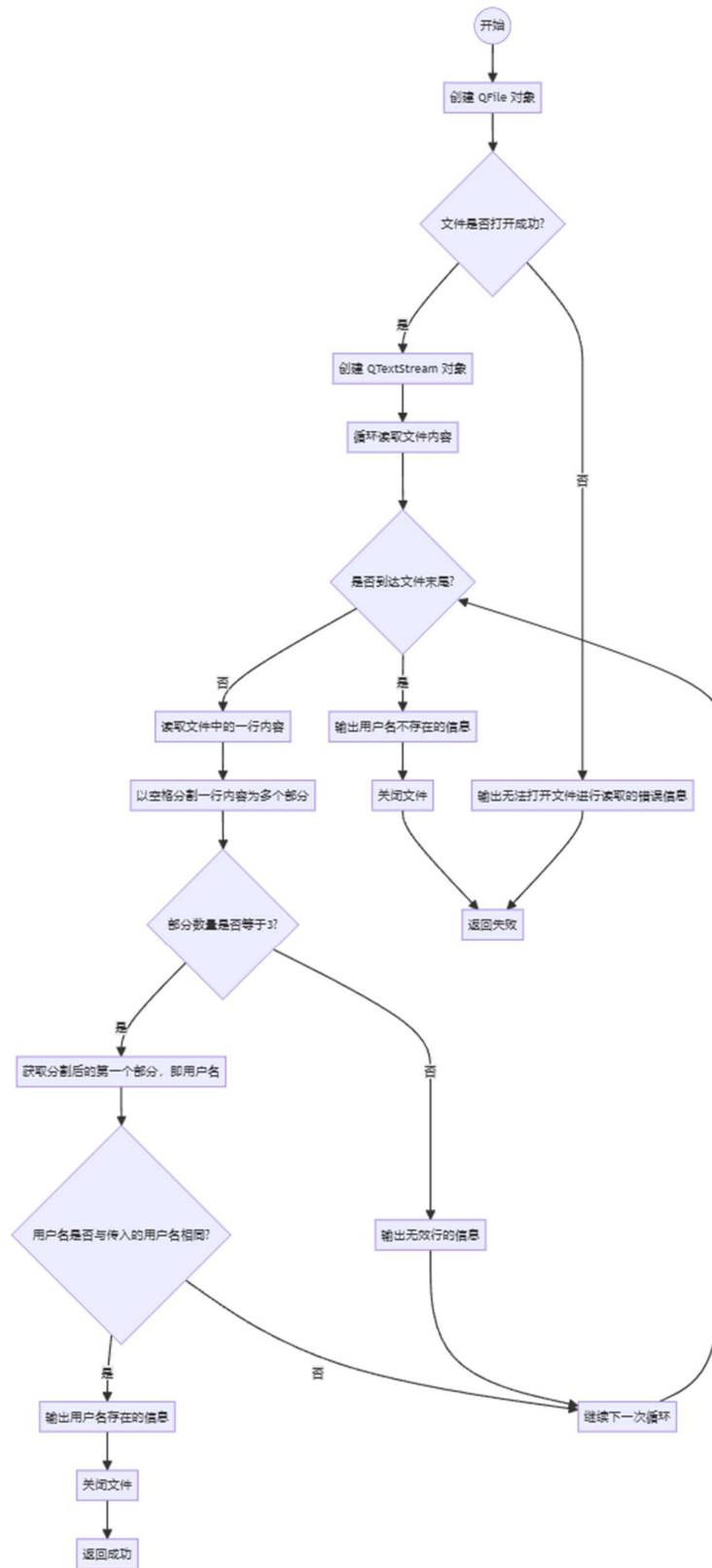


图 3-5 判断用户是否已经存在

本系统对用户输入的密码进行了 SHA-256 加密，使原密码对任何人不可见，大大提高了系统的安全性。

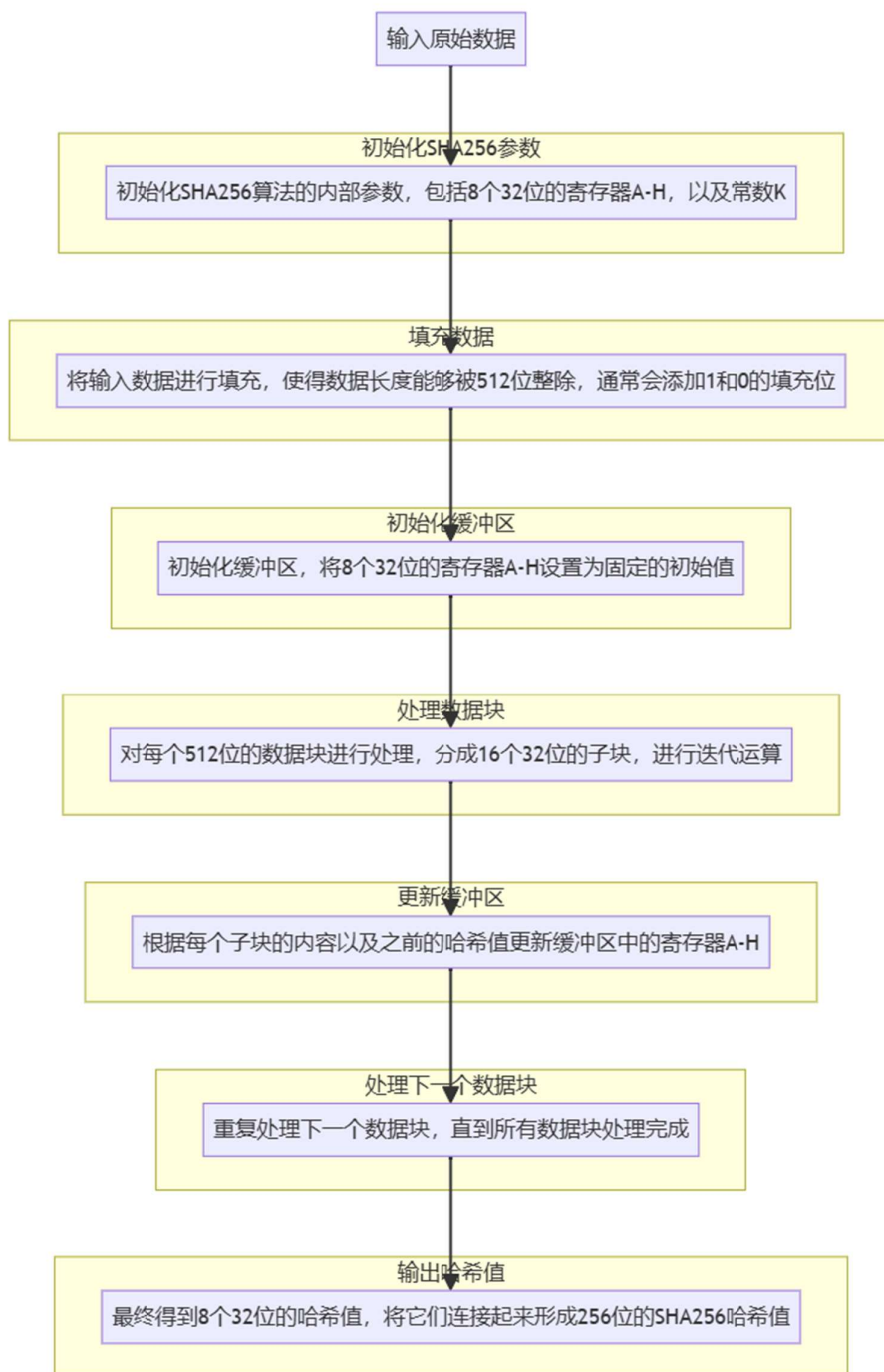


图 3-6 SHA-256 加密流程

3.2 程序主界面设计



图 3-7 主界面

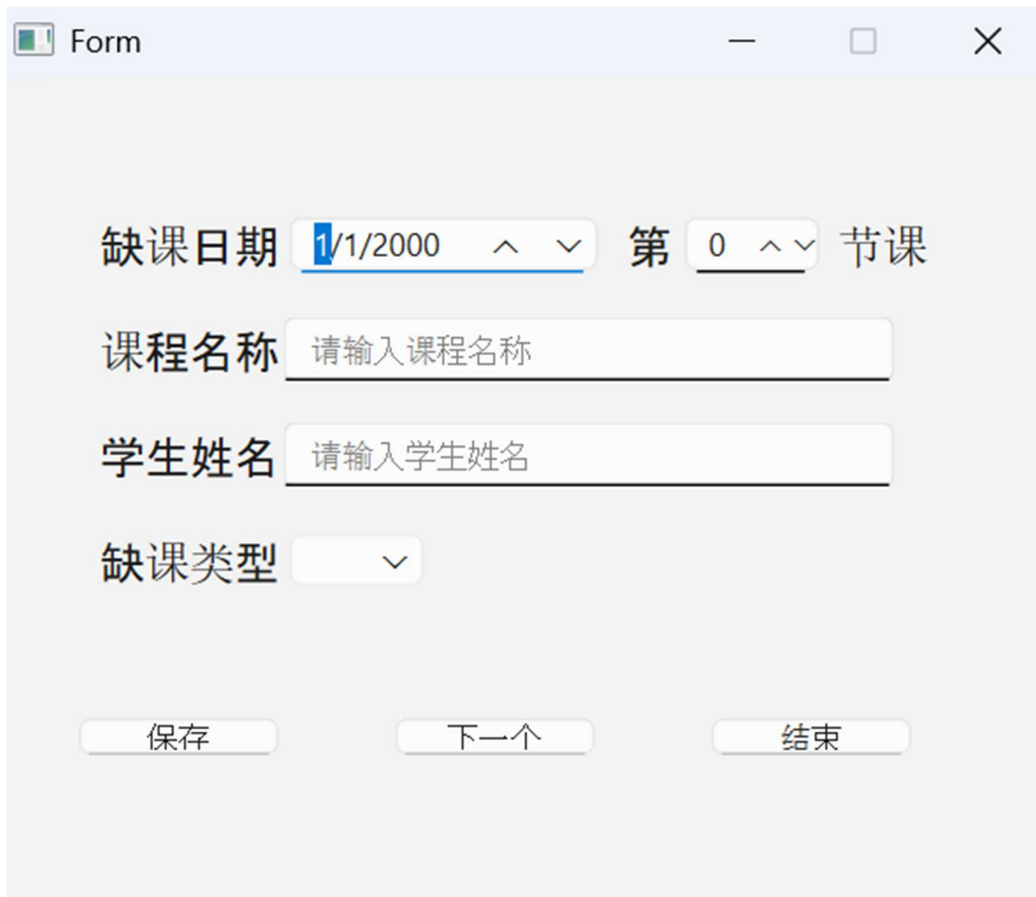
程序的主界面包含了四个按钮：

- 录入考勤信息
- 修改考勤信息
- 查询考勤信息
- 统计考勤信息

分别对应了四个功能：

- 录入信息
- 删除和修改信息
- 查看和查找信息
- 统计信息

3.3 录入考勤信息



Form

缺课日期 1/1/2000 ^ v 第 0 ^ v 节课

课程名称 请输入课程名称

学生姓名 请输入学生姓名

缺课类型 v

保存 下一个 结束

图 3-8 信息录入界面

3.3.1 信息 info 类设计

Info 类声明如下：

```
1. class info
2. {
3. public:
4.     int id;
5.     QString date;
6.     int ClassNum;
7.     QString ClassName;
8.     QString StuName;
9.     QString Method;
```

```

10.         info(int id, QString date, int ClassNum, QString ClassName, QStr
            ing StuName, QString Method): id(id), date(date), ClassNum(ClassNum), Cl
            assName(ClassName), StuName(StuName), Method(Method){}
11.         info(){};
12.
13.         void tofile(const QString &filepath);
14.
15.
16.
17.         ~info()
18.         {
19.             qDebug() << "Destructor called" << Qt::endl;
20.         }
21.     };

```

其中，info 类包括以下元素：

- int id : 信息的序号
- QString date : 日期
- int ClassNum : 课时
- QString ClassName: 课程名称
- QString StuName : 姓名
- QString Method : 缺勤方式
- void tofile(const QString &filepath);

对于函数：

```

1.         void tofile(const QString &filepath);

```

该函数实现了将对象 info 内的数据格式化保存在文件中的功能，具体实现如下：

```

1.         /**
2.          * @brief 将信息写入文件
3.          *
4.          * 将信息写入指定文件中。信息格式为：
5.          *
            id date ClassNum ClassName StuName Method。

```

```

6.      * @param filepath 文件路径
7.      */
8.      void info::tofile(const QString &filepath){
9.          // 打开文件
10.         QFile file(filepath);
11.         if (!file.open(QIODevice::Append | QIODevice::Text)) {
12.             qDebug() << "Failed to open file for appending:" << file.errorString();
13.             return;
14.         }
15.
16.         // 创建文本流
17.         QTextStream out(&file);
18.
19.         // 将信息写入文件
20.         out << this->id << ' ' << this->date << ' ' << this->ClassNum <<
            ' ' << this->ClassName << ' ' << this->StuName << ' ' << this->Method <
            < Qt::endl;
21.
22.         // 关闭文件
23.         file.close();
24.     }

```

该函数的流程图如下：

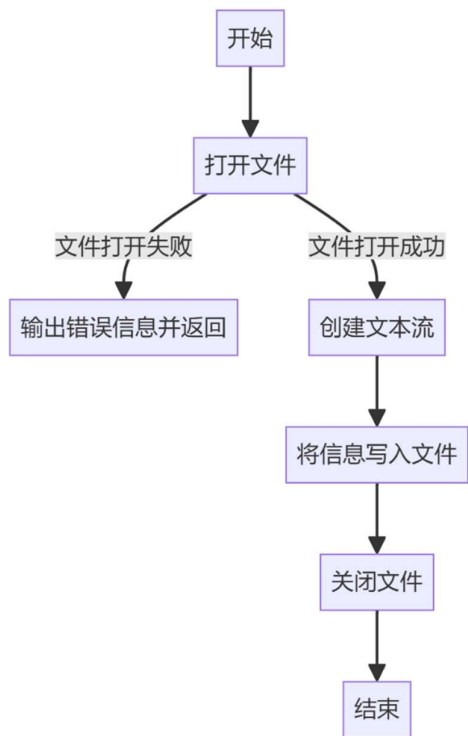


图 3-9 信息保存函数

3.3.2 保存按钮实现

用户在点击保存按钮时，执行以下操作：

- 创建一个 `info` 对象
- 通过 `tofile()` 函数将该对象中的数据保存在 `info.txt` 中

具体实现如下：

```
1. void input::on_save_released()
2. {
3.     info info(id_s, date_s, ClassNum_s, ClassName_s, StuName_s, Method_s);
4.     info.tofile(QString(INFOPATH));
5. }
```

3.4 修改考勤信息

	ID	日期	课时	课程名称	姓名	缺课类型
1	1	30/07/2024	2	fg	shsfdh	旷课
2	2	30/07/2024	3	x'fg'd'r'fj'h	d'hj'fg'd'j	旷课
3	3	01/01/2002	1	111	111	旷课

图 3-10 修改考勤信息

3.4.1 删除信息

用户在点击删除按钮时，程序根据用户事先选择的要删除的信息 ID 删除 info.txt 中 ID 相同的信息，并重新计算文件中所有信息的 ID。

具体实现如下：

```
1. void change::on_delet_clicked()
2. {
3.     if(confirmOperation(this)){
4.         deleteRowAndRecomputeId(QString(INFOPATH), id_c);
5.         fillTableFromTextFile(ui->tableWidget,QString(INFOPATH));
6.         ui->id->clear();
7.     }
8. }
```

对于函数：


```
1. void deleteRowAndRecomputeId(const QString &filename, int idToDelete
    )
```

其中:

- const QString &filename 代表要操作的文件路径
- int idToDelete 代表要删除的行的 ID

下面是该函数的具体实现:

```
1. /**
2.  * @brief 删除指定 ID 的行并重新计算 ID
3.  * 从指定的文件中删除具有指定 ID 的行，并重新计算剩余行的 ID。
4.  * @param filename 要操作的文件路径
5.  * @param idToDelete 要删除的行的 ID
6.  */
7. void deleteRowAndRecomputeId(const QString &filename, int idToDelete
    ) {
8.     // 创建一个 QFile 对象，使用提供的文件路径。
9.     QFile file(filename);
10.    // 如果文件无法以读写文本模式打开，则直接返回。
11.    if (!file.open(QIODevice::ReadWrite | QIODevice::Text))
12.        return;
13.    // 创建一个字符串列表，用于存储文件中的每一行数据。
14.    QStringList lines;
15.    // 创建一个与文件关联的 QTextStream 对象。
16.    QTextStream in(&file);
17.    // 循环直到到达文件结尾。
18.    while (!in.atEnd()) {
19.        // 从文件中读取一行并将其存储在 'line' 变量中，然后添加
        // 到 'lines' 列表中。
```

```

20.         QString line = in.readLine();
21.         lines.append(line);
22.     }
23.     file.resize(0);
24.     // 初始化当前ID 为1。
25.     int currentId = 1;
26.     // 创建一个与文件关联的 QTextStream 对象, 用于写入新的行数据。
27.     QTextStream out(&file);
28.     // 遍历 'lines' 列表中的每一行。
29.     for (const QString &line : lines) {
30.         // 使用空格作为分隔符将该行分割成字段, 并将它们存储在 'fields' 列
        表中。
31.         QStringList fields = line.split(' ');
32.         // 如果字段列表为空, 则继续到下一行。
33.         if (fields.isEmpty())
34.             continue;
35.         // 将该行的第一个字段 (ID) 转换为整数类型。
36.         int id = fields[0].toInt();
37.         // 如果该行的 ID 不等于要删除的ID, 则将其添加到文件中, 并更新其 ID
        为当前ID 值。
38.         if (id != idToDelete) {
39.             fields[0] = QString::number(currentId++);
40.             out << fields.join(' ') << '\n';
41.         }
42.     }
43.     // 关闭文件。
44.     file.close();
45. }

```

流程图如下:

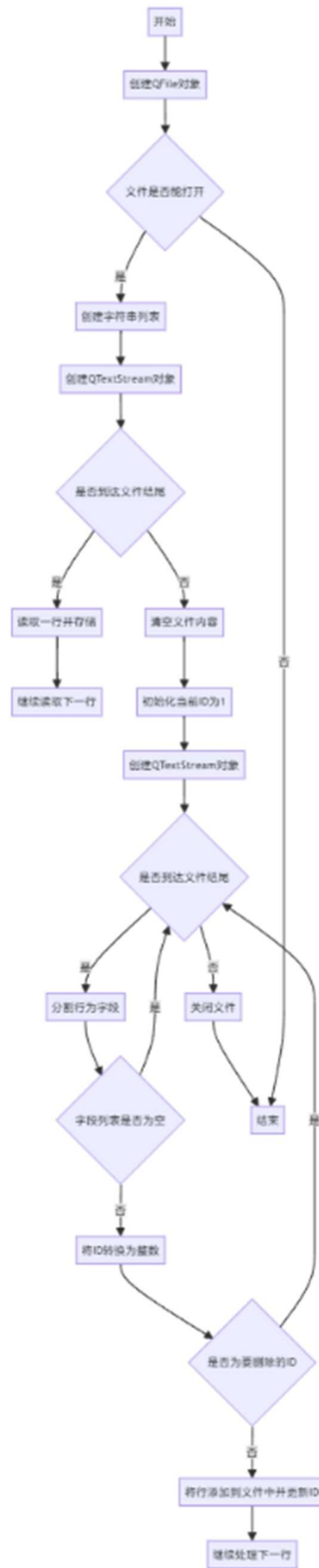


图 3-11 删除信息流程

3.4.2 修改信息

The screenshot shows a Qt application window titled "Form". It contains a table with the following data:

ID	日期	课时	课程名称	姓名	缺课类型
1 1	30/07/2024	2	fg	shsfdh	旷课
2 2	30/07/2024	3	xfa'd'cf'fh	dh'f'fa'd'i	旷课
3 3	01/01/2002	1			

Below the table is a modal dialog box titled "Form" for editing a record. It contains the following fields and controls:

- ID: A dropdown menu showing "0" and a "加载" (Load) button.
- 缺课日期: A date picker showing "1/1/2000" and a "第" (No.) dropdown showing "0" followed by "节课" (class).
- 课程名称: A text input field with the placeholder "请输入课程名称".
- 学生姓名: A text input field with the placeholder "请输入学生姓名".
- 缺课类型: A dropdown menu.
- Buttons: "确定" (OK) and "取消" (Cancel).

图 3-12 修改信息页面

在该功能上，用户应先输入想要修改的信息 ID，后点击加载按钮，则该条信息就会被自动填入下面的控件，用户可按需进行修改。

下面是加载按钮的触发函数：

```
1. void changeInfo::on_load_released()
2. {
3.     qDebug()<< id_i << Qt::endl;
4.     info = getInfoById(QString(INFOPATH), id_i);
5.     qDebug()<< info.date << Qt::endl;
6.     qDebug()<< QDate::fromString(info.date, "dd/MM/yyyy") << Qt::endl
7.     ;
8.     ui->date_c->setDate(QDate::fromString(info.date, "dd/MM/yyyy"));
9.     ui->classnum_c->setValue(info.ClassNum);
10.    ui->classname_c->setText(info.ClassName);
11.    ui->stuname_c->setText(info.StuName);
12.    ui->method_c->setCurrentText(info.Method);
13. }
```

加载按钮通过函数

```
1.     info getInfoById(const QString &filename, int idToFind)
```

实现在 info.txt 中通过 id 查找信息。

其中：

- const QString &filename 代表要搜索的文件路径
- int idToFind 代表要查找的信息 ID

具体实现如下：

```
1.     /**
2.      * @brief 通过 ID 从文件中获取信息
3.      *
4.      * 从指定的文件中查找具有指定 ID 的信息，并返回对应的 info 对象。
5.      * 如果找不到对应 ID 的信息，则返回一个默认构造的 info 对象。
6.      *
7.      * @param filename 要搜索的文件路径
8.      * @param idToFind 要查找的信息的 ID
9.      * @return 包含指定 ID 信息的 info 对象，如果找不到则返回默认构造的 info 对
      象
10.     */
11.     info getInfoById(const QString &filename, int idToFind) {
12.         // 创建一个 QFile 对象，使用提供的文件路径。
13.         QFile file(filename);
14.         // 如果文件无法以只读文本模式打开，则返回默认构造的 info 对象。
15.         if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
16.             return info(0, "", 0, "", "", "");
17.         // 创建一个与文件关联的 QTextStream 对象。
18.         QTextStream in(&file);
```

```

19.      // 循环直到到达文件结尾。
20.      while (!lin.atEnd()) {
21.          // 从文件中读取一行并将其存储在 'line' 变量中。
22.          QString line = in.readLine();
23.          // 使用空格作为分隔符将该行分割成字段，并将它们存储在 'fields' 列表
          表中。
24.          QStringList fields = line.split(' ');
25.          // 如果字段列表为空，则继续到下一行。
26.          if (fields.isEmpty())
27.              continue;
28.          // 将该行的第一个字段 (ID) 转换为整数类型。
29.          int id = fields[0].toInt();
30.          // 如果该行的 ID 等于要查找的 ID，则根据字段创建一个 info 对象并返回。
31.          if (id == idToFind) {
32.              return info(id,
33.                          fields[1],
34.                          fields[2].toInt(),
35.                          fields[3],
36.                          fields[4],
37.                          fields[5]);
38.          }
39.      }
40.      // 如果找不到对应 ID 的信息，则返回默认构造的 info 对象。
41.      return info(0, "", 0, "", "", "");
42.  }

```

流程图如下：

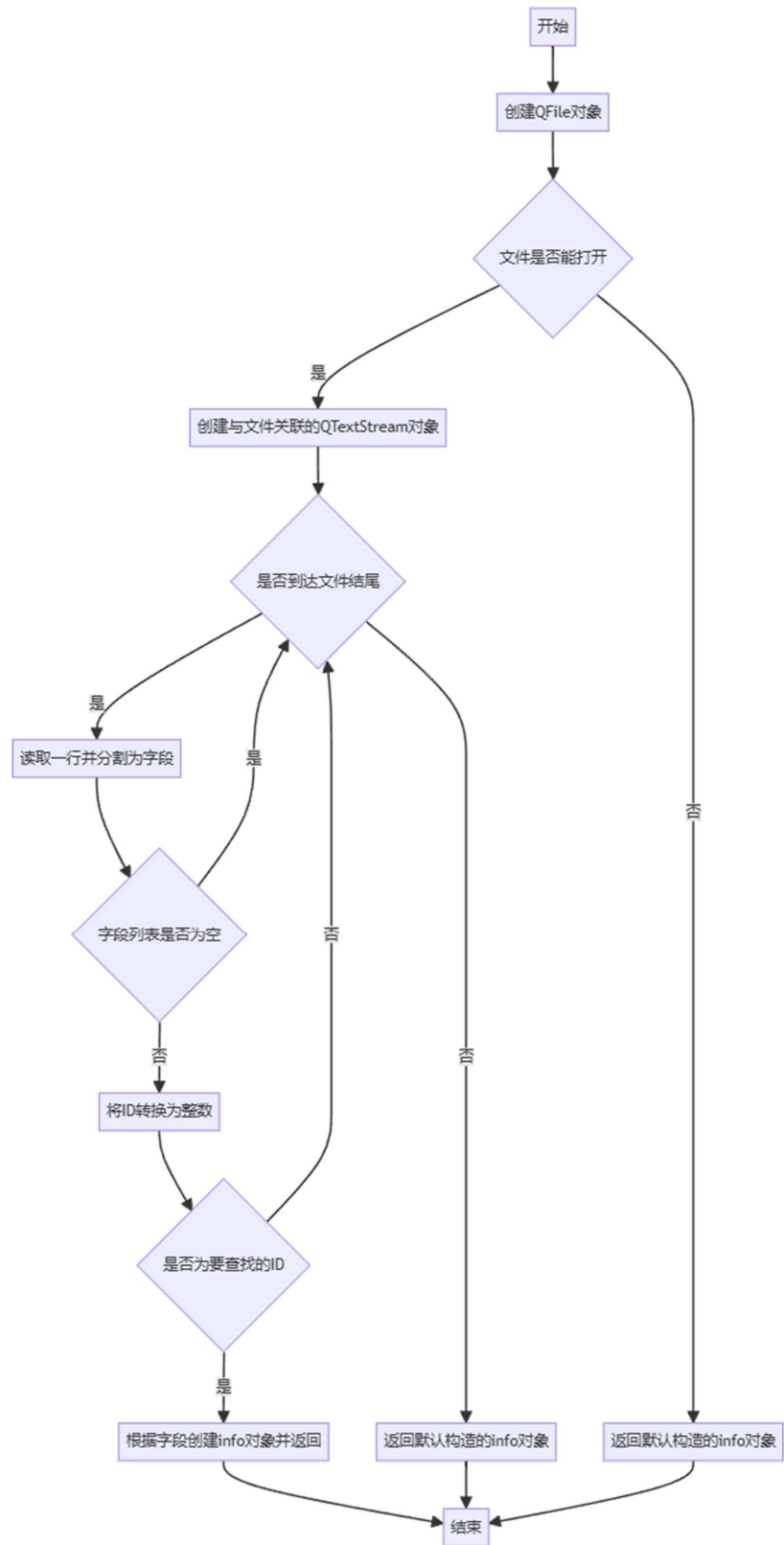
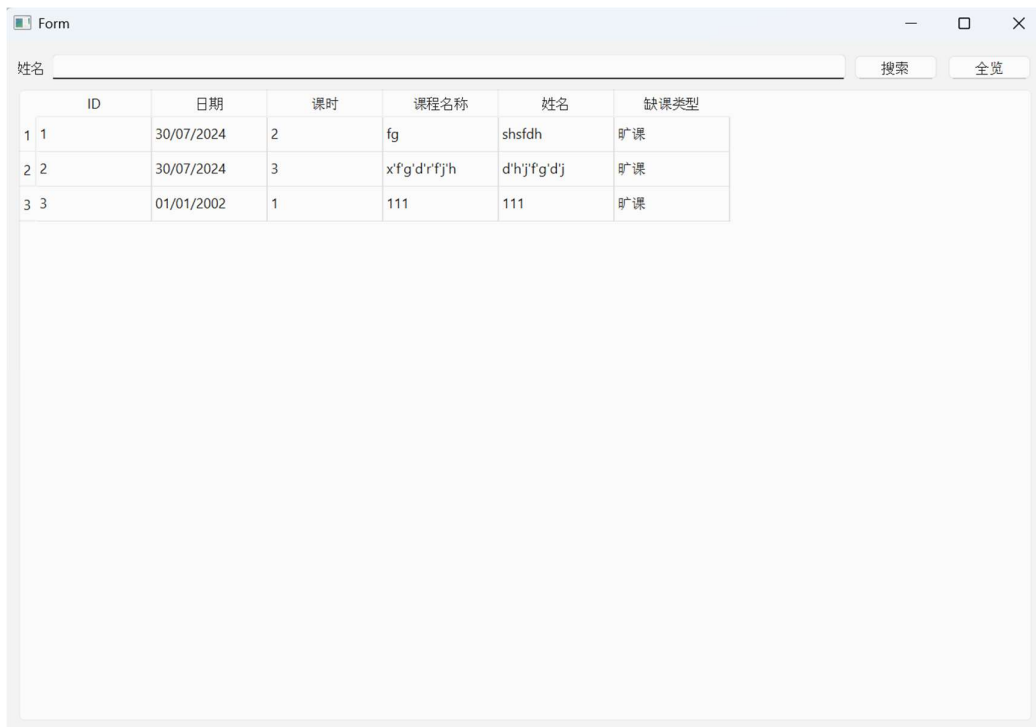


图 3-13 通过 ID 查找信息

3.5 查看出勤信息



	ID	日期	课时	课程名称	姓名	缺课类型
1	1	30/07/2024	2	fg	shsfdh	旷课
2	2	30/07/2024	3	x'fg'd'r'fj'h	d'h'j'fg'd'j	旷课
3	3	01/01/2002	1	111	111	旷课

图 3-14 查询界面

3.5.1 全览功能

用户通过点击全览按钮进行全部信息的浏览，具体实现如下：

```
1. void view::on_view_t_released()  
2. {  
3.     fillTableFromTextFile(ui->tableWidget,QString(INFOPATH));  
4. }
```

其中，函数

```
1. void fillTableFromTextFile(QTableWidget *tableWidget, const QString  
    &filename)
```

中：

- QTableWidget *tableWidget 代表要填充数据的 QTableWidget 指针、
- const QString &filename 代表要读取数据的文本文件路径

具体实现如下：

```
1.  /**
2.   * @brief 从文本文件中填充表格
3.   *
4.   * 从指定的文本文件中读取数据，并将数据填充到给定的表格中。
5.   * 每行数据应包含 6 个字段：ID、日期、课时、课程名称、姓名、缺课类型。
6.   *
7.   * @param tableWidget 要填充数据的 QTableWidgetItem 指针
8.   * @param filename 要读取数据的文本文件路径
9.   */
10. void fillTableFromTextFile(QTableWidgetItem *tableWidget, const QString
    &filename) {
11.     // 创建一个 QFile 对象，使用提供的文件路径。
12.     QFile file(filename);
13.     // 如果文件无法以只读文本模式打开，则直接返回。
14.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
15.         return;
16.
17.     // 清空表格内容，设置行数为 0，列数为 6。
18.     tableWidget->clear();
19.     tableWidget->setRowCount(0);
20.     tableWidget->setColumnCount(6);
21.
22.     // 设置表格的水平表头标签。
23.     QStringList headers;
24.     headers << "ID" << "日期" << "课时" << "课程名称" << "姓名" << "缺
        课类型";
25.     tableWidget->setHorizontalHeaderLabels(headers);
26. }
```

```

27.      // 创建一个与文件关联的 QTextStream 对象。
28.      QTextStream in(&file);
29.      int row = 0;
30.      // 循环直到到达文件结尾。
31.      while (!in.atEnd()) {
32.          // 从文件中读取一行并将其存储在 'line' 变量中。
33.          QString line = in.readLine();
34.          // 使用空格作为分隔符将该行分割成字段，并将它们存储在 'fields' 列
          表中。
35.          QStringList fields = line.split(' ');
36.          // 如果字段数大于等于6，则将该行数据插入到表格中。
37.          if (fields.size() >= 6) {
38.              tableWidget->insertRow(row);
39.              for (int column = 0; column < 6; ++column) {
40.                  // 创建一个新的 QTableWidgetItem 对象，将字段值设置为该项
                      的文本。
41.                  QTableWidgetItem *item = new QTableWidgetItem(fields
                      [column]);
42.                  // 将该项添加到表格的指定行和列。
43.                  tableWidget->setItem(row, column, item);
44.              }
45.              ++row;
46.          }
47.      }
48.
49.      // 关闭文件。
50.      file.close();
51.  }

```

流程图如下：

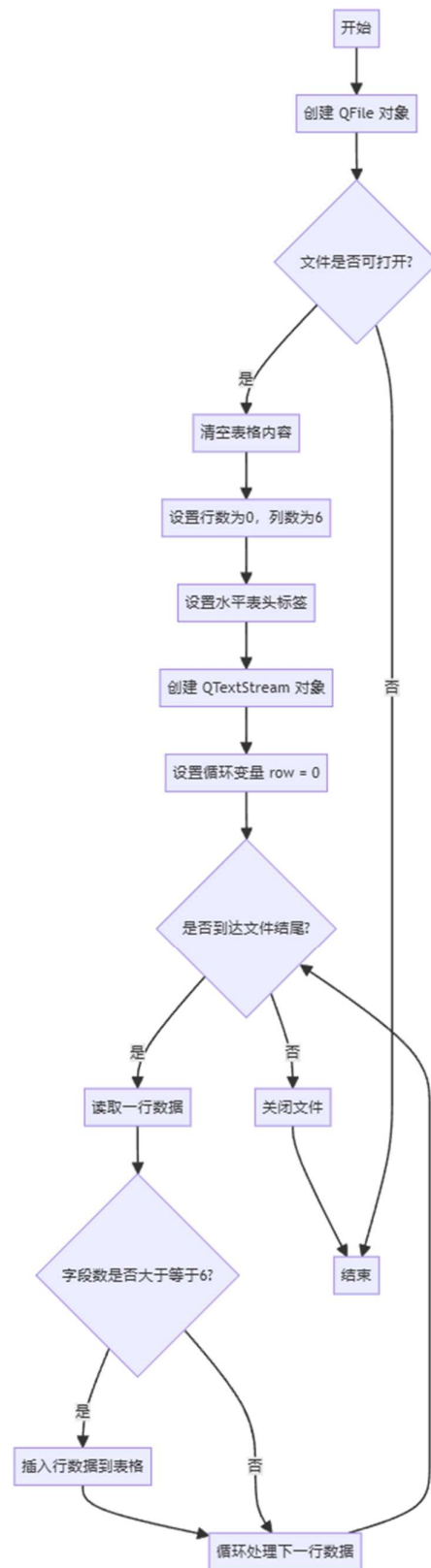


图 3-15 全览流程

3.5.2 查找功能

用户可以通过输入姓名来查找信息。

其中，搜索按钮的触发函数如下：

```
1. void view::on_search_v_released()  
2. {  
3.     qDebug() << name_v << Qt::endl;  
4.     loadTableData(QString(INFOPATH), name_v, ui->tableWidget);  
5. }
```

搜索操作调用了函数：

```
1. void loadTableData(const QString &filePath, const QString &name, QTableWidget *tableWidget)
```

其中：

- const QString &filePath 代表要读取数据的文件路径
- const QString &name 代表要筛选的姓名
- QTableWidget *tableWidget 代表要填充数据的 QTableWidget 指针

该函数的具体实现如下：

```
1. /**  
2.  * @brief 加载表格数据  
3.  *  
4.  * 从指定的文件中加载数据到表格中，仅加载指定姓名的行数据。  
5.  *  
6.  * @param filePath 要读取数据的文件路径  
7.  * @param name 要筛选的姓名  
8.  * @param tableWidget 要填充数据的 QTableWidget 指针  
9.  */
```

```

10. void loadTableData(const QString &filePath, const QString &name, QTa
    bleWidget *tableWidget) {
11.     // 创建一个 QFile 对象, 使用提供的文件路径。
12.     QFile file(filePath);
13.     // 如果文件无法以只读文本模式打开, 则打印错误信息并返回。
14.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
15.         qDebug() << "Cannot open file for reading";
16.         return;
17.     }
18.     // 设置表格行数为 0。
19.     tableWidget->setRowCount(0);
20.     // 创建一个与文件关联的 QTextStream 对象。
21.     QTextStream in(&file);
22.     int row = 0;
23.     // 循环直到到达文件结尾。
24.     while (!in.atEnd()) {
25.         // 从文件中读取一行并将其存储在 'line' 变量中。
26.         QString line = in.readLine();
27.         // 使用空格作为分隔符将该行分割成字段, 并将它们存储在 'fields' 列
            表中。
28.         QStringList fields = line.split(' ');
29.         // 如果字段数为 6 且第 5 个字段 (姓名) 等于指定姓名, 则将该行数据添加
            到表格中。
30.         if (fields.size() == 6 && fields[4] == name) {
31.             // 插入一行到表格中。
32.             tableWidget->insertRow(row);
33.             // 遍历该行的字段并将每个字段添加到表格对应位置。
34.             for (int col = 0; col < fields.size(); ++col) {
35.                 tableWidget->setItem(row, col, new QTableWidgetItem(
                    fields[col]));

```

```
36.         }
37.         row++;
38.     }
39. }
40. // 关闭文件。
41. file.close();
42. }
```

流程图如下：

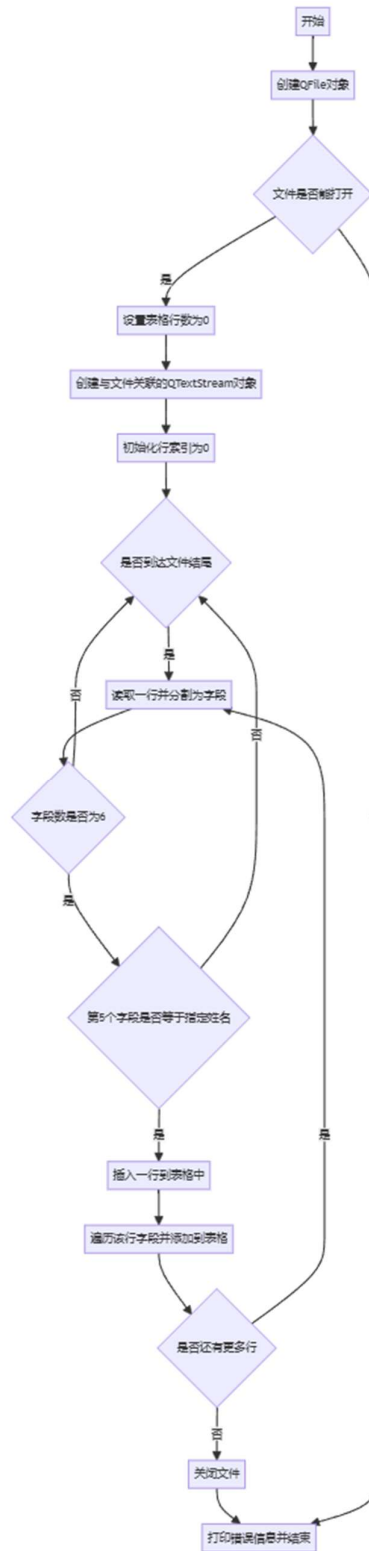


图 3-16 姓名筛选

3.6 统计出勤信息

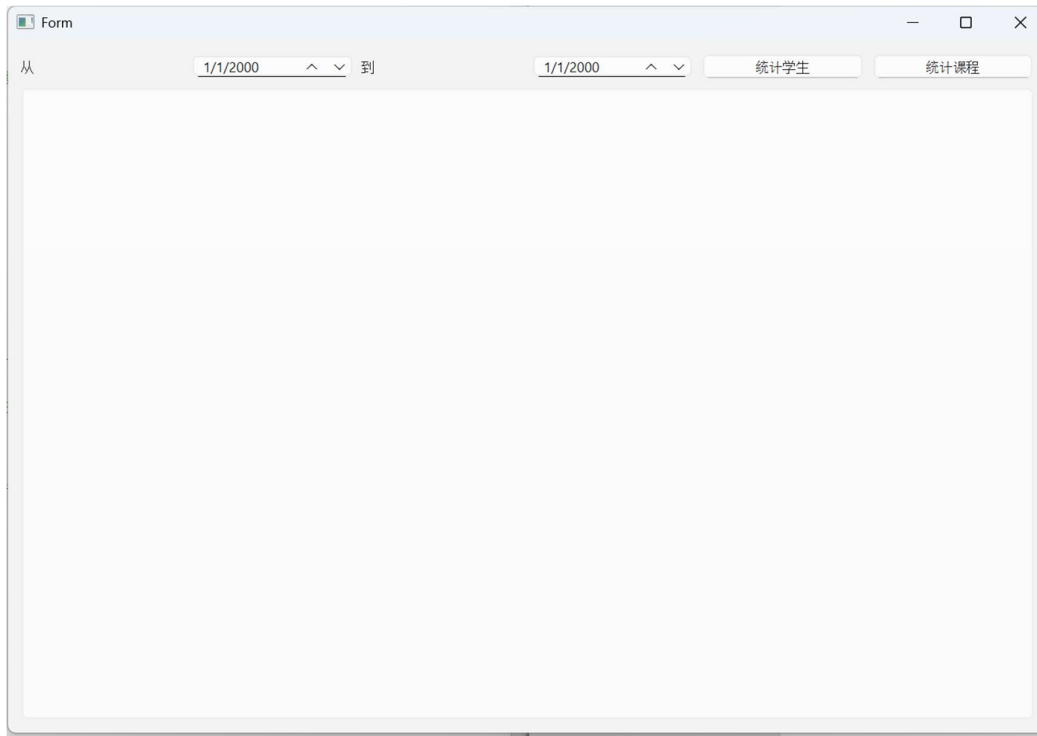


图 3-17 统计页面

统计页面实现了在某一时间区间内统计每个学生缺勤次数和每门课程缺勤人次并降序排列在表格中显示的功能。

用户必须指定时间区间才可以使用此功能

3.6.1 学生统计

“统计学生”按钮的触发函数如下：

```
1. void statistics::on_pushButton_released()
2. {
3.     if(fromischange && toischange){
4.         countNamesInTXT(ui->tableWidget, QString(INFOPATH), from, to
5.         );
6.         fromischange = false;
7.         toischange = false;
8.     }
9.     else
```



```

9.         QMessageBox::critical(nullptr, "错误", "请设置起止时间");
10.    }

```

其中，调用了函数：

```

1.    void countNamesInTXT(QTableWidget *tableWidget, const QString &filePath,
    QDate &from, QDate &to)

```

在此函数中：

- QTableWidget *tableWidget 代表要填充数据的 QTableWidget 指针
- const QString &filePath 代表要读取的文本文件路径
- QDate &from 代表开始日期范围
- QDate &to 代表结束日期范围

其具体实现如下：

```

1.    /**
2.     * @brief 统计文本文件中指定日期范围内学生出勤次数
3.     *
4.     * 从指定的文本文件中统计在指定日期范围内每位学生的出勤次数，并将结果填充到
    表格中显示。
5.     *
6.     * @param tableWidget 要填充数据的 QTableWidget 指针
7.     * @param filePath 要读取的文本文件路径
8.     * @param from 开始日期范围
9.     * @param to 结束日期范围
10.    */
11.    void countNamesInTXT(QTableWidget *tableWidget, const QString &filePath,
    QDate &from, QDate &to) {
12.        // 打开文件
13.        QFile file(filePath);
14.        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {

```

```

15.         return; // 文件打开失败, 直接返回
16.     }
17.
18.     // 统计学生名字出现的次数
19.     QMap<QString, int> studentCountMap;
20.     QTextStream in(&file);
21.     while (!in.atEnd()) {
22.         QString line = in.readLine();
23.         QStringList fields = line.split(' ');
24.         if (fields.size() >= 6) {
25.             QString dateStr = fields[1];
26.             QString stuName = fields[4];
27.             QDate date = QDate::fromString(dateStr, "dd/MM/yyyy");
28.
29.             if (date.isValid() && date >= from && date <= to) {
30.                 studentCountMap[stuName]++;
31.             }
32.         }
33.     }
34.     file.close();
35.
36.     // 将统计结果转换为可排序的列表
37.     QList<QPair<QString, int>> studentCountList;
38.     for (auto it = studentCountMap.begin(); it != studentCountMap.en
        d(); ++it) {
39.         studentCountList.append(qMakePair(it.key(), it.value()));
40.     }
41.
42.     // 按出现次数降序排序
43.     std::sort(studentCountList.begin(), studentCountList.end(), [](c
        onst QPair<QString, int>& a, const QPair<QString, int>& b) {
44.         return b.second > a.second;
45.     });
46.
47.     // 设置表格行数和列数

```

```
48.         tableWidget->setRowCount(studentCountList.size());
49.         tableWidget->setColumnCount(2);
50.         tableWidget->setHorizontalHeaderLabels(QStringList() << "姓名
        " << "缺勤次数");
51.
52.         // 填充表格数据
53.         for (int i = 0; i < studentCountList.size(); ++i) {
54.             QTableWidgetItem* nameItem = new QTableWidgetItem(studentCou
                ntList[i].first);
55.             QTableWidgetItem* countItem = new QTableWidgetItem(QString::
                number(studentCountList[i].second));
56.             tableWidget->setItem(i, 0, nameItem);
57.             tableWidget->setItem(i, 1, countItem);
58.         }
59.
60.         // 调整列宽以适应内容
61.         tableWidget->resizeColumnsToContents();
62.     }
```

流程图如下：

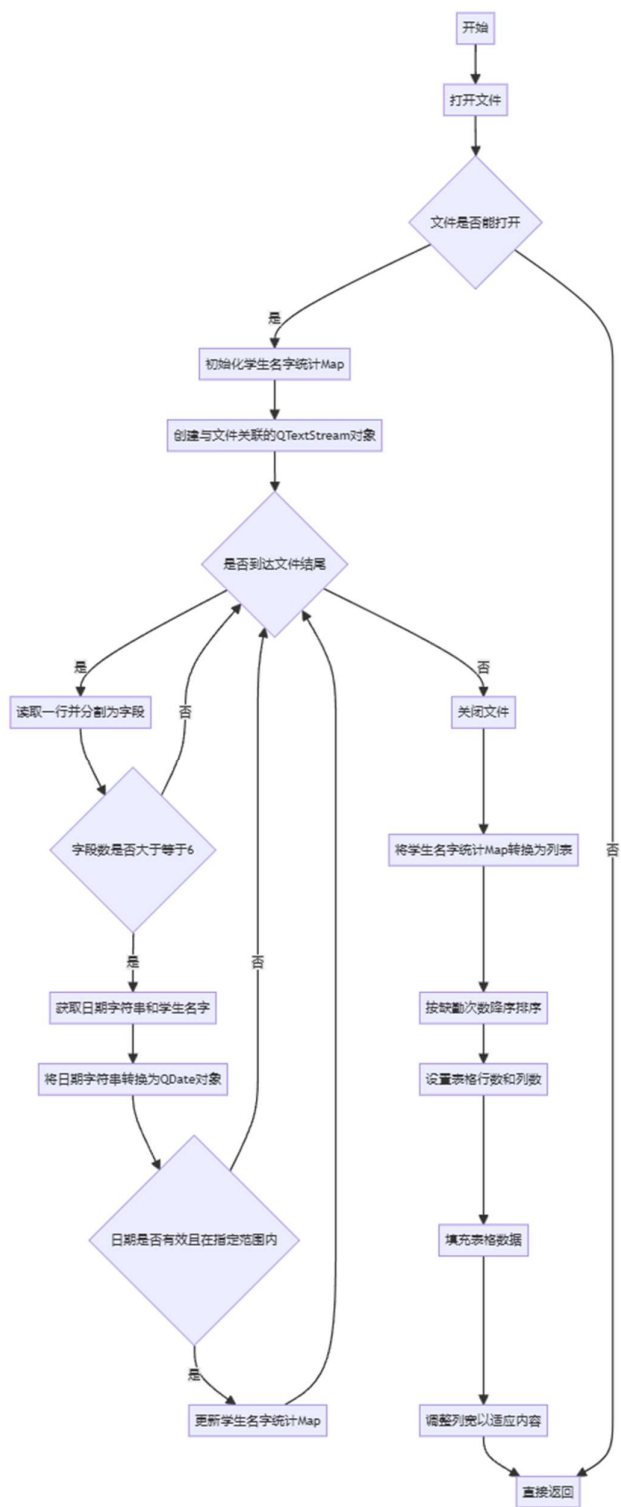


图 3-18 统计学生流程

3.6.2 统计课程

“统计课程”按钮触发如下：

```
1. void statistics::on_pushButton_2_released()
2. {
3.     if(fromismatch && toismatch){
4.         displayCourseStatistics(ui->tableWidget, QString(INFOPATH),
           from, to);
5.         fromismatch = false;
6.         toismatch = false;
7.     }
8.     else
9.         QMessageBox::critical(nullptr, "错误", "请设置起止时间");
10. }
```

其中，调用了函数：

```
1. void displayCourseStatistics(QTableWidget* tableWidget, const QString& filePath, QDate& from, QDate& to)
```

在此函数中：

- QTableWidget* tableWidget 代表要填充数据的 QTableWidget 指针
- const QString& filePath 代表要读取的文本文件路径
- QDate& from 代表开始日期范围
- QDate& to 代表结束日期范围

其具体实现为：

```
1. /**
2.  * @brief 显示课程统计信息
3.  *
4.  * 从指定的文件中读取数据，并统计在指定日期范围内每个课程的缺勤人次，
5.  * 然后将统计结果填充到表格中显示。
6.  *
7.  * @param tableWidget 要填充数据的 QTableWidget 指针
```

```

8.      * @param filePath 要读取的文本文件路径
9.      * @param from 开始日期范围
10.     * @param to 结束日期范围
11.     */
12.     void displayCourseStatistics(QTableWidget* tableWidget, const QString& filePath, QDate& from, QDate& to) {
13.         // 打开文件
14.         QFile file(filePath);
15.         if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
16.             qWarning("Cannot open file for reading");
17.             return;
18.         }
19.
20.         QTextStream in(&file);
21.         QMap<QString, int> courseCount;
22.
23.         // 逐行读取文件并统计课程名字
24.         while (!in.atEnd()) {
25.             QString line = in.readLine();
26.             QStringList fields = line.split(' ');
27.             if (fields.size() >= 6) {
28.                 QString dateStr = fields[1];
29.                 QDate date = QDate::fromString(dateStr, "dd/MM/yyyy");
30.                 if (date.isValid() && date >= from && date <= to) {
31.                     QString courseName = fields[3];
32.                     courseCount[courseName]++;
33.                 }
34.             }
35.         }
36.
37.         // 关闭文件
38.         file.close();
39.
40.         // 将统计结果转为 QVector<QPair<QString, int>> 以便排序

```

```

41.     QVector<QPair<QString, int>> courseList;
42.     for (auto it = courseCount.begin(); it != courseCount.end(); ++i
t) {
43.         courseList.append(qMakePair(it.key(), it.value()));
44.     }
45.
46.     // 按出现次数降序排序
47.     std::sort(courseList.begin(), courseList.end(), [](const QPair<Q
String, int>& a, const QPair<QString, int>& b) {
48.         return b.second > a.second;
49.     });
50.
51.     // 设置表格行数和列数
52.     tableWidget->setRowCount(courseList.size());
53.     tableWidget->setColumnCount(2);
54.
55.     // 设置表格头部
56.     QStringList headers;
57.     headers << "课程名称" << "缺勤人次";
58.     tableWidget->setHorizontalHeaderLabels(headers);
59.
60.     // 填充表格数据
61.     for (int i = 0; i < courseList.size(); ++i) {
62.         tableWidget->setItem(i, 0, new QTableWidgetItem(courseList[i
].first));
63.         tableWidget->setItem(i, 1, new QTableWidgetItem(QString::num
ber(courseList[i].second)));
64.     }
65.
66.     // 调整列宽以适应内容
67.     tableWidget->resizeColumnsToContents();
68. }

```

流程图如下：

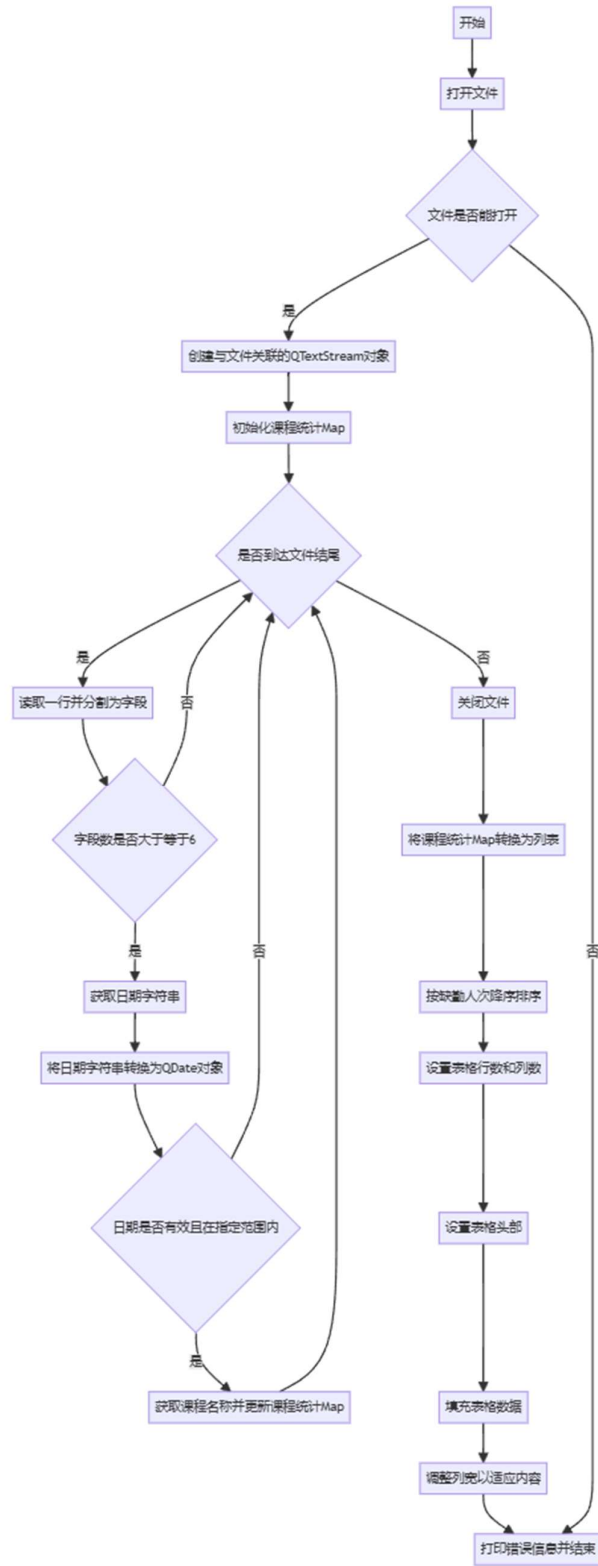


图 3-19 统计课程流程

3.7 学生端页面

Form

姓名

搜索

全览

	ID	日期	课时	课程名称	姓名	缺课类型
1	1	30/07/2024	2	fg	shsfdh	旷课
2	2	30/07/2024	3	x'fg'd'r'f'j'h	d'h'j'fg'd'j	旷课
3	3	01/01/2002	1	111	111	旷课

图 3- 20 学生端页面

区别于教师端，学生端只有信息浏览和查找权限，而没有修改和统计权限。

4. 编码设计

4.1 开发环境的设置和建立

下载安装 Visual Studio 2022 并配置相关环境；下载安装 qtcreator 4.7.0 并配置相关环境。

4.2 程序设计时的注意事项

在开发考勤管理系统的过程中，程序设计需要注意以下事项：

4.2.1 需求分析与明确：

在编写代码前，要充分理解和明确系统的需求，包括功能需求和非功能需求，确保所有学生和老师的的需求都能得到满足。

4.2.2 模块化设计：

将系统功能划分为若干模块，每个模块负责特定的功能，例如用户管理、考勤记录管理、数据统计等。模块之间应保持低耦合、高内聚，以便于维护和扩展。

4.2.3 用户体验：

设计简洁、直观的用户界面，确保用户操作方便，交互界面友好，易于上手。

4.2.4 数据安全与隐私：

实现用户密码哈希加密存储，防止数据泄露和未经授权的访问，确保数据的安全性和隐私保护。

4.2.5 数据持久化：

设计可靠的数据存储机制，确保系统能够长时间稳定运行，利用文件存储用户信息和考勤数据等，确保数据能够准确保存和读取。

4.2.6 性能优化：

考虑系统的性能要求，优化数据查询和处理算法，确保系统在高并发环境下仍能高效运行。对关键模块进行性能测试和优化。

4.2.7 测试与调试：

在开发过程中，进行充分的单元测试、集成测试和系统测试，及时发现和修复问题，确保系统的稳定性和可靠性。

4.2.8 代码规范：

遵循良好的编码规范和代码风格，确保代码可读性和可维护性。注释清晰，变量命名规范，函数设计合理。

4.3 关键构件/插件的特点和使用

在开发设计本项目时，我组采用 QT 作为关键构件。

4.3.1 QT 的特点

4.3.1.1 跨平台支持：

Qt 是一个跨平台的开发框架，支持 Windows、Linux、macOS 等多个操作系统。使用 Qt 开发的应用程序可以在不同平台上无缝运行，极大地提升了应用的兼容性和可移植性。

4.3.1.2 丰富的组件库：

Qt 提供了丰富的 UI 组件库，包括按钮、文本框、标签、表格、树形视图等，能够快速构建用户友好的图形界面。此外，Qt 还支持自定义组件，满足特定需求。

4.3.1.3 强大的图形能力：

Qt 的图形绘制能力强大，支持 2D 和 3D 图形绘制，可以实现复杂的图形界面和动画效果，提升用户体验。

4.3.1.4 信号与槽机制：

Qt 独特的信号与槽机制（Signal-Slot）使得事件驱动编程更加直观和高效，便于实现界面与业务逻辑的分离，代码结构更加清晰。

4.3.2 QT 的使用情况

4.3.2.1 用户界面设计：

在考勤管理系统中，使用 Qt Designer 设计用户界面，通过拖拽方式快速创建窗口和控件布局，生成的 UI 文件可以直接在代码中加载和使用。

4.3.2.2 用户登录与管理:

利用 Qt 的输入框、按钮和对话框组件，实现用户登录界面和用户管理功能。通过信号与槽机制，处理用户交互事件，如登录按钮点击、表单提交等。

4.3.2.3 考勤记录展示与管理:

使用 Qt 的表格视图 (QTableView) 显示考勤记录，通过模型-视图架构分离数据和界面，实现数据的动态更新和交互。

4.3.2.4 数据持久化与读取:

结合 Qt 的文件操作类 (如 QFile、QTextStream) 实现考勤数据的持久化存储和读取，确保数据的安全和持久性。

4.3.2.5 统计与图表展示:

利用 Qt Charts 模块，生成考勤数据的统计图表，提供直观的图形化展示，帮助用户更好地理解和分析考勤数据。

4.4 主要程序的代码设计及注释

info.h 代码如下:

```
1.  #ifndef INFO_H
2.  #define INFO_H
3.
4.  // 引入所需的 Qt 库
5.  #include "QString"      // 提供字符串处理类
6.  #include "QDebug"      // 提供调试输出功能
7.  #include "QFile"       // 提供文件处理类
8.  #include "QTableWidget" // 提供表格控件类
9.  #include "QDate"       // 提供日期处理类
10.
11. // 定义一个表示考勤信息的类
12. class info {
13. public:
14.     int id;           // 学生 ID
15.     QString date;     // 日期
16.     int ClassNum;     // 课时编号
17.     QString ClassName; // 课程名称
```

```

18.     QString StuName;    // 学生姓名
19.     QString Method;    // 缺课类型

20.

21.     // 带参数的构造函数, 用于初始化 info 对象的成员变量
22.     info(int id, QString date, int ClassNum, QString ClassName, QStr
        ing StuName, QString Method)
23.         : id(id), date(date), ClassNum(ClassNum), ClassName(ClassNam
            e), StuName(StuName), Method(Method) {}

24.

25.     // 默认构造函数, 用于创建空的 info 对象
26.     info() {};

27.

28.     // 将 info 对象的数据写入文件
29.     void tofile(const QString &filepath);

30.

31.     // 析构函数, 用于调试目的, 析构时输出调试信息
32.     ~info() {
33.         qDebug() << "Destructor called" << Qt::endl;
34.     }
35. };

36.

37.     // 获取文件中最后一个 ID
38.     int lastid(const QString &filepath);

39.

40.     // 从文本文件填充 QTableWidgetItem
41.     void fillTableFromTextFile(QTableWidgetItem *tableWidget, const QString
        &filename);

42.

43.     // 删除指定 ID 的行并重新计算 ID
44.     void deleteRowAndRecomputeId(const QString &filename, int idToDelete
        );

45.

46.     // 显示确认操作对话框
47.     bool confirmOperation(QWidget *parent);

48.

```

```

49. // 修改指定 ID 的 info 数据
50. void modifyInfoData(const QString &filename, const info &infoObj, int idToModify);
51.
52. // 根据 ID 获取 info 对象
53. info getInfoById(const QString &filename, int idToFind);
54.
55. // 加载表数据到 QTableWidgetItem, 可以根据姓名筛选
56. void loadTableData(const QString &filePath, const QString &name, QTableWidgetItem *tableWidget);
57.
58. // 统计 TXT 文件中名字在指定时间范围内出现的次数
59. void countNamesInTXT(QTableWidgetItem *tableWidget, const QString &filePath, QDate &from, QDate &to);
60.
61. // 显示指定时间范围内的课程统计
62. void displayCourseStatistics(QTableWidgetItem *tableWidget, const QString &filePath, QDate &from, QDate &to);
63.
64. #endif // INFO_H

```

info.cpp 代码如下:

```

1. #include "info.h"
2. #include "QFile"
3. #include "QDebug"
4. #include "QMessageBox"
5. #include <QMap>
6. #include <QStringList>
7. #include <QTextStream>
8. #include <QTableWidgetItem>
9.
10. // 将 info 对象的数据写入文件
11. void info::tofile(const QString &filepath) {
12.     QFile file(filepath);
13.     if (!file.open(QIODevice::Append | QIODevice::Text)) {
14.         qDebug() << "Failed to open file for appending:" << file.errorString();

```

```

15.         return;
16.     }
17.     QTextStream out(&file);
18.     // 写入 info 对象的数据到文件
19.     out << this->id << ' ' << this->date << ' ' << this->ClassNum <<
        ' ' << this->ClassName << ' ' << this->StuName << ' ' << this->Method
        << Qt::endl;
20.     file.close();
21. }

22.

23. // 获取文件中最后一个 ID
24. int lastid(const QString &filepath) {
25.     QFile file(filepath);
26.     QString lastId = "0"; // 初始化最后的 ID 为 0
27.     if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
28.         QTextStream in(&file);
29.         while (!in.atEnd()) {
30.             QString line = in.readLine();
31.             QStringList parts = line.split(" ");
32.             if (!parts.isEmpty()) {
33.                 QString id = parts.first();
34.                 lastId = id; // 更新最后的 ID
35.             }
36.         }
37.     }
38.     file.close();
39.     return lastId.toInt(); // 返回最后 ID 的整数值
40. }

41.

42. // 从文本文件填充 QTableWidgetItem
43. void fillTableFromTextFile(QTableWidgetItem *tableWidget, const QString
    &filename) {
44.     QFile file(filename);
45.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
46.         return;

47.

48.     tableWidget->clear();
49.     tableWidget->setRowCount(0);
50.     tableWidget->setColumnCount(6);

51.

```

```

52.     // 设置表头
53.     QStringList headers;
54.     headers << "ID" << "日期" << "课时" << "课程名称" << "姓名" << "缺
    课类型";
55.     tableWidget->setHorizontalHeaderLabels(headers);

56.

57.     QTextStream in(&file);
58.     int row = 0;
59.     while (!lin.atEnd()) {
60.         QString line = in.readLine();
61.         QStringList fields = line.split(' ');
62.         if (fields.size() >= 6) {
63.             tableWidget->insertRow(row);
64.             for (int column = 0; column < 6; ++column) {
65.                 QTableWidgetItem *item = new QTableWidgetItem(fields
        [column]);
66.                 tableWidget->setItem(row, column, item);
67.             }
68.             ++row;
69.         }
70.     }

71.

72.     file.close();
73. }

74.

75. // 删除指定 ID 的行并重新计算 ID
76. void deleteRowAndRecomputeId(const QString &filename, int idToDelete
    ) {
77.     QFile file(filename);
78.     if (!file.open(QIODevice::ReadWrite | QIODevice::Text))
79.         return;
80.
81.     QStringList lines;
82.     QTextStream in(&file);
83.     while (!lin.atEnd()) {
84.         QString line = in.readLine();
85.         lines.append(line);
86.     }
87.     file.resize(0); // 清空文件内容
88.     int currentId = 1;
89.     QTextStream out(&file);

```



```

90.     for (const QString &line : lines) {
91.         QStringList fields = line.split(' ');
92.         if (fields.isEmpty())
93.             continue;
94.         int id = fields[0].toInt();
95.         if (id != idToDelete) {
96.             fields[0] = QString::number(currentId++); // 重新计算ID
97.             out << fields.join(' ') << '\n';
98.         }
99.     }
100.    file.close();
101. }

102.

103. // 显示确认操作对话框
104. bool confirmOperation(QWidget *parent) {
105.     QMessageBox::StandardButton reply;
106.     reply = QMessageBox::question(parent, "注意", "你确定要删除吗",
107.                                   QMessageBox::Yes | QMessageBox::No
108.     );
109.     return reply == QMessageBox::Yes;
110. }

111. // 修改指定ID的info数据
112. void modifyInfoData(const QString &filename, const info &infoObj, int
    idToModify) {
113.     QFile file(filename);
114.     if (!file.open(QIODevice::ReadWrite | QIODevice::Text))
115.         return;
116.
117.     QStringList lines;
118.     QTextStream in(&file);
119.     while (!in.atEnd()) {
120.         QString line = in.readLine();
121.         lines.append(line);
122.     }
123.     file.resize(0); // 清空文件内容
124.     QTextStream out(&file);
125.     for (const QString &line : lines) {
126.         QStringList fields = line.split(' ');
127.         if (fields.isEmpty())
128.             continue;

```

```

129.         int id = fields[0].toInt();
130.         if (id == idToModify) {
131.             // 写入修改后的info 数据
132.             out << infoObj.id << ' ' << infoObj.date << ' ' << infoObj.ClassNum << ' ' << infoObj.ClassName << ' ' << infoObj.StuName << ' ' << infoObj.Method << '\n';
133.         } else {
134.             out << line << '\n';
135.         }
136.     }
137.     file.close();
138. }

139.

140. // 根据 ID 获取 info 对象
141. info getInfoById(const QString &filename, int idToFind) {
142.     QFile file(filename);
143.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
144.         return info(0, "", 0, "", "", "");
145.
146.     QTextStream in(&file);
147.     while (!in.atEnd()) {
148.         QString line = in.readLine();
149.         QStringList fields = line.split(' ');
150.         if (fields.isEmpty())
151.             continue;
152.         int id = fields[0].toInt();
153.         if (id == idToFind) {
154.             // 返回找到的info 对象
155.             return info(id, fields[1], fields[2].toInt(), fields[3], fields[4], fields[5]);
156.         }
157.     }
158.     return info(0, "", 0, "", "", ""); // 未找到返回空对象
159. }

160.

161. // 加载表数据到QTableWidget, 可以根据姓名筛选
162. void loadTableData(const QString &filePath, const QString &name, QTableWidget *tableWidget) {
163.     QFile file(filePath);
164.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
165.         qDebug() << "Cannot open file for reading";
166.         return;

```

```

167.     }
168.     tableWidget->setRowCount(0);
169.     QTextStream in(&file);
170.     int row = 0;
171.     while (!in.atEnd()) {
172.         QString line = in.readLine();
173.         QStringList fields = line.split(' ');
174.         if (fields.size() == 6 && fields[4] == name) {
175.             tableWidget->insertRow(row);
176.             for (int col = 0; col < fields.size(); ++col) {
177.                 tableWidget->setItem(row, col, new QTableWidgetItem(
178.                     fields[col]));
179.                 row++;
180.             }
181.         }
182.         file.close();
183.     }

184.

185. // 统计 TXT 文件中名字在指定时间范围内出现的次数
186. void countNamesInTXT(QTableWidget *tableWidget, const QString &filePath,
187.     QDate &from, QDate &to) {
188.     QFile file(filePath);
189.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
190.         return; // 文件打开失败, 直接返回
191.     }

192. // 统计学生名字出现的次数
193. QMap<QString, int> studentCountMap;
194. QTextStream in(&file);
195. while (!in.atEnd()) {
196.     QString line = in.readLine();
197.     QStringList fields = line.split(' ');
198.     if (fields.size() >= 6) {
199.         QString dateStr = fields[1];
200.         QString stuName = fields[4];
201.         QDate date = QDate::fromString(dateStr, "dd/MM/yyyy");

202.

203.         if (date.isValid() && date >= from && date <= to) {
204.             studentCountMap[stuName]++;
205.         }

```

```

206.     }
207. }
208.     file.close();
209.
210.     // 将统计结果转换为可排序的列表
211.     QList<QPair<QString, int>> studentCountList;
212.     for (auto it = studentCountMap.begin(); it != studentCountMap.en
        d(); ++it) {
213.         studentCountList.append(qMakePair(it.key(), it.value()));
214.     }
215.
216.     // 按出现次数降序排序
217.     std::sort(studentCountList.begin(), studentCountList.end(), [](c
        onst QPair<QString, int>& a, const QPair<QString, int>& b) {
218.         return b.second > a.second;
219.     });
220.
221.     // 设置表格行数和列数
222.     tableWidget->setRowCount(studentCountList.size());
223.     tableWidget->setColumnCount(2);
224.     tableWidget->setHorizontalHeaderLabels(QStringList() << "姓名
        " << "缺勤次数");
225.
226.     // 填充表格数据
227.     for (int i = 0; i < studentCountList.size(); ++i) {
228.         QTableWidgetItem* nameItem = new QTableWidgetItem(studentCou
            ntList[i].first);
229.         QTableWidgetItem* countItem = new QTableWidgetItem(QString::
            number(studentCountList[i].second));
230.         tableWidget->setItem(i, 0, nameItem);
231.         tableWidget->setItem(i, 1, countItem);
232.     }
233.
234.     // 调整列宽以适应内容
235.     tableWidget->resizeColumnsToContents();
236. }
237.
238. // 显示指定时间范围内的课程统计

```

```

239. void displayCourseStatistics(QTableWidget* tableWidget, const QStrin
    g& filePath, QDate& from, QDate& to) {
240.     QFile file(filePath);
241.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
242.         qWarning("Cannot open file for reading");
243.         return;
244.     }
245.
246.     QTextStream in(&file);
247.     QMap<QString, int> courseCount;
248.
249.     // 逐行读取文件并统计课程名字
250.     while (!in.atEnd()) {
251.         QString line = in.readLine();
252.         QStringList fields = line.split(' ');
253.         if (fields.size() >= 6) {
254.             QString dateStr = fields[1];
255.             QDate date = QDate::fromString(dateStr, "dd/MM/yyyy");
256.             if (date.isValid() && date >= from && date <= to) {
257.                 QString courseName = fields[3];
258.                 courseCount[courseName]++;
259.             }
260.         }
261.     }
262.
263.     file.close();
264.
265.     // 将统计结果转为 QVector<QPair<QString, int>> 以便排序
266.     QVector<QPair<QString, int>> courseList;
267.     for (auto it = courseCount.begin(); it != courseCount.end(); ++i
        t) {
268.         courseList.append(qMakePair(it.key(), it.value()));
269.     }
270.
271.     // 按出现次数降序排序
272.     std::sort(courseList.begin(), courseList.end(), [](const QPair<Q
        String, int>& a, const QPair<QString, int>& b) {
273.         return b.second > a.second;
274.     });

```

```

275.
276.     // 设置表格行数和列数
277.     tableWidget->setRowCount(courseList.size());
278.     tableWidget->setColumnCount(2);
279.
280.     // 设置表格头部
281.     QStringList headers;
282.     headers << "课程名称" << "缺勤人次";
283.     tableWidget->setHorizontalHeaderLabels(headers);
284.
285.     // 填充表格数据
286.     for (int i = 0; i < courseList.size(); ++i) {
287.         tableWidget->setItem(i, 0, new QTableWidgetItem(courseList[i]
            ].first));
288.         tableWidget->setItem(i, 1, new QTableWidgetItem(QString::num
            ber(courseList[i].second)));
289.     }
290.
291.     // 调整列宽以适应内容
292.     tableWidget->resizeColumnsToContents();
293. }

```

4.5 解决的技术难点

利用函数通过将统计结果转换为可排序的数据结构，然后利用标准库中的 `std::sort` 函数对结果进行排序，最后将排序后的结果显示在 `QTableWidget` 中，实现了对学生缺勤次数和课程缺勤人次的统计和展示。

```

1.     void countNamesInTXT(QTableWidget *tableWidget, const QString &filePath,
    2.         QDate &from, QDate &to) {
3.         QFile file(filePath);
4.         if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
5.             return; // 文件打开失败，直接返回
6.         }
7.         // 统计学生名字出现的次数
8.         QMap<QString, int> studentCountMap;
9.         QTextStream in(&file);
10.        while (!in.atEnd()) {

```

```

11.         QString line = in.readLine();
12.         QStringList fields = line.split(' ');
13.         if (fields.size() >= 6) {
14.             QString dateStr = fields[1];
15.             QString stuName = fields[4];
16.             QDate date = QDate::fromString(dateStr, "dd/MM/yyyy");
17.
18.             if (date.isValid() && date >= from && date <= to) {
19.                 studentCountMap[stuName]++;
20.             }
21.         }
22.     }
23.     file.close();
24.
25.     // 将统计结果转换为可排序的列表
26.     QList<QPair<QString, int>> studentCountList;
27.     for (auto it = studentCountMap.begin(); it != studentCountMap.end(); ++it) {
28.         studentCountList.append(qMakePair(it.key(), it.value()));
29.     }
30.
31.     // 按出现次数降序排序
32.     std::sort(studentCountList.begin(), studentCountList.end(), [](const QPair<QString, int>& a, const QPair<QString, int>& b) {
33.         return b.second > a.second;
34.     });
35.
36.     // 设置表格行数和列数
37.     tableWidget->setRowCount(studentCountList.size());
38.     tableWidget->setColumnCount(2);
39.     tableWidget->setHorizontalHeaderLabels(QStringList() << "姓名"
40.         << "缺勤次数");
41.
42.     // 填充表格数据
43.     for (int i = 0; i < studentCountList.size(); ++i) {
44.         QTableWidgetItem* nameItem = new QTableWidgetItem(studentCountList[i].first);
45.         QTableWidgetItem* countItem = new QTableWidgetItem(QString::number(studentCountList[i].second));

```

```

45.         tableWidget->setItem(i, 0, nameItem);
46.         tableWidget->setItem(i, 1, countItem);
47.     }

48.
49.     // 调整列宽以适应内容
50.     tableWidget->resizeColumnsToContents();
51. }

52.
53.
54. void displayCourseStatistics(QTableWidget* tableWidget, const QString& filePath, QDate& from, QDate& to) {
55.     QFile file(filePath);
56.     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
57.         qWarning("Cannot open file for reading");
58.         return;
59.     }

60.
61.     QTextStream in(&file);
62.     QMap<QString, int> courseCount;

63.
64.     // 逐行读取文件并统计课程名字
65.     while (!in.atEnd()) {
66.         QString line = in.readLine();
67.         QStringList fields = line.split(' ');
68.         if (fields.size() >= 6) {
69.             QString dateStr = fields[1];
70.             QDate date = QDate::fromString(dateStr, "dd/MM/yyyy");
71.             if (date.isValid() && date >= from && date <= to) {
72.                 QString courseName = fields[3];
73.                 courseCount[courseName]++;
74.             }
75.         }
76.     }

77.
78.     // 关闭文件
79.     file.close();

80.

```



```

81.      // 将统计结果转为 QVector<QPair<QString, int>> 以便排序
82.      QVector<QPair<QString, int>> courseList;
83.      for (auto it = courseCount.begin(); it != courseCount.end(); ++i
t) {
84.          courseList.append(qMakePair(it.key(), it.value()));
85.      }

86.

87.      // 按出现次数降序排序
88.      std::sort(courseList.begin(), courseList.end(), [](const QPair<Q
String, int>& a, const QPair<QString, int>& b) {
89.          return b.second > a.second;
90.      });

91.

92.      // 设置表格行数和列数
93.      tableWidget->setRowCount(courseList.size());
94.      tableWidget->setColumnCount(2);

95.

96.      // 设置表格头部
97.      QStringList headers;
98.      headers << "课程名称" << "缺勤人次";
99.      tableWidget->setHorizontalHeaderLabels(headers);

100.

101.     // 填充表格数据
102.     for (int i = 0; i < courseList.size(); ++i) {
103.         tableWidget->setItem(i, 0, new QTableWidgetItem(courseList[i
].first));
104.         tableWidget->setItem(i, 1, new QTableWidgetItem(QString::num
ber(courseList[i].second)));
105.     }

106.

107.     // 调整列宽以适应内容
108.     tableWidget->resizeColumnsToContents();
109. }

```

这个算法主要中包含了以上两个函数: countNamesInTXT 和 displayCourseStatistics, 它们分别用于统计学生缺勤次数和课程缺勤人次, 并将结果显示在 QTableWidgetItem 中。

对于 countNamesInTXT 函数: 我们先利用函数打开指定的文本文件, 并按行读取其中的内容; 再对每一行进行分割, 提取出日期和学生姓名, 并将日期转换为 QDate 类型; 如

果日期在指定范围内，则将学生姓名作为键，缺勤次数作为值，存储在 QMap 类型的 studentCountMap 中；然后将 studentCountMap 中的统计结果转换为可排序的列表 studentCountList，其中每个元素是一个键值对，包含学生姓名和缺勤次数；接着使用 std::sort 函数对 studentCountList 进行降序排序，根据学生的缺勤次数进行排序，最后将排序后的结果填充到 QTableWidgetItem 中，显示学生姓名和缺勤次数，并调整列宽以适应内容。

对于 displayCourseStatistics 函数：我们先利用函数同样打开指定的文本文件，并按行读取其中的内容；再对每一行进行分割，提取出日期和课程名称，并将日期转换为 QDate 类型；如果日期在指定范围内，则将课程名称作为键，缺勤人次作为值，存储在 QMap 类型的 courseCount 中；然后将 courseCount 中的统计结果转换为可排序的列表 courseList，其中每个元素是一个键值对，包含课程名称和缺勤人次；接着使用 std::sort 函数对 courseList 进行降序排序，根据课程的缺勤人次进行排序；最后将排序后的结果填充到 QTableWidgetItem 中，显示课程名称和缺勤人次，并调整列宽以适应内容。

5. 测试时出现过的问题及其解决方法

遇到的问题：修改学生的考勤记录后系统不能自动刷新

问题原因：系统不能自动刷新的原因是系统未能在修改操作后自动再次读取文件中更新后的数据，故而为解决该问题，我们只需让系统能够在修改操作之后再次读取文件中更新后的数据即可。

解决方法：新增一个刷新按钮，用户可以自助手动刷新。

6. 总结

6.1 课程设计完成情况

已完成的部分：用户的注册、登录和权限管理；录入、修改、查询以及统计分析和排序学生的缺课记录；使用文件存储用户信息及学生考勤情况；图形化的用户友好交互式界面的实现；实现模块化设计程序；编程风格良好。

未完成的部分：无。

6.2 创新功能

创新功能：新增用户管理模块，添加了用户注册、登录和权限管理的功能；利用文件写入读取数据，包括用户的相关信息和学生的考勤情况等，确保数据的长期保存和安全；添加了图形化用户友好交互式界面，界面简洁清晰，易于上手，方便用户进行操作和数据查看。

6.3 小组成员分工及成绩自评排序

在人员分工上，我组汪钰轩同学负责底层逻辑代码的编写、项目报告书的编写、调试代码、代码后期的完善和维护，宋冠庭同学负责图形化代码的实现、调试代码、代码后期的完善和维护。经过自评，成绩排序如下：1.汪钰轩 2.宋冠庭。

6.4 项目开发心得

在开发设计考勤管理系统的过程中，我们不仅收获了丰富的编程知识和团队协作经验，还深刻体会到软件开发的复杂与精彩。从需求分析、系统设计到代码实现，每一步都充满了挑战和收获。我们在解决用户身份验证、数据持久化等难题时，提升了技术能力，增强了逻辑思维。在不断的调试与优化中，我们认识到细节的重要性和严谨工作的必要性，同时学会了有效的项目管理和时间规划。这次项目不仅加深了我们对 C++ 和 Qt 的理解，还让我们更全面地掌握了软件开发的流程和方法。通过分工合作，我们体验到了团队合作的力量，学会了如何在紧迫的时间内高效完成任务。此外，我们也认识到良好的文档记录和代码规范的重要性，这些宝贵的经验和教训将成为我们未来工作的基石。总体而言，这次项目不仅提升了我们的技术水平，也培养了我们团队协作和项目管理的能力，为未来的职业发展奠定了坚实的基础。

1. 参考文献

[1] <https://wiki.qt.io/Main>