高级语言程序设计

实验报告

南开大学 计算机大类

宋卓伦

2311095

0928

2024年5月10日

目录

一、作业题目

二、开发软件

三、课题要求

四、主要流程

1. 整体流程

2. 算法或公式

（1）LoginFirst类头文件

（2）FormDetail类头文件

（3）Moments类头文件

（4）Accounts类头文件

（5）DaysMaster类头文件

（6）ToDoList类头文件

（7）一些功能的函数

3.单元测试

五、单元测试

六、收获

1. 程序学习的收获

2. 个人成长的收获

**一、作业题目**

Record Your Daily Life

一个可以完成日记本（手账）、记账本、倒数日记录、待办事件等功能的程序

**二、开发软件**

Qt 6.6.3（Qt Creator 13.0.1 & Qt Design Studio）

**三、课题要求**

1. 面向对象。
2. 单元测试。
3. 模型部分。
4. 验证。

**四、主要流程**

**1、整体流程**

a. 创建LoginFirst类为登陆界面，包括退出按钮、打开详情信息、打开选择界面和打开历史记录；

b. 创建FormDetail类来查询版本信息；

c. 创建Select类为选择界面，选择Moments、Accounts、DaysMaster和ToDoList四种功能，还有期待后续开发和退出；

d. 创建Moments类为记事（手账）界面，编辑文本，包含新建、打开、保存、另存为等功能；

e. 创建Accounts类为记账界面，记录支出明细，包含添加、删除、导出等功能；

f. 创建DaysMaster类为倒数日界面，查询重要节点，同时可以导出至Moments类进行编辑；

g. 创建ToDoList 类为待办事项界面，添加/勾选/清空待办事项。

**2、算法及公式**

定义七个基类LoginFirst、FormDetail、Select、Moments、Accounts、DaysMaster和ToDoList 。

对应的关系和继承关系：

**LoginFirst类头文件：**

QT\_BEGIN\_NAMESPACE  
namespace Ui {  
class LoginFirst;  
}  
QT\_END\_NAMESPACE  
​  
class LoginFirst : public QMainWindow  
{  
   Q\_OBJECT  
​  
public:  
   LoginFirst(QWidget \*parent = nullptr);  
   ~LoginFirst();  
​  
private slots:  
   void on\_pushButton\_2\_clicked(); //退出按钮；  
   void on\_pushButton\_3\_clicked(); //打开详情信息；  
   void on\_pushButton1\_clicked(); //打开选择界面；  
   void on\_pushButton\_clicked(); //打开历史记录；  
​  
private:  
   Ui::LoginFirst \*ui; //界面打开；  
   FormDetail\* fd = new FormDetail; //包含FormDetail类；  
   Select\* s = new Select; //包含Select类；  
   Moments\* mt; //包含Moments类；  
};  
​  
#endif // LOGINFIRST\_H

**FormDetail类头文件：**

namespace Ui {  
class FormDetail;  
}  
​  
class FormDetail : public QWidget  
{  
   Q\_OBJECT  
​  
public:  
   explicit FormDetail(QWidget \*parent = nullptr);  
   ~FormDetail();  
​  
private slots:  
   void on\_pushButton\_clicked(); //关闭页面；  
​  
private:  
   Ui::FormDetail \*ui; //界面打开；  
};  
​  
#endif // FORMDETAIL\_H

**Select类头文件：**

namespace Ui {  
class Select;  
}  
​  
class Select : public QWidget  
{  
   Q\_OBJECT  
​  
public:  
   explicit Select(QWidget \*parent = nullptr);  
   ~Select();  
​  
private slots:  
   void on\_pushButton\_3\_clicked(); //选择Moments界面；  
   void on\_pushButton\_5\_clicked(); //选择Accounts界面；  
   void on\_pushButton\_clicked(); //选择DaysMaster界面；  
   void on\_pushButton\_4\_clicked(); //选择ToDoList界面；  
   void on\_pushButton\_6\_clicked(); //选择Continue(后续开发)界面；  
   void on\_pushButton\_2\_clicked(); //选择退出；  
​  
private:  
   Ui::Select \*ui; //创建页面；  
   Moments\* mt = new Moments; //包含Moments类；  
   Accounts\* at = new Accounts; //包含Accounts类；  
   DaysMaster\* dm = new DaysMaster; //包含DaysMaster类；  
   ToDoList\* tdl = new ToDoList; //包含ToDoList类；  
};  
​  
#endif // SELECT\_H

**Moments类头文件：**

namespace Ui {  
class Moments;  
}  
​  
class Moments : public QMainWindow  
{  
   Q\_OBJECT  
​  
public:  
   explicit Moments(QWidget \*parent = nullptr);  
   QTextEdit\* textEdit;  
   ~Moments();  
​  
public slots:  
   void onActionNewTriggered(bool checked); //新建；  
   void onActionOpenTriggered(bool checked); //打开；  
   void onActionSaveTriggered(bool checked); //保存；  
   void onActionOtherSaveTriggered(bool checked); //另存为；  
   void onActionPasteTriggered(bool checked); //粘贴；  
   void onActionUndoTriggered(bool checked); //撤销；  
   void onActionRedoTriggered(bool checked); //重做；  
   void onActionCopyTriggered(bool checked); //复制；  
​  
   void receiveText(const QString &text);//将DaysMaster的文本传递到Moments；  
   void receiveTxt(const QString &text);//将历史记录打开的txt文件的文本传递到Moments；  
​  
private slots:  
   void on\_pushButton\_clicked(); //确认是否退出（以按钮形式）；  
   void on\_pushButton1\_clicked(bool checked); //保存（以按钮形式）；  
   void on\_pushButton\_2\_clicked(); //清空（以按钮形式）；  
   void onEdited(); //反映文本已修改；  
   void on\_pushButton\_3\_clicked(); //插入（以按钮形式）；  
   void on\_pushButton\_4\_clicked(); //打开（以按钮形式）；  
​  
signals:  
   void open(); //一个信号；  
​  
private:  
   Ui::Moments \*ui; //创建页面；  
   Moments \*mt; //包含Moments类；  
   QString savePath; //保存路径；  
   bool saved; //判断是否保存；  
   QString filename; //文件名；  
   QFile file; //文件；  
};  
​  
#endif // MOMENTS\_H

**Accounts类头文件：**

namespace Ui {  
class Accounts;  
}  
​  
class Accounts : public QMainWindow  
{  
   Q\_OBJECT  
​  
public:  
   explicit Accounts(QWidget \*parent = nullptr);  
   ~Accounts();  
​  
private slots:  
   void on\_pushButton\_clicked(); //新建记录；  
   void extracted(bool &ok, int &number, QStringList &parts);  
   void on\_pushButton\_2\_clicked(); //上面两个都是删除记录，同时更新已花掉的额度；  
   void on\_standardEdit\_returnPressed(); //获取限定额度；  
   void on\_standardEdit\_textChanged(const QString &stanDard); //当限额变化时返回当前值；  
   void on\_loadTrButton\_clicked(); //清空记录；  
   void on\_saveTrButton\_clicked(); //保存为txt记录；  
   void on\_pushButton\_3\_clicked(); //关闭页面；  
​  
private:  
   Ui::Accounts \*ui; //创建页面；  
   QList<QString> transactions; //交易清单；  
};  
​  
#endif // ACCOUNTS\_H

**DaysMaster类头文件：**

namespace Ui {  
class DaysMaster;  
}  
​  
class DaysMaster : public QMainWindow  
{  
   Q\_OBJECT  
​  
public:  
   explicit DaysMaster(QWidget \*parent = nullptr);  
   ~DaysMaster();  
​  
private slots:  
   void onCountdown(); //日期记录；  
   void on\_pushButton\_2\_clicked(); //清空记录；  
   void on\_pushButton\_3\_clicked(); //关闭页面；  
   void on\_pushButton\_4\_clicked(); //保存为txt文件；  
   void on\_pushButton\_5\_clicked(); //直接在Moments里面打开；  
​  
signals:  
   void textToTransfer(const QString &text); //传递信号  
​  
private:  
   Ui::DaysMaster \*ui; //创建页面；  
   DaysMaster \*dm; //包含DaysMaster类；  
   QDateEdit \*targetDateEdit; //设置时间显示的文本框；  
   QLabel \*countdownLabel; //设置标签显示距离天数；  
   Moments \*mt; //包含Moments类；  
};  
​  
#endif // DAYSMASTER\_H

**ToDoList类头文件：**

namespace Ui {  
class ToDoList;  
}  
​  
class ToDoList : public QMainWindow  
{  
   Q\_OBJECT  
​  
public:  
   explicit ToDoList(QWidget \*parent = nullptr);  
   ~ToDoList();  
​  
private slots:  
   void on\_addButton\_clicked(); //设置额度；  
   void on\_pushButton\_clicked(); //清空记录；  
   void on\_pushButton\_2\_clicked(); //关闭页面；  
​  
private:  
   Ui::ToDoList \*ui; //创建页面；  
   QVBoxLayout \*checkLayout; // 水平布局  
   QVBoxLayout \*lineeditLayout; // 水平布局  
   ToDoList \*tdl; //包含ToDoList类；  
};  
​  
#endif // TODOLIST\_H  
​

上面七段代码反映了各槽函数与类之间的关系。

**一些功能的函数：**

//打开历史记录；  
#include "loginfirst.h"  
​  
void LoginFirst::on\_pushButton\_clicked()  
{  
   QMessageBox judge;  
   judge.setStyleSheet("QLabel{""min-width:300px;""min-height:120px;"  
                       "font-size:30px;""}");  
   judge.setText("   请选择打开方式！");  
   judge.addButton("moments",QMessageBox::AcceptRole);  
   judge.addButton("其它文件",QMessageBox::RejectRole);  
​  
   int res=judge.exec();  
   switch(res)  
  {  
   case QMessageBox::AcceptRole:  
  {  
       qDebug()<<"打开文件…"<<Qt::Checked<<Qt::endl;  
       QString fileName = QFileDialog::getOpenFileName  
          (this, tr("打开文件"), "", tr("文本文件(\*.txt);;word文件(\*.doc);;word文件(\*.docx)"));  
       if (!fileName.isEmpty()) {  
           // 用户选择了文件，尝试打开并读取  
           QFile file(fileName);  
           if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {  
               // 无法打开文件，显示错误消息  
               QMessageBox::warning(this, "错误", "无法打开文件！");  
               return;  
          }  
           else{  
               mt = new Moments;  
               // 使用 QTextStream 读取文件内容  
               QTextStream in(&file);  
               QString fileContent = in.readAll(); // 读取所有内容  
               // 将文件内容设置到 QTextEdit 控件中  
               mt->receiveText(fileContent);  
               // 假设你的 QTextEdit 对象名为 textEdit  
               mt->show();  
               mt->setWindowTitle("Moments");  
          }  
      }break;  
  }  
   case QMessageBox::RejectRole:  
  {  
       qDebug()<<"打开文件…"<<Qt::Checked<<Qt::endl;  
       // 弹出文件对话框，让用户选择文本文件  
       QString fileName = QFileDialog::getOpenFileName  
          (this, tr("打开文件"), "", tr("文本文件(\*.txt);;word文件(\*.doc);;word文件(\*.docx)"));  
       if (!fileName.isEmpty()) {  
           // 用户选择了文件，尝试打开并读取  
           QFile file(fileName);  
           if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {  
               // 无法打开文件，显示错误消息  
               QMessageBox::warning(this, "错误", "无法打开文件！");  
               return;  
          }  
           else{  
               QUrl url = QUrl::fromLocalFile(fileName);  
               if (!QDesktopServices::openUrl(url)) {  
                   qDebug() << "无法打开文件：" << fileName;  
              }  
          }  
      }break;  
  }  
   default:break;  
  }  
}

//文件打开、保存；  
#include "moments.h"  
​  
//打开文件；  
void Moments::on\_pushButton\_4\_clicked()  
{  
   qDebug()<<"打开文件…"<<Qt::Checked<<Qt::endl;  
   // 弹出文件对话框，让用户选择文本文件  
   QString fileName = QFileDialog::getOpenFileName  
      (this, tr("打开文件"), "",tr("文本文件 (\*.txt);;word文件(\*.doc);;word文件(\*.docx)"));  
   if (!fileName.isEmpty()) {  
       // 用户选择了文件，尝试打开并读取  
       QFile file(fileName);  
       if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {  
           // 无法打开文件，显示错误消息  
           QMessageBox::warning(this, "错误", "无法打开文件！");  
           return;  
      }  
       // 使用 QTextStream 读取文件内容  
       QTextStream in(&file);  
       QString fileContent = in.readAll(); // 读取所有内容  
       // 将文件内容设置到 QTextEdit 控件中  
       ui->textEdit->setPlainText(fileContent); // 假设你的 QTextEdit 对象名为 textEdit  
  }  
}  
​  
//保存所写的内容；  
void Moments::on\_pushButton1\_clicked(bool checked)  
{  
   qDebug()<<"进行保存:"<<checked<<Qt::endl;  
   if(savePath.isEmpty())  
  {  
       QString file\_path;  
       while(file\_path.isEmpty()){  
           file\_path = QFileDialog::getSaveFileName(this,"选择路径",".","\*.txt;;\*.doc;;\*.docx");  
           if (file\_path.isEmpty()) {  
               //用户取消对话框或未选择文件  
               return;  
          }  
           break;  
      }  
       qDebug()<<file\_path<<Qt::endl;  
       this->savePath = file\_path;  
  }  
​  
   QFile txt(this->savePath);  
​  
   bool tag = txt.open(QIODevice::WriteOnly);  
   if(tag){  
       txt.write(ui->textEdit->toPlainText().toStdString().data());  
  }  
   else{  
       qDebug()<<"打开文件失败"<<Qt::endl;  
       QMessageBox::critical(NULL,"错误","无法打开文件，请检查权限",QMessageBox::Yes,QMessageBox::Yes);  
  }  
   txt.close();  
   this->saved = true;  
}

//新建和删除；  
#include "accounts.h"  
​  
//新建记录；  
void Accounts::on\_pushButton\_clicked()  
{  
   QString description = ui->descriptionlineEdit->text();  
   QString amountStr = ui->amountlineEdit->text();  
   bool valid;  
   double amount = amountStr.toDouble(&valid);  
​  
   if (!description.isEmpty() && valid) {  
       QString transaction = description + " - " + amountStr;  
       transactions.append(transaction);  
       ui->listWidget->addItem(transaction);  
       ui->descriptionlineEdit->clear();  
       ui->amountlineEdit->clear();  
       //这里设计了一个限定额度；  
       amount0+=amount;  
       if(amount0 >= standard){  
           QMessageBox::warning(this, "消费过高", "请注意钱财余额！");  
      }  
  }  
   else {  
       QMessageBox::warning(this, "非法输入", "请输入正确的描述和额度！");  
  }  
}  
​  
​  
//删除记录；  
//获取删除记录中的金额值；  
void Accounts::extracted(bool &ok, int &number, QStringList &parts) {  
   if (!parts.isEmpty()) {  
       // 尝试将最后一个部分转换为整数  
       number = parts.last().toInt(&ok);  
  }  
}  
//删除相应行，同时更新已经花掉金钱的额度；  
void Accounts::on\_pushButton\_2\_clicked() {  
   QString amountStr = ui->amountlineEdit->text();  
   bool valid;  
   amount1 = amountStr.toDouble(&valid);  
​  
   int currentRow = ui->listWidget->currentRow();  
   if (currentRow != -1) {  
       QString itemText = ui->listWidget->currentItem()->text();  
       transactions.removeAt(currentRow);  
       ui->listWidget->takeItem(currentRow);  
​  
       //假设已经获取了当前项的文本，进行分割文本  
       QStringList parts = itemText.split('-', Qt::SkipEmptyParts);  
​  
       bool ok = false;  
       int number = 0;  
       extracted(ok, number, parts);  
​  
       if (ok) {  
           amount0 -= number;  
      } else {  
           // 提取失败，可能因为最后一个部分不是有效的数字  
           qDebug() << "未能找到数字！";  
      }  
  }  
}  
//获取限定额度；  
void Accounts::on\_standardEdit\_returnPressed()  
{  
   QString stanDard = ui->standardEdit->text();  
   standard = stanDard.toDouble();  
}  
//当限定额度变化时返回当前值；  
void Accounts::on\_standardEdit\_textChanged(const QString &stanDard)  
{  
   standard = stanDard.toDouble();  
}

//日期获取和导入至moments类；  
#include "daysmaster.h"  
#include "moments.h"  
​  
DaysMaster::DaysMaster(QWidget \*parent)  
  : QMainWindow(parent)  
  , ui(new Ui::DaysMaster)  
{  
   ui->setupUi(this);  
  ......  
     
   // 连接按钮的点击信号到槽函数  
   connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(onCountdown()));  
​  
}  
​  
//计算日期；  
void DaysMaster::onCountdown()  
{  
   //目标日和预计日期；  
   QDateTime targetDateTime = targetDateEdit->dateTime();  
   QDateTime currentDateTime = QDateTime::currentDateTime();  
​  
   //计算剩余的天数；  
   qint64 msecs = targetDateTime.msecsTo(currentDateTime.addDays(0)) / 1000;  
   qint64 days = msecs / (24 \* 60 \* 60);  
​  
   if (targetDateTime > currentDateTime) {  
       //更新标签显示剩余天数；  
       countdownLabel->setText("距离这一天还有 " + QString::number(-days+1) + " 天");  
  }  
   else{  
       //更新标签显示剩余天数；  
       countdownLabel->setText("这一天已经过去了 " + QString::number(days) + " 天");  
  }  
}  
​  
//为了不多此一举，可以直接在moments里面打开；  
void DaysMaster::on\_pushButton\_5\_clicked()  
{  
   QMessageBox judge;  
   judge.setWindowTitle("注意");  
   judge.setStyleSheet("QLabel{""min-width:200px;""min-height:80px;"  
                       "font-size:20px;""}");  
   judge.setText("   是否关闭？");  
   judge.setStandardButtons(QMessageBox::Yes|QMessageBox::No);  
   judge.setDefaultButton(QMessageBox::No);  
​  
   int res=judge.exec();  
   if(res==QMessageBox::Yes)  
  {  
       mt = new Moments;  
       mt->show();  
       mt->setWindowTitle("Moments");  
       mt->receiveText(ui->dateEdit->text()+"\n"  
                       +ui->label->text()+"\n"+ui->lineEdit->text());  
       this->close();  
  }  
}

//记账功能具体实现；  
#include "todolist"  
​  
//设置额度；  
static int i = 0;  
void ToDoList::on\_addButton\_clicked()  
{  
   QString amountStr = ui->lineEdit->text();  
   if(i>amountStr.toInt()){  
       QMessageBox judge;  
       judge.setStyleSheet("QLabel{""min-width:200px;""min-height:80px;"  
                           "font-size:20px;""}");  
       judge.setText(" 太累了，别这么忙！");  
       judge.setStandardButtons(QMessageBox::Yes);  
       judge.exec();  
  }  
   else{  
       // 创建一个新的  
       QLineEdit \*newLineEdit = new QLineEdit("New LineEdit", this);  
       lineeditLayout->addWidget(newLineEdit);  
       // 可选：更新布局  
       lineeditLayout->update();  
       newLineEdit->clear();  
       newLineEdit->setStyleSheet("font-size: 12pt;");  
       newLineEdit->setStyleSheet("background-image: url(:/new/prefix3/R-C.jpg);");  
​  
       // 创建一个新的  
       QCheckBox \*newCheckBox = new QCheckBox("New CheckBox", this);  
       checkLayout->addWidget(newCheckBox);  
       // 可选：更新布局  
       checkLayout->update();  
       newCheckBox->setText("BINGO!");  
       newCheckBox->setStyleSheet("background-image: url(:/new/prefix3/R-C.jpg);");  
  }  
   i++;  
}

**3、单元测试**

针对每个类型设计单独案例，测试每个功能能否正常运行。

针对类型之间切换，测试其是否能够正常进行。

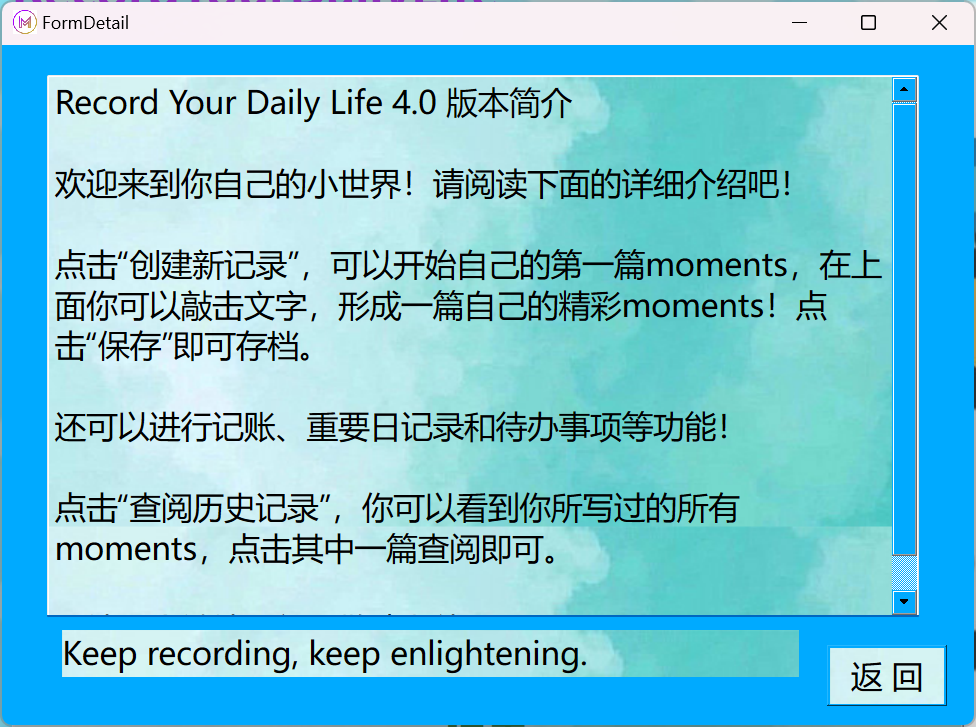
**五、单元测试**

**以下是各页面打开的样式：**

LoginFirst打开的样式：



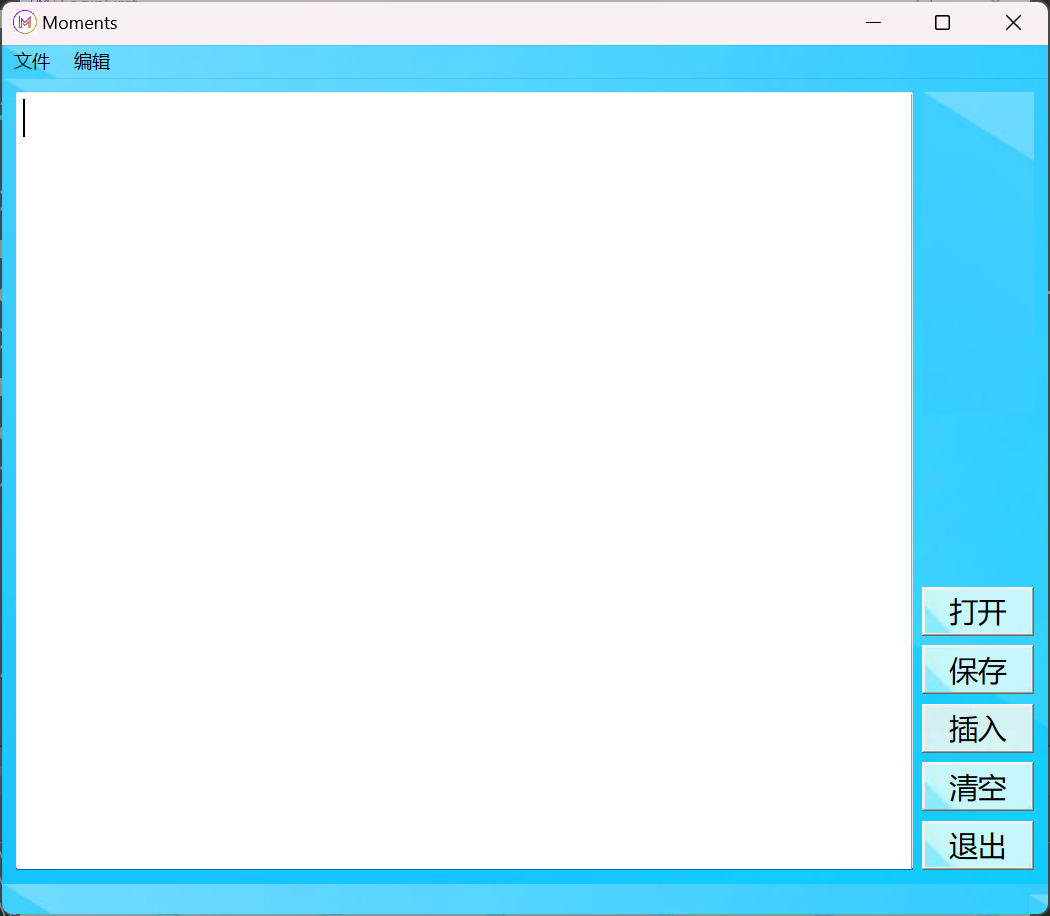
FormDetail打开的样式：



Select打开的样式：



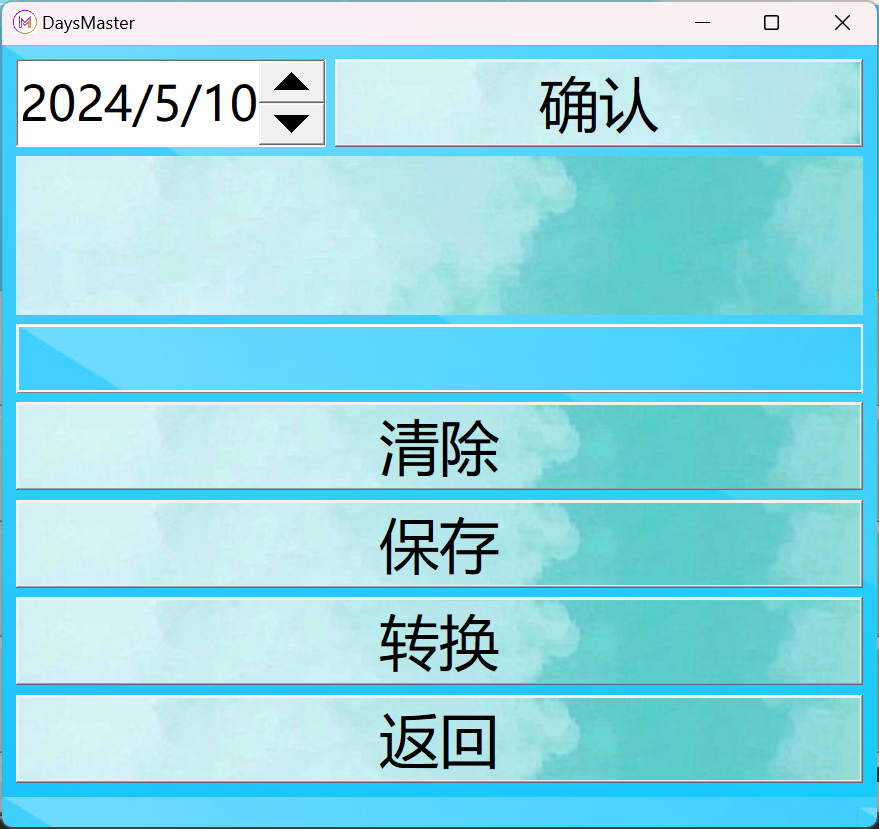
Moments打开的样式：



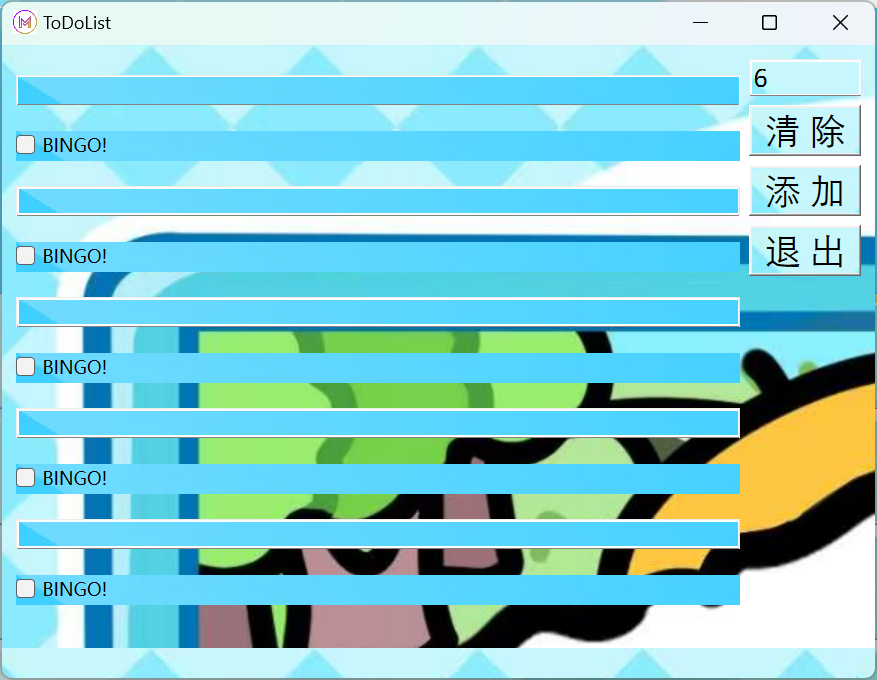
Accounts打开的样式：



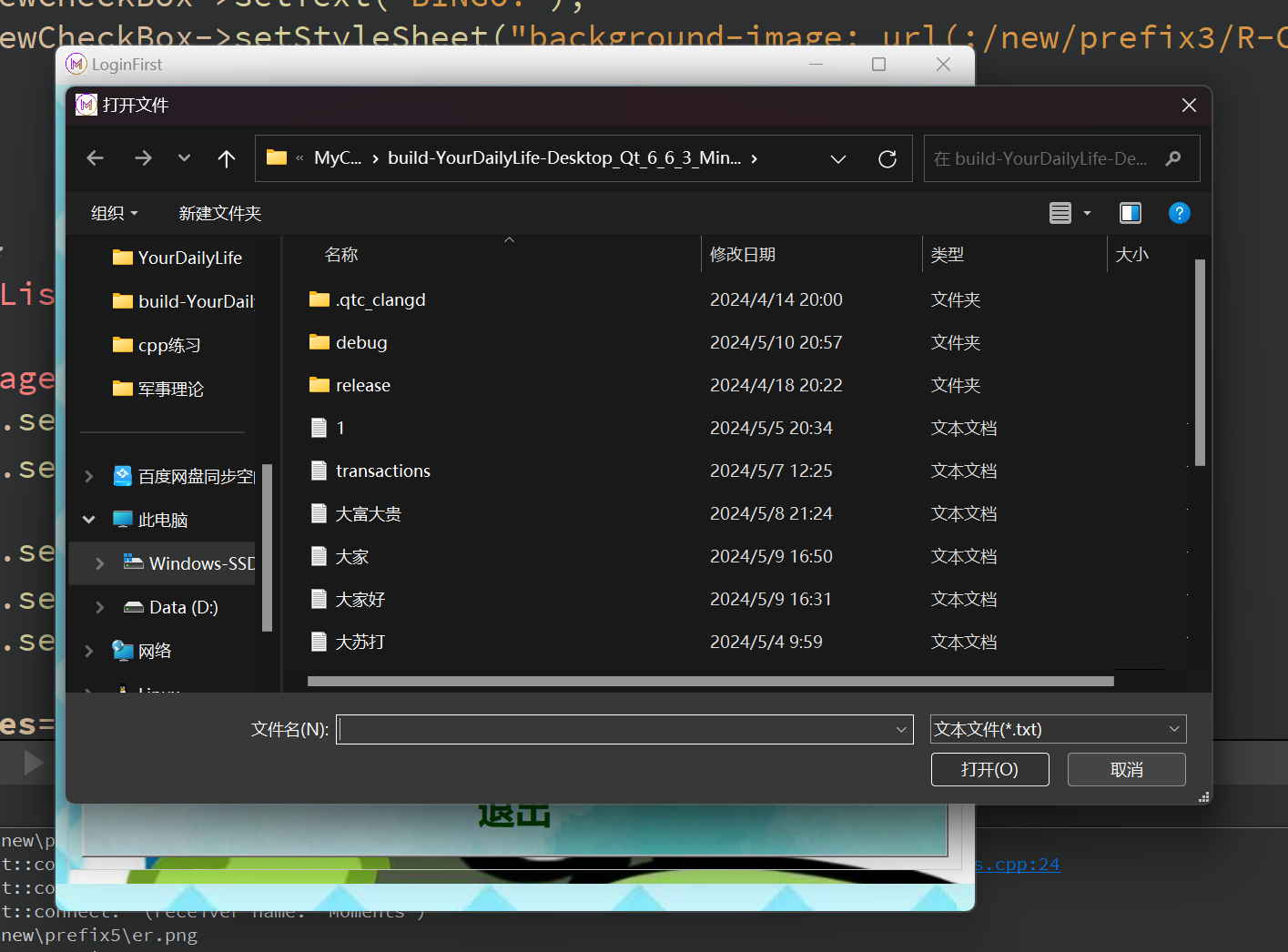
DaysMaster打开的样式：



ToDoList打开的样式：



保存记录的打开窗口：



**六、收获**

**1、程序学习的收获**

较为熟练掌握了在Qt程序里面编辑C++语言，并借助ui的form形式来编辑页面。

**几处代码的改进：**

static int i = 0;  
void ToDoList::on\_addButton\_clicked()  
{  
 //i++; 不应当放在这里;  
 QString amountStr = ui->lineEdit->text();  
 if(i>amountStr.toInt())  
 {  
 QMessageBox judge;  
 judge.setStyleSheet("QLabel{""min-width:200px;""min-height:80px;"  
 "font-size:20px;""}");  
 judge.setText(" 太累了，别这么忙！");  
 judge.setStandardButtons(QMessageBox::Yes);  
 judge.exec();  
 }  
 else  
 {  
 ......  
 i++;  
 }  
}

这里的i=0处应当把i++这个语句放在else语句中的图示位置，从而在更新计数时如果出现超额的情况，i的数量不会增加。

#include <QMessageBox>  
  
//关闭页面；  
void ToDoList::on\_pushButton\_2\_clicked()  
{  
 QMessageBox judge;  
 judge.setWindowTitle("注意");  
 judge.setStyleSheet("QLabel{""min-width:200px;""min-height:80px;"  
 "font-size:20px;""}");  
 judge.setText(" 是否清空记录？");  
 judge.setStandardButtons(QMessageBox::Yes|QMessageBox::No);  
 judge.setDefaultButton(QMessageBox::No);  
  
 int res=judge.exec();  
 if(res==QMessageBox::Yes)  
 {  
 this->close();  
 }  
}

我们可以通过QMessageBox库的使用来打开一个提示框，为用户提供一个退出/清空的确认。还可以设置judge的大小和按钮情况。

//源类;  
  
void DaysMaster::on\_pushButton\_5\_clicked()  
{  
 QMessageBox judge;  
 judge.setWindowTitle("注意");  
 judge.setStyleSheet("QLabel{""min-width:200px;""min-height:80px;"  
 "font-size:20px;""}");  
 judge.setText(" 是否关闭？");  
 judge.setStandardButtons(QMessageBox::Yes|QMessageBox::No);  
 judge.setDefaultButton(QMessageBox::No);  
  
 int res=judge.exec();  
 if(res==QMessageBox::Yes)  
 {  
 mt = new Moments;  
 mt->show();  
 mt->setWindowTitle("Moments");  
 mt->receiveText(ui->dateEdit->text()+"\n"  
 +ui->label->text()+"\n"+ui->lineEdit->text());  
 this->close();  
 }  
}

//目标类;  
  
void Moments::receiveText(const QString &text) {  
 ui->textEdit->setPlainText(text);  
}

通过setPlainText和设置一个receiveText的函数来传递你所需要的文本，这种方法非常好用！

void LoginFirst::on\_pushButton\_clicked()  
{  
 QMessageBox judge;  
 judge.setStyleSheet("QLabel{""min-width:300px;""min-height:120px;"  
 "font-size:30px;""}");  
 judge.setText(" 请选择打开方式！");  
 judge.addButton("moments",QMessageBox::AcceptRole);  
 judge.addButton("其它文件",QMessageBox::RejectRole);  
  
 int res=judge.exec();  
 switch(res)  
 {  
 case QMessageBox::AcceptRole:  
 {  
 ......  
 break;  
 }  
 case QMessageBox::RejectRole:  
 {  
 ......  
 break;  
 }  
 default:break;  
 }  
}

这里通过switch语句判断res值的情况从而确定不同的处理方式，但是不要忘记break语句！

void DaysMaster::onCountdown()  
{  
 //目标日和预计日期；  
 QDateTime targetDateTime = targetDateEdit->dateTime();  
 QDateTime currentDateTime = QDateTime::currentDateTime();  
  
 //计算剩余的天数；  
 qint64 msecs = targetDateTime.msecsTo(currentDateTime.addDays(1)) / 1000;  
 qint64 days = msecs / (24 \* 60 \* 60);  
  
 if (targetDateTime > currentDateTime) {  
 //更新标签显示剩余天数；  
 countdownLabel->setText("距离这一天还有 " + QString::number(-days) + " 天");  
 }  
 else{  
 //更新标签显示剩余天数；  
 countdownLabel->setText("这一天已经过去了 " + QString::number(days) + " 天");  
 }  
}

这是之前的代码，未能解决选择当天的问题。

void DaysMaster::onCountdown()  
{  
 //目标日和预计日期；  
 QDateTime targetDateTime = targetDateEdit->dateTime();  
 QDateTime currentDateTime = QDateTime::currentDateTime();  
  
 //计算剩余的天数；  
 qint64 msecs = targetDateTime.msecsTo(currentDateTime.addDays(0)) / 1000;  
 qint64 days = msecs / (24 \* 60 \* 60);  
  
 if (targetDateTime > currentDateTime) {  
 //更新标签显示剩余天数；  
 countdownLabel->setText("距离这一天还有 " + QString::number(-days+1) + " 天");  
 }  
 else{  
 //更新标签显示剩余天数；  
 countdownLabel->setText("这一天已经过去了 " + QString::number(days) + " 天");  
 }  
}

更新后的代码，避免了选定日期时当天的情况。

**2、个人成长的收获**

这次的大作业给了我无与伦比的体验。我开始学着用一名计算机专业学生的视角去解决问题，同时尝试用编程语言去完成并实现各种各样的功能。在这个过程中，由于本人第一次接触Qt，再加上很多C++的专业知识还不是很掌握，里面的功能没有搞得很清楚，所以在起步阶段是非常吃力的。但是在这一个月的探索中，我逐渐找到了开发项目的乐趣，我会为了实现一个心仪已久的功能而沾沾自喜，开始学会在网上查找资料，学习前辈的作品里面的亮点，用各种大模型去解决我实际过程中遇到的种种问题，而不是一味地求问学长学姐……这一切都培养了我独立自主思考并解决问题的能力，或许能够为我以后可能的科研之路打下坚实的基础……