索引：

Properties:成员变量
Public Function:公有的成员函数
Reimplemented Public Function:公有虚函数
Public Slots:槽函数
Signals:信号函数
Static Public Member:静态成员
Reimplemented Protected Function:保护的虚函数
Detailed Description:详细说明

类

QApplication 应用程序
QWidgets 控制基类
QObject 顶层基类
QPushButton 按钮
QTexCodec 编码转换
QMainWindow 主窗口
QDialog 对话框
Qstring 字符串
QSlider 滑块
QSpinBox 选值框
QLineEdit 行编辑控件(常见)
QDoubleValidator 数字验证器
QFont 字体
QHBoxLayout 水平布局器
QVBoxLayout 垂直布局
QFram 显示框架(Qlabel 的基类)
QRect 矩形(x,y,w,h)
QPoint 位置(x,y)
QSize 大小(w,h)
QPainter Qt 的二维图形引擎
QImage 图片
QDir 目录操作
QMutex 互斥锁
QSemaphore 信号量
QReadWriteLock 读写锁
QWaitCondition 等待条件

目录

Day01

一、第一个 QT 程序

Hello

main.cpp

```
#include <QApplication>
#include <QLabel> //标签控件

int main(int argc,char** argv){
    //构建 Qt 应用程序
    QApplication app(argc,argv);

    //创建标签控件(图形对象,部件,构件,组件)
    QLabel label("Hello Qt!");
    //显示标签控件
    label.show();

    //让应用程序进入事件循环
    return app.exec();
}
```

hello.pro

```
#############################################################################
# Automatically generated by qmake (3.0) Thu May 7 14:04:09 2020
#############################################################################
QT += widgets
TEMPLATE = app
TARGET = Hello
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

main.cpp_____qmak -project_____>Hello.pro 添加 QT += widgets)_____qmake_____>Makefile _____>make_____>Hello(可执行程序)_____>./Hello

Button

main.cpp

```
#include <QApplication>
#include <QPushButton>

int main(int argc,char* argv[]){
        QApplication app(argc,argv);

        //创建按钮控件
        QPushButton button("Button");
        //显示按钮控件
        button.show();

        return app.exec();
}
```

查看当前默认用的编码方式:
底行命令：set fileencoding
设置当前用的编码方式:
底行命令：set fileencoding=gbk(euc-cn)

Button2

Main.cpp

```
#include <QApplication>
#include <QPushButton>
#include <QTextCodec>

int main(int argc,char* argv[]){
        QApplication app(argc,argv);

        //1)创建编码转换对象
        QTextCodec* codec = QTextCodec::codecForName("GBK");
        //2)将要显示的字符串转换为 Qt 内部的 unicode 编码
        QString str = codec->toUnicode("下午听课有点困");

        //创建按钮控件
        QPushButton button(str);
        //显示按钮控件
        button.show();

```

```
18      return app.exec();
19 }
```

二、字符编码转换

Codec

Maincpp

```
1 #include <QApplication>
2 #include <QLabel>
3 #include <QPushButton>
4 #include <QTextCodec>
5 int main(int argc,char** argv){
6       QApplication app(argc,argv);
7
8       //创建编码转换对象
9       QTextCodec* codec = QTextCodec::codecForName("GBK");
10      //将要显示文本字符串转换为 unicode 编码再使用
11      QLabel label( codec->toUnicode("我是标签") );
12      label.show();
13      QPushButton button( codec->toUnicode("我是按钮") );
14      button.show();
15
16      return app.exec();
17 }
```

三、父窗口创建

Parent

Main.cpp

```
1 #include <QApplication>
2 #include <QLabel>
3 #include <QPushButton>
4 #include <QMainWindow>//主窗口
5 #include <QDialog>//对话框
6 #include <QWidget>//控件基类
7
8 int main(int argc,char** argv){
9       QApplication app(argc,argv);
10      //创建父窗口
11      //QMainWindow parent;
12      QDialog parent;
13      //QWidget parent;
14
15      //创建标签和按钮控件并停靠在父窗口上面
```

```
16      QLabel label("我是标签",&parent);
17      QPushButton button("我是按钮",&parent);
18
19      //设置父窗口和每个控件大小和位置
20      parent.resize(480,320);
21      label.move(30,50);
22      button.move(30,200);
23
24      //显示父窗口,上面停靠的控件也会一起被显示
25      parent.show();
26      return app.exec();
27 }
```

四、信号和槽函数

信号的定义

• Qt 类库中，预定义了若干个信号，可以直接使用，也可以根据需要自定义信号函数，如果自定义信号需要满足如下要求

– 继承 QObject 或者其子类

– 类中包含 Q_OBJECT 宏，用于启动元对象系统，属于语法扩展，对应 moc

注：moc 全称是 Meta-Object Compiler，也就是“元对象编译器”

– 信号需要使用 signals 标记，只需声明，不能写定义，元对象编译器(moc)会提供必要的代码实现

```
class XX:public QObject{
Q_OBJECT
signals:
void signal_func(..); //信号函数
};知
识
讲
解
```

槽的定义

• Qt 类库中，预定义了若干个槽函数，可以直接使用，也可以根据需要自定义槽函数，如果自定义槽需要满足如下要求

– 继承 QObject 或者其子类

– 类中包含 Q_OBJECT 宏，用于启动元对象系统

– 槽函数需要使用 slots 标记，其声明、定义和普通成员函数相同，槽函数可以连接到某个信号上，当信号被发射时，槽函数将被触发和执行，另外槽函数也可以当做普通的成员函数直接调用

```
class XX:public QObject{
Q_OBJECT
public slots:
void slot_func(...){...}//槽函数
```

};知
识
讲
解
信号和槽的连接
• 不同对象的信号或槽的定义时是彼此独立的，不会有任何关联，如果希望借助信号和槽机制实现交互功能，必须事先建立信号和槽的连接
• 利用 QObject 中静态成员函数 connect 即可方便的建立连接：

```
QObject::connect(
const QObject * sender,//信号发送对象指针
const char * signal, //要发送的信号函数，
const QObject * receiver, //信号的接收对象指针
const char * method)；//接收信号后要执行的槽函数
```

注：
信号函数需要使用“SIGNAL(...)” 宏将其转换为 const char*
槽函数需要使用“SLOT(...)”宏将其转换为 const char*

Close
Main.cpp

```
#include <QApplication>
#include <QPushButton>
#include <QLabel>
#include <QDialog>
int main(int argc,char** argv){
    QApplication app(argc,argv);
    QDialog parent;//父窗口
    QLabel label("我是一个标签",&parent);//栈对象
    QPushButton* button = new QPushButton("我是一个标签",&parent);//堆对象
    parent.resize(480,320);//父窗口大小
    label.move(50,50);//标签位置
    button->move(50,200);//按钮位置

    QPushButton* button2 = new QPushButton("关闭",&parent);//关闭按钮
    button2->move(240,200);

    parent.show();//显示父窗口
    //实现点击按钮关闭标签功能
    QObject::connect(button,SIGNAL(clicked(void)),
                     &label,SLOT(close(void)));
    //练习:增加一个"关闭"按钮控件,实现点击该按钮关闭应用程序
    QObject::connect(button2,SIGNAL(clicked(void)),
                     //&app,SLOT(closeAllWindows(void)));
                     &app,SLOT(quit(void)));
```

```
25
26      return app.exec();
27 }
```

Day02

五、滑块和选值框

Changed

Main.cpp

```
 1 #include <QApplication>
 2 #include <QSlider>//滑块
 3 #include <QSpinBox>//选值框
 4 #include <QDialog>
 5
 6 int main(int argc,char** argv){
 7     QApplication app(argc,argv);
 8
 9     QDialog parent;//父窗口
10     parent.resize(200,100);
11
12     //创建水平滑块控件并停靠在父窗口上面
13     QSlider slider(Qt::Horizontal,&parent);
14     slider.move(20,30);
15     slider.setRange(0,50);//设置数据变化范围
16     //创建选值框控件并停靠在父窗口上面
17     QSpinBox spin(&parent);
18     spin.move(120,30);
19     spin.setRange(0,50);//设置数据变化范围
20     //显示父窗口
21     parent.show();
22
23     //滑块滑动,选值框数值随之改变
24     QObject::connect(&slider,SIGNAL(valueChanged(int)),
25                      &spin,SLOT(setValue(int)));
26     //选值框数值改变,滑块随之滑动
27     QObject::connect(&spin,SIGNAL(valueChanged(int)),
28                      &slider,SLOT(setValue(int)));
29
30     return app.exec();
31 }
```

六、加法计算器

CalculatorDialog.h

```
1 #ifndef __CALCULATORDIALOG_H
2 #define __CALCULATORDIALOG_H
3
4 #include <QDialog>
5 #include <QLabel>
6 #include <QPushButton>
7 #include <QLineEdit>//行编辑控件
8 #include <QDoubleValidator>//数字验证器
9 #include <QHBoxLayout>//水平布局器
10 #include <QFont>//字体
11 //继承 QDialog,当前类就也是父窗口类
12 class CalculatorDialog:public QDialog{
13         Q_OBJECT //moc
14 public:
15         //构造函数
16         CalculatorDialog(void);
17 public slots://自定义槽函数
18         //设置按钮为正常或禁用状态的槽函数
19         void enableButton(void);
20         //点击按钮计算结果的槽函数
21         void calcClicked(void);
22 private:
23         QLineEdit* m_editX;//左操作数
24         QLineEdit* m_editY;//右操作数
25         QLineEdit* m_editZ;//显示计算结果
26         QLabel* m_label;//加号"+"
27         QPushButton* m_button;//等号"="
28         QHBoxLayout* m_hBoxLayout;//水平布局器
29 };
30 #endif//__CALCULATORDIALOG_H
```

CalculatorDialog.cpp

```
1 #include "CalculatorDialog.h"
2 //构造函数
3 CalculatorDialog::CalculatorDialog(void){
4         //界面初始化
5         setWindowTitle("计算器");//窗口标题
6         QFont font;//字体
7         font.setPointSize(20);//字体大小:20
8         setFont(font);//设置字体
9         //左操作数,参数 this 表示父窗口指针
10        m_editX = new QLineEdit(this);
11        //设置文本对齐方式:右对齐
12        m_editX->setAlignment(Qt::AlignRight);
```

```
13      //设置数字验证器,只能输入数字形式的文本内容
14      m_editX->setValidator(new QDoubleValidator(this));
15      //右操作数
16      m_editY = new QLineEdit(this);
17      m_editY->setAlignment(Qt::AlignRight);
18      m_editY->setValidator(new QDoubleValidator(this));
19      //显示计算结果
20      m_editZ = new QLineEdit(this);
21      m_editZ->setAlignment(Qt::AlignRight);
22      m_editZ->setReadOnly(true);//设置只读
23      //"+"
24      m_label = new QLabel("+",this);
25      //"="
26      m_button = new QPushButton("=",this);
27      m_button->setEnabled(false);//设置按钮禁用状态
28      //布局器:自动调整大小和位置
29      m_hBoxLayout = new QHBoxLayout(this);
30      //将控件按水平方向添加到布局器中
31      m_hBoxLayout->addWidget(m_editX);
32      m_hBoxLayout->addWidget(m_label);
33      m_hBoxLayout->addWidget(m_editY);
34      m_hBoxLayout->addWidget(m_button);
35      m_hBoxLayout->addWidget(m_editZ);
36      //设置布局器
37      setLayout(m_hBoxLayout);
38
39      //连接信号和槽函数
40      //左右操作数文本改变时,将会发送 textChanged 信号,
41      //连接设置按钮为正常或禁用状态的槽函数
42      connect(m_editX,SIGNAL(textChanged(QString)),
43              this,SLOT(enableButton(void)));
44      connect(m_editY,SIGNAL(textChanged(QString)),
45              this,SLOT(enableButton(void)));
46      //点击等号按钮,发送 clicked 信号,连接计算结果的槽函数
47      connect(m_button,SIGNAL(clicked(void)),this,SLOT(calcClicked(void)));
48 }
49 //设置按钮为正常或禁用状态的槽函数
50 void CalculatorDialog::enableButton(void){
51      bool bXOk,bYOk;
52      //text():获取文本内容(QString)
53      //toDouble():将 QString 转换为字符串,如果转换失败将 bXOk 置 false,成功置 true
54      m_editX->text().toDouble(&bXOk);
55      m_editY->text().toDouble(&bYOk);
56      //如果左右操作数都输入有效数据,设置按钮为正常状态否则设置禁用状态
```

```
57          m_button->setEnabled(bXOk && bYOk);
58 }
59 //点击按钮计算结果的槽函数
60 void CalculatorDialog::calcClicked(void){
61          //计算相加结果
62          double res = m_editX->text().toDouble() + m_editY->text().toDouble();
63          //将结果转为 QString,number:将参数的数字转换为 QString
64          QString str = QString::number(res);
65          //显示结算结果
66          m_editZ->setText(str);
67 }
```

main.cpp

```
 1 #include <QApplication>
 2 #include "CalculatorDialog.h"
 3
 4 int main(int argc,char** argv){
 5          QApplication app(argc,argv);
 6
 7          CalculatorDialog dialog;
 8          dialog.show();
 9
10          return app.exec();
11 }
```

七、获取系统时间

Time

TimeDialog.h

```
 1 #ifndef __TIMEDIALOG_H
 2 #define __TIMEDIALOG_H
 3
 4 #include <QDialog>
 5 #include <QLabel>
 6 #include <QPushButton>
 7 #include <QVBoxLayout>//垂直布局器
 8 #include <QFont>
 9 #include <QTime> //时间
10
11 class TimeDialog:public QDialog{
12          Q_OBJECT //moc
13 public:
14          TimeDialog(void);//构造函数
15 public slots:
```

```
16      void getTime(void);//获取系统时间的槽函数
17 private:
18      QLabel* m_label;//显示时间标签
19      QPushButton* m_button;//获取时间的按钮
20      QVBoxLayout* m_vBoxLayout;//垂直布局器
21 };
22
23 #endif//__TIMEDIALOG_H
```

TimeDialog.cpp

```
 1 #include "TimeDialog.h"
 2 //构造函数
 3 TimeDialog::TimeDialog(void){
 4      //界面初始化
 5      setWindowTitle("系统时间");//设置窗口标题
 6      QFont font;
 7      font.setPointSize(20);
 8      setFont(font);//设置字体大小
 9
10      m_label = new QLabel(this);//显示系统时间标签
11      //设置标签边框样式:凹陷面板
12      m_label->setFrameStyle(QFrame::Panel|QFrame::Sunken);
13      //设置文本对齐方式:水平和垂直都居中
14      m_label->setAlignment(Qt::AlignHCenter|Qt::AlignVCenter);
15
16      m_button = new QPushButton("获取系统时间",this);
17
18      //设置垂直布局器:自动调正大小和位置
19      m_vBoxLayout = new QVBoxLayout(this);
20      m_vBoxLayout->addWidget(m_label);
21      m_vBoxLayout->addWidget(m_button);
22      setLayout(m_vBoxLayout);
23
24      //信号和槽连接,点击获取时间按钮,发送 clicked 信号连接到获取时间的槽函数
25      connect(m_button,SIGNAL(clicked(void)),this,SLOT(getTime(void)));
26 }
27
28 //获取系统时间的槽函数
29 void TimeDialog::getTime(void){
30      QTime time = QTime::currentTime();//获取系统时间
31      QString str = time.toString("hh:mm:ss");//将时间转换为 QString
32      m_label->setText(str);//将字符串时间显示到 label 控件上
33 }
```

main.cpp

```
1 #include <QApplication>
2 #include "TimeDialog.h"
3
4 int main(int argc,char** argv){
5         QApplication app(argc,argv);
6
7         TimeDialog dialog;
8         dialog.show();
9
10        return app.exec();
11 }
```

八、designer 设计师（day02+day03）

继承和组合的方式 CalculatorDialog2.tar.gz//继承方式
CalculatorDialog2_2.tar.gz//组合方式

九、QT 创造器(qtcreator)

alt+0 ------>显示和关闭项目栏
F4 ------>头文件源文件切换
ctrl+r ------>快速运行可执行程序
ctrl+s -------->保存
ctrl+b -------->构建不运行
ctrl+shift+F11---->全屏/恢复
光标停在函数上按下 F1 就可以打开具体函数，按两下 F1 全屏打开

十、QT 时间处理机制

绘图事件处理：

1.把图片资源转换成源代码，一起编译
1.1 用 rcc 把图片资源转换成 C++源代码
1.2 重写 paintevent 事件，用 paintevent 事件的 Qprint 绘图工具
第一步：打开 qtcreator,新建工程文件
第二步：向工程目录里添加资源文件：
1.2.1 将资源文件拷贝到工程目录里
1.2.2 点击文件-->新建文件或项目---->QT--->QT Resource File
1.2.3 showinage.qec--->添加前缀--->添加后缀(必须是当前工程文件的文件)
第三步：编写代码
bug:图片错开(frame 和 painter 坐标没对齐)

showinagedialog.h

```
#ifndef SHOWIMAGEDIALOG_H
#define SHOWIMAGEDIALOG_H

#include <QDialog>
#include <QPainter>//画家类(QT 的二维图形引擎)
#include <QDebug>

namespace Ui {
class ShowImageDialog;
}

class ShowImageDialog : public QDialog
{
    Q_OBJECT

public:
    explicit ShowImageDialog(QWidget *parent = 0);
    ~ShowImageDialog();

private slots:
    //转到槽自动生成:上一张按钮对应的槽函数
    void on_m_prevButton_clicked();
    //转到槽自动生成:下一张按钮对应的槽函数
    void on_m_nextButton_clicked();
private:
    //绘图事件处理函数[virtual]
    void paintEvent(QPaintEvent *);
private:
    Ui::ShowImageDialog *ui;
    int m_index;//记录图片的索引
};

#endif // SHOWIMAGEDIALOG_H
```

showinagedialog.cpp

```
#include "showimagedialog.h"
#include "ui_showimagedialog.h"

ShowImageDialog::ShowImageDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::ShowImageDialog)
{
```

```
 8      ui->setupUi(this);
 9      m_index = 0;
10 }
11
12 ShowImageDialog::~ShowImageDialog()
13 {
14      delete ui;
15 }
16 //转到槽自动生成:上一张按钮对应的槽函数
17 void ShowImageDialog::on_m_prevButton_clicked()
18 {
19      if(--m_index < 0){
20          m_index = 9;
21      }
22      update();//更新界面(触发绘图事件)
23 }
24 //转到槽自动生成:下一张按钮对应的槽函数
25 void ShowImageDialog::on_m_nextButton_clicked()
26 {
27      if(++m_index > 9){
28          m_index = 0;
29      }
30      update();//更新界面(触发绘图事件)
31 }
32 //绘图事件处理函数
33 void ShowImageDialog::paintEvent(QPaintEvent *)
34 {
35      //创建画家对象,参数表示绘制设备(可以是当前父窗口)
36      QPainter painter(this);
37      //获取绘图的矩形区域
38      QRect rect = ui->frame->frameRect();
39      //平移 rect 记录坐标,让其和父窗口(painter)坐标系一致
40      rect.translate( ui->frame->pos());
41
42      //qDebug() << "frame 位置:" << ui->frame->pos();
43      //qDebug() << "rect 记录位置:" << rect.x() << "," << rect.y();
44
45      //准备要绘制图片
46      //":"表示相对资源文件路径,":/images/x.jpg"
47      QImage image(":/images/"+QString::number(m_index)+".jpg");
48      //把准备好的图像(image)画到 rect 矩形区域中
49      painter.drawImage(rect,image);
50 }
```

main.cpp

```
1 #include "showimagedialog.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6         QApplication a(argc, argv);
7         ShowImageDialog w;
8         w.show();
9
10        return a.exec();
11 }
```

2. 运行时加载图片

十一、定时器事件

1. 打开 qtcreator 创建工程目录(lottery)

2. 左侧”项目”--->勾选掉 Shaow build

3. 将 photo 拷贝到 lottery 工程目录

4. 打开设计模式进入 designer

5. 右点击“开始”button--->转到槽

6. 编译

lotterydialog.h

```
1 #ifndef LOTTERYDIALOG_H
2 #define LOTTERYDIALOG_H
3
4 #include <QDialog>
5 #include<QDir>          //目录操作
6 #include<QPainter> //画家类
7 #include<QVector>    //向量容器(类似数组)
8 #include<QTime>       //时间(设置随机数的种子)
9 #include<QDebug>      //打印调用
10
11 namespace Ui {
12 class LotteryDialog;
13 }
```

```
14
15 class LotteryDialog : public QDialog
16 {
17     Q_OBJECT
18
19 public:
20     explicit LotteryDialog(QWidget *parent = 0);
21     ~LotteryDialog();
22
23 private slots:
24     //开始按钮对应的槽函数
25     void on_pushButton_clicked();
26 private:
27     //从 path 目录下加载候选者的图片
28     void loadImages(const QString& path);
29     //定时器事件处理函数
30     void timerEvent(QTimerEvent *);
31     //绘图事件处理函数
32     void paintEvent(QPaintEvent *);
33 private:
34     Ui::LotteryDialog *ui;
35     QVector<QImage> m_vecImages;//保存图片对象的容器
36     int m_index;//记录图片在容器中的索引
37     int m_timerId;//定时器 id
38     bool isStarted;//标记摇奖状态，true:开启摇奖，false:停止摇奖
39 };
40
41 #endif // LOTTERYDIALOG_H
```

lotterydialog.cpp

```
1 #include "lotterydialog.h"
2 #include "ui_lotterydialog.h"
3 //注：到项目模式，去掉"Shaow build"选项,将 photos 目录拷贝到工程目录下
4
5 LotteryDialog::LotteryDialog(QWidget *parent) :
6     QDialog(parent),
7     ui(new Ui::LotteryDialog)
8 {
9     ui->setupUi(this);
10    isStarted = false;//未开始摇奖
11    qsrand(QTime::currentTime().msec());//设置随机种子
12    m_index = 0;//初始化索引编号
13    loadImages("./photos");//从 photos 目录下加载图片数据到 m_vecImages 容器
```

```
14      qDebug()<<"加载到图片的个数:"<< m_vecImages.size();
15 }
16 LotteryDialog::~LotteryDialog()
17 {
18      delete ui;
19 }
20 //开始按钮对应的槽函数
21 void LotteryDialog::on_pushButton_clicked()
22 {
23      if(isStarted == false){
24          isStarted = true;//开始摇奖
25          ui->pushButton->setText("停止");//修改按钮文本：停止
26          m_timerId = startTimer(50);//开启定时器，每隔 50 毫秒切换一个图片
27      }
28      else{
29          isStarted = false;//停止摇奖
30          ui->pushButton->setText("开始");//修改按钮文本：开始
31          killTimer(m_timerId);//关闭定时器
32      }
33 }
34 //从 path 目录下加载候选者的图片
35 void LotteryDialog::loadImages(const QString & path)
36 {
37      QDir dir(path);//创建目录对象
38      //加载当前目录下的所有图片,"Files"指定只访问普通文件不包含子目录
39      QStringList list1 = dir.entryList(QDir::Files);
40
41      for(int i = 0;i<list1.size();i++){
42          //根据文件名创建图片对象，at(i)获取容器中第 i 个文件名
43          QImage image(path+"/"+list1.at(i));
44          //将图片对象添加到容器中
45          m_vecImages << image;
46      }
47          //递归加载子目录的所有图片,参数致命只访问子目录但不包括".",".."
48      QStringList list2 = dir.entryList(QDir::Dirs|QDir::NoDotAndDotDot);
49      for(int i = 0;i<list2.size();i++){
50          //list2.at(i);获取第 i 个子目录的文件名
51          loadImages(path+"/"+list2.at(i));
52
53      }
54 }
55 //定时器事件处理函数
56 void LotteryDialog::timerEvent(QTimerEvent *)
57 {
```

```
58      //随机获取一个候选者图片在容器中的索引
59      m_index = qrand()% m_vecImages.size();
60      //触发绘图事件
61      update();
62 }
63 //绘图事件处理函数
64 void LotteryDialog::paintEvent(QPaintEvent *)
65 {
66      QPainter painter(this);//创建画家对象
67      QRect rect = ui->frame->frameRect();//绘制矩形区域
68      rect.translate(ui->frame->pos());//坐标平移，让 rect 和父窗口坐标系一致
69      painter.drawImage(rect,m_vecImages[m_index]);//绘制和索引对应的图片
70 }
```

main.cpp

```
1 #include "lotterydialog.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6      QApplication a(argc, argv);
7      LotteryDialog w;
8      w.show();
9
10     return a.exec();
11 }
```

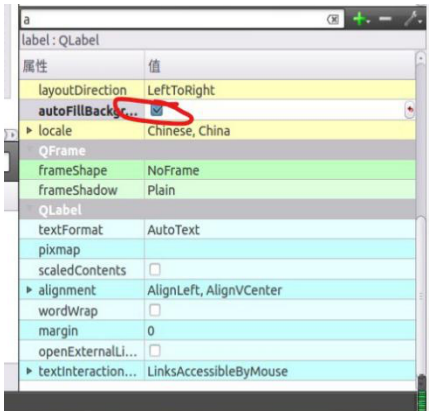
DAY04

十二、鼠标事件

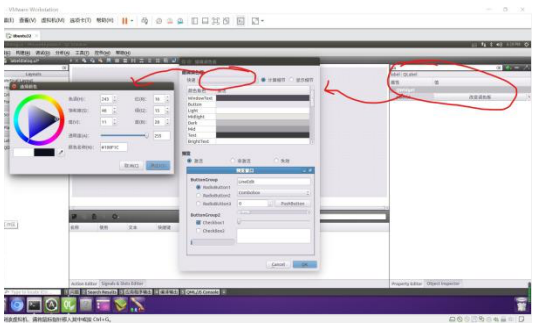
1) 鼠标事件(QMouseEvent)
   mousePressEvent()?mouseReleaseEvent()/mouseDoubleClickEvent()/moseMoveEvent()
2) 键盘事件(QKeyEvent)
   keyPressEvent()/keyReleaseEvent()

设置控件的背景颜色颜色：
方法一：使用调色板(palette)

------>autoFillBackground

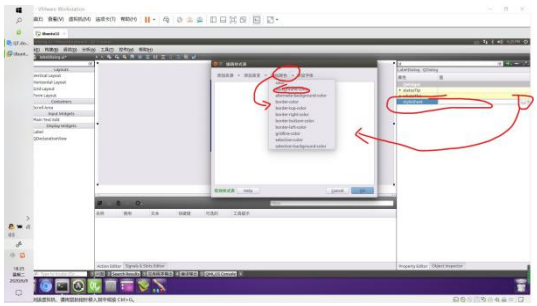


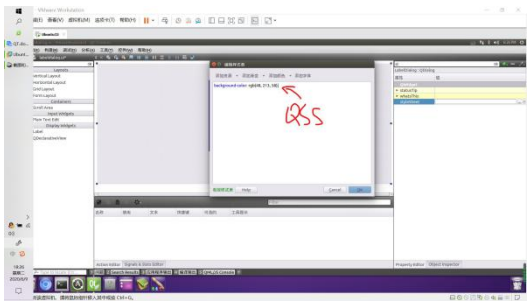
------>palette



方法二：使用样式表(styleSheet)//QSS

------>styleSheet------->backgroundcolor





Labeldialog.cpp

```
1 #include "labeldialog.h"
2 #include "ui_labeldialog.h"
3
4 LabelDialog::LabelDialog(QWidget *parent) :
5         QDialog(parent),
```

```
 6      ui(new Ui::LabelDialog)
 7 {
 8      ui->setupUi(this);
 9      m_drag = false;
10 }
11
12 LabelDialog::~LabelDialog()
13 {
14      delete ui;
15 }
16 //鼠标按下时执行的事件处理函数
17 void LabelDialog::mousePressEvent(QMouseEvent *event)
18 {
19      //qDebug("鼠标按下");
20      //是否为鼠标左键
21      if(event->button() == Qt::LeftButton){
22          //获取 label 所在矩形区域
23          QRect rect = ui->label->frameRect();
24          //平移 rect 坐标值,让其和窗口坐标系相同
25          rect.translate(ui->label->pos());
26          //contains():判断参数位置(鼠标位置)是否在 rect 矩形区域中
27          if(rect.contains(event->pos())==true){
28              m_drag = true;
29              //相对位置  = label 位置-鼠标位置;
30              m_pos = ui->label->pos() - event->pos();
31          }
32      }
33 }
34 //鼠标抬起时执行的事件处理函数
35 void LabelDialog::mouseReleaseEvent(QMouseEvent *event)
36 {
37      //qDebug("鼠标抬起");
38      if(event->button() == Qt::LeftButton){
39          m_drag = false;
40      }
41 }
42 //鼠标移动时执行的事件处理函数
43 void LabelDialog::mouseMoveEvent(QMouseEvent *event)
44 {
45      //qDebug("鼠标移动");
46      if(m_drag == true){
47          //计算 label 要拖拽移动的新位置
48          QPoint newPos = event->pos() + m_pos;
49
```

```
50              //设置 label 移动时不能超出父窗口,移动最大范围:
51              //x:0~窗口宽-label 宽
52              //y:0~窗口高-label 高
53              QSize s1 = size();//获取窗口大小
54              QSize s2 = ui->label->size();//获取 label 大小
55              if(newPos.x() < 0){
56                  newPos.setX(0);
57              }
58              else if(newPos.x() > s1.width()-s2.width()){
59                  newPos.setX(s1.width()-s2.width());
60              }
61              if(newPos.y() < 0){
62                  newPos.setY(0);
63              }
64              else if(newPos.y() > s1.height()-s2.height()){
65                  newPos.setY(s1.height()-s2.height());
66              }
67
68              //移动 label 到新位置
69              ui->label->move(newPos);
70          }
71 }
72 //键盘按键按下时执行的事件处理函数
73 void LabelDialog::keyPressEvent(QKeyEvent *event)
74 {
75          //qDebug("键盘按键按下");
76          //获取 label 当前位置
77          int x = ui->label->pos().x();
78          int y = ui->label->pos().y();
79          if(event->key() == Qt::Key_Up){//方向键上↑
80              ui->label->move(x,y-10);
81          }
82          else if(event->key() == Qt::Key_Down){//方向键下↓
83              ui->label->move(x,y+10);
84          }
85          else if(event->key() == Qt::Key_Left){//方向键左←
86              ui->label->move(x-10,y);
87          }
88          else if(event->key() == Qt::Key_Right){//方向键右→
89              ui->label->move(x+10,y);
90              while(1);
91          }
92 }
```

labeldialog.h

```
1 #ifndef LABELDIALOG_H
2 #define LABELDIALOG_H
3
4 #include <QDialog>
5 #include <QMouseEvent>
6 #include <QKeyEvent>
7
8 namespace Ui {
9 class LabelDialog;
10 }
11 //鼠标拖拽 label 移动,键盘方向键控制 label 移动
12 class LabelDialog : public QDialog
13 {
14     Q_OBJECT
15
16 public:
17     explicit LabelDialog(QWidget *parent = 0);
18     ~LabelDialog();
19 private:
20     //鼠标按下时执行的事件处理函数
21     void mousePressEvent(QMouseEvent *event);
22     //鼠标抬起时执行的事件处理函数
23     void mouseReleaseEvent(QMouseEvent *event);
24     //鼠标移动时执行的事件处理函数
25     void mouseMoveEvent(QMouseEvent *event);
26     //键盘按键按下时执行的事件处理函数
27     void keyPressEvent(QKeyEvent *event);
28 private:
29     Ui::LabelDialog *ui;
30     bool m_drag;//是否允许鼠标拖拽,当鼠标左键选中 label 时才能拖拽
31     QPoint m_pos;//记录鼠标拖拽和 label 的相对位置
32 };
33
34 #endif // LABELDIALOG_H
```

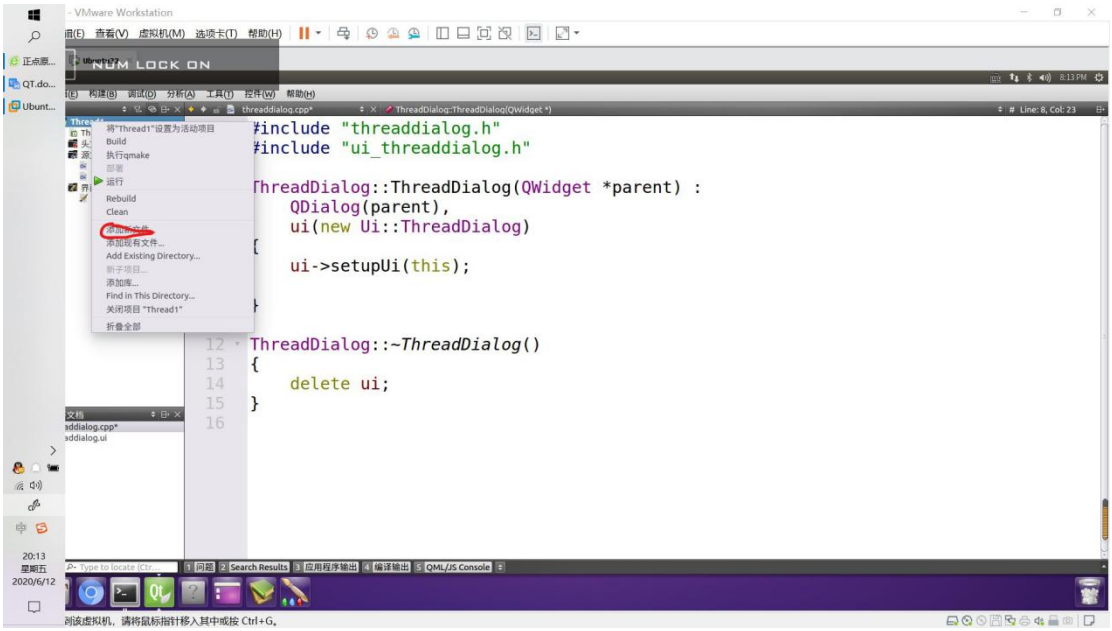
main.cpp

```
1 #include "labeldialog.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     LabelDialog w;
```

```
 8         w.show();
 9
10         return a.exec();
11 }
```

十三、Qt 多线程

添加子线程



方法 1：继承 QThread,重写线程入口函数 run

thread1

threaddialog.h

```
 1 #ifndef THREADDIALOG_H
 2 #define THREADDIALOG_H
 3
 4 #include <QDialog>
 5 #include "workerthread.h"
 6
 7 namespace Ui {
 8 class ThreadDialog;
 9 }
10
11 class ThreadDialog : public QDialog
12 {
13         Q_OBJECT
14
15 public:
16         explicit ThreadDialog(QWidget *parent = 0);
```

```
17          ~ThreadDialog();
18
19 private:
20          Ui::ThreadDialog *ui;
21          WorkerThread thread;//子线程
22 };
23
24 #endif // THREADDIALOG_H
```

threaddialog.cpp

```
 1 #include "threaddialog.h"
 2 #include "ui_threaddialog.h"
 3
 4 ThreadDialog::ThreadDialog(QWidget *parent) :
 5          QDialog(parent),
 6          ui(new Ui::ThreadDialog)
 7 {
 8          ui->setupUi(this);
 9          qDebug() << "主线程:" << QThread::currentThreadId();
10          //点击按钮连接开启子线程槽函数
11          connect(ui->pushButton,SIGNAL(clicked()),&thread,SLOT(start()));
12 }
13
14 ThreadDialog::~ThreadDialog()
15 {
16          delete ui;
17 }
```

workerdialog.h

```
 1 #ifndef WORKERTHREAD_H
 2 #define WORKERTHREAD_H
 3
 4 #include <QThread>
 5 #include <QDebug>
 6
 7 //继承 QThread
 8 class WorkerThread:public QThread
 9 {
10 public:
11          WorkerThread();
12          ~WorkerThread();
13 protected:
14          //重写的线程入口函数
15          void run(void);
```

```
16 };
17
18 #endif // WORKERTHREAD_H
```

workerdialog.cpp

```
 1 #include "workerthread.h"
 2
 3 WorkerThread::WorkerThread()
 4 {
 5
 6 }
 7
 8 WorkerThread::~WorkerThread()
 9 {
10
11 }
12
13 //重写的线程入口函数
14 void WorkerThread::run(void)
15 {
16         while(1){
17                 qDebug() << "子线程:" << currentThreadId();
18                 sleep(1);
19         }
20 }
```

main.cpp

```
 1 #include "threaddialog.h"
 2 #include <QApplication>
 3
 4 int main(int argc, char *argv[])
 5 {
 6         QApplication a(argc, argv);
 7         ThreadDialog w;
 8         w.show();
 9
10         return a.exec();
11 }
```

方法 2：QObject::moveToThread

thread2

threaddialog.h

```
1 #ifndef THREADDIALOG_H
2 #define THREADDIALOG_H
3
4 #include <QDialog>
5 #include "worker.h"
6
7 namespace Ui {
8 class ThreadDialog;
9 }
10
11 class ThreadDialog : public QDialog
12 {
13     Q_OBJECT
14
15 public:
16     explicit ThreadDialog(QWidget *parent = 0);
17     ~ThreadDialog();
18
19 private:
20     Ui::ThreadDialog *ui;
21     QThread thread;//子线程
22     Worker work;//需要放到子线程中工作的对象
23 };
24
25 #endif // THREADDIALOG_H
```

threaddialog.cpp

```
1 #include "threaddialog.h"
2 #include "ui_threaddialog.h"
3
4 ThreadDialog::ThreadDialog(QWidget *parent) :
5     QDialog(parent),
6     ui(new Ui::ThreadDialog)
7 {
8     ui->setupUi(this);
9     qDebug() << "主线程的 ID:" << QThread::currentThreadId();
10    //将工作对象移动到子线程中
11    work.moveToThread(&thread);
12    //点击按钮发送 clicked 信号,连接到工作对象的槽函数
13    connect(ui->pushButton,SIGNAL(clicked()),&work,SLOT(doWork()));
14    //开启子线程,将来通过信号触发 dowork 将会在子线程中被执行
15    thread.start();
16 }
17
```

```
18 ThreadDialog::~ThreadDialog()
19 {
20         delete ui;
21 }
```

worker.h

```
 1 #ifndef WORKER_H
 2 #define WORKER_H
 3
 4 #include <QObject>
 5 #include <QThread>
 6 #include <QDebug>
 7
 8 class Worker : public QObject
 9 {
10         Q_OBJECT
11 public:
12         explicit Worker(QObject *parent = 0);
13         ~Worker();
14
15 signals:
16 public slots:
17         //自定义槽函数:将来要放到子线程中执行
18         void doWork(void);
19 };
20
21 #endif // WORKER_H
```

worker.cpp

```
 1 #include "worker.h"
 2
 3 Worker::Worker(QObject *parent) : QObject(parent)
 4 {
 5
 6 }
 7
 8 Worker::~Worker()
 9 {
10
11 }
12
13 //自定义槽函数:将来要放到子线程中执行
14 void Worker::doWork(void)
15 {
```

```
16          while(1){
17                  qDebug() << "子线程 ID:" << QThread::currentThreadId();
18                  QThread::sleep(1);
19          }
20 }
```

main.cpp

```
 1 #include "threaddialog.h"
 2 #include <QApplication>
 3
 4 int main(int argc, char *argv[])
 5 {
 6          QApplication a(argc, argv);
 7          ThreadDialog w;
 8          w.show();
 9
10          return a.exec();
11 }
```

十四、Qt 线程同步控制

1）互斥锁 QMutex
2）信号量 QSemaphore
3）读写锁 QReadwriteLock
4）等待条件 QWaitCondition
Semaphore（信号量）

shell:qmake -project
shell:qtcreator Semaphore.pro

main.cpp

```
 1 #include <QCoreApplication>
 2 #include <QThread>
 3 #include <QSemaphore>
 4
 5 const int DataSize = 20;//要生产的产品个数
 6 const int BuffSize = 5;//仓库大小
 7 int buffer[BuffSize];//仓库
 8
 9 //控制生产者信号量
10 QSemaphore freeSpace(BuffSize);
11 //控制消费者信号量
12 QSemaphore usedSpace(0);
13
```

```
14 //生产者线程
15 class threadProducer:public QThread{
16         void run(void){
17                 for(int i=0;i<DataSize;i++){
18                         freeSpace.acquire();//-1,获取空闲位置成功继续生产,否则等待
19                         buffer[i%BuffSize] = i+1;
20                         qDebug("生产产品:%d",buffer[i%BuffSize]);
21                         msleep(100);
22                         usedSpace.release();//+1,多了一个可以消费的产品
23                 }
24         }
25 };
26 //消费者线程
27 class threadConsumer:public QThread{
28         void run(void){
29                 for(int i=0;i<DataSize;i++){
30                         usedSpace.acquire();//-1,获取一个可以消费的产品,没有产品则等待
31                         qDebug("消费产品:%d",buffer[i%BuffSize]);
32                         msleep(400);
33                         freeSpace.release();//+1,多一个可以生产的空闲位置
34                 }
35         }
36 };
37
38 int main(int argc,char** argv){
39         QCoreApplication app(argc,argv);//没有界面的应用程序
40         threadProducer producer;//生产者
41         threadConsumer consumer;//消费者
42         producer.start();//开始生产
43         consumer.start();//开销消费
44
45         producer.wait();
46         consumer.wait();
47
48         return app.exec();
49 }
```

DAY05

十五、Qt 数据库

shell:sqlite 3

Sqlite 的操作指令

- 进入 SQLite 的命令行界面后，可以输入两种指令，一种是自身配置和格式控制相关指令，这些指令都以**“.”**开头；另外一种指令是 SQL 语言(Sql 语句)，实现对数据库的增删改查等操作，这些指令以**“;”**结束
- 输入.help 或 .h 可以获取以“.”开头指令的帮助信息
- 输入.exit 或.quit 退出 SQLite 的命令界面，回到系统的控制终端
- 如果需要清屏，可以使用“ctrl+L”

以“.”开头的常用命令 (不区分大小写)

- .database //查看数据库的名字和对应的文件名

- .table //查看数据表的名字

- .schema //查看数据表创建时的详细信息

- .mode //设置显示模式，如 tab/list/column/html/..-------->.help 查看

- .nullvalue //设置空白字段显示的字符串-------->.nullvalue “字符串”

- .header on //显示数据表的表头

--select * from company；//查看 company 数据表的数据------>没有电点

注：可以将格式显示配置命令写入配置文件(.sqliterc)，下次启动时会自动从该配置文件中加载配置。------>在/home/tarena 目录中的隐藏文件

vi /home/tarena/.sqliterc

```
1 .mode tab
2 .nullvalue "NULL"
3 .header on
```

创建数据表

- 语法格式

CREATE TABLE 表名(列名 1 类型 [约束], 列名 2 类型 [约束],...);

- 常用的类型

INT(整型)、TEXT(字符串)、REAL(浮点数)

- 常用的约束：

PRIMARY KEY(主键):表示该列数据唯一,可以加快数据访问.

NOT NULL(非空):该列数据不能为空

//创建数据列表

```
sqlite> create table student (
   ...> id INT PRIMARY KEY NOT NULL,
   ...> name TEXT NOT NULL,
   ...> score REAL NOT NULL );
```

删除数据表

- 语法格式

DROP TABLE 表名;

注：慎用，数据表一旦删除里面数据也将随之消失.

向数据表中插入数据

- 语法格式

INSERT INTO 表名 (列名 1,列名 2,...) VALUES(数值 1,数值 2,...);

INSERT INTO 表名 VALUES(数值 1,数值 2,...);

注：如果要为表中的所有列添加值，并且插入列的顺序和创建表的顺序相同，

可以不需要在 Sqlite 查询中指定列名

- 插入数据实例

INSERT INTO company (id,name,age,salary) VALUES(10021,'小乔',24,11000); INSERT INTO company VALUES(10022,'大乔',29,'江西',13000);字符串单引号双引号都可以，不能有中文

从数据表删除数据

- 语法格式

DELETE FROM 表名 WHERE 条件表达式;

注：可以没有 WHERE 子句，但数据表的所有数据都将被删除

注：如果有多个条件可以使用逻辑与(and)或逻辑或(or)连接

- 删除数据实例

DELETE FROM company WHERE id=10029;

DELETE FROM company WHERE address='成都' and salary<1000;

修改数据表中的数据

- 语法格式

UPDATE 表名 SET 列名 1=新数值,列名 2=新数值,... WHERE 条件表达式;

注：可以没有 WHERE 子句，但数据表的所有数据都将被修改

注：“新数值”可以是一个常数，也可以是一个表达式

- 修改数据实例

UPDATE company SET age = 45 WHERE id = 10011;

UPDATE company SET salary=salary+2000 WHERE age>=30 and age<=35;

查询数据表中的数据（面试题）

- 语法格式

SELECT 列名 1,列名 2,... FROM 表名;
SELECT 列名 1,列名 2,... FROM 表名 WHERE 条件表达式;
SELECT 列名 1,列名 2,... FROM 表名 ORDER BY 列名 排序方式; //asc 表示升序排列，desc 表示降序注：
ORDER BY 子句可以和 WHERE 子句配合使用，也可以独立使用
_ 查询操作实例
SELECT * FROM company WHERE salary>10000 or salary<3500;
SELECT * FROM company ORDER BY id ASC;

模糊查询

_ 语法格式
SELECT 列名 1,列名 2,... FROM 表名 WHERE 列名 LIKE 模糊匹配条件;
_ 模糊匹配通配符
百分号 （%）：代表零个、一个或多个数字或字符
下划线 （_）：代表一个单一的数字或字符
注：它们可以被组合使用
_ 模糊查询实例
SELECT * FROM company WHERE salary LIKE '%8%';
SELECT * FROM company WHERE name LIKE '关_';
SELECT * FROM company WHERE salary WHERE SALARY LIKE ' _2%3 ';

在 Qt 中使用 Sqlite 数据库

• 建立 Qt 应用程序和数据库连接(QSqlDatabase)
– 添加数据库驱动
db = QSqlDatabase::addDatabase("QSQLITE");
– 设置数据库名字
db.setDatabaseName("menu.db");
– 打开数据库
db.open();
注：使用 Qt 数据库模块需要在工程文件中添加 “QT += sql”

在 Qt 中使用 Sqlite 数据库

• 执行数据库操作的 SQL 语句（QSqlQuery）
QSqlQuery query;
QString str(“SELECT、DELETE、INSERT、UPDATE 等 SQL 语句”);
query.exec(str);
• 获取查询结果集（QSqlQueryModel）
QString str = QString (“SELECT * FROM 表名");
QSqlQueryModel *model = new QSqlQueryModel;
model->setQuery(str);//执行查询操作，并将结果集保存到 model 对象中
ui->tableView->setModel(model);//显示查询结果集到 QTabelView 控件

vi studentdialog.h

```
 1 #ifndef STUDENTDIALOG_H
 2 #define STUDENTDIALOG_H
 3
 4 #include <QDialog>
 5 //工程文件需要添加:QT += sql
 6 #include <QSqlDatabase>
 7 #include <QSqlQuery>
 8 #include <QSqlQueryModel>
 9 #include <QSqlError>
10 #include <QDebug>
11
12 namespace Ui {
13 class StudentDialog;
14 }
15
16 class StudentDialog : public QDialog
17 {
18         Q_OBJECT
19
20 public:
21         explicit StudentDialog(QWidget *parent = 0);
22         ~StudentDialog();
23 private:
24         //创建或连接 sqlite 数据库
25         void createDB();
26         //创建学生成绩数据表
27         void createTable();
28         //查询和显示数据表的内容
29         void queryTable();
30 private slots:
31         //插入
32         void on_insertButton_clicked();
33         //删除
34         void on_deleteButton_clicked();
35         //修改
36         void on_updateButton_clicked();
37         //排序
38         void on_sortButton_clicked();
39
40 private:
41         Ui::StudentDialog *ui;
42         QSqlDatabase db;//建立应用程序和数据库连接
43         QSqlQueryModel model;//保存查询的结果集
```

```
44 };
45
46
47 #endif // STUDENTDIALOG_H
```

vi studentdialog.cpp

```
 1 #include "studentdialog.h"
 2 #include "ui_studentdialog.h"
 3
 4 StudentDialog::StudentDialog(QWidget *parent) :
 5     QDialog(parent),
 6     ui(new Ui::StudentDialog)
 7 {
 8     ui->setupUi(this);
 9     createDB();
10     createTable();
11     queryTable();
12 }
13
14 StudentDialog::~StudentDialog()
15 {
16     delete ui;
17 }
18
19 //创建或连接 sqlite 数据库
20 void StudentDialog::createDB()
21 {
22     //添加 sqlite 数据库驱动
23     db = QSqlDatabase::addDatabase("QSQLITE");
24     //设置数据库名字,如果"student.db"不存在会自动创建
25     db.setDatabaseName("student.db");
26     //打开数据库
27     if(db.open() == false){
28         qDebug() << db.lastError().text();
29     }
30     else{
31         qDebug() << "创建或连接数据库成功!";
32     }
33 }
34 //创建学生成绩数据表
35 void StudentDialog::createTable()
36 {
37     //准备创建数据表 sql 语句的字符串
38     QString str = QString("CREATE TABLE student ("
```

```
39                                      "id INT PRIMARY KEY NOT NULL,"
40                                      "name TEXT NOT NULL,"
41                                      "score REAL NOT NULL)");
42      //执行 sql 语句
43      QSqlQuery query;
44      if(query.exec(str) == false){
45          qDebug() << str;
46      }
47      else{
48          qDebug() << "创建数据表成功!";
49      }
50 }
51 //查询和显示数据表的内容
52 void StudentDialog::queryTable()
53 {
54      //准备查询的 sql 语句字符串
55      QString str = QString("SELECT * FROM student");
56      //执行查询语句并保存结果集到 model 对象
57      model.setQuery(str);
58      //将结果集显示到界面
59      ui->tableView->setModel(&model);
60 }
61 //插入
62 void StudentDialog::on_insertButton_clicked()
63 {
64      //准备插入操作的 sql 语句字符串
65      int id = ui->idEdit->text().toInt();//学生学号
66      QString name = ui->nameEdit->text();//学生姓名
67      double score = ui->scoreEdit->text().toDouble();//学生成绩
68      QString str = QString("INSERT INTO student VALUES(%1,'%2',%3)"
69                                      ).arg(id).arg(name).arg(score);
70      //执行 sql 语句
71      QSqlQuery query;
72      if(query.exec(str) == false){
73          qDebug() << str;
74      }
75      else{
76          qDebug() << "插入数据成功!";
77          queryTable();
78      }
79 }
80 //删除:根据 ID 删除一条数据
81 void StudentDialog::on_deleteButton_clicked()
82 {
```

```
83      //准备删除操作的 sql 语句字符串
84      int id = ui->idEdit->text().toInt();//学生学号
85      QString str = QString("DELETE FROM student WHERE id=%1").arg(id);
86      //执行 sql 语句
87      QSqlQuery query;
88      if(query.exec(str) == false){
89          qDebug() << str;
90      }
91      else{
92          qDebug() << "删除数据成功!";
93          queryTable();
94      }
95 }
96 //修改:根据 ID 修改成绩
97 void StudentDialog::on_updateButton_clicked()
98 {
99      //准备修改操作的 sql 语句字符串
100     int id = ui->idEdit->text().toInt();//学生学号
101     double score = ui->scoreEdit->text().toDouble();
102     QString str = QString("UPDATE student SET score=%1 WHERE id=%2"
103                           ).arg(score).arg(id);
104     //执行 sql 语句
105     QSqlQuery query;
106     if(query.exec(str) == false){
107         qDebug() << str;
108     }
109     else{
110         qDebug() << "修改数据成功!";
111         queryTable();
112     }
113 }
114 //排序
115 void StudentDialog::on_sortButton_clicked()
116 {
117     //准备排序查询的 sql 语句字符串
118     QString value = ui->valueComboBox->currentText();//获取排序列名
119     QString condition;//排序方式:升序(0)/降序(1)
120     if(ui->condComboBox->currentIndex() == 0){
121         condition = "ASC";
122     }
123     else{
124         condition = "DESC";
125     }
126     QString str = QString("SELECT * FROM student ORDER BY %1 %2"
```

```
127                                     ).arg(value).arg(condition);
128         //执行查询语句并保存结果集到 model 对象
129         model.setQuery(str);
130         //将结果集显示到界面
131         ui->tableView->setModel(&model);
132 }
```

十六、Qt 网络编程(重点)

网络协议模型

• OSI(Open System Interconnection)的七层协议
– 应用层，为应用程序提供服务，处理业务逻辑，如 http、ftp、pop3、smtp
– 表示层，处理在两个通信系统中交换信息的方式，如 ASCII、JPEG、MJPG
– 会话层，建立、管理、终止会话，如 SMB
– 传输层，向用户提供可靠的端到端服务，如 TCP/UDP
– 网络层，为网络不同的主机提供通信，从中选择最适当的路径，如：IP 协议
– 数据链路层，实现物理寻址，建立通信链路，如 MAC
– 物理层，为数据端设备提供传送的物理媒体，对网卡的硬件驱动

TCP/IP 协议族

• TCP/IP 是当今最流行的组网形式，TCP/IP 不是专指某一个协议，是一组协议的代名词，共同组成了 TCP/IP 协议簇，主要包含的协议
– HTTP：超文本传输协议，广泛用于 Web 通信
– TCP：传输控制协议，面向连接，可靠的全双工字节流
– UDP：用户数据报协议，无连接，不可靠的全双工消息包
– ICMP：互联网控制消息协议
– IPv4/IPv6：IP 协议
– ARP：地址解析协议 ，根据 IP 地址获取物理地址(MAC)
– ...

IP 地址

• IP 地址是计算机在互联网中的唯一标识，具体可分为 IPv4 协议地址和 IPv6 协议地址：
– IPv4：32 位整数，如“0xC0A80F64”
– IPv6：128 位整数，如“fe80:0000:0000:0000:dad9:b217:f207:bc35”
• Qt 中使用 QHostAddress 对象来表示一个 IP 地址，包括 IPv4 和 IPv6，可以通过 toString 成员函数，将其转换为点分字符串的形式，也就是平时看到的地址，例如 IPv4 地址“0xC0A80F64”，对应点分字符串为“192.168.15.100”
• QHostAddress 的构造函数
– QHostAddress(quint32 ip4Addr)
– QHostAddress(const QString & address)

– ...

• 查询主机 IP 地址命令

– linux 中使用“ifconfig”

– windows 中“ipconfig”

• 特殊的 IP 地址

– QHostAddress::LocalHost

注：对应“127.0.0.1”表示本地环回地址，常用于本地测试

– QHostAddress::Any

注：对应“0.0.0.0”表示任意 IP 地址，常用于服务器

– QHostAddress::Broadcast

注：对应“255.255.255.255”表示广播地址，用于局域网广播消息

• 通过 IP 地址判断两台主机是否可以通信

– ping 对方 IP 地址/域名

注：用于确定本地主机是否能与对方主机成功交换(发送与接收)数据包，再根据返回的信息，就可以推断通信响应速度、网络是否通畅

– 正常连通时可以类似如下信息：

```
~$ ping 192.168.3.54
PING 192.168.3.54 (192.168.3.54) 56(84) bytes of data.
64 bytes from 192.168.3.54: icmp_seq=1 ttl=64 time=0.047 ms
64 bytes from 192.168.3.54: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 192.168.3.54: icmp_seq=3 ttl=64 time=0.092 ms
64 bytes from 192.168.3.54: icmp_seq=4 ttl=64 time=0.029 ms
```

UDP 通信

• UDP 协议简介

– UDP （User Data Protocol，UDP）即用户数据报协议，是 OSI 参考模型中一种无连接的传输层协议，它是一种简单轻量级、不可靠、面向数据报、无连接的传协议，可以应用在可靠性要求不苛刻的场景

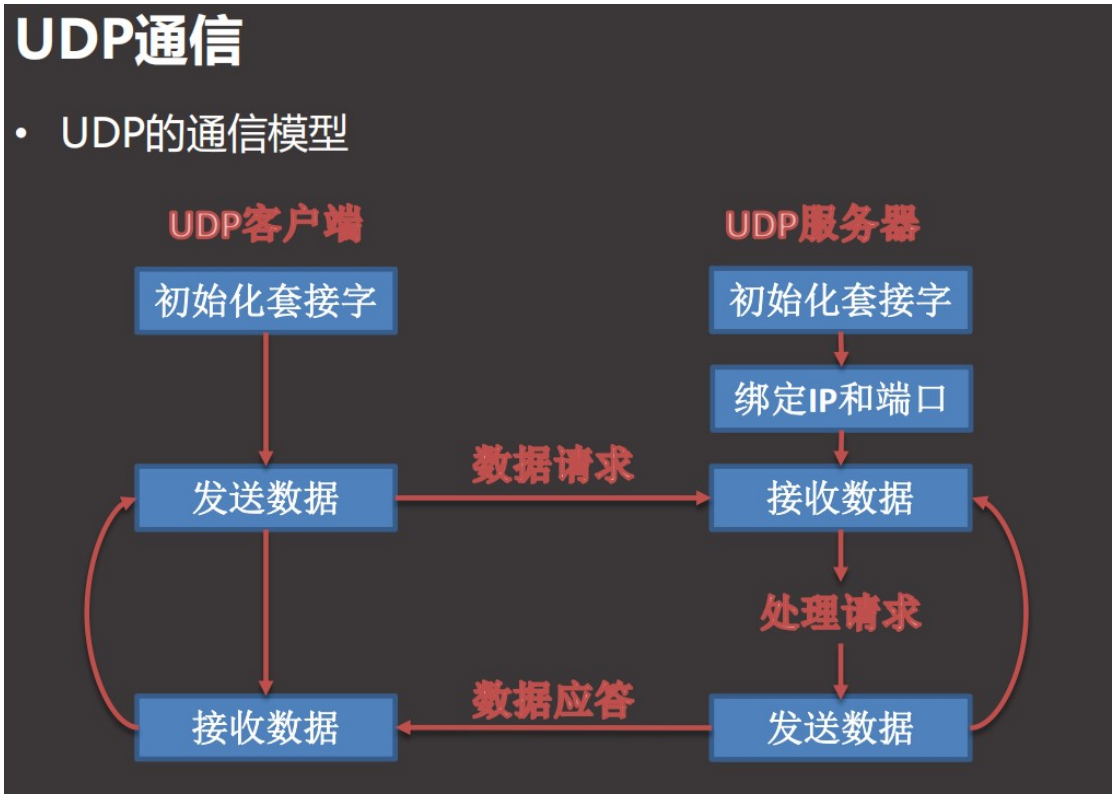
• 适合使用 UDP 通信的情况

– 通信的交互数据大多为短消息

– 拥有大量的客户端

– 对数据安全性无特殊要求

– 网络负担重，但对响应实时性要求较高



UDP 通信

• QUdpSocket 类

– 该类封装了 UDP 套接字，可以非常方便的建立 UDP 通信的连接，通过 bind()函数实现绑定一个 IP 和端口，然后调用 writeDatagram()函数实现发送消息，以及用 readDatagram()实现从 UDP 套接字读取消息

– 如果想使用 QIODevice 类继承的 read()/write()等标准的数据读写函数，必须首先调用 connectToHost()，建立套接字的连接

– 当 QUdpSocket 套接字有数据到来时，将会发送 readyRead() 信号，所以不必单独开启线程来处理连接，只需要在收到该信号时，再去读取数据报

– 当信号到来以后， hasPendingDatagrams()函数将返回 true，这时可以调用 pendingDatagramSize()获得第一个的数据报的大小，然后再通过 readDatagram()函数读取它

练习 UDP 通信

• UDP 网络广播通信

– 广播发送端(Sender)

1）指定广播地址：255.255.255.255

2）指定广播端口：8888

3）输入广播消息，每隔 1 秒发送一次广播(定时器)

– 广播接收端(Receiver)

1）绑定接收广播端口：8888

2）实时接收广播消息并显示

cd Sender

vi senderdialog.h

```
 1 #ifndef SENDERDIALOG_H
 2 #define SENDERDIALOG_H
 3
 4 #include <QDialog>
 5 //工程文件添加: QT += network
 6 #include <QUdpSocket>
 7 #include <QHostAddress>
 8 #include <QTimer>//定时器
 9 namespace Ui {
10 class SenderDialog;
11 }
12
13 class SenderDialog : public QDialog
14 {
15     Q_OBJECT
16
17 public:
18     explicit SenderDialog(QWidget *parent = 0);
19     ~SenderDialog();
20
21 private slots:
22     //开始广播按钮对应的槽函数
23     void on_pushButton_clicked();
24     //定时发送广播消息的槽函数
25     void sendMessage();
26 private:
27     Ui::SenderDialog *ui;
28     QUdpSocket* udpSocket;//UDP 通信套接字
29     QTimer* timer;//定时器
30     bool isStarted;//状态标记:true 开始广播,false 停止广播
31 };
32
33 #endif // SENDERDIALOG_H
```

vi senderdialog.cpp

```
 1 #include "senderdialog.h"
 2 #include "ui_senderdialog.h"
 3
 4 SenderDialog::SenderDialog(QWidget *parent) :
 5     QDialog(parent),
 6     ui(new Ui::SenderDialog)
```

```
 7 {
 8     ui->setupUi(this);
 9     udpSocket = new QUdpSocket(this);
10     timer = new QTimer(this);
11     isStarted = false;
12     //定时器到时发送信号 timeout,连接到发送广播消息的槽函数
13     connect(timer,SIGNAL(timeout()),this,SLOT(sendMessage()));
14 }
15 SenderDialog::~SenderDialog()
16 {
17     delete ui;
18 }
19 //开始广播按钮对应的槽函数
20 void SenderDialog::on_pushButton_clicked()
21 {
22     if(isStarted == false){
23         isStarted = true;//开始广播
24         timer->start(1000);//开启定时器,每隔 1 秒广播一次
25         ui->pushButton->setText("停止广播");//修改按钮文本"停止广播"
26         ui->messageEdit->setEnabled(false);//禁用消息输入
27         ui->portEdit->setEnabled(false);//禁用端口输入
28     }
29     else{
30         isStarted = false;//停止广播
31         timer->stop();//停止广播
32         ui->pushButton->setText("开始广播");//修改按钮文本"开始广播"
33         ui->messageEdit->setEnabled(true);//恢复消息输入
34         ui->portEdit->setEnabled(true);//恢复端口输入
35     }
36 }
37 //定时发送广播消息的槽函数
38 void SenderDialog::sendMessage()
39 {
40     //获取消息
41     QString msg = ui->messageEdit->text();
42     if(msg == ""){
43         return;
44     }
45     //获取端口号,quint16->unsigned short
46     quint16 port = ui->portEdit->text().toShort();
47     //使用 udp 套接字向广播地址发送消息
48     //参数:
49     //1)消息包(QByteArray,类似 char[]),toUtf8:将 QString 转换为 QByteArray
50     //2)广播地址(QHostAddress),Broadcast:"255.255.255.255"
```

```
51          //3)广播端口(quint16)
52          udpSocket->writeDatagram(msg.toUtf8(),QHostAddress::Broadcast,port);
```

cd Receiver

vi receiverdialog.h

```
 1 #ifndef RECEIVERDIALOG_H
 2 #define RECEIVERDIALOG_H
 3
 4 #include <QDialog>
 5 #include <QUdpSocket>
 6 #include <QHostAddress>
 7 namespace Ui {
 8 class ReceiverDialog;
 9 }
10
11 class ReceiverDialog : public QDialog
12 {
13          Q_OBJECT
14
15 public:
16          explicit ReceiverDialog(QWidget *parent = 0);
17          ~ReceiverDialog();
18
19 private slots:
20          //开始接收按钮对应的槽函数
21          void on_pushButton_clicked();
22          //接收广播消息的槽函数
23          void receiveMessage();
24 private:
25          Ui::ReceiverDialog *ui;
26          QUdpSocket* udpSocket;//UDP 套接字
27          bool isStarted;//标记接收状态:true 开始接收,false 停止接收
28 };
29
30 #endif // RECEIVERDIALOG_H
```

vi receiverdialog.cpp

```
 1 #include "receiverdialog.h"
 2 #include "ui_receiverdialog.h"
 3
 4 ReceiverDialog::ReceiverDialog(QWidget *parent) :
 5          QDialog(parent),
 6          ui(new Ui::ReceiverDialog)
 7 {
```

```
 8      ui->setupUi(this);
 9      udpSocket = new QUdpSocket(this);
10      isStarted = false;//停止接收状态
11 }
12
13 ReceiverDialog::~ReceiverDialog()
14 {
15      delete ui;
16 }
17 //开始接收按钮对应的槽函数
18 void ReceiverDialog::on_pushButton_clicked()
19 {
20      if(isStarted == false){
21          isStarted = true;//开始接收广播消息
22          //获取接收广播的端口
23          quint16 port = ui->lineEdit->text().toShort();
24          if(udpSocket->bind(QHostAddress::Any,port) == false){
25              qDebug("绑定接收端口失败!");
26              return;
27          }
28          //当套接字收到广播消息时发送信号 readyRead,连接到接收广播消息的槽函数
29          connect(udpSocket,SIGNAL(readyRead()),this,SLOT(receiveMessage()));
30
31          ui->pushButton->setText("停止接收");//修改按钮文本
32          ui->lineEdit->setEnabled(false);//禁用端口输入
33      }
34      else{
35          isStarted = false;//停止接收广播消息
36          udpSocket->close();//关闭通信套接字
37          ui->pushButton->setText("开始接收");//修改按钮文本
38          ui->lineEdit->setEnabled(true);//恢复端口输入
39      }
40 }
41 //接收广播消息的槽函数
42 void ReceiverDialog::receiveMessage()
43 {
44      //hasPendingDatagrams():判断是否等待读取的数据包
45      while(udpSocket->hasPendingDatagrams()){
46          //准备接收数据包的缓冲区
47          QByteArray buf;//类似  char buf[?]
48          //pendingDatagramSize:获取等待读取数据包的大小
49          buf.resize(udpSocket->pendingDatagramSize());
50          //读取数据包,参数:
51          //1)接收数据缓冲区首地址(char*),data():将 QByteArray 转换为 char*
```

```
52              //2)要接收数据包的字节数(qint64)
53              udpSocket->readDatagram(buf.data(),buf.size());
54              //显示广播消息到界面
55              ui->listWidget->addItem(buf);
56              //回滚显示最底部消息(最新消息)
57              ui->listWidget->scrollToBottom();
58          }
59 }
```

DAY06

回顾：

1 常用 SQL 语句

1）创建数据表

CREATE TABLE 表明 （列名 类型 【约束】，...）；

2）删除数据表

DROP TABLE 表名；

3）向数据表插入数据

INSERT INTO 表名（列名,...）VALUES（数值，...）；

4）从数据表删除数据

DELETE FORM 表名 WHERE 条件表达式；

5）修改数据表中数据

UPDATE 表名 SET 列名=新数值 WHERE 条件表达式；

6）查询数据表中数据

SELECT 列名，...FROM 表名 WHERE 条件表达式 ORDER BY 列名 ASC/DESC;

SELECT 列名，...FROM 表名 WHERE LIKE “匹配模式” (‘曹%’);

2 QT 使用数据库

```
QSQLDatabase db = QSQLDatabase::addDatabase(“QSQLITE”);
db.setDatabaseName(“test.db”);
db.open();
QSqlQuery query;
query.exec(“sql 语句”);
QSqlQueryModel model;
model.setQuery(“select 语句”);
```

3 网络编程

1）网络协议模型(OSI 七层)

应用层(HTTP)、表示层、会话层

传输层：UDP,TCP

网络层：IPV4

数据链路层

物理层

2）IP 地址（QHostAddress）

3）UDP 通信（QUdpSocket）

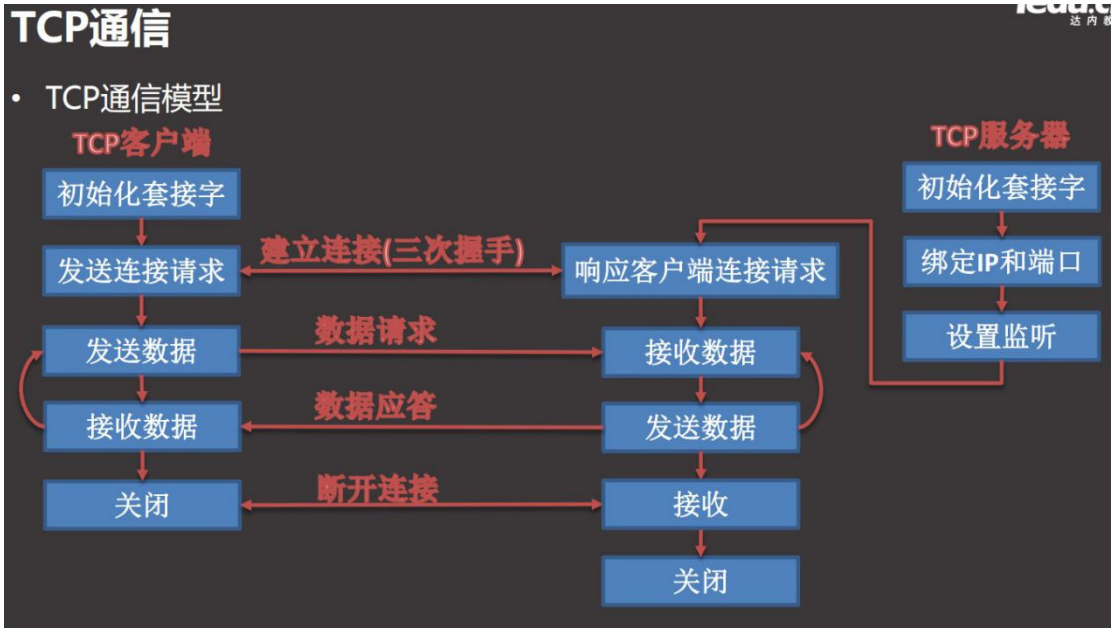
TCP 通信

• TCP 协议简介

_ TCP（ Transmission Control Protocol）传输控制协议）是一个可靠的、基于字节流的，面向连接的传输层协议，在 TCP/IP 协议族中，TCP 位于 IP 协议层之上，应用层之下的中间层

• 适合 TCP 通信的情况

_ 不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是 IP 层不提供这样的流机制，TCP 协议恰恰符合这一要求，特别适合连续传输数据，同时能够确保数据的安全性，广泛使用的 HTTP 协议就是基于 TCP 协议而实现



TCP 通信

• QTcpSocket

– 该类提供了一个基于 TCP 的套接字，它是 QAbstractSocket 的一个子类，通过它可以非常方便建立一个基于 TCP 连接和数据流传输。

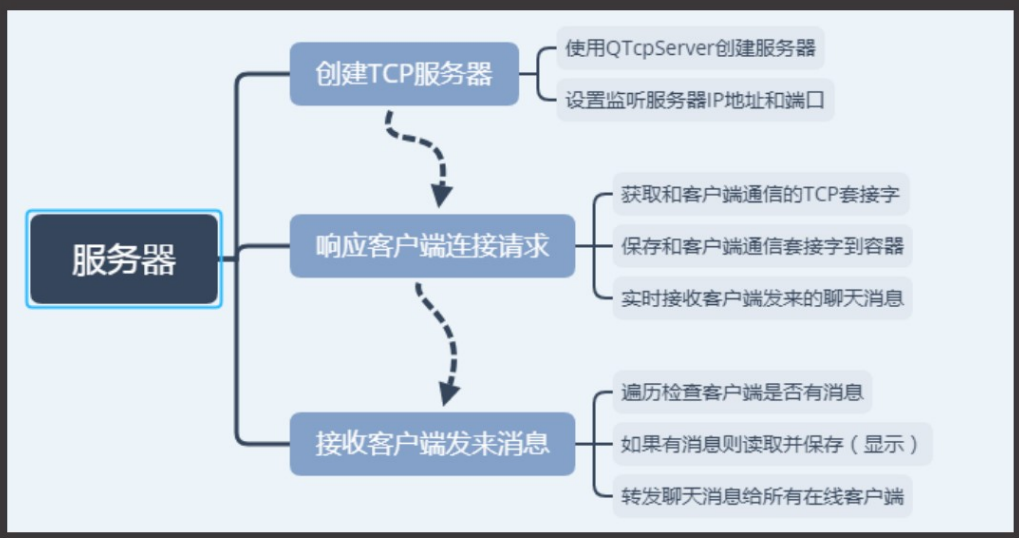
• QTcpServer

– 该类提供了一个基于 TCP 的服务器，通过该类可以快速建立 TCP 服务器，并接受客户端的连接请求

– QTcpServer::listen()函数可以设置监听服务器的 IP 和端口号，每当监听到有客户端和服务器建立连接，该类将会自动发送信号，newConnection()，可以连接自定义的槽函数处理和客户端的连接请求

TCP网络聊天室

- TCP聊天室服务器设计概要



vi serverdialog.h

```
1 #ifndef SERVERDIALOG_H
2 #define SERVERDIALOG_H
3
4 #include <QDialog>
5 //QT += network
6 #include <QTcpServer>
7 #include <QTcpSocket>
8 #include <QDebug>
9 #include <QTimer>
10
11 namespace Ui {
12 class ServerDialog;
13 }
14
15 class ServerDialog : public QDialog
16 {
17         Q_OBJECT
18
19 public:
20         explicit ServerDialog(QWidget *parent = 0);
21         ~ServerDialog();
22
23 private slots:
24         //创建服务器按钮对应的槽函数
25         void on_pushButton_clicked();
```

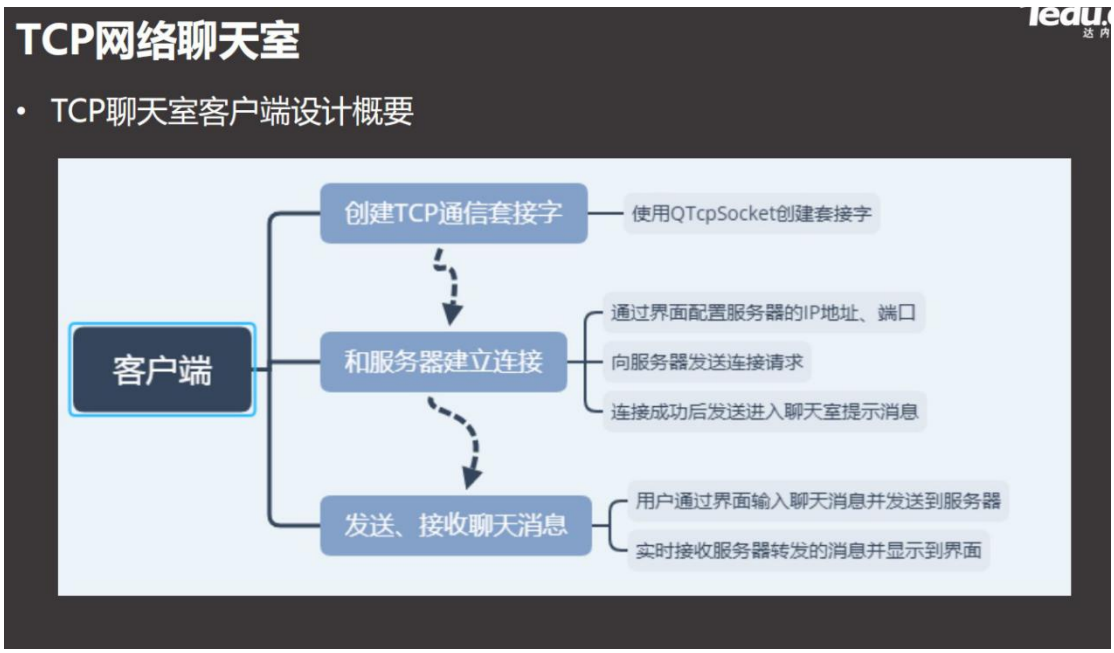
```
26      //响应客户端连接请求的槽函数
27      void onNewConnection();
28      //接收客户端聊天消息的槽函数
29      void onReadyRead();
30      //转发聊天消息给其它客户端
31      void sendMessage(const QByteArray& msg);
32      //定时器到时执行的槽函数:检容器中套接字是否断开连接的容器
33      void onTimeout();
34 private:
35      Ui::ServerDialog *ui;
36      QTcpServer tcpServer;//TCP 服务器
37      quint16 port;//服务器端口
38      QList<QTcpSocket*> tcpClientList;//列表容器:保存和客户端通信的套接字
39      QTimer timer;//定时器:定义检查容器的套接字是否断开连接
40 };
41
42 #endif // SERVERDIALOG_H
```

vi serverdialog.cpp

```
 1 #include "serverdialog.h"
 2 #include "ui_serverdialog.h"
 3
 4 ServerDialog::ServerDialog(QWidget *parent) :
 5      QDialog(parent),
 6      ui(new Ui::ServerDialog)
 7 {
 8      ui->setupUi(this);
 9 }
10
11 ServerDialog::~ServerDialog()
12 {
13      delete ui;
14 }
15 //创建服务器按钮对应的槽函数
16 void ServerDialog::on_pushButton_clicked()
17 {
18      //获取端口
19      port = ui->lineEdit->text().toShort();
20      //设置监听服务器的 IP 和端口
21      if(tcpServer.listen(QHostAddress::Any,port) == false){
22           qDebug("服务器创建失败!");
23           return;
24      }
25      qDebug("服务器创建成功!");
```

```
26      ui->pushButton->setEnabled(false);//设置按钮禁用
27      ui->lineEdit->setEnabled(false);//禁用端口输入
28      //当有客户端和服务器建立连接请求,发送信号 newConnection,连接到响应客户端请
            求的槽函数
29      connect(&tcpServer,SIGNAL(newConnection()),this,SLOT(onNewConnection()));
30      //定时器到时发送信号 timeout,连接到定时检查容器中套接字是否断开连接的容器
31      connect(&timer,SIGNAL(timeout()),SLOT(onTimeout()));
32      //开启定时器,每隔 3 秒检查一次
33      timer.start(3000);
34 }
35 //响应客户端连接请求的槽函数
36 void ServerDialog::onNewConnection()
37 {
38      //nextPendingConnection():获取和客户端通信的套接字
39      QTcpSocket* tcpClient = tcpServer.nextPendingConnection();
40      //保存和客户端通信套接字容器
41      tcpClientList.append(tcpClient);
42      //当和客户端通信的套接字给服务器发送消息,发送信号 readyRead,连接到接收聊天
            消息的槽函数
43      connect(tcpClient,SIGNAL(readyRead()),this,SLOT(onReadyRead()));
44 }
45 //接收客户端聊天消息的槽函数
46 void ServerDialog::onReadyRead()
47 {
48      //遍历检查容器中哪个客户端有消息到来
49      for(int i=0;i<tcpClientList.size();i++){
50          //bytesAvailable:获取第 i 个套接字可以读取的消息字节数
51          //如果返回 0,表示该套接字没有消息,如果返回大于 0,表示有消息到来
52          if(tcpClientList.at(i)->bytesAvailable()){
53              //读取消息
54              QByteArray buf = tcpClientList.at(i)->readAll();
55              //显示消息到界面
56              ui->listWidget->addItem(buf);
57              ui->listWidget->scrollToBottom();
58              //转发消息给其它客户端
59              sendMessage(buf);
60          }
61      }
62 }
63 //转发聊天消息给其它客户端
64 void ServerDialog::sendMessage(const QByteArray& msg)
65 {
66      //将参数(聊天消息)转发所有的客户端
67      for(int i=0;i<tcpClientList.size();i++){
```

```
68                 tcpClientList.at(i)->write(msg);
69         }
70 }
71
72 //定时器到时执行的槽函数:检容器中套接字是否断开连接的容器
73 void ServerDialog::onTimeout()
74 {
75         //遍历检查容器每个套接字的连接状态
76         for(int i=0;i<tcpClientList.size();i++){
77                 //state():获取套接字连接状态,如果返回 UnconnectedState 表示已经断开连接
78                 if(tcpClientList.at(i)->state() == QAbstractSocket::UnconnectedState){
79                         //将已断开连接的套接字从容器中删除
80                         tcpClientList.removeAt(i);
81                         i--;
82                 }
83         }
84         //容器:s1 s2 s4 s5
85         //下标: 0    1    2    3    4
86 }
```



vi clientdialog.h

```
1 #ifndef CLIENTDIALOG_H
2 #define CLIENTDIALOG_H
3
4 #include <QDialog>
5 //QT += network
6 #include <QTcpSocket>
7 #include <QHostAddress>
```

```
 8 #include <QMessageBox>
 9 #include <QDebug>
10
11 namespace Ui {
12 class ClientDialog;
13 }
14
15 class ClientDialog : public QDialog
16 {
17     Q_OBJECT
18
19 public:
20     explicit ClientDialog(QWidget *parent = 0);
21     ~ClientDialog();
22
23 private slots:
24     //发送按钮对应的槽函数
25     void on_sendButton_clicked();
26     //连接服务器按钮对应的槽函数
27     void on_connectButton_clicked();
28     //和服务器连接成功时执行的槽函数
29     void onConnected();
30     //和服务器断开连接时执行的槽函数
31     void onDisconnected();
32     //接收聊天消息的槽函数
33     void onReadyRead();
34     //网络异常时执行的槽函数
35     void onError();
36 private:
37     Ui::ClientDialog *ui;
38     QTcpSocket tcpSocket;//和服务器通信的套接字
39     QHostAddress serverIp;//服务器 ip 地址
40     quint16 serverPort;//服务器端口
41     QString username;//聊天室昵称
42     bool status;//标记连接状态,true:在线状态  false:离线状态
43 };
44
45 #endif // CLIENTDIALOG_H
```

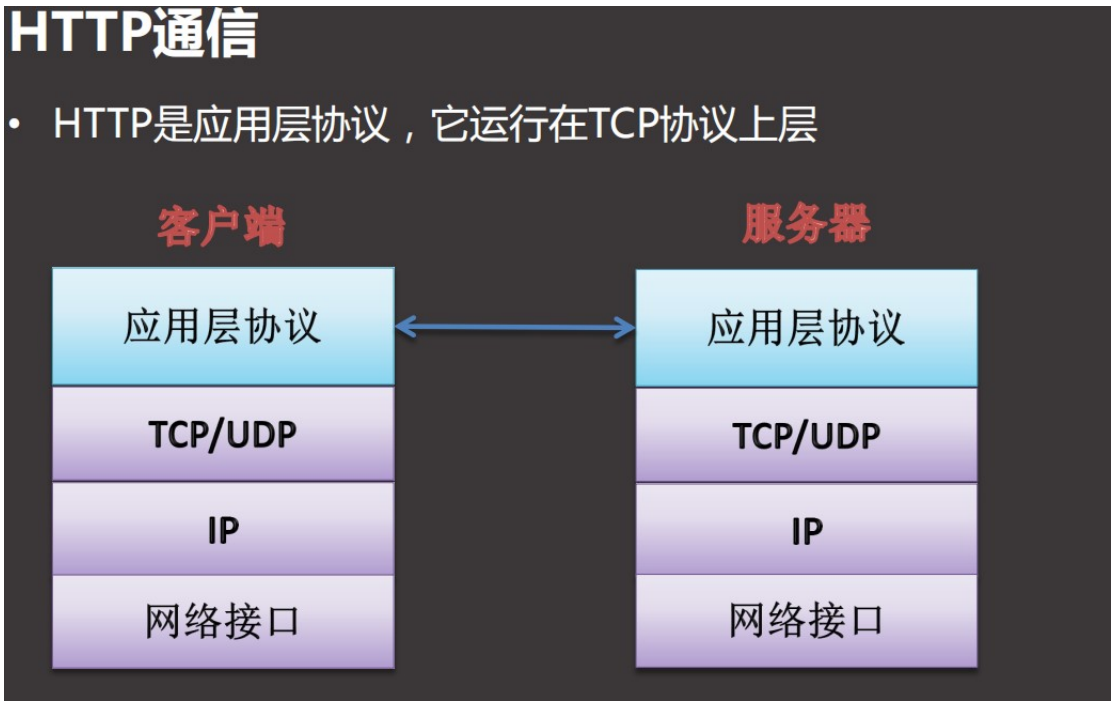
vi clientdialog.cpp

```
 1 #include "clientdialog.h"
 2 #include "ui_clientdialog.h"
 3
 4 ClientDialog::ClientDialog(QWidget *parent) :
```

```
5       QDialog(parent),
6       ui(new Ui::ClientDialog)
7 {
8       ui->setupUi(this);
9       status = false;//标记离线状态
10      //连接服务器成功时,发送信号 connected,连接到和服务器连接成功时执行的槽函数
11      connect(&tcpSocket,SIGNAL(connected()),this,SLOT(onConnected()));
12 //和服务器连接连接时,发送信号 disconnected,连接到和服务器断开连接时执行的槽函数
13      connect(&tcpSocket,SIGNAL(disconnected()),this,SLOT(onDisconnected()));
14      //收到聊天消息时,发送信号 readyRead,连接到接收聊天消息的槽函数
15      connect(&tcpSocket,SIGNAL(readyRead()),this,SLOT(onReadyRead()));
16      //网络通信异常时,发送 error 信号,连接到处理网络异常的槽函数
17      connect(&tcpSocket,SIGNAL(error(QAbstractSocket::SocketError)),
18              this,SLOT(onError()));
19 }
20
21 ClientDialog::~ClientDialog()
22 {
23      delete ui;
24 }
25 //发送按钮对应的槽函数
26 void ClientDialog::on_sendButton_clicked()
27 {
28      //获取用户输入的聊天消息
29      QString msg = ui->messageEdit->text();
30      if(msg == ""){
31          return;
32      }
33      msg = username + ":" + msg;
34      //发送消息
35      tcpSocket.write(msg.toUtf8());
36      //清空已输入的消息
37      ui->messageEdit->clear();
38 }
39 //连接服务器按钮对应的槽函数
40 void ClientDialog::on_connectButton_clicked()
41 {
42      if(status == false){//如果当前是离线状态,则建立连接
43          //设置服务器 IP 地址
44          if(serverIp.setAddress(ui->serverIpEdit->text()) == false){
45              QMessageBox::critical(this,"Error","IP 地址格式错误!");
46              return;
47          }
48          //设置服务器端口
```

```
49              serverPort = ui->serverPortEdit->text().toShort();
50              if(serverPort < 1024){
51                  QMessageBox::critical(this,"Error","端口格式错误!");
52                  return;
53              }
54              //设置聊天室昵称
55              username = ui->usernameEdit->text();
56              if(username == ""){
57                  QMessageBox::critical(this,"Error","聊天室昵称不能为空!");
58                  return;
59              }
60
61              //向服务器发送连接请求
62              //如果连接成功,tcpSocket 发送信号 connected
63              //如果连接失败,tcpSocket 发送信号 error
64              tcpSocket.connectToHost(serverIp,serverPort);
65          }
66          else{//如果当前是在线状态,则断开
67              //发送离开聊天室提示消息
68              QString msg = username + ":离开了聊天室!";
69              tcpSocket.write(msg.toUtf8());
70              //和服务器断开连接,断开连接后会发送信号 disconnected
71              tcpSocket.disconnectFromHost();
72          }
73 }
74 //和服务器连接成功时执行的槽函数
75 void ClientDialog::onConnected()
76 {
77          status = true;//标记在线状态
78          ui->sendButton->setEnabled(true);//恢复发送消息按钮
79          ui->serverIpEdit->setEnabled(false);//禁用 ip 输入
80          ui->serverPortEdit->setEnabled(false);//禁用端口输入
81          ui->usernameEdit->setEnabled(false);//禁用端口输入
82          ui->connectButton->setText("离开服务器");//修改按钮文本
83          //向服务器发送进入聊天室的提示消息
84          QString msg = username + ":进入了聊天室!";
85          //toUtf8:QString(unicode)转换为 QByteArray(utf-8)
86          tcpSocket.write(msg.toUtf8());
87 }
88 //和服务器断开连接时执行的槽函数
89 void ClientDialog::onDisconnected()
90 {
91          status = false;//标记离线状态
92          ui->sendButton->setEnabled(false);//禁用发送消息按钮
```

```
93          ui->serverIpEdit->setEnabled(true);//恢复 ip 输入
94          ui->serverPortEdit->setEnabled(true);//恢复端口输入
95          ui->usernameEdit->setEnabled(true);//恢复端口输入
96          ui->connectButton->setText("连接服务器");//修改按钮文本
97 }
98 //接收聊天消息的槽函数
99 void ClientDialog::onReadyRead()
100 {
101         //bytesAvailable 获取等待读取消息的字节数
102         if(tcpSocket.bytesAvailable()){
103                 //读取消息
104                 QByteArray buf = tcpSocket.readAll();
105                 //显示消息到界面
106                 ui->listWidget->addItem(buf);
107                 ui->listWidget->scrollToBottom();
108         }
109 }
110 //网络异常时执行的槽函数
111 void ClientDialog::onError()
112 {
113         //errorString():获取套接字通信异常原因的字符串
114         QMessageBox::critical(this,"Error",tcpSocket.errorString());
115 }
```



HTTP 通信

• HTTP 协议简介

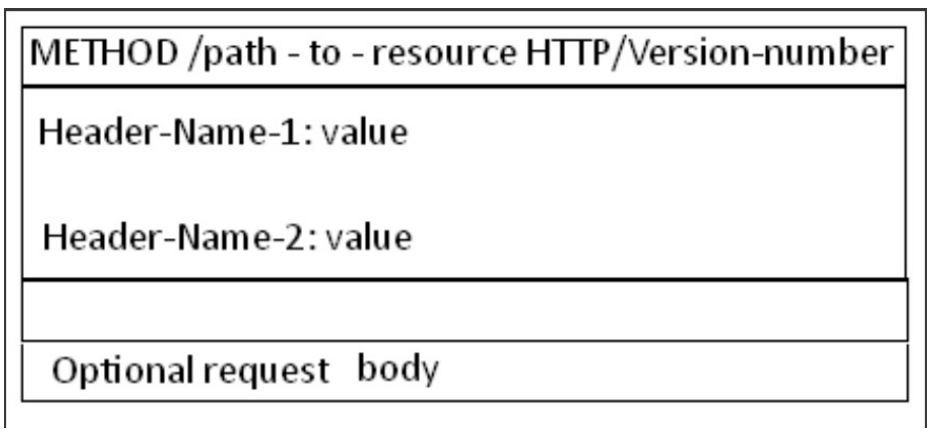
– 超文本传输协议（HyperText Transfer Protocol），是互联网上应用最为广泛的一种网络协议，所有的 HTML 文件都必须遵守这个标准

– HTTP 是一个客户端和服务器端请求和应答的标准，客户端是终端用户，服务器端是网站，客户端通过使用 Web 浏览器或者其它的工具，发起一个到服务器上指定端口（默认端口为 80）的 HTTP 请求。HTTP 服务器监听客户端发送过来的请求，一旦收到请求，服务器将发回一个状态行，比如“HTTP/1.1 200 OK”，和响应消息，响应消息体可能是请求数据、文件、错误消息等

HTTP 通信

• HTTP 协议请求

– HTTP 协议请求（request）是由客户端向服务器发送的数据包，基于 HTTP 协议的传输过程总是由客户端的请求开始，一个请求数据包的格式如下：



• **HTTP 请求格式 (如上图)**

– 请求行，例如 GET /images/logo.gif HTTP/1.1，表示从/images 目录下请求 logo.gif 这个文件
– 请求头，例如 Accept-Language: en
– 空行“\r\n”
– 请求消息体

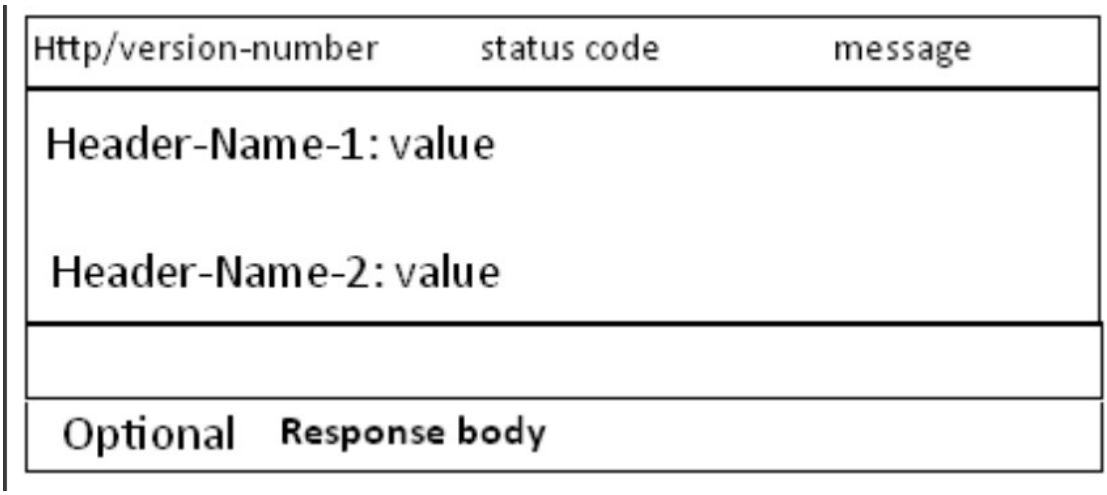
• **HTTP 协议请求案例：**

– 请求行使用了 GET 方法获取 http 服务器根目录下的 index.html 文件，使用 HTTP 1.1 版本的协议
– 请求头使用了 Host 指明虚拟主机
– 接下来是空行
– 最后是请求的消息体，为空

GET /index.html HTTP/1.1
Host: www.example.com
(空行)
(空行)

• **HTTP 协议响应格式：**

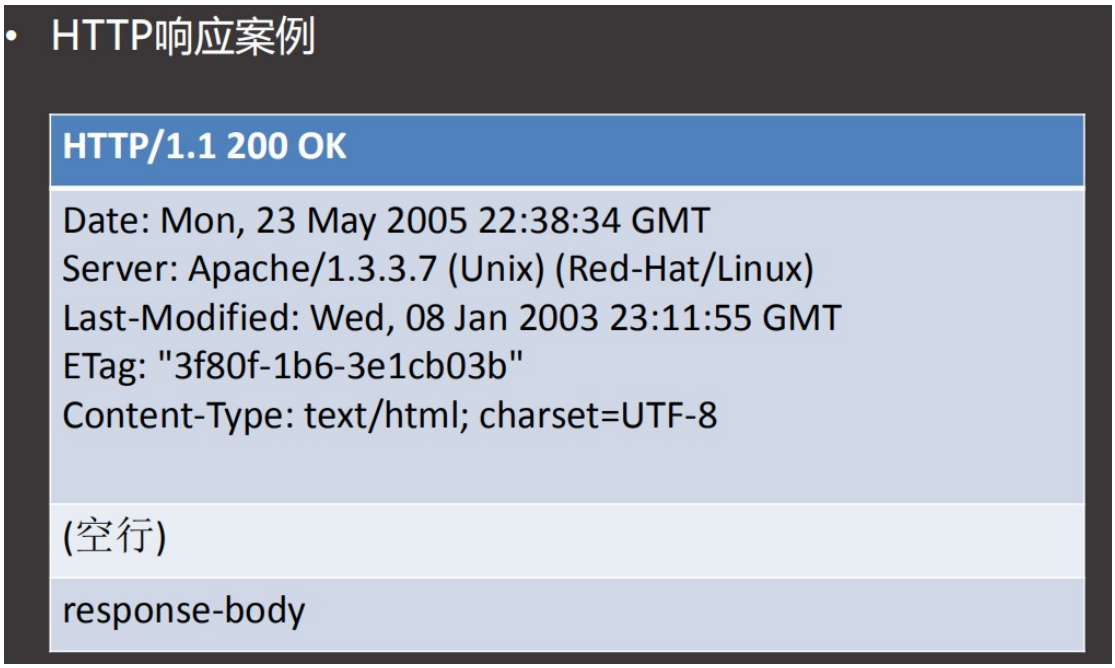
– HTTP 协议响应（response）是由服务器给客户端返回的消息，当服务器收到客户端的请求以后，会根据请求内容，给客户端返回“适当”的数据



• **HTTP 响应格式**

– 状态行，例如 HTTP/1.0 200 OK，表示请求成功
– 响应头，通常服务器需要传递许多附加信息，这些信息不能全放在状态行里。因此，需要另行定义响应头，用来描述这些附加信息

– 空行“\r\n”
– 响应消息体，服务器将响应头域传给客户端后，接着传递的就是客户端请求的数据，如果客户端请求一个网页，服务器就会传递网页，如果请求的是文件，服务器就会发送文件给客户端。



• Qt5 采用统一的网络配置，删除了 Qt4 中 QFtp、QHttp 等类，所有应用层网络通信的 API 都是是围绕一个 QNetworkAccessManager 对象来进行，比如代理和缓存配置，以及处理相关的信号，监视网络的运行状态，网络相关操作的应答信号，通过该类可以管理整个通信过程
• QT 网络应用开发相关类
– QNetworkAccessManager：管理应用程序的发送请求和接收响应数据
– QNetworkRequest：根据 URL 封装请求数据包
– QNetworkReply：接收和请求对应的响应数据
– QUrl：网络地址（Uniform Resource Locator，即统一资源定位地址）
• 使用 QNetworkAccessManager 对象创建以后，应用程序可以使用它来在网络上发送的请求(QNetworkRequest)，然后将返回一个用于接收响应数据的 QNetworkReply 对象

```
//创建管理通信的 manager 对象
QNetworkAccessManager *manager = new QNetworkAccessManager(this);
//根据 URL 初始化请求
QNetworkRequest request(QUrl(“http://qt-project.org”));
//使用 get 方法发送请求，并返回用于接收响应数据的 reply 对象指针
QNetworkReply *reply = manager->get(QNetworkRequest(url));
```

vi mainwindow.h

```
 1 #ifndef MAINWINDOW_H
 2 #define MAINWINDOW_H
 3
 4 #include <QMainWindow>
 5 //QT += network
 6 #include <QNetworkAccessManager> //管理通信
 7 #include <QNetworkRequest> //请求
 8 #include <QNetworkReply> //响应
 9 #include <QUrl> //网络地址
10 #include <QAuthenticator> //登录认证
11 #include <QFileInfo> //文件信息
12 #include <QFile> //文件操作
13 #include <QDebug> //打印调试
14
15 namespace Ui {
16 class MainWindow;
17 }
18
19 class MainWindow : public QMainWindow
20 {
21         Q_OBJECT
22
23 public:
24         explicit MainWindow(QWidget *parent = 0);
25         ~MainWindow();
26 public slots:
27         //向服务器发送请求
28         void sendRequest();
```

```
29      //处理登录认证的槽函数
30      void onAuthenticationRequired(QNetworkReply*,QAuthenticator*);
31      //接收响应数据的槽函数
32      void onReadyRead();
33      //接收响应数据结束的槽函数
34      void onFinished();
35
36      //处理目录链接的槽函数,参数为点击链接 url 地址
37      void onAnchorclicked(const QUrl& url);
38
39      //下载文件操作
40      void downloadFile(const QUrl& fileUrl);
41      //接收文件内容的槽函数
42      void receiveFile();
43      //打印当前下载进度槽函数,参数:已收到字节数/总字节数
44      void onDownloadProgress(qint64,qint64);
45      //接收文件内容结束执行的槽函数
46      void receiveFileFinished();
47
48 private:
49      Ui::MainWindow *ui;
50      QNetworkAccessManager *manager;//管理 HTTP 通信的请求和响应
51      QNetworkRequest request;//请求
52      QNetworkReply* reply;//响应
53      QUrl currentUrl;//记录当前 URL 地址
54      QByteArray buf;//数据缓冲区:接收服务器返回的影响数据
55      QFile* file;//保存要下载的文件
56 };
57
58 #endif // MAINWINDOW_H
```

vi mainwindow.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3 #include "logindialog.h"
 4
 5 MainWindow::MainWindow(QWidget *parent) :
 6      QMainWindow(parent),
 7      ui(new Ui::MainWindow)
 8 {
 9      ui->setupUi(this);
10      //创建管理通信的 Manager 对象
11      manager = new QNetworkAccessManager(this);
12      //初始化请求 URL
```

```
13      request.setUrl(QUrl("http://code.tarena.com.cn/"));
14      //发送请求
15      sendRequest();
16
17      //点击界面的链接时,发送信号 anchorClicked,连接到处理目录链接的槽函数
18      connect(ui->textBrowser,SIGNAL(anchorClicked(QUrl)),
19              this,SLOT(onAnchorclicked(QUrl)));
20
21 }
22
23 MainWindow::~MainWindow()
24 {
25      delete ui;
26 }
27 //向服务器发送请求
28 void MainWindow::sendRequest()
29 {
30      //通过 manager 对象,使用 GET 方法向服务器发送请求,返回用于接收响应数据的
reply 对象指针
31      reply = manager->get(request);
32      //如果请求的 URL 服务器需要登录认证,发送信号 authenticationRequired 连接到处理
登录认证的槽函数
33      connect(manager,SIGNAL(authenticationRequired(
34                              QNetworkReply*,QAuthenticator*)),
35              this,SLOT(onAuthenticationRequired(
36                              QNetworkReply*,QAuthenticator*)));
37      //当服务器返回响应数据,reply 将会发送 readyRead 信号,连接到接收响应的槽函数
38      connect(reply,SIGNAL(readyRead()),this,SLOT(onReadyRead()));
39 //当接收响应数据完成,reply 将会发送 finished 信号,连接接收响应数据结束的槽函数
40      connect(reply,SIGNAL(finished()),this,SLOT(onFinished()));
41 }
42 //处理登录认证的槽函数
43 void MainWindow::onAuthenticationRequired(
44          QNetworkReply*,QAuthenticator* a)
45 {
46      //创建登录认证的窗口
47      LoginDialog login(this);
48 //显示登录子窗口并进入事件循环,点击上面 Ok 或者 Cancel 都会退出事件循环,但是
49      //返回值不同,如果点击 Ok 验证登录,点击 Cancel 不验证.
50      if(login.exec() == QDialog::Accepted){
51          //qDebug() << login.getUsername();
52          a->setUser(login.getUsername());
53          //qDebug() << login.getPassword();
54          a->setPassword(login.getPassword());
```

```
55          }
56 }
57 //接收响应数据的槽函数
58 void MainWindow::onReadyRead()
59 {
60          //接收响应数据并保存,+=追加保存
61          buf += reply->readAll();
62          //记录当前 URL 地址
63          currentUrl = reply->url();
64 }
65 //接收响应数据结束的槽函数
66 void MainWindow::onFinished()
67 {
68          //显示
69          ui->textBrowser->setText(buf);
70          //清空 buf
71          buf.clear();
72          //销毁 reply 对象
73          reply->deleteLater();//delete this;
74 }
75
76 //处理目录链接的槽函数,参数为点击链接 url 地址
77 void MainWindow::onAnchorclicked(const QUrl& url)
78 {
79          //qDebug() << "当前 URL:" << currentUrl.toString();
80          //qDebug() << "点击 URL:" << url.toString();
81          QUrl newUrl;//要进入的新的链接 URL 地址
82          //如果点击时不是"../"
83          if(url.toString() != "../"){
84                    //newUrl =  当前 URL+点击 URL
85                    newUrl = currentUrl.toString() + url.toString();
86          }
87          else{//处理../
88                    //如果当前 URL 在首界面,点击"../"什么也不做
89                    if(currentUrl.toString() == "http://code.tarena.com.cn/"){
90                              return;
91                    }
92                    //如果不在首界面,点击"../"去掉最后一级链接路径
93                    //当前 URL: "http://code.tarena.com.cn/CSDCode/csd2002/"
94                    //点击../: "http://code.tarena.com.cn/CSDCode/
95                    //查找倒数第二次出现"/"位置
96                    int pos = currentUrl.toString().lastIndexOf("/",-2);
97                    //字符串截断,截取到 pos 所在位置
98                    newUrl = currentUrl.toString().mid(0,pos+1);
```

```
 99      }
100
101     //判断 newUrl 是否为文件链接,如果是文件链接,则执行下载文件操作
102     //判断方式:如果点击 url 不是目录链接,就是要下载的文件链接
103     if(url.toString().lastIndexOf("/") == -1){
104         //qDebug() << "下载文件操作:" << newUrl;
105         downloadFile(newUrl);
106     }
107     else{
108         //设置请求为 newUrl
109         request.setUrl(newUrl);
110         //发送新的请求
111         sendRequest();
112     }
113 }
114
115 //下载文件操作
116 void MainWindow::downloadFile(const QUrl& fileUrl)
117 {
118     //根据获取 fileUrl 获取文件名
119     QFileInfo fileInfo = fileUrl.path();
120     QString filename = fileInfo.fileName();
121     //在本地创建同名的文件
122     file = new QFile(filename,this);
123     //打开本地同名文件,如果不存在将会自动创建
124     file->open(QIODevice::WriteOnly);
125
126     //设置请求 URl 为下载文件的 URL
127     request.setUrl(fileUrl);
128     //发送下载文件连接请求
129     reply = manager->get(request);
130     //当收到服务器返回的响应数据(文件内容),发送信号 readyRead(),连接到接收文件
内容的槽函数
131     connect(reply,SIGNAL(readyRead()),this,SLOT(receiveFile()));
132     //伴随文件下载 reply 会发送下载进度信号 downloadProgress,连接到打印当前下载进
度槽函数
133     connect(reply,SIGNAL(downloadProgress(qint64,qint64)),
134             this,SLOT(onDownloadProgress(qint64,qint64)));
135     //文件下载完成,发送信号 finished,连接到接收文件内容结束执行的槽函数
136     connect(reply,SIGNAL(finished()),this,SLOT(receiveFileFinished()));
137 }
138 //接收文件内容的槽函数
139 void MainWindow::receiveFile()
140 {
```

```
141         //读取响应数据并写入本地同名文件
142         file->write( reply->readAll() );
143 }
144 //打印当前下载进度槽函数,参数:已收到字节数/总字节数
145 void MainWindow::onDownloadProgress(qint64 readBytes,qint64 totalBytes)
146 {
147         //计算下载进度的百分比
148         qint64 progress = readBytes*100 / totalBytes;
149         //打印进度
150         qDebug() << file->fileName() << ":" << progress << "%...";
151 }
152 //接收文件内容结束执行的槽函数
153 void MainWindow::receiveFileFinished()
154 {
155         qDebug() << file->fileName() << ":文件下载完成!";
156         file->flush();//刷新文件流
157         file->close();
158         reply->deleteLater();
159 }
160
161 //不足:一次只能下载一个文件
162 //扩展:
163 //1)使用多线程,将下载文件操作放到子线程中,实现多个文件同时下载
164 //2)将要下载的文件,指定保存在/home/tarena/Downloads
165 //参考:refer.tar.gz
```

vi logindialog.h

```
  1 #ifndef LOGINDIALOG_H
  2 #define LOGINDIALOG_H
  3
  4 #include <QDialog>
  5
  6 namespace Ui {
  7 class LoginDialog;
  8 }
  9
 10 class LoginDialog : public QDialog
 11 {
 12         Q_OBJECT
 13
 14 public:
 15         explicit LoginDialog(QWidget *parent = 0);
 16         ~LoginDialog();
 17
```

```
18 private slots:
19         //ok 按钮对应的槽函数
20         void on_buttonBox_accepted();
21         //Cancel 按钮对应的槽函数
22         void on_buttonBox_rejected();
23 public:
24         //获取用户名
25         const QString& getUsername();
26         //获取密码
27         const QString& getPassword();
28 private:
29         Ui::LoginDialog *ui;
30         QString m_username;//记录用户名
31         QString m_password;//记录密码
32 };
33
34 #endif // LOGINDIALOG_H
```

vi logindialog.cpp

```
 1 #include "logindialog.h"
 2 #include "ui_logindialog.h"
 3
 4 LoginDialog::LoginDialog(QWidget *parent) :
 5         QDialog(parent),
 6         ui(new Ui::LoginDialog)
 7 {
 8         ui->setupUi(this);
 9 }
10
11 LoginDialog::~LoginDialog()
12 {
13         delete ui;
14 }
15 //ok 按钮对应的槽函数
16 void LoginDialog::on_buttonBox_accepted()
17 {
18         //从登录界面获取用户名和密码
19         m_username = ui->usernameEdit->text();
20         m_password = ui->passwdEdit->text();
21         //退出登录窗口的事件循环,返回 QDialog::Accepted
22         accept();
23 }
24 //Cancel 按钮对应的槽函数
25 void LoginDialog::on_buttonBox_rejected()
```

```
26 {
27         //退出登录窗口的事件循环,返回 QDialog::Rejected
28         reject();
29 }
30 //获取用户名
31 const QString& LoginDialog::getUsername()
32 {
33         return m_username;
34 }
35 //获取密码
36 const QString& LoginDialog::getPassword()
37 {
38         return m_password;
39 }
```

DAY07

1　TCP 编程

1）QTcpserver //TCP 服务器

2）QTcpSocket //TCP 套接字

2　HTTP 编程

1）QNetworkManagerAccess //管理通信

2）QnetworkRequest //请求

3）QNetworkReply //响应

代码下载器拓展任务：增加多线程，目录操作

vi download.h

```
 1 #ifndef Download_H
 2 #define Download_H
 3
 4 #include <QThread>
 5 #include <QNetworkAccessManager>
 6 #include <QNetworkRequest>
 7 #include <QNetworkReply>
 8 #include <QUrl>
 9 #include <QDir>
10 #include <QFile>
11 #include <QFileInfo>
12 #include <mainwindow.h>
13
14 class Download : public QObject
15 {
```

```
16      Q_OBJECT
17 public:
18      //构造函数，参数表示要下载的文件连接
19      Download(const QUrl& url);
20      ~Download();
21 private slots:
22      //接收文件的槽函数
23      void ReceiveFile();
24      //更新显示文件下载进度的槽函数
25      //参数:已收到数据的字节数/总字节数
26      void onDownloadProgress(qint64,qint64);
27      //接收文件完成的槽函数
28      void ReceiveFileFinished();
29
30 signals:
31      //自定义信号，文件下载完成时发送,只需声明,不能写定义
32      void downloadFinished();
33 private:
34      QNetworkRequest request;//请求
35      QNetworkReply* reply;//响应
36      QUrl fileUrl;//下载文件的 URL 地址
37      QFile* file;
38
39      //将 MainWindow 类声明为当前类的友元，友元类可以访问当前类的任何成员
40      friend class MainWindow;
41 };
42
43 #endif // Download_H
```

vi download.cpp

```
 1 #include "download.h"
 2 #include "mainwindow.h"
 3
 4 Download::Download(const QUrl& url):fileUrl(url){
 5      //根据 URL 获取文件名
 6      QFileInfo fileInfo = fileUrl.path();
 7      QString filename = fileInfo.fileName();
 8
 9      //设置要下载文件的路径
10      QDir dir = QDir::home();
11      //判断是否存在 Download 目录
12      if(dir.exists("Downloads")==false){
13           //如果该目录不存在则创建
14           if(dir.mkdir("Downloads") == false){
```

```
15                  qDebug("创建目录失败");
16                  return;
17              }
18      }
19      //指定下载文件放到主目录的 Download 下面
20      QString path = QDir::homePath() + "/Downloads/"+ filename;
21      //在本地创建同名的文件
22      file = new QFile(path);
23      //以写的方式打开文件
24      file->open(QIODevice::WriteOnly);
25 }
26 Download::~Download()
27 {
28      //qDebug() << "~Download";
29 }
30 //接收文件的槽函数
31 void Download::ReceiveFile()
32 {
33      if(reply->bytesAvailable()){
34          file->write(reply->readAll());
35      }
36 }
37 //参数:已收到数据的字节数/总字节数
38 void Download::onDownloadProgress(
39              qint64 readBytes,qint64 totalBytes)
40 {
41      qint64 progress=readBytes*100/totalBytes;//百分比
42      qDebug() << file->fileName() << ":" << progress << "%....";
43 }
44 //接收文件完成的槽函数
45 void Download::ReceiveFileFinished()
46 {
47      qDebug() << file->fileName() << "文件下载完成";
48      file->flush();//刷新文件流
49      file->close();//关闭文件
50      delete file;//销毁文件对象
51      reply->deleteLater();//销毁响应对象
52      emit downloadFinished();//发送信号，表示文件下载完成
53 }//emit:标记发送信号
```

vi mainwindow.h

```
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
```

```
 4 #include <QMainWindow>
 5 #include <QNetworkAccessManager>
 6 #include <QNetworkRequest>
 7 #include <QNetworkReply>
 8 #include <QUrl>
 9 #include <QAuthenticator>
10 #include <QFile>
11 #include <QFileInfo>
12 #include <QDebug>
13 #include <download.h>
14
15 namespace Ui {
16 class MainWindow;
17 }
18
19 class MainWindow : public QMainWindow
20 {
21     Q_OBJECT
22
23 public:
24     explicit MainWindow(QWidget *parent = 0);
25     ~MainWindow();
26     //向服务器发送请求
27     void sendRequest();
28 private slots:
29     //处理登录认证的槽函数
30     void onAuthenticationRequired(
31             QNetworkReply*,QAuthenticator*);
32     //接收响应数据的槽函数
33     void onReadyRead();
34     //接收响应数据完成时执行的槽函数
35     void onFinished();
36     //处理目录链接的槽函数
37     void onAnchorClicked(const QUrl& url);
38
39 private:
40     //下载文件
41     void downloadFile(const QUrl& fileUrl);
42
43 private:
44     Ui::MainWindow *ui;
45 public:
46     QNetworkAccessManager* manager;//管理通信
47 private:
```

```
48          QNetworkRequest request;//请求
49          QNetworkReply* reply;//响应
50          QUrl currentUrl;//记录当前的 Url 地址
51          QByteArray buf;//保存接收的响应数据
52 };
53
54 #endif // MAINWINDOW_H
```

vi mainwindow.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3 #include "logindialog.h"
 4 #include "download.h"
 5
 6 MainWindow::MainWindow(QWidget *parent) :
 7          QMainWindow(parent),
 8          ui(new Ui::MainWindow)
 9 {
10          ui->setupUi(this);
11
12          //创建管理通信的 manager 对象
13          manager = new QNetworkAccessManager(this);
14          //初始化请求网址
15          request.setUrl(QUrl("http://code.tarena.com.cn/"));
16          //向服务器发送请求
17          sendRequest();
18          //点击界面的链接时,发送信号 anchorClicked,
19          //参数表示点击链接的 URL 地址
20          connect(ui->textBrowser,SIGNAL(anchorClicked(QUrl)),
21                  this,SLOT(onAnchorClicked(QUrl)));
22 }
23 MainWindow::~MainWindow()
24 {
25          delete ui;
26 }
27
28 //向服务器发送请求
29 void MainWindow::sendRequest()
30 {
31          //发送请求时,禁用界面,避免连续发送请求而导致的异常结束
32          ui->textBrowser->setEnabled(false);
33          //向服务器发送请求
34          reply = manager->get(request);
35          //如果服务器需要进行登录认证,manager 会发送认证
```

```
36      //信号:authenticationRequired
37      connect(manager,SIGNAL(
38              authenticationRequired(
39                      QNetworkReply*,QAuthenticator*)),
40              this,SLOT(onAuthenticationRequired(
41                      QNetworkReply*,QAuthenticator*)));
42      //如果认证成功,响应数据到来,发送信号 readyRead
43      connect(reply,SIGNAL(readyRead()),
44              this,SLOT(onReadyRead()));
45      //响应数据接收结束,发送信号 finished
46      connect(reply,SIGNAL(finished()),
47              this,SLOT(onFinished()));
48 }
49 //处理登录认证的槽函数
50 void MainWindow::onAuthenticationRequired(
51              QNetworkReply*,QAuthenticator* authenticator)
52 {
53      //qDebug("onAuthenticationRequired");
54      //从登录认证子窗口中,获取用户名和密码在进行认证
55      LoginDialog login(this);
56      //显示登录窗口,并进入事件循环,点击上面 Ok/Cancel
57      //时都会退出登录窗口,但是返回值不同.
58      //如果点击 Ok 按钮退出,返回 QDialog::Accepted
59      if(login.exec() == QDialog::Accepted){
60              authenticator->setUser(login.getUsername());
61              authenticator->setPassword(login.getPassword());
62      }
63 }
64 //接收响应数据的槽函数
65 void MainWindow::onReadyRead()
66 {
67      //qDebug("onReadyRead");
68      //读取响应数据,并保存
69      buf += reply->readAll();
70      //保存当前 URl 地址
71      currentUrl = reply->url();
72 }
73 //接收响应数据完成时执行的槽函数
74 void MainWindow::onFinished()
75 {
76      //qDebug("onFinished");
77      //显示响应数据
78      ui->textBrowser->setText(buf);
79      //清空 buf
```

```
80      buf.clear();
81      //销毁 reply 对象
82      reply->deleteLater();
83      //恢复界面
84      ui->textBrowser->setEnabled(true);
85 }
86
87 //处理目录链接的槽函数
88 void MainWindow::onAnchorClicked(
89              const QUrl &url){
90      //qDebug()<<"当前的 URL:"<<currentUrl.toString();
91      //qDebug()<<"点击的 URL:"<<url.toString();
击的 URL:"<<url.toString();
92
93      QUrl newUrl;
94      //如果点击是不是"../",新的 URL=当前 URL+点击 URL
95      if(url.toString() != "../"){
96          newUrl = currentUrl.toString() +
97                          url.toString();
98      }
99      //如果点击是"../"
100     else{
101         //如果当前在首页,什么也不做
102         if(currentUrl.toString() ==
103             "http://code.tarena.com.cn/"){
104             return;
105         }
106         //如果不再首页,去掉最后一级链接路径
107         //查找目录路径中倒数第二次出现"/"位置
108         int pos = currentUrl.toString(
109                         ).lastIndexOf("/",-2);
110         //字符串截断,去掉后面的路径
111         newUrl =
112             currentUrl.toString().mid(0,pos+1);
113     }
114     //判断点击 URL 如果不是目录则执行文件下载操作.
115     if(url.toString().lastIndexOf("/")==-1){
116         downloadFile(newUrl);
117         return;
118     }
119     //设置新的请求 URL
120     request.setUrl(newUrl);
121     //发送新的请求
122     sendRequest();
```

```
123 }
124
125 //下载文件
126 void MainWindow::downloadFile(const QUrl& fileUrl)
127 {
128         //创建下载文件线程
129         QThread *thread = new QThread;
130         //创建下载文件对象
131         Download* download = new Download(fileUrl);
132         //将下载文件对象移动到子线程中
133         download->moveToThread(thread);
134         //下载文件完成时，让线程退出
135         connect(download,SIGNAL(downloadFinished()),
136                         thread,SLOT(quit()));
137         //线程结束时，删除下载文件的对象
138         connect(thread,SIGNAL(finished()),
139                         download,SLOT(deleteLater()));
140         //线程结束时，删除线程对象
141         connect(thread,SIGNAL(finished()),
142                         thread,SLOT(deleteLater()));
143         //设置下载文件请求的 URL
144         download->request.setUrl(fileUrl);
145
146         //发送获取下载文件的请求
147         download->reply = manager->get(download->request);
148
149         //响应数据到来，在子线程中完成下载操作
150         //download 已经移动到子线程中，通过信号触发里面槽将在子线程中运行
151         connect(download->reply,SIGNAL(readyRead()),
152                         download,SLOT(ReceiveFile()));
153         connect(download->reply,SIGNAL(finished()),
154                         download,SLOT(ReceiveFileFinished()));
155         connect(download->reply,SIGNAL(downloadProgress(qint64,qint64)),
156                         download,SLOT(onDownloadProgress(qint64,qint64)));
157         //开启子线程
158         thread->start();
159 }
```

vi logindialog.h

```
  1 #ifndef LOGINDIALOG_H
  2 #define LOGINDIALOG_H
  3
  4 #include <QDialog>
  5
```

```
 6 namespace Ui {
 7 class LoginDialog;
 8 }
 9
10 class LoginDialog : public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     explicit LoginDialog(QWidget *parent = 0);
16     ~LoginDialog();
17
18 private slots:
19     //Ok 按钮对应的槽函数
20     void on_buttonBox_accepted();
21     //Cancel 按钮对应的槽函数
22     void on_buttonBox_rejected();
23 public:
24     //获取用户名
25     const QString& getUsername();
26     //获取密码
27     const QString& getPassword();
28 private:
29     Ui::LoginDialog *ui;
30     QString username;//保存用户名
31     QString password;//保存密码
32 };
33
34 #endif // LOGINDIALOG_H
```

vi logindialog.cpp

```
 1 #include "logindialog.h"
 2 #include "ui_logindialog.h"
 3
 4 LoginDialog::LoginDialog(QWidget *parent) :
 5     QDialog(parent),
 6     ui(new Ui::LoginDialog)
 7 {
 8     ui->setupUi(this);
 9 }
10
11 LoginDialog::~LoginDialog()
12 {
13     delete ui;
```

```
14 }
15 //Ok 按钮对应的槽函数
16 void LoginDialog::on_buttonBox_accepted()
17 {
18         username = ui->usernameEdit->text();
19         password = ui->passwordEdit->text();
20         accept();//退出,返回 QDialog::Accepted
21 }
22 //Cancel 按钮对应的槽函数
23 void LoginDialog::on_buttonBox_rejected()
24 {
25         reject();//退出,返回 QDialog::Rejected
26 }
27 //获取用户名
28 const QString& LoginDialog::getUsername()
29 {
30         return username;
31 }
32 //获取密码
33 const QString& LoginDialog::getPassword()
34 {
35         return password;
36 }
```

多窗口编程：

1）先显示主窗口，在弹出子窗口 //Windows1

2）先显示子窗口，再进入主窗口(例如 qq)//Windows2

3）同时显示多个窗口//Windows3

Windows1

vi mainwindow.h

```
 1 #ifndef MAINWINDOW_H
 2 #define MAINWINDOW_H
 3
 4 #include <QMainWindow>
 5
 6 namespace Ui {
 7 class MainWindow;
 8 }
 9
```

```
10 class MainWindow : public QMainWindow
11 {
12         Q_OBJECT
13
14 public:
15         explicit MainWindow(QWidget *parent = 0);
16         ~MainWindow();
17
18 private slots:
19         void on_action_triggered();
20
21 private:
22         Ui::MainWindow *ui;
23 };
24
25 #endif // MAINWINDOW_H
```

vi mainwindow.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3 #include "dialog.h"
 4 MainWindow::MainWindow(QWidget *parent) :
 5         QMainWindow(parent),
 6         ui(new Ui::MainWindow)
 7 {
 8         ui->setupUi(this);
 9 }
10
11 MainWindow::~MainWindow()
12 {
13         delete ui;
14 }
15 //菜单选项
16 void MainWindow::on_action_triggered()
17 {
18 #if 0
19         Dialog dialog(this);//子窗口
20         dialog.exec();//显示子窗口并进入事件循环(主窗口不能继续响应事件循环)
21 #else
22         Dialog* pdialog = new Dialog(this);
23         pdialog->show();//显示子窗口,不影响主窗口的事件循环
24 #endif
25 }
```

vi dialog.h

```
 1 #ifndef DIALOG_H
 2 #define DIALOG_H
 3
 4 #include <QDialog>
 5
 6 namespace Ui {
 7 class Dialog;
 8 }
 9
10 class Dialog : public QDialog
11 {
12         Q_OBJECT
13
14 public:
15         explicit Dialog(QWidget *parent = 0);
16         ~Dialog();
17
18 private:
19         Ui::Dialog *ui;
20 };
21
22 #endif // DIALOG_H
```

vi dailog.cpp

```
 1 #include "dialog.h"
 2 #include "ui_dialog.h"
 3
 4 Dialog::Dialog(QWidget *parent) :
 5         QDialog(parent),
 6         ui(new Ui::Dialog)
 7 {
 8         ui->setupUi(this);
 9 }
10
11 Dialog::~Dialog()
12 {
13         delete ui;
14 }
```

Windows2

vi mainwindows.h

```
 1 #ifndef MAINWINDOW_H
 2 #define MAINWINDOW_H
```

```
 3
 4 #include <QMainWindow>
 5
 6 namespace Ui {
 7 class MainWindow;
 8 }
 9
10 class MainWindow : public QMainWindow
11 {
12         Q_OBJECT
13
14 public:
15         explicit MainWindow(QWidget *parent = 0);
16         ~MainWindow();
17
18 private:
19         Ui::MainWindow *ui;
20 };
21
22 #endif // MAINWINDOW_H
```

vi mainwindows.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3
 4 MainWindow::MainWindow(QWidget *parent) :
 5         QMainWindow(parent),
 6         ui(new Ui::MainWindow)
 7 {
 8         ui->setupUi(this);
 9 }
10
11 MainWindow::~MainWindow()
12 {
13         delete ui;
14 }
```

vi logindialog.h

```
 1 #ifndef LOGINDIALOG_H
 2 #define LOGINDIALOG_H
 3
 4 #include <QDialog>
 5
 6 namespace Ui {
```

```
 7 class LoginDialog;
 8 }
 9
10 class LoginDialog : public QDialog
11 {
12         Q_OBJECT
13
14 public:
15         explicit LoginDialog(QWidget *parent = 0);
16         ~LoginDialog();
17
18 private slots:
19         void on_buttonBox_accepted();
20
21         void on_buttonBox_rejected();
22
23 private:
24         Ui::LoginDialog *ui;
25 };
26
27 #endif // LOGINDIALOG_H
```

vi logindialog.cpp

```
 1 #include "logindialog.h"
 2 #include "ui_logindialog.h"
 3
 4 LoginDialog::LoginDialog(QWidget *parent) :
 5         QDialog(parent),
 6         ui(new Ui::LoginDialog)
 7 {
 8         ui->setupUi(this);
 9 }
10
11 LoginDialog::~LoginDialog()
12 {
13         delete ui;
14 }
15 //ok
16 void LoginDialog::on_buttonBox_accepted()
17 {
18         if("ok"){
19                 accept();
20         }
21 }
```

```
22 //Cancel
23 void LoginDialog::on_buttonBox_rejected()
24 {
25         close();
26 }
```

vi main.cpp

```
 1 #include "mainwindow.h"
 2 #include <QApplication>
 3 #include "logindialog.h"
 4
 5 int main(int argc, char *argv[])
 6 {
 7         QApplication a(argc, argv);
 8
 9         LoginDialog login;
10         if(login.exec() == QDialog::Accepted){
11                 MainWindow w;
12                 w.show();
13
14                 return a.exec();
15         }
16 }
```

Windows3

vi mainwindows1.h

```
 1 #ifndef MAINWINDOW1_H
 2 #define MAINWINDOW1_H
 3
 4 #include <QMainWindow>
 5
 6 namespace Ui {
 7 class MainWindow1;
 8 }
 9
10 class MainWindow1 : public QMainWindow
11 {
12         Q_OBJECT
13
14 public:
15         explicit MainWindow1(QWidget *parent = 0);
16         ~MainWindow1();
17
18 private:
```

```
19          Ui::MainWindow1 *ui;
20 };
21
22 #endif // MAINWINDOW1_H
```

vi mainwindows1.cpp

```
 1 #include "mainwindow1.h"
 2 #include "ui_mainwindow1.h"
 3
 4 MainWindow1::MainWindow1(QWidget *parent) :
 5          QMainWindow(parent),
 6          ui(new Ui::MainWindow1)
 7 {
 8          ui->setupUi(this);
 9 }
10
11 MainWindow1::~MainWindow1()
12 {
13          delete ui;
14 }
```

vi mainwindows2.h

```
 1 #ifndef MAINWINDOW2_H
 2 #define MAINWINDOW2_H
 3
 4 #include <QMainWindow>
 5
 6 namespace Ui {
 7 class MainWindow2;
 8 }
 9
10 class MainWindow2 : public QMainWindow
11 {
12          Q_OBJECT
13
14 public:
15          explicit MainWindow2(QWidget *parent = 0);
16          ~MainWindow2();
17
18 private:
19          Ui::MainWindow2 *ui;
20 };
21
22 #endif // MAINWINDOW2_H
```

vi mainwindows2.cpp

```
 1 #include "mainwindow2.h"
 2 #include "ui_mainwindow2.h"
 3
 4 MainWindow2::MainWindow2(QWidget *parent) :
 5         QMainWindow(parent),
 6         ui(new Ui::MainWindow2)
 7 {
 8         ui->setupUi(this);
 9 }
10
11 MainWindow2::~MainWindow2()
12 {
13         delete ui;
14 }
```

vi main.cpp

```
 1 #include "mainwindow1.h"
 2 #include "mainwindow2.h"
 3 #include <QApplication>
 4
 5 int main(int argc, char *argv[])
 6 {
 7         QApplication a(argc, argv);
 8         MainWindow1 w1;
 9         w1.show();
10         MainWindow2 w2;
11         w2.show();
12
13         return a.exec();
14 }
```

十七、QT 语言学家

/home/tarena/.config 文件的 QrProject 是配置 Qt 的环境

什么是 Qt 语言家

• Qt 应用程序开发中，源代码或界面中经常会出现一些字符串，比如控件上需要显式的文本字符串，打印的消息提示字符串，在不同的语言环境下，可能会需要显示不同语言对应的文本

• 针对多语言切换的需求，Qt 提供了语言家(Qt Linguist)，在所有需要翻译的文本处都使用 QObject::tr()函数，Qt 语言家会提取 QObject::tr()的参数字符串，使用 Qt 语言家对其进行翻译再发布和使用，Qt 程序可以在运行时加载发

布的翻译文件以更新文本字符串
• 简单的说 QT 语言家是 Qt 的一个附加组件，就是解决语言国际化问题，让文本字符串得到正确的显示
• Qt 语言家系统主要是利用 Qt 提供的工具 lupdate、linguist 和 lrelease 协助翻译工作并生成最后需要的“.qm”翻译文件
– lupdate：用于从源码或界面中扫描并提取需要翻译的字符串，生成“.ts”文件
注: 运行 lupdate 时需要在工程文件添加环境变量 TRANSLATIONS，指定.ts 文件
– linguist：用于协助完成翻译工作，即打开前面用 lupdate 生成的“.ts”文件，对其中的字符串逐条进行翻译并保存。
注:“.ts”文件采用了 XML 格式，也可以使用其他编辑器来打开 ts 文件并翻译
– lrelease：用于处理翻译好的“.ts”文件，生成或更新“.qm”翻译文件
注:“.qm”是 Qt 翻译器(QTranslator)最终需要使用资源文件

Qt 语言家的使用

• 第一步：在工程中添加“.ts”文件
– 新版本的 qtcreator 可以再创建工程时指定需要的“.ts”文件

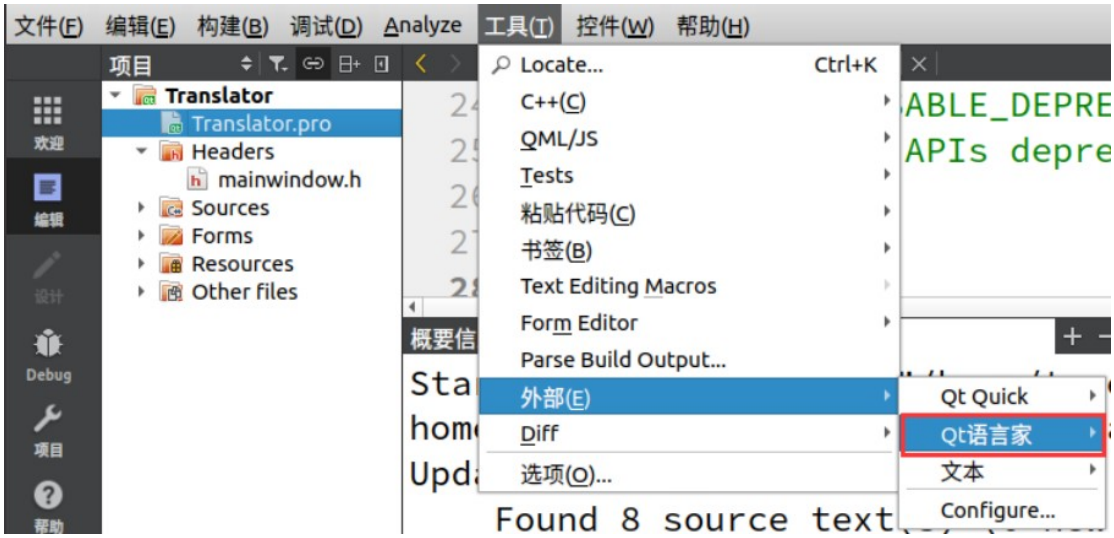
```
SOURCES += \
        main.cpp \
        mainwindow.cpp

HEADERS += \
        mainwindow.h

FORMS += \
        mainwindow.ui

TRANSLATIONS = translation_ch.ts
```

• 第二步：创建或更新“.ts”文件
– 在 qtcreator 环境中，在菜单栏位置，依次执行：
– 工具(Tools)->外部(External)->Qt 语言家(Linguist)->更新翻译(lupdate)
知
注：如果“.ts”不存在将会自动创建，如果存在则更新

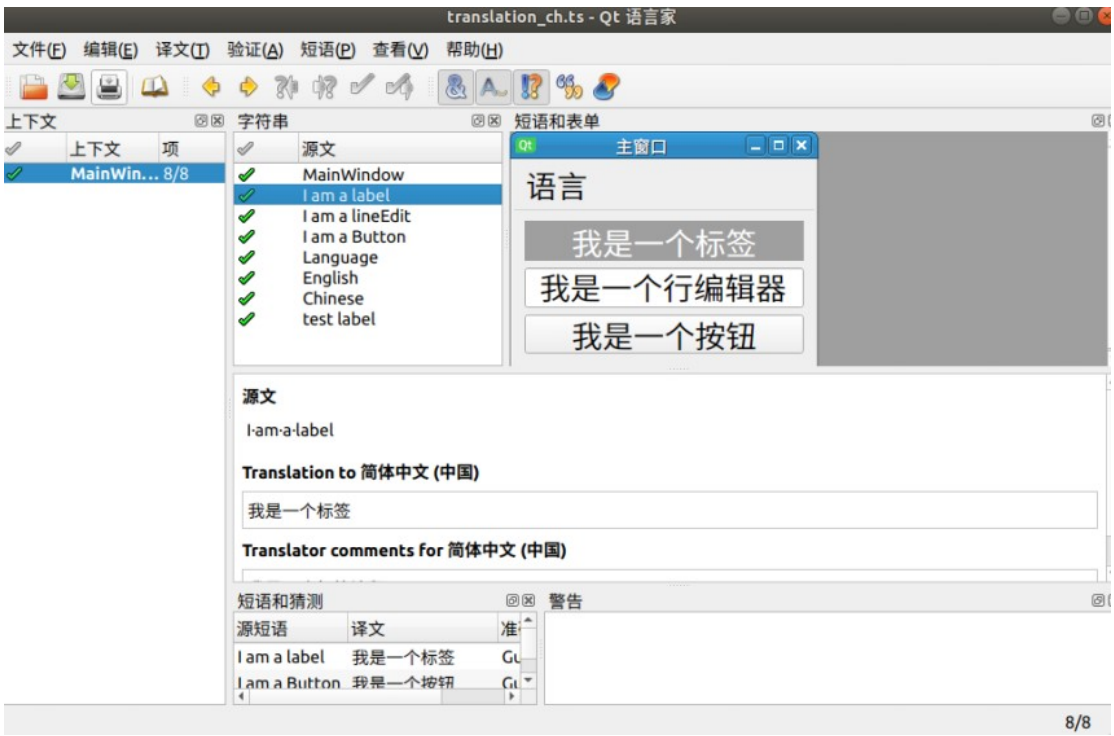


- 第三步：使用 Qt 语言家工具(Qt Linguist)翻译所有需要显示的字符串
  - 在工程目录下执行“linguist”命令，打开.ts 文件

```
Translator$ ls
main.cpp         mainwindow.ui      translation_ch.ts
mainwindow.cpp   resource.qrc       Translator.pro
mainwindow.h     translation_ch.qm  Translator.pro.user
Translator$ linguist translation_ch.ts
Translator$
```

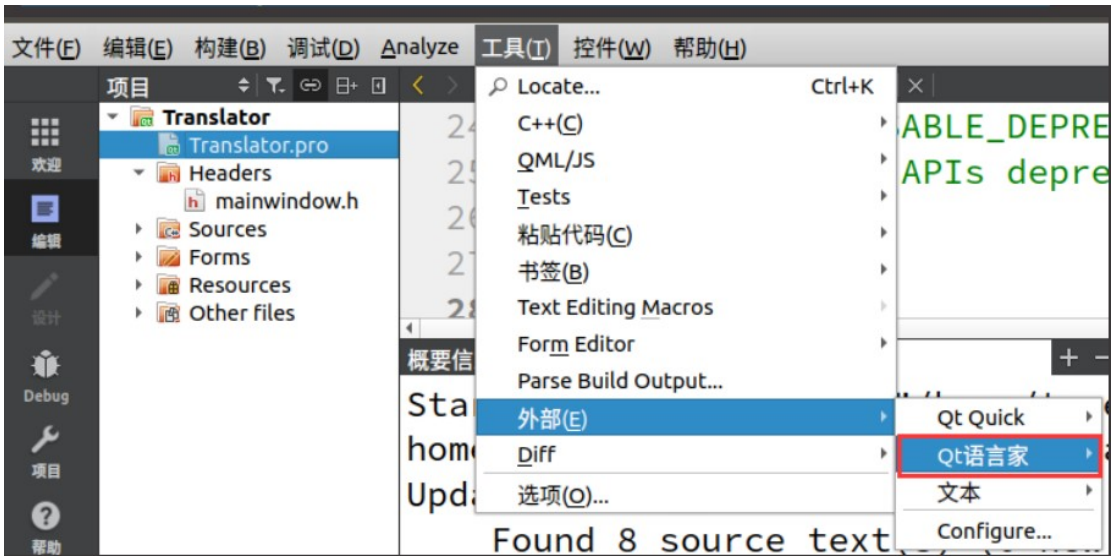
注：也可以将“.ts”文件添加到工程中，在 qtcreator 环境中选择使用“Qt Linguist”打开.ts 文件

- 在 Qt 语言家界面如下所示，翻译之后要保存

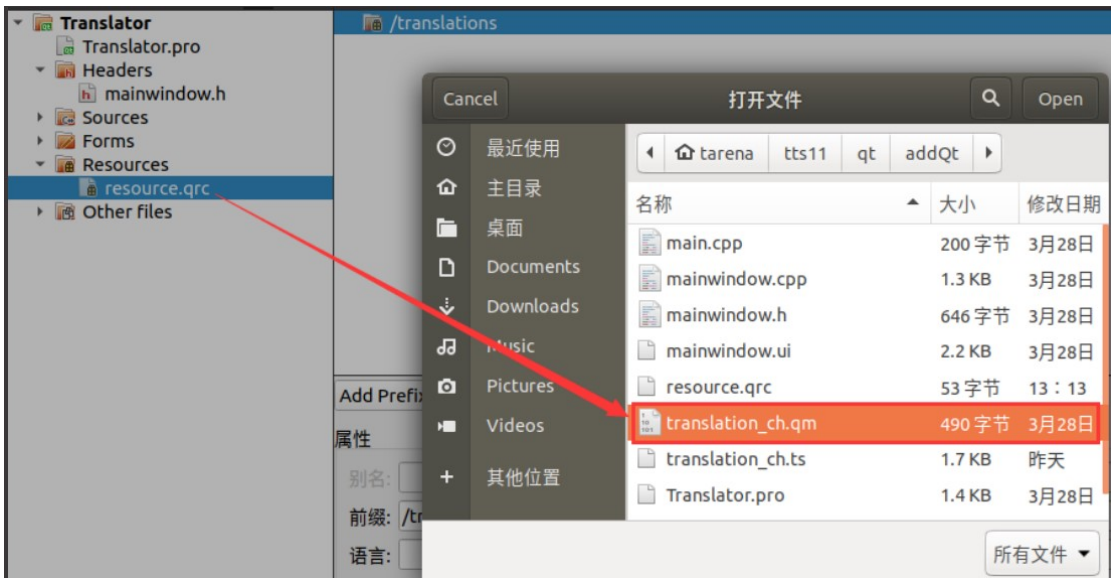


- 第四步：创建或更新“.qm”文件
  - 在 qtcreator 环境中，在菜单栏位置，依次执行：

– 工具(Tools)->外部(External)->Qt 语言家(Linguist)->发布翻译(lrelease)
知
注：如果“.qm”不存在将会自动创建，如果存在则更新



• 第五步：向工程中添加资源文件，并“.qm”将文件添加到资源文件中



• 第六步：编程使用翻译文件
– 创建翻译对象
QTranslator *chineseTranslator = new QTranslator(this);
– 从资源文件中加载翻译文件
chineseTranslator->load(":/translations/translation_ch.qm");
– 安装翻译器
qApp->installTranslator(chineseTranslator);
– 卸载翻译器
qApp->removeTranslator(chineseTranslator);

• 总结 Qt 语言家使用过程
– 1）在工程中添加“.ts”文件

– 2）使用 lupdate 创建或更新“.ts”文件
– 3）使用 linguist 翻译所有需要显示的字符串
– 4）使用 lrelease 创建或更新“.qm”文件
– 5）将“.qm”文件添加资源文件中
– 6）编程使用翻译文件(QTranslator)

在工程文件 Translator.pro 中添加 TRANSLATIONS = translation_ch.ts

vi mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTranslator>
#include <QDebug>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
        Q_OBJECT

public:
        explicit MainWindow(QWidget *parent = 0);
        ~MainWindow();

private slots:
        //切换为英文
        void on_actionEnglish_triggered();
        //切换为中文
```

```
24          void on_actionChinese_triggered();
25
26 private:
27          Ui::MainWindow *ui;
28          QTranslator* chineseTranslator;//中文翻译器
29 };
30
31 #endif // MAINWINDOW_H
```

vi mainwindow.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3
 4 MainWindow::MainWindow(QWidget *parent) :
 5          QMainWindow(parent),
 6          ui(new Ui::MainWindow)
 7 {
 8          ui->setupUi(this);
 9          //创建中文翻译器
10          chineseTranslator = new QTranslator(this);
11          //加载翻译文件(.qm)
12          chineseTranslator->load(":/translations/translation_ch.qm");
13 }
14
15 MainWindow::~MainWindow()
16 {
17          delete ui;
18 }
19 //切换为英文
20 void MainWindow::on_actionEnglish_triggered()
21 {
22          //卸载中文翻译器(恢复默认为英文)
23          qApp->removeTranslator(chineseTranslator);
24          //重新翻译界面中文本字符串
25          ui->retranslateUi(this);
26          qDebug() << tr("test");
27 }
28 //切换为中文
29 void MainWindow::on_actionChinese_triggered()
30 {
31          //为应用安装中文翻译器
32          qApp->installTranslator(chineseTranslator);
33          //重新翻译界面中文本字符串
34          ui->retranslateUi(this);
```

```
35      //tr():根据 qm 文件,翻译参数字符串
36      qDebug() << tr("test");
37 }
```

十七、Qt 插件

什么是插件

• 插件(Plugin)是一种遵循一定规范的应用程序接口编写出来的程序，主要为了扩展应用程序功能，通常一个插件表现为一个动态库（Linux 中是*.so，Windows 中是*.dll）
• 应用程序可以在运行时加载和使用插件，扩展功能只需增加插件，一个应用程序可以包含很多插件，每个都可以独立的编译和维护。

插件和动态库

• Qt 插件文件本质就是动态库(.so)，但是相比一般的库文件，Qt 自定义了一组专用的声明接口，用于从动态库中导出，以便 Qt 的插件管理体系查找和调用
• 定义 Qt 插件是必须包括以下宏，否则 Qt 插件是无法被加载和使用的
– Q_PLUGIN_METADATA(IID PLUGININTERFACE_IID)
– Q_INTERFACES(PluginInterface)
• 当选择 Qt 插件项目模板时， Qt 系统会自动插入专用的插件接口相关的代码，可以事先约定好需要导出什么函数或什么类，并为其定义了一套完整的编程规范；而如果实现一个普通的动态库，要想调用，总是需要先找到库中的函数，再导出才能使用

如何创建插件

• 第一步：定义插件接口类(纯抽象类)

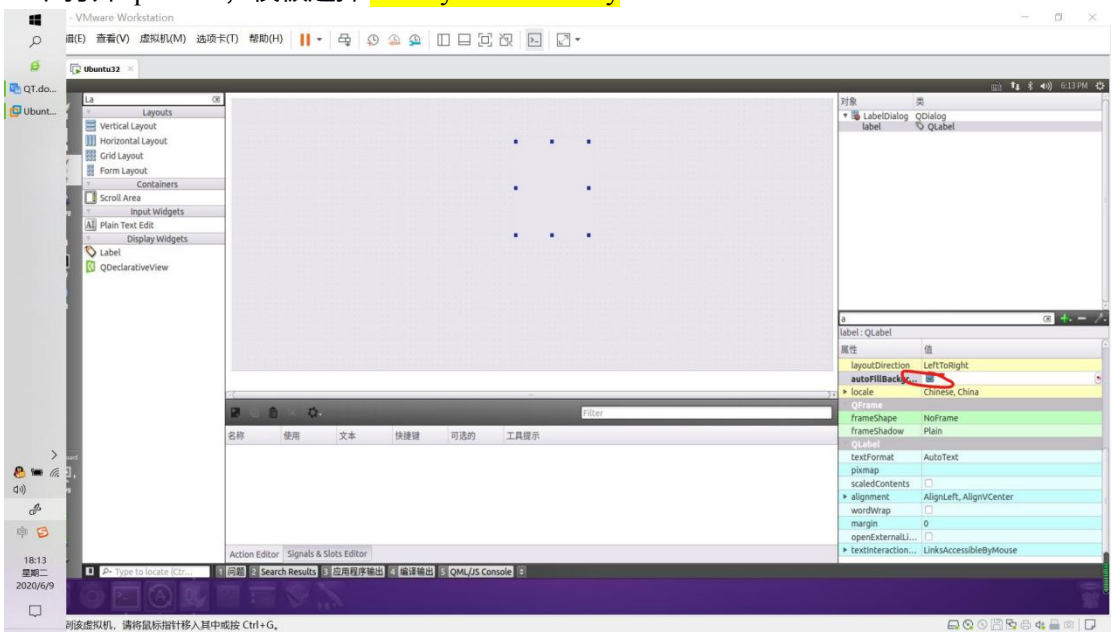
```
//plugininterface.h
class PluginInterface{
public:
virtual ~PluginInterface(){}
virtual QString description() = 0;
virtual double calculate(double left,double right)=0;
};
#define PLUGININTERFACE_IID "calculator.pluginInterface“
//将该类定义为接口, PLUGININTERFACE_IID 是与该接口对应的唯一字符串
Q_DECLARE_INTERFACE(PluginInterface,PLUGININTERFACE_IID)

cd plugin
vi PluginInterface.h
  1 #ifndef __PLUGININTERFACE_H
  2 #define __PLUGININTERFACE_H
  3
  4 #include <QObject>
```

```
5 #include <QString>
6 //插件接口(纯抽象类)
7 class PluginInterface{
8 public:
9         virtual ~PluginInterface(){}
10        //描述插件信息
11        virtual QString description() = 0;
12        //计算
13        virtual double calculate(double left,double right) = 0;
14 };
15 #define PLUGININTERFACE_IID "calculator.pluginInterface"
16 //将 PluginInterface 声明为 Qt 中插件接口
17 //参数:
18 //1)接口类的名字
19 //2)接口的标识 ID,唯一的字符串形式 ID(可以自定义)
20 Q_DECLARE_INTERFACE(PluginInterface,PLUGININTERFACE_IID)
21
22 #endif//__PLUGININTERFACE_H
```

- 第二步：使用接口类创建插件工程

– 1、打开 qtcreator，模板选择:Library->C++ Library



- 第二步：使用接口类创建插件工程

– 2、在工程配置中，使用 CONFIG 指明是一个插件

CONFIG += plugin

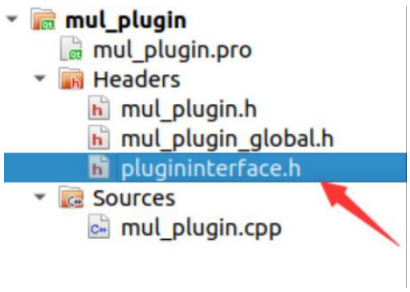
```
QT -= gui

TEMPLATE = lib
DEFINES += MUL_PLUGIN_LIBRARY

CONFIG += c++11 plugin
```

- 第二步：使用接口类创建插件

– 3、将接口文件 plugininterface.h 拷贝工程到工程目录下，并将其添加到插件工程中



- 第二步：使用接口类创建插件

– 4、修改插件头文件(mul_plugin.h)添加如下代码:

```
class MUL_PLUGIN_EXPORT Mul_plugin
        :public QObject,public PluginInterface
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID PLUGININTERFACE_IID)
    Q_INTERFACES(PluginInterface)
public:
    Mul_plugin();
    ~Mul_plugin();
    QString description();
    double calculate(double left,double right);
```

vi add_plugin.h

```
1 #ifndef ADD_PLUGIN_H
2 #define ADD_PLUGIN_H
3
4 #include "add_plugin_global.h"
5 #include "pluginInterface.h"
6
7 class ADD_PLUGINSHARED_EXPORT Add_plugin
```

```
 8                :public QObject,public PluginInterface
 9 {
10         //moc
11         Q_OBJECT
12         //添加关于插件元数据,Qt 插件中应该只出现一次
13         Q_PLUGIN_METADATA(IID PLUGININTERFACE_IID)
14         //使用声明插件接口
15         Q_INTERFACES(PluginInterface)
16 public:
17         Add_plugin();
18         QString description();
19         double calculate(double left, double right);
20 };
21
22 #endif // ADD_PLUGIN_H
23 //练习:增加乘法器和除法器插件
```

vi add_plugin.cpp

```
 1 #include "add_plugin.h"
 2
 3
 4 Add_plugin::Add_plugin()
 5 {
 6 }
 7 QString Add_plugin::description(){
 8         return "这是一个加法器插件";
 9 }
10 double Add_plugin::calculate(double left, double right)
11 {
12         return left+right;
13 }
```

• 第三步：构建工程，完成插件的创建

– 构建工程，得到和平台对应的插件文件，操作系统不同，插件文件的扩展名可能所有区别，例如 windows 系统是.dll，linux 系统是.so

注：插件工程不是一个完整的应用程序，不可以直接运行

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
mul_plugin$ ls
libmul_plugin.so    moc_predefs.h         mul_plugin.o
Makefile            mul_plugin.cpp        mul_plugin.pro
moc_mul_plugin.cpp  mul_plugin_global.h   mul_plugin.pro.user
moc_mul_plugin.o    mul_plugin.h          plugininterface.h
mul_plugin$
```

Qt 插件的使用

• **在 Qt 应用中使用插件**

– 加载插件

QPluginLoader pluginLoader(“./plugins/libmul_plugin.so”);

– 实例化插件对象,并返回指向插件对象的基类指针

PluginInterface *plugin =

dynamic_cast<PluginInterface*>(pluginLoader.instance());

– 调用插件中的计算接口

QString str = plugin->description()

double res = plugin->calculate(100, 200);

注：选择不同的插件，将会得到不同的计算结果，而应用程序不需要任何修改

• **总结 Qt 插件的创建和使用步骤**

– 1）定义接口类（抽象类）

– 2）继承接口类，创建插件

– 3）使用插件，编写应用程序

– 4）增加插件，扩展应用程序

QFileInfoList fileInfoList = filtersDir.entryInfoList(QDir::Files);

fileInfoList==>QFileInfo,QFileInfo,QFileInfo,QFileInfo ...

```
for(int i = 0;i<fileInfoList.size();i++)
{
    fileInfoList.at(i);
    ...
}
    foreach (QFileInfo fileInfo,fileInfoList) {
        fileInfo
        ...
    }
```

cd PluginUser
vi mainwinow.h

```
 1 #ifndef MAINWINDOW_H
 2 #define MAINWINDOW_H
 3
 4 #include <QMainWindow>
 5 #include "pluginInterface.h"
 6 #include <QDir>
 7 #include <QFileInfoList>
 8 #include <QMessageBox>
 9 #include <QPluginLoader>
10 #define PLUGIN_SUBFOLDER "/plugins/"
11
12 namespace Ui {
13 class MainWindow;
14 }
15
16 class MainWindow : public QMainWindow
17 {
18         Q_OBJECT
19
20 public:
21         explicit MainWindow(QWidget *parent = 0);
22         ~MainWindow();
23 private:
24         //从"plugins"目录下获取所有可用的插件列表
25         void getPluginList();
26 private slots:
27         //获取插件信息的槽函数
28         void on_helpButton_clicked();
29         //获取计算结果的槽函数
30         void on_calculateButton_clicked();
31
32 private:
33         Ui::MainWindow *ui;
34 };
35
36 #endif // MAINWINDOW_H
```

vi mainwindow.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3
 4 MainWindow::MainWindow(QWidget *parent) :
```

```
 5      QMainWindow(parent),
 6      ui(new Ui::MainWindow)
 7 {
 8      ui->setupUi(this);
 9      //设置左右操作数只能输入数字形式的内容
10      ui->leftEdit->setValidator(new QDoubleValidator(this));
11      ui->rightEdit->setValidator(new QDoubleValidator(this));
12      //获取插件列表
13      getPluginList();
14 }
15
16 MainWindow::~MainWindow()
17 {
18      delete ui;
19 }
20 //从"plugins"目录下获取所有可用的插件列表
21 void MainWindow::getPluginList()
22 {
23      //创建插件所在目录对象
24      //qApp->applicationDirPath():获取应用程序所在的路径
25      //注:去掉项目模式 shadow build 选项,让工程和应用程序放在同一目录
26      QDir filtersDir(qApp->applicationDirPath()+PLUGIN_SUBFOLDER);
27      //遍历访问目录下的所有文件
28      QFileInfoList fileInfoList = filtersDir.entryInfoList(QDir::Files);
29      //foreach:遍历访问容器中的每个元素
30      foreach(QFileInfo fileInfo,fileInfoList){
31          //判断 fileInfo 是不是库文件
32          if(QLibrary::isLibrary(fileInfo.absoluteFilePath())){
33              //加载插件
34              QPluginLoader pluginLoader(fileInfo.absoluteFilePath());
35//instance:创建插件对象,如果 dynamic_cast 转换成功说明,当前是一个可用插件
36              if(dynamic_cast<PluginInterface*>(pluginLoader.instance())){
37                  //将该插件文件名显示到列表窗口上
38                  ui->pluginList->addItem(fileInfo.fileName());
39                  //卸载插件
40                  pluginLoader.unload();
41              }
42              else{
43                  QString str = QString("确保%1 是一个正确的插
件!").arg(fileInfo.fileName());
44                  QMessageBox::warning(this,"Warning",str);
45              }
46          }
47      }
```

```
48      //获取列表窗口上可用的插件个数
49      if(ui->pluginList->count() <= 0){
50          QMessageBox::critical(this,"Error","请确保在 plugins 目录下有可用的插件");
51          //设置禁用界面任何操作
52          this->setEnabled(false);
53      }
54 }
55 //获取插件信息的槽函数
56 void MainWindow::on_helpButton_clicked()
57 {
58      //currentRow():获取列表窗口的行数
59      if(ui->pluginList->currentRow() >= 0){
60          //加载列中选中插件
61          QPluginLoader pluginLoader(
62                  qApp->applicationDirPath()    //获取应用程序路径
63                  + PLUGIN_SUBFOLDER              //获取插件子目录
64                  + ui->pluginList->currentItem()->text());//获取选中插件文件名
65          PluginInterface* plugin =
66                  dynamic_cast<PluginInterface*>(pluginLoader.instance());
67          if(plugin){
68              QMessageBox::information(this,"插件信息",plugin->description());
69          }
70          else{
71              QMessageBox::warning(this,"Waring","请确保选中的插件是有效的!");
72          }
73      }
74 }
75 //获取计算结果的槽函数
76 void MainWindow::on_calculateButton_clicked()
77 {
78      //获取左右操作数,并确保是有效的数字
79      bool blOk,brOk;
80      double leftValue = ui->leftEdit->text().toDouble(&blOk);
81      double rightValue = ui->rightEdit->text().toDouble(&brOk);
82      if(blOk==false || brOk==false){
83          QMessageBox::warning(this,"Warning","请输入有效的左右操作数");
84          return;
85      }
86      if(ui->pluginList->currentRow() >= 0){
87          QPluginLoader pluginLoader(
88                  qApp->applicationDirPath()
89                  + PLUGIN_SUBFOLDER
90                  + ui->pluginList->currentItem()->text());
91          PluginInterface* plugin =
```

```
92                  dynamic_cast<PluginInterface*>(pluginLoader.instance());
93              if(plugin){
94                  //计算和显示结果
95                  double res = plugin->calculate(leftValue,rightValue);
96                  QMessageBox::information(this,"计算结果",QString::number(res));
97              }
98              else{
99                  QMessageBox::warning(this,"Warning","请确保选中插件是有效的!");
100             }
101         }
102 }
```

十八、Qt 项目：远程视频监控系统

项目开发流程

- 一、需求分析：明确系统的功能要求
- 二、概要设计：设计系统的整体架构和模块划分
- 三、详细设计：设计每个模块的组织结构和执行流程
- 四、编写代码：编程实现每个模块的具体功能
- 五、测试运行：测试功能，修改 BUG
- 六、升级维护：优化程序，增加功能

需求分析

- 使用 uvc 摄像头，实时获取视频，通过 http 协议传输视频数据，并实时显示到 web 浏览器或者 PC 客户端

– 在 ubuntu 系统架设视频服务器（mjpg-streamer），实现抓取视频图像数据

– 利用 HTTP 协议实现和视频服务器的数据交互，编写客户端程序（C++），向视频服务器发送获取视频的请求，并接收响应数据

– 分析视频服务器的响应数据格式，从返回的数据中剥离出保存一帧图像，即一张 JPEG 格式的图片，获取的图片可以用图形查看工具直接打开

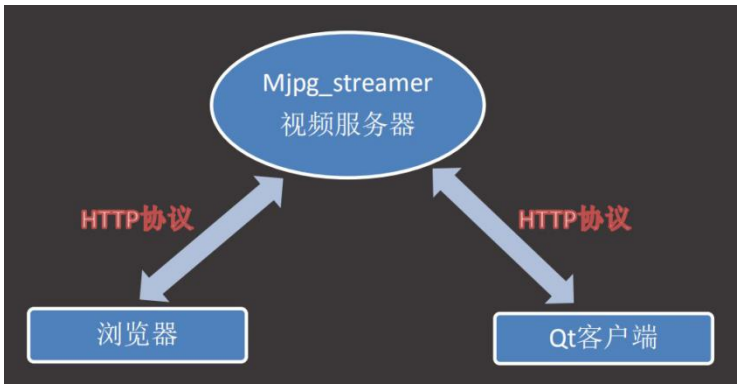
– 使用 QT 的网络编程框架，向视频服务器发送获取视频或截图请求，并通过 QT 的信号和槽机制实时接收服务返回的响应数据，从中剥离出 JPEG 图像帧并显示到 QT 界面，通过不断刷新显示 JPEG 图像帧的方式，实现视频监控的客户端

- 通过浏览器查看视频监控画面

• 通过 Qt 客户端查看视频监控画面

概要设计

• 远程视频监控系统结构框图



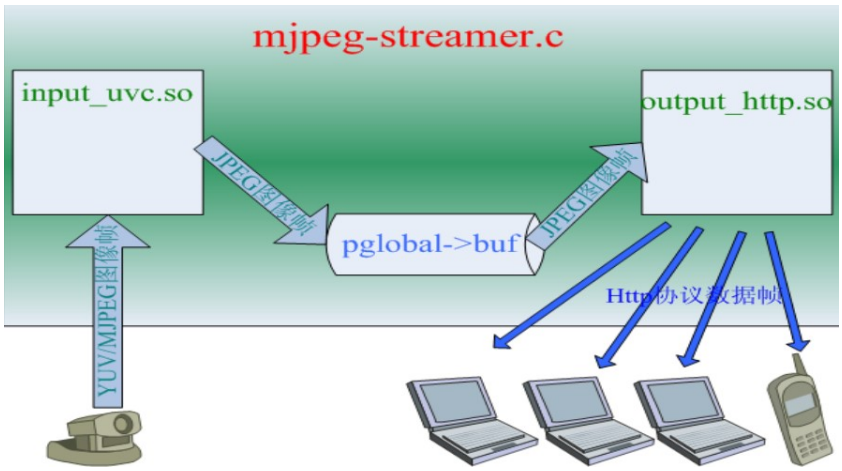
详细设计

• Mjpg-streamer 简介

– 网络视频服务器(Mjpg-streamer)，采用多线程的工作方式，可以获取 UVC 摄像头的视频数据，并利用网络将视频数据发送给客户端

– Mjpg-streamer 内置 HTTP 服务器（web server），用户可以通过浏览器直接访问服务器从而获取视频，也可以通过 HTTP 协议，编写 Qt 客户端，实现获取和显示视频流

• Mjpg-streamer 框架



• Mjpg-streamer 采用高内聚低耦合的系统架构，将输入插件和输出插件分离，编译维护

– 输入插件只能有一个，从 UVC 摄像头插件（intput_uvc.so）、测试图片插件（intput_testpicture.so）等选择一个，作为视频图像获取来源

– 输出插件可以是多个，如将视频流通过 http 输出插件（output_http.so）同时发送给多个客户端，同时也可以用输出文件插件将视频数据保存起来知

• UVC 摄像头简介

– UVC 该驱动适用于符合 USB 视频类(USB Video Class)规范的摄像头设备，它包括 V4L2 内核设备驱动和用户空间工具补丁。大多数大容量存储器设备（如优盘）都遵循 USB 规范，因而仅用一个单一驱动就可以操作它们。与此类似，UVC 兼容外设只需要一个通用驱动即可

– USB 摄像头大体上可以分为 UVC cameras 和 non-UVC cameras，UVC 是一个开放的标准，拥有维护良好的驱动，它属于内核代码的一部分。插入摄像头后就可以工作，而无须编译或安装额外的驱动

 logic c270

• HTTP 协议简介

– HTTP 是 Hyper Text Transfer Protocol（超文本转移协议）

– 是一个应用层协议，由请求和响应构成

– HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记

– http 默认采用 80 端口

• MJPG 简介

– MJPG 是 MJPEG“Motion Joint Photographic Experts Group”的缩写，即联合运动摄影专家小组，MJPEG 还可以表示文件格式扩展名，是一种视频编码格式

– MJPG 视频格式本质就是由系列 JPEG 图像帧组成的视频，所以 MJPG 每一帧数据实质就是一张 JPEG 格式的图片

• JPEG 图片

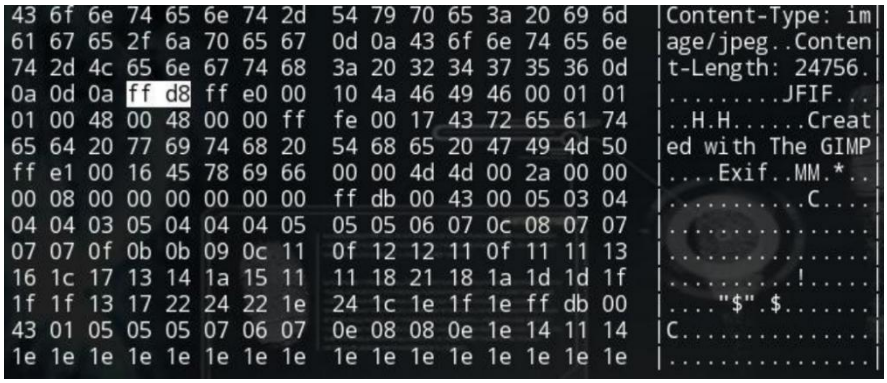
– JPEG 文件大体上有两部分组成：标记码（TAG）和压缩数据，标记码部分给出了 JPEG 图像的所有信息，如图像的 宽、高、huffman 表等

– 标记码由两个字节构成，其前一个字节是固定值 0xFF,后一个字节则根据不同意义有不同数值。一个完整的两字节的标记码后，就是标记码对应的压缩数据流，

记录了关于文件的各种信息。

• 从视频服务器获取图像帧分析

– hexdump -C /tmp/test.mjpeg | less



• Qt 视频监控客户端

– 创建 QT 应用程序与服务器建立连接

– 向服务器发送获取视频请求

"GET /?action=stream HTTP/1.1\r\n\r\n“

– 接收服务器的响应数据

– 从响应数据中剥离出一张干净的 JPEG 图像帧

– 将 JEPG 图像帧显示到 QT 界面

– 重复以上操作，不断更新显示 QT 界面的图像帧，最终看到的将是视频流

编写代码

安装 cmake 指令：sudo apt-get install cmake

mjpg-streamer 编译和配置

1) tar -xvf mjpg-streamer-experimental.tar.gz
2) cd mjpg-streamer-experimental
3) rm -r _build
4) make

5)　测试图片插件编译(单独编译)

*6) sudo make install

测试:

//运行视频服务器

./start.sh

//使用 uvc 摄像头作为插件

./mjpg_streamer -i “./input_uvc.so -n -f 10 -r 320x240 -d /dev/video0” -o”

./output_http.so -w ./www”

//使用测试图片作为输入插件

./mjpg_streamer -i ”./input_uvc.so -n -f 10 -r 320x240 -d /dev/video0” -o”

./output_http.so -w ./www” -b

在浏览器查看结果:

http://127.0.0.1:8080/stream.html

编译 mjpg-streamer 问题：

1）提示缺少“jpeglib.h”，解决方法安装 jpeg 库

sudo apt-get install libjpeg-dev

sudo apt-get update

- 在 ubuntu 搭建 mjpg 视频视频服务器
  - 执行 make，完成源代码编译
  - 使用 UVC 摄像头运行服务器

mjpg_streamer -i "./input_testpicture.so -d 500 –r 320x240" -o

"./output_http.so -w ./www -"

  - 使用测试图片运行服务器

./mjpg_streamer -i "./input_uvc.so -d /dev/video0 -f 30 –r 480x272 " -

o "./output_http.so -w www –p 80“

注意摄像头的设备节点不一定是/dev/video0，也可能是/dev/video1，要根据实际选择

- Qt 视频监控客户端编码
  - 使用 QT 网络应用开发相关类 QNetworkAccessManager、QNetworkRequest、

QNetworkReply 管理 HTTP 通信过程

  - 发送获取视频或快照请求
  - 接收响应数据并保存
  - 从响应数据中剥离图像帧
  - 在 QLabel 控件上显示图像帧或快照
  - 重复以上过程即可看到实时视频监控画面

测试运行

- Qt 视频监控客户端运行效果图

升级维护

- 未来扩展
  - 自己编写视频服务器
  - 优化界面
  - 多语言支持
  - 图像识别(opencv)
  - 多线程

.....

shell:ctag -R *

代码见 day08