QT第四天

一、定时器

所谓定时器，就是在一定时间后，系统会自动调用相应的函数，执行相应的功能，在开启定时器时，要给定超时时间，系统会根据这个时间，每隔该时间段会自动执行某些功能。该功能的实现方式有两种，分别是类对象版本，和定时器事件版本。

1.1 定时器类对象版本

```
1>  使用QTimer类，实例化一个定时器
2>  使用该类的成员函数void QTimer::start(int msec)函数，启动一个定时器
3>  一旦启动定时器后,每隔 msec毫秒后，会自动发射一个timeout的信号
4>  将timeout的信号连接到对应的槽函数中
5>  效果为：每隔msec毫秒后，系统会自动调用槽函数
6>  可以使用成员函数void QTimer::stop()，停止该定时器
7>  需要使用的头文件：
        定时器类：QTimer

举个例子：
    QTimer *timer = new QTimer(this);
      connect(timer, SIGNAL(timeout()), this, SLOT(processOneThing()));
      timer->start();
```

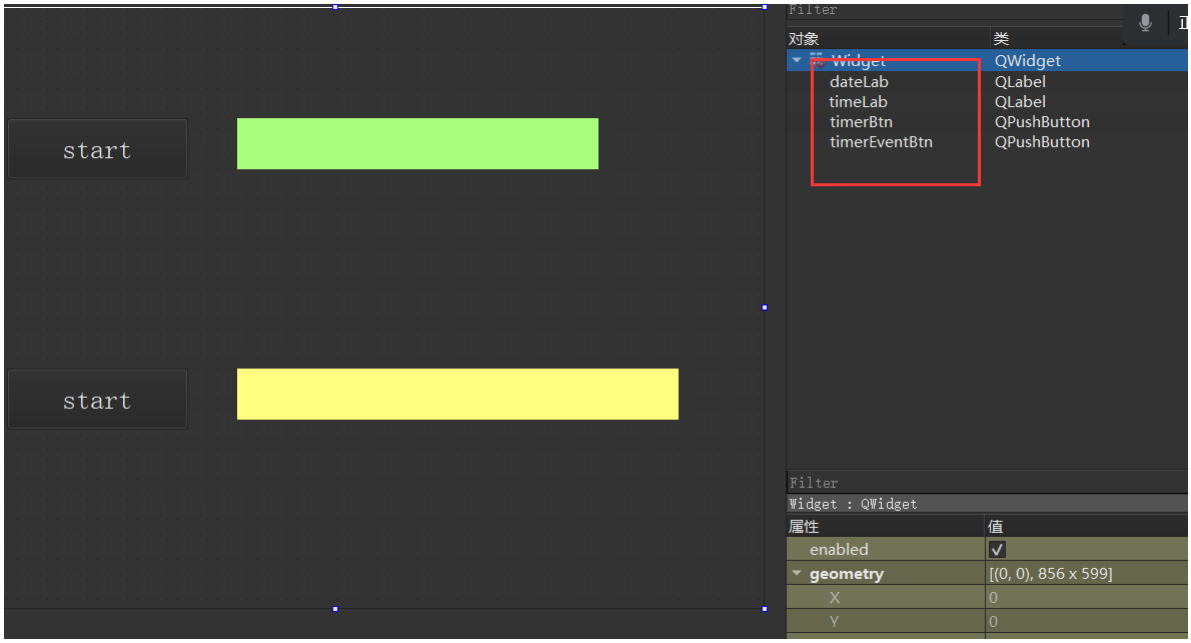
1.2 定时器事件

```
1>  需要重写基类提供的虚函数：void timerEvent(QTimerEvent *event)
2>  在程序对应的地方，调用基类提供的公共成员函数：int QObject::startTimer(int interval)
      该函数会启动一个定时器，并返回该定时器的唯一标识id
      该函数执行后，会每隔interval毫秒，系统会自动调用定时器事件处理函数timerEvent
3>  当要关闭该定时器时，调用基类提供的公共成员函数：void QObject::killTimer(int id)
    参数：要关闭的定时器的id号
4>  所需头文件：
        定时器事件类：QTimerEvent

    举个例子：
```

```
        class MyObject : public QObject
{
    Q_OBJECT

public:
    MyObject(QObject *parent = nullptr);

protected:
    void timerEvent(QTimerEvent *event) override;
};

MyObject::MyObject(QObject *parent)
    : QObject(parent)
{
    startTimer(50);     // 50-millisecond timer
    startTimer(1000);   // 1-second timer
    startTimer(60000);  // 1-minute timer

    using namespace std::chrono;
    startTimer(milliseconds(50));
    startTimer(seconds(1));
    startTimer(minutes(1));

    // since C++14 we can use std::chrono::duration literals, e.g.:
    startTimer(100ms);
    startTimer(5s);
    startTimer(2min);
    startTimer(1h);
}

void MyObject::timerEvent(QTimerEvent *event)
{
    qDebug() << "Timer ID:" << event->timerId();
}

```

案例:

1> ui界面



2> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QTimer>            //定时器类
#include<QTime>             //时间类
#include<QTimerEvent>      //定时器事件类
#include<QDateTime>            //日期类

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    //重写定时器事件处理函数
    void timerEvent(QTimerEvent *event);

private slots:
    void on_timerBtn_clicked();

    void on_timerEventBtn_clicked();

```

```
    void on_t1_timeout();            //自定义的关于timeout信号的槽函数

private:
    Ui::Widget *ui;

    QTimer *t1;              //定义定时器类对象指针，因为多个成员函数要使用

    int timerId;          //定时器事件所启动的定时器的标识

};
#endif // WIDGET_H

```

3> 源文件

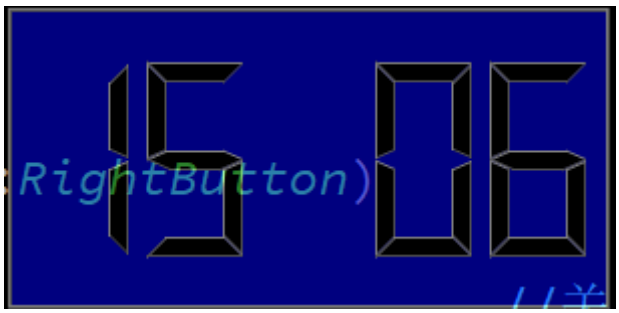
```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);

    //实例化一个定时器类对象
    t1 = new QTimer(this);

    //将t1定时器发射的信号函数与槽函数进行连接
    connect(t1, &QTimer::timeout, this, &Widget::on_t1_timeout);
}

Widget::~Widget()
{
    delete ui;
}



//类对象版按钮对应的槽函数
void Widget::on_timerBtn_clicked()
{

```

```
28
29      if(ui->timerBtn->text() == "start")
30      {
31          //启动定时器
32          t1->start(1000);
33          //该函数会启动定时器t1，每隔1秒后，会自动发射一个timeout的信号
34          //将该信号，连接到对应的槽函数中，可以去处理相关逻辑
35          //该连接只需要进行一次，所以，可以写到构造函数中去
36
37
38          ui->timerBtn->setText("stop");
39      }else if(ui->timerBtn->text() == "stop")
40      {
41          //关闭定时器
42          t1->stop();
43
44
45          ui->timerBtn->setText("start");
46      }
47  }
48
49
50  //timeout信号对应的槽函数
51  void Widget::on_t1_timeout()
52  {
53      //将当前的系统时间，展示到timeLab中
54      QTime time = QTime::currentTime();      //获取当前系统时间，并
    将其放入time对象中
55
56      //将Qtime类型转换为QString类型
57      QString time_str = time.toString("hh:mm:ss");
58
59      //将时间展示到lab中
60      ui->timeLab->setText(time_str);
61  }
62
63
64
65  //定时器事件按钮对应的槽函数
66  void Widget::on_timerEventBtn_clicked()
67  {
68      if(ui->timerEventBtn->text() == "start")
69      {
70          //启动定时器
71          timerId = startTimer(1000);
```

```
72          //功能：启动一个定时器
73          //参数：超时时间，以毫秒为单位，每隔1秒后，会自动调用定时器时事件处理函数
74          //返回值：该定时器的标识，每个定时器都有唯一的标识
75
76          ui->timerEventBtn->setText("stop");
77      }else if(ui->timerEventBtn->text() == "stop")
78      {
79          //关闭定时器
80          killTimer(timerId);
81          //功能：关闭给定的定时器
82          //参数：要关闭的定时器的标识
83
84          ui->timerEventBtn->setText("start");
85      }
86  }
87
88  //定时器事件处理函数
89  void Widget::timerEvent(QTimerEvent *event)
90  {
91      //判断是哪个定时器超时
92      if(event->timerId() == timerId)
93      {
94          //将当前的日期及时间展示到lab中
95          QDateTime data = QDateTime::currentDateTime();    //获取当前的系统日期
96          //将日期转换为字符串
97          QString data_str = data.toString("yyyy:MM:dd-hh:mm:ss");
98          //将字符串展示到lab中
99          ui->dateLab->setText(data_str);
100     }
101 }
102
103
```

二、电子钟实例

1>　ui界面



2> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QLCDNumber>
#include<QTime>           //时间类
#include<QTimerEvent>     //定时器事件类
#include<QMouseEvent>     //鼠标事件处理类

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QLCDNumber
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    void timerEvent(QTimerEvent * e);       //声明定时器事件处理函数

    void mousePressEvent(QMouseEvent *e);     //鼠标按下事件处理函数

    void mouseMoveEvent(QMouseEvent *e);    //鼠标移动事件处理函数




private:
    Ui::Widget *ui;

    //定义定时器的标志
```

```
    int timerId;

    //定义属性，用来判断是否展示冒号
    bool isOk;

    //定义左边对象，记录鼠标起始点的坐标
    QPoint startPoint;

};
#endif // WIDGET_H

```

3> 源文件

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QLCDNumber(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    this->isOk = true;            //给定是否展示初始值

    //设置固定大小
    this->setFixedSize(300,150);

    //改变背景色
    //定义一个调色板
    QPalette p = this->palette();        //获取页面自带的调色板

    //给调色板设置颜色
    p.setColor(QPalette::Window, Qt::blue);    //设置为天蓝色

    //将该调色板设置到页面中
    this->setPalette(p);

    //将页面的头部取消
    this->setWindowFlags(Qt::FramelessWindowHint);

    //设置透明度
    this->setWindowOpacity(0.5);

    //启动定时器
    timerId = startTimer(1000);
```

```
    //会每隔1秒后，自动调用定时器处理函数，我们将逻辑写到定时器处理函数中

}

Widget::~Widget()
{
    delete ui;
}

//定时器事件处理函数
void Widget::timerEvent(QTimerEvent * e)
{
    if(e->timerId() == timerId)
    {
        //调取系统时间，显示到页面中去
        QTime currentTime = QTime::currentTime();         //调取系统目前时间
        //将时间转换为字符串
        QString strTime = currentTime.toString("hh:mm");


        //对冒号是否展示，做出判断
        if(isOk == true)
        {
            strTime[2] = ':';
            isOk = false;              //展示完毕，将判断设为假
        }else
        {
            strTime[2] = ' ';
            isOk = true;
        }


        //将时间展示到页面中
        this->display(strTime);
    }
}


void Widget::mousePressEvent(QMouseEvent *e)     //鼠标按下事件处理函数
{
    if(e->button() == Qt::LeftButton)
    {
        startPoint = e->pos();         //保存鼠标起始位置
```

```
    }

    if(e->button() == Qt::RightButton)
    {
        this->close();                          //关闭页面
    }
}

void Widget::mouseMoveEvent(QMouseEvent *e)  //鼠标移动事件处理函数
{
    QPoint pos = e->globalPos() - startPoint;          //计算出页面移动的起始坐标

    this->move(pos);           //将页面进行移动

}

```

三、绘制事件

```
5> 当有下列情况之一发生时，窗口部件会收到绘制事件，
   即 QWidget类的 paintEvent()虚函数会被调用
   最终调用的是子类中重写的PaintEvent()事件函数
   窗口被创建以后第一次显示出来
   窗口由隐藏状态转变为可见状态
   窗口由最小化状态转变为正常或最大化状态
   窗口超出屏幕边界的区域进入屏幕范围之内
   窗口被遮挡的区域因某种原因重新暴露出来
   窗口因尺寸大小的变化需要呈现更多的内容
   QWidget类的 update()成员函数被调用
6> 作为 QWidget类的子类，可以在对该虚函数的覆盖版本中
   实现诸如显示文本、绘制图形、濾染图像等操作
   void ShowPicsDlg::paintEvent(QPaintEvent *e)
7> QPainter类是Qt的二维图形引擎，该类具有如下功能
   绘制矢量文字
   绘制几何图形
   绘制响度映射和图像
   反走样，像素混合，渐变和矢量路径
   平移、选择、错切、缩放等线性变换
8> QPainter类通过构造函数接收绘制设备，即在什么上画
   QPainter::QPainter(QPaintDevice* device);
9> QPainter类用于渲染图像的众多成员函数之一
```

```
    void QPainter::drawImage(const QRect& rect,const QImage& image);
```

3.2 画家类

```
QPainter类 ---> 画家类
    void SimpleExampleWidget::paintEvent(QPaintEvent *)
    {
        QPainter painter(this);
        painter.setPen(Qt::blue);
        painter.setFont(QFont("Arial", 30));
        painter.drawText(rect(), Qt::AlignCenter, "Qt");
    }
```

案例:

1> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QPaintEvent>        //绘制事件
#include<QDebug>
#include<QPainter>          //画家类
#include<QPen>             //画笔类

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    //重写绘制事件处理函数
    void paintEvent(QPaintEvent *event) override;



private:
```

```
    Ui::Widget *ui;
};
#endif // WIDGET_H

```

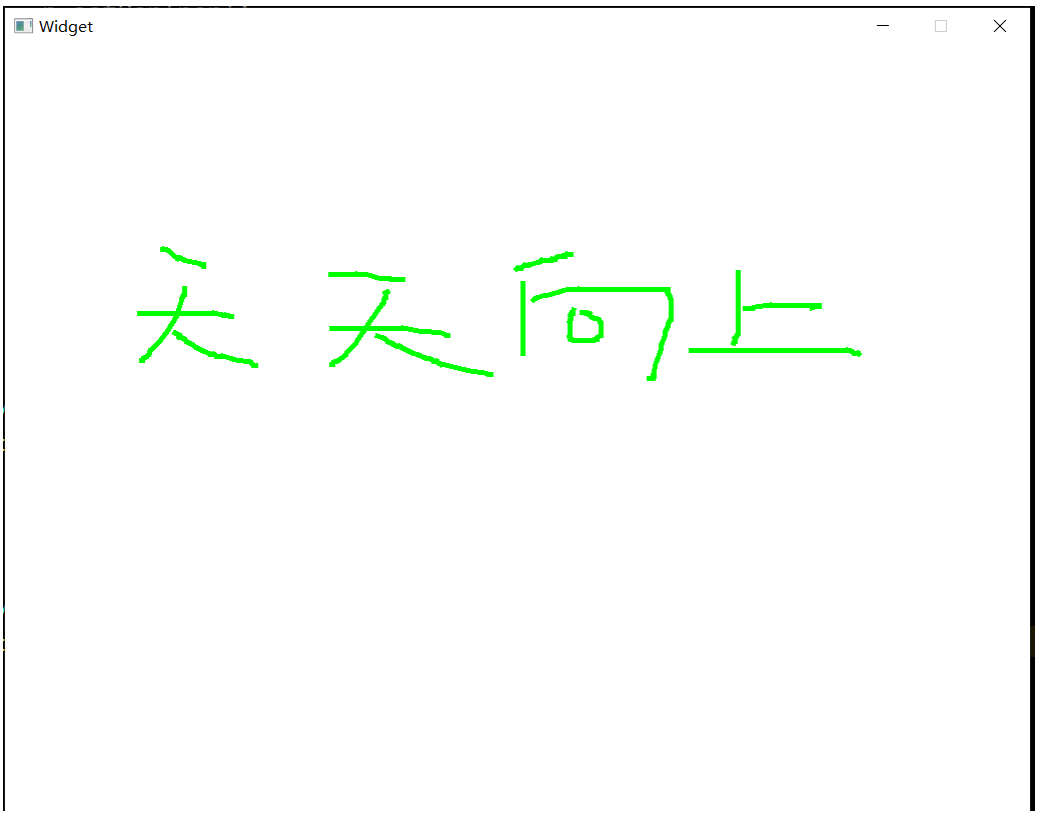
2> 源文件

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
}

Widget::~Widget()
{
    delete ui;
}

//重写的绘制事件处理函数
void Widget::paintEvent(QPaintEvent *event)
{
    //qDebug()<<"hello world";

    //实例化有个画家
    QPainter painter(this);

    //给画家设置画笔
    painter.setPen(Qt::blue);
    //设置画家的字体
    painter.setFont(QFont("隶书", 30));
    //绘制适量文字
    painter.drawText(this->rect(), Qt::AlignCenter, "华清远见");

    //想要绘制一个矩形框
    QPen pen;
    //设置画笔的风格
    pen.setStyle(Qt::DashDotDotLine);
    //设置画笔的粗细
    pen.setWidth(10);
    //设置画笔的颜色
```

```
    pen.setColor(QColor(100,100,0));

    //更新画家的画笔
    painter.setPen(pen);

    //绘制矩形框
    painter.drawRect(20,20, this->width()-40, this->height()-40);


}

```

四、画图板

1> 效果图



2> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QPaintEvent>
#include<QPainter>
#include<QMouseEvent>
#include<QPixmap>
```

```
9
10 QT_BEGIN_NAMESPACE
11 namespace Ui { class Widget; }
12 QT_END_NAMESPACE
13
14 class Widget : public QWidget
15 {
16     Q_OBJECT
17
18 public:
19     Widget(QWidget *parent = nullptr);
20     ~Widget();
21
22     //鼠标移动事件
23     void mouseMoveEvent(QMouseEvent *event) override;
24     //鼠标按压事件
25     void mousePressEvent(QMouseEvent *event) override;
26     //绘制事件
27     void paintEvent(QPaintEvent *event) override;
28
29 private:
30     Ui::Widget *ui;
31
32     QPixmap *pixmap;          //绘画容器
33
34     QPoint startPoint;         //鼠标起始点坐标
35 };
36 #endif // WIDGET_H
37
```

3> 源文件

```
1 #include "widget.h"
2 #include "ui_widget.h"
3
4 Widget::Widget(QWidget *parent)
5     : QWidget(parent)
6     , ui(new Ui::Widget)
7 {
8     ui->setupUi(this);
9
10     //设置页面大小
11     this->setFixedSize(1024,768);
12
13     //给绘图容器申请空间
```

```
    pixmap = new QPixmap(this->size());      //将该容器的大小与页面一致

    //设置填充颜色
    pixmap->fill(Qt::white);         //用白色填充
}

Widget::~Widget()
{
    delete ui;
}

//鼠标移动事件
void Widget::mouseMoveEvent(QMouseEvent *event)
{
    //实例化一个画家类，以pixmap作为父组件
    QPainter p(pixmap);


    //定义画笔
    QPen pen;
    pen.setWidth(5);
    pen.setColor(QColor(0,255,0));

    //给画家设置画笔
    p.setPen(pen);

    //从起始点到鼠标所在位置划线
    p.drawLine(startPoint, event->pos());

    //更新起始点坐标
    startPoint = event->pos();

    //调用绘制事件
    update();

}
//鼠标按压事件
void Widget::mousePressEvent(QMouseEvent *event)
{
    startPoint = event->pos();         //给起始点初始值

}
//绘制事件
void Widget::paintEvent(QPaintEvent *event)
```

```
{
    //定义一个画家类，用来将pixmap绘制到窗口上
    QPainter painter(this);

    //将绘制好的pixmap画到窗口上
    painter.drawPixmap(0,0, *pixmap);

}

```

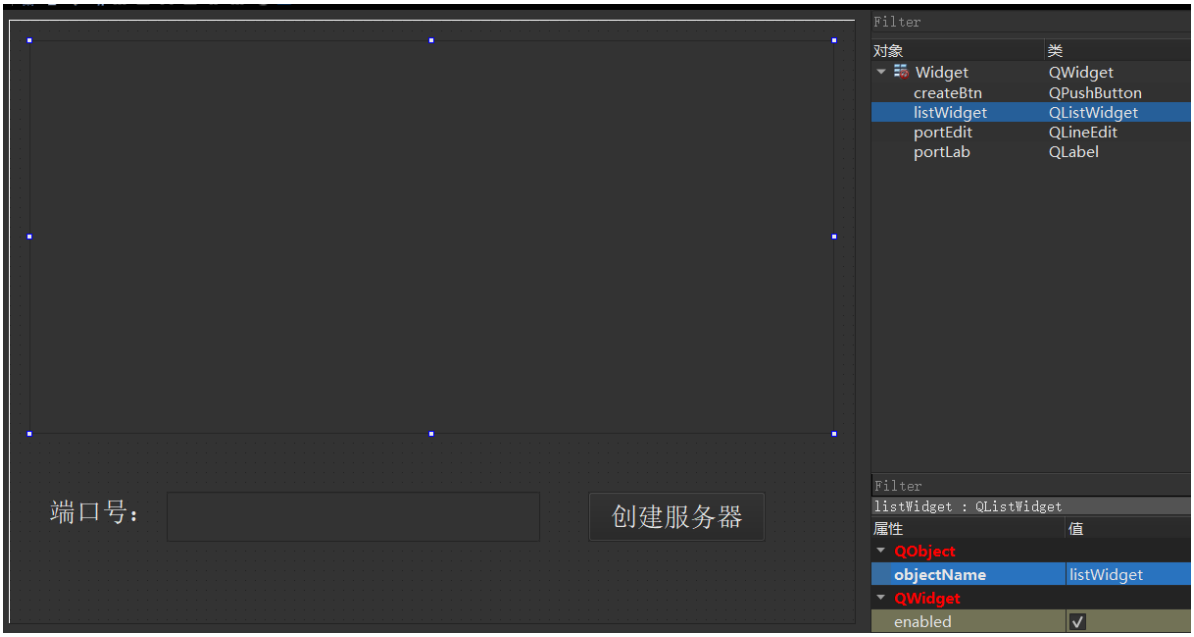
五、基于tcp的网络聊天室

服务器端原理图

客户端原理图

5.1 服务器端

1>　ui界面



2> 配置文件

```
QT       += core gui network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    widget.cpp

HEADERS += \
```

```
    widget.h

FORMS += \
    widget.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

```

3> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QTcpServer>         //服务器类
#include<QTcpSocket>          //客户端类
#include<QDebug>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    void sendToAll(QByteArray msg);        //声明广播函数

private slots:
    void on_createBtn_clicked();

    void on_tcpServer_newConnection();          //newConnection信号对应的槽函数

    void on_socket_readyRead();                 //readyRead信号对应的槽函数

private:
```

```
    Ui::Widget *ui;

    //定义服务器的指针，用来指向服务器
    QTcpServer *tcpServer;

    //定义链表存储连接过来的客户端
    QList<QTcpSocket *> tcpSockets;


};
#endif // WIDGET_H

```

4> 源文件

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);

    //创建出服务器对象
    tcpServer = new QTcpServer(this);
}

Widget::~Widget()
{
    delete ui;
}

//定义广播函数，将获取到的消息发送给每个客户端
void Widget::sendToAll(QByteArray msg)
{
    for(int i=0; i<tcpSockets.length(); i++)
    {
        tcpSockets.at(i)->write(msg);
        //将消息写到套接字中，功能是将服务器消息发送给客户端
    }

}

```

```
//创建服务器按钮对应的槽函数
void Widget::on_createBtn_clicked()
{
    quint16 port = ui->portEdit->text().toUInt();
    //从ui界面中，获取服务器的端口号
    //touint是QString类中的成员函数，功能是将字符串转换为无符号整数

    //将服务器设置为监听状态
    if(tcpServer->listen(QHostAddress::Any, port))
    {
        //功能：将服务器设置为监听状态
        //参数1：允许任何主机连接该服务器
        //参数2：提供的端口号，如果不设置，默认为0，允许任意端口号进行访问
        //返回值：成功返回true，失败返回false
        qDebug()<<"创建服务器成功";

        //将断开输入框和创建服务器按钮设为不可用
        ui->portEdit->setEnabled(false);
        ui->createBtn->setEnabled(false);
    }else
    {
        qDebug()<<"创建服务器失败";
    }

    //此时服务器一直处于监听状态，如果有客户端连接进来
    //该服务器就会发射一个newConnection信号，我们将该信号连接到对应的槽函数中
    //在槽函数中处理相应逻辑
    connect(tcpServer, &QTcpServer::newConnection,
            this, &Widget::on_tcpServer_newConnection);
    //功能：将服务器发射的newConnection与自定义的槽函数连接
    //参数1：信号发射者
    //参数2：发射的信号
    //参数3：信号接收者
    //参数4：槽函数
}

//处理newConnection信号的槽函数的定义
void Widget::on_tcpServer_newConnection()
{
    //获取最新连接的客户端
    QTcpSocket *socket = tcpServer->nextPendingConnection();
    //QTcpSocket *nextPendingConnection():获取最新连接的客户端socket
```

```
    //参数：无
    //返回值：新连接进来的客户端的套接字

    //将该客户端放入客户端链表中
    tcpSockets.push_back(socket);
    //链表操作，尾插法

    //此时，成功将客户端和服务器建立起联系了
    //自此以后，就可以进行数据的收发了
    //当客户端有消息发送过来时，就会自动触发一个readyRead的信号
    //将该信号连接到对应的槽函数中，就可以进行数据的读写了
    connect(socket, &QTcpSocket::readyRead, this,
&Widget::on_socket_readyRead);
    //参数1：客户端套接字作为信号发射者
    //参数2：要发射的是客户端中的readyRead信号
    //参数3：该页面接受信号
    //参数4：处理readyRead信号的自定义槽函数
}

//处理readyRead信号的槽函数
void Widget::on_socket_readyRead()
{
    //移除无效的客户端，该客户端已经断开连接了，但是，客户端链表中还存放该
客户端信息
    for(int i=0; i<tcpSockets.size(); i++)
    {
        //判断第i个客户端是否有效，如果无效，则进行删除
        if(tcpSockets.at(i)->state() == false)
        {
            //state()：判断客户端的状态，如果可以则是该客户端，不可用
为false
            tcpSockets.removeAt(i);
            //功能：从链表中移除第i个元素
            //参数：链表的下标，从0开始
            //返回值：无
        }
    }

    //去判断到低是哪个客户端发送来的消息
    for(int i=0; i<tcpSockets.size(); i++)
    {
        //判断该客户端中是否有数据
        if(tcpSockets.at(i)->bytesAvailable())
        {
            //qint64 bytesAvailable():判断客户端中是否有数据
```

```
                //参数：无
                //返回值：客户端中数据的大小，如果为0，则说明，该客户端没有数据可以被读取

                //读取该客户端中的数据，可以用客户端的成员函数，readAll函数读取数据
                QByteArray msg = tcpSockets.at(i)->readAll();
                //功能：读取套接字中所有的数据
                //参数：无
                //返回值：QByteArray字节数组

                //将该数据展示到页面中去
                ui->listWidget->addItem(QString::fromLocal8Bit(msg));
                //页面要展示数据，所需要的是QString类型的数据
                //但是，readAll()函数返回的是QByteArray类型的数据
                //所以，需要调用QString的成员函数，来将字节数组转换为字符串

                //将该消息广播给所有客户端
                sendToAll(msg);

            }
        }

}

```

5> ui文件

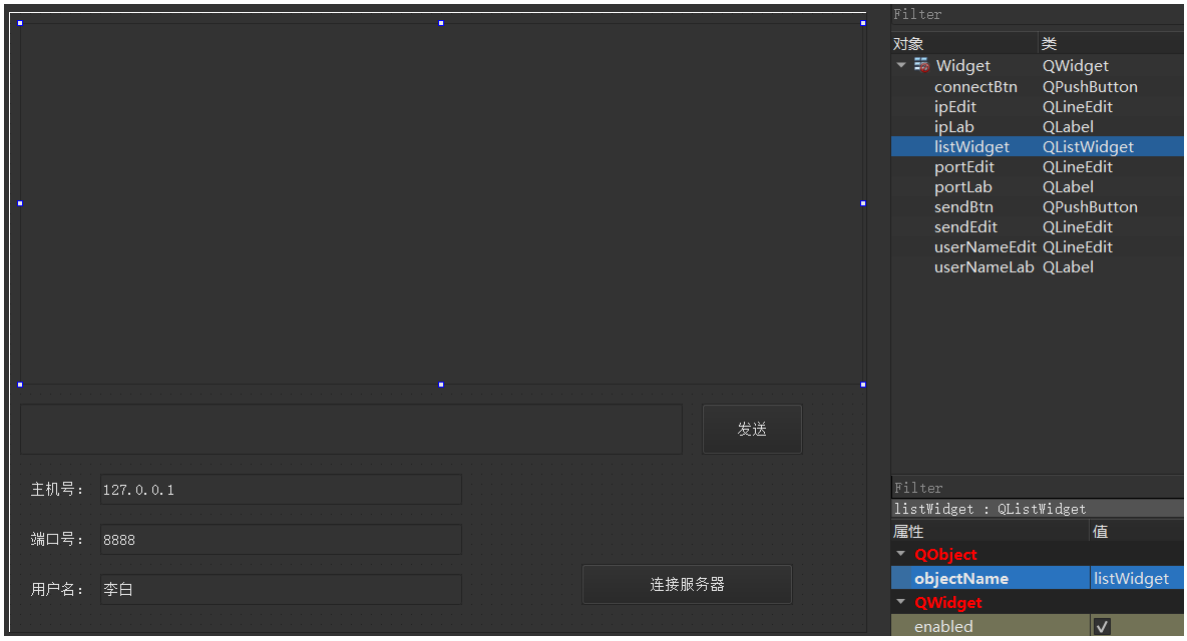
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Widget</class>
 <widget class="QWidget" name="Widget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>862</width>
    <height>614</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <pointsize>16</pointsize>
```

```
 </font>
</property>
<property name="windowTitle">
 <string>Widget</string>
</property>
<widget class="QListWidget" name="listWidget">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>20</y>
   <width>821</width>
   <height>401</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="portLab">
 <property name="geometry">
  <rect>
   <x>40</x>
   <y>470</y>
   <width>101</width>
   <height>61</height>
  </rect>
 </property>
 <property name="text">
  <string>端口号：</string>
 </property>
</widget>
<widget class="QLineEdit" name="portEdit">
 <property name="geometry">
  <rect>
   <x>160</x>
   <y>480</y>
   <width>381</width>
   <height>51</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="createBtn">
 <property name="geometry">
  <rect>
   <x>590</x>
   <y>480</y>
   <width>181</width>
   <height>51</height>
```

```
    </rect>
   </property>
   <property name="text">
    <string>创建服务器</string>
   </property>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>

```

5.2 客户端

1> ui界面



2> 配置文件

```
QT       += core gui network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the

```

```
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    widget.cpp

HEADERS += \
    widget.h

FORMS += \
    widget.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

```

3> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QTcpSocket>
#include<QMessageBox>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

```

```
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

private slots:
    void on_connectBtn_clicked();

    void on_tcpSocket_connected();          //处理connected信号的槽函数

    void on_tcpSocket_readyRead();         //处理readyRead信号的槽函数

    void on_sendBtn_clicked();

    void on_tcpSocket_disconnected();        //处理disconnected信号的槽函数

private:
    Ui::Widget *ui;

    //定义一个客户端指针
    QTcpSocket *tcpSocket;

    //定义变量接受用户名
    QString userName;

};
#endif // WIDGET_H

```

4> 源文件

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);

    //初始时，发送信息的编辑框和按钮不可用
    ui->sendBtn->setEnabled(false);
    ui->sendEdit->setEnabled(false);
```

```
13
14      //实例化一个客户端
15      tcpSocket = new QTcpSocket(this);
16
17      //如果客户端连接服务器成功的话，会自动发射一个connected的信号函数
18      //可以在该信号函数中处理相关逻辑
19      connect(tcpSocket, &QTcpSocket::connected, this,
    &Widget::on_tcpSocket_connected);
20      //参数1：该客户端作为信号发射者
21      //参数2：要发射的是connected信号
22      //参数3：该页面接受信号
23      //参数4：处理该信号的槽函数
24
25
26
27      //在客户端与服务器建立起连接后
28      //如果客户端收到服务器发送来的数据，就会自动触发一个readyRead信号
29      //我们在该信号对应的槽函数中，可以读取服务器发送过来的数据，以及处理相
    关逻辑
30      connect(tcpSocket, &QTcpSocket::readyRead, this,
    &Widget::on_tcpSocket_readyRead);
31      //参数1：该客户端
32      //参数2：收到服务器消息后，发射的信号
33      //参数3：该页面接受到信号
34      //参数4：处理相关逻辑的槽函数
35
36
37      //当客户端与服务器断开连接后，会触发一个disconnected的信号，
38      //我们可以在该信号对应的槽函数中处理相关逻辑
39      connect(tcpSocket, &QTcpSocket::disconnected, this,
    &Widget::on_tcpSocket_disconnected);
40  }
41
42  Widget::~Widget()
43  {
44      delete ui;
45
46
47  }
48
49  //连接服务器按钮对应的槽函数
50  void Widget::on_connectBtn_clicked()
51  {
52      if(ui->connectBtn->text() == "连接服务器")
53      {
```

```
        //编写连接服务器的逻辑
        QString ip = ui->ipEdit->text();    //获取界面ip地址
        quint16 port = ui->portEdit->text().toUInt();    //获取页面上的端口号

        //连接到服务器
        tcpSocket->connectToHost(ip, port, QTcpSocket::ReadWrite, QTcpSocket::AnyIPProtocol);
        //功能：将该客户端连接到服务器
        //参数1：要连接的主机ip地址
        //参数2：要连接的端口号
        //参数3：连接模式：读写都可以
        //参数4：可以连接到任意ip
        //如果连接服务器成功，就会触发一个connected的信号
        //将该信号与对应的槽函数进行连接，只需连接一次，所以写在构造函数中了



        ui->connectBtn->setText("断开服务器");
    }else if(ui->connectBtn->text() == "断开服务器")
    {
        userName = ui->userNameEdit->text();       //获取用户名

        QString msg = userName +": 离开聊天室";
        tcpSocket->write(msg.toLocal8Bit());

        //编写断开服务器的逻辑,调用成员函数disconnectFromHost函数完成断开连接
        tcpSocket->disconnectFromHost();
        //功能：断开客户端的连接
        //参数：无
        //返回值：无



        ui->connectBtn->setText("连接服务器");
    }

}

//connected信号对应的槽函数的定义
void Widget::on_tcpSocket_connected()
{
```

```
95      //qDebug()<<"连接服务器成功";
96      QMessageBox::information(this,"连接服务器", "恭喜你连接成
    功! ");
97
98      //将发送输入框和按钮设为可用状态
99      ui->sendBtn->setEnabled(true);
100     ui->sendEdit->setEnabled(true);
101
102     //将ip、端口号、用户名的输入框设为不可用
103     ui->ipEdit->setEnabled(false);
104     ui->portEdit->setEnabled(false);
105     ui->userNameEdit->setEnabled(false);
106
107     //获取用户名
108     userName = ui->userNameEdit->text();
109
110     QString msg = userName +": 加入群聊";
111
112     //将消息发送给服务器
113     tcpSocket->write(msg.toLocal8Bit());
114
115 }
116
117 //readyRead信号对应的槽函数的定义
118 void Widget::on_tcpSocket_readyRead()
119 {
120     QByteArray msg =  tcpSocket->readAll();
121     //功能：读取套接字中的数据
122
123     //将消息展示到自己的页面上
124     ui->listWidget->addItem(QString::fromLocal8Bit(msg));
125
126 }
127
128
129 //发送按钮对应的槽函数
130 void Widget::on_sendBtn_clicked()
131 {
132     userName = ui->userNameEdit->text();
133
134     //获取界面上编辑的消息
135     QString msg = userName + ": " + ui->sendEdit->text();
136
137     //将该消息发送给服务器
138     tcpSocket->write(msg.toLocal8Bit());
```

```
139
140     //将编辑信息框中的消息清除
141     ui->sendEdit->clear();
142 }
143
144 //disconnected信号对应的槽函数定义
145 void Widget::on_tcpSocket_disconnected()
146 {
147     //将发送信息的编辑框和按钮禁用
148     ui->sendBtn->setEnabled(false);
149     ui->sendEdit->setEnabled(false);
150
151     //将用户信息设为可用状态
152     ui->ipEdit->setEnabled(true);
153     ui->portEdit->setEnabled(true);
154     ui->userNameEdit->setEnabled(true);
155
156 }
157
```

5>　ui文件

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>Widget</class>
4   <widget class="QWidget" name="Widget">
5    <property name="geometry">
6     <rect>
7      <x>0</x>
8      <y>0</y>
9      <width>854</width>
10     <height>618</height>
11    </rect>
12   </property>
13   <property name="windowTitle">
14    <string>Widget</string>
15   </property>
16   <widget class="QListWidget" name="listWidget">
17    <property name="geometry">
18     <rect>
19      <x>10</x>
20      <y>10</y>
21      <width>841</width>
22      <height>361</height>
23     </rect>
```

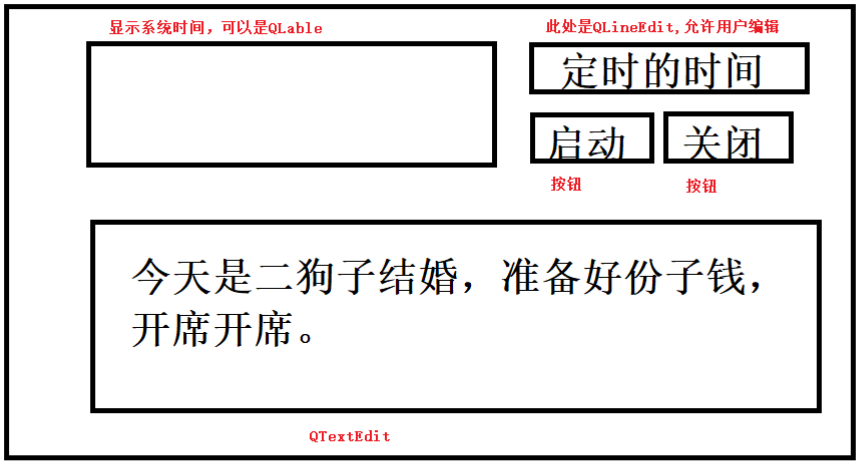
```
 </property>
</widget>
<widget class="QLineEdit" name="sendEdit">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>390</y>
   <width>661</width>
   <height>51</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="sendBtn">
 <property name="geometry">
  <rect>
   <x>690</x>
   <y>390</y>
   <width>101</width>
   <height>51</height>
  </rect>
 </property>
 <property name="text">
  <string>发送</string>
 </property>
</widget>
<widget class="QLabel" name="ipLab">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>460</y>
   <width>81</width>
   <height>31</height>
  </rect>
 </property>
 <property name="text">
  <string>主机号：</string>
 </property>
</widget>
<widget class="QLineEdit" name="ipEdit">
 <property name="geometry">
  <rect>
   <x>90</x>
   <y>460</y>
   <width>361</width>
   <height>31</height>
```

```
    </rect>
   </property>
   <property name="text">
    <string>127.0.0.1</string>
   </property>
  </widget>
  <widget class="QLineEdit" name="portEdit">
   <property name="geometry">
    <rect>
     <x>90</x>
     <y>510</y>
     <width>361</width>
     <height>31</height>
    </rect>
   </property>
   <property name="text">
    <string>8888</string>
   </property>
  </widget>
  <widget class="QLabel" name="portLab">
   <property name="geometry">
    <rect>
     <x>20</x>
     <y>510</y>
     <width>81</width>
     <height>31</height>
    </rect>
   </property>
   <property name="text">
    <string>端口号：</string>
   </property>
  </widget>
  <widget class="QLabel" name="userNameLab">
   <property name="geometry">
    <rect>
     <x>20</x>
     <y>560</y>
     <width>81</width>
     <height>31</height>
    </rect>
   </property>
   <property name="text">
    <string>用户名：</string>
   </property>
  </widget>
```

```
  <widget class="QLineEdit" name="userNameEdit">
   <property name="geometry">
    <rect>
     <x>90</x>
     <y>560</y>
     <width>361</width>
     <height>31</height>
    </rect>
   </property>
   <property name="text">
    <string>李白</string>
   </property>
  </widget>
  <widget class="QPushButton" name="connectBtn">
   <property name="geometry">
    <rect>
     <x>570</x>
     <y>550</y>
     <width>211</width>
     <height>41</height>
    </rect>
   </property>
   <property name="text">
    <string>连接服务器</string>
   </property>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>

```

作业

实现闹钟功能



要求：在没有点击启动按钮前，系统时间正常显示，关闭按钮禁用状态，设置时间和文本编辑器为可用状态

在点击启动按钮后，启动按钮和设置时间以及文本编辑器为不可用状态，关闭按钮为可用状态

当点击关闭按钮时，关闭按钮不可用，设置时间、启动按钮、文本编辑器为可用状态