# 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
   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
4
   5> 效果为:每隔msec毫秒后,系统会自动调用槽函数
6 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:
7
         定时器类: QTimer
8
9
10 举个例子:
      QTimer *timer = new QTimer(this);
11
12
        connect(timer, SIGNAL(timeout()), this,
   SLOT(processOneThing()));
13
       timer->start();
```

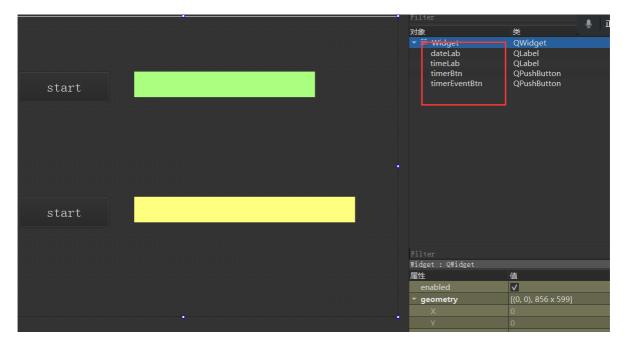
### 1.2 定时器事件

```
1 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
3
       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
4
  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
6
7
  4> 所需头文件:
8
         定时器事件类: QTimerEvent
9
      举个例子:
10
```

```
class MyObject : public QObject
11
      {
12
13
          Q_OBJECT
14
15
      public:
          MyObject(QObject *parent = nullptr);
16
17
18
      protected:
19
          void timerEvent(QTimerEvent *event) override;
20
      };
21
22
      MyObject::MyObject(QObject *parent)
          : QObject(parent)
23
      {
24
25
          startTimer(50); // 50-millisecond timer
          startTimer(1000); // 1-second timer
26
          startTimer(60000); // 1-minute timer
27
28
29
          using namespace std::chrono;
          startTimer(milliseconds(50));
30
          startTimer(seconds(1));
31
32
          startTimer(minutes(1));
33
          // since C++14 we can use std::chrono::duration
34
   literals, e.g.:
          startTimer(100ms);
35
          startTimer(5s);
36
37
          startTimer(2min);
          startTimer(1h);
38
      }
39
40
      void MyObject::timerEvent(QTimerEvent *event)
41
42
      {
43
          qDebug() << "Timer ID:" << event->timerId();
44
      }
45
```

#### 案例:

1> ui界面



```
1 #ifndef WIDGET_H
 2
   #define WIDGET_H
 3
  #include <QWidget>
 4
 5
  #include<QTimer>
                            //定时器类
  #include<QTime>
 6
                              //时间类
   #include<QTimerEvent>
 7
                            //定时器事件类
   #include<QDateTime>
                                 //日期类
 8
9
10
   QT_BEGIN_NAMESPACE
   namespace Ui { class Widget; }
11
12
   QT_END_NAMESPACE
13
   class Widget: public QWidget
14
15
   {
       Q_OBJECT
16
17
   public:
18
       widget(Qwidget *parent = nullptr);
19
20
       ~Widget();
21
22
       //重写定时器事件处理函数
23
       void timerEvent(QTimerEvent *event);
24
25
    private slots:
       void on_timerBtn_clicked();
26
27
       void on_timerEventBtn_clicked();
28
29
```

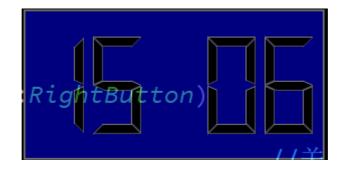
```
30 void on_t1_timeout(); //自定义的关于timeout信号的槽
   函数
31
32
  private:
33
      Ui::Widget *ui;
34
                       //定义定时器类对象指针,因为多个成员函数
35
      QTimer *t1;
  要使用
36
37
      int timerId; //定时器事件所启动的定时器的标识
38
  };
39
40
  #endif // WIDGET_H
41
```

```
1 #include "widget.h"
   #include "ui_widget.h"
 2
 3
4
   widget::Widget(QWidget *parent)
       : QWidget(parent)
 5
       , ui(new Ui::Widget)
 6
 7
   {
8
       ui->setupUi(this);
9
       //实例化一个定时器类对象
10
11
       t1 = new QTimer(this);
12
       //将t1定时器发射的信号函数与槽函数进行连接
13
14
       connect(t1, &QTimer::timeout, this,
   &Widget::on_t1_timeout);
   }
15
16
   Widget::~Widget()
17
18
19
       delete ui;
20
   }
21
22
23
24
   //类对象版按钮对应的槽函数
   void Widget::on_timerBtn_clicked()
25
26
   {
27
```

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

```
//功能:启动一个定时器
72
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
           //返回值: 该定时器的标识,每个定时器都有唯一的标识
74
75
76
           ui->timerEventBtn->setText("stop");
77
        }else if(ui->timerEventBtn->text() == "stop")
78
       {
79
           //关闭定时器
80
           killTimer(timerId);
           //功能: 关闭给定的定时器
81
           //参数:要关闭的定时器的标识
82
83
84
           ui->timerEventBtn->setText("start");
85
       }
86
    }
87
88
    //定时器事件处理函数
    void Widget::timerEvent(QTimerEvent *event)
89
90
       //判断是哪个定时器超时
91
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
           //将日期转换为字符串
96
97
           QString data_str = data.toString("yyyy:MM:dd-
    hh:mm:ss");
           //将字符串展示到lab中
98
99
           ui->dateLab->setText(data_str);
100
       }
101
102
103
```

## 二、电子钟实例



```
1 #ifndef WIDGET_H
   #define WIDGET_H
 3
4 #include <QWidget>
5 #include<QLCDNumber>
6 #include<QTime>
                          //时间类
7 #include<QTimerEvent> //定时器事件类
   #include<QMouseEvent> //鼠标事件处理类
9
   QT_BEGIN_NAMESPACE
10
11
   namespace Ui { class Widget; }
12
   QT_END_NAMESPACE
13
14
   class Widget: public QLCDNumber
   {
15
16
       Q_OBJECT
17
18
   public:
19
       widget(Qwidget *parent = nullptr);
       ~Widget();
20
21
       void timerEvent(QTimerEvent * e); //声明定时器事件处理函
22
   数
23
24
       void mousePressEvent(QMouseEvent *e); //鼠标按下事件处理函
   数
25
26
       void mouseMoveEvent(QMouseEvent *e); //鼠标移动事件处理函数
27
28
29
30
   private:
31
       Ui::Widget *ui;
32
33
34
       //定义定时器的标志
```

```
int timerId;
35
36
       //定义属性,用来判断是否展示冒号
37
       bool isOk;
38
39
       //定义左边对象,记录鼠标起始点的坐标
40
       QPoint startPoint;
41
42
43
   };
   #endif // WIDGET_H
44
45
```

```
1 #include "widget.h"
2
  #include "ui_widget.h"
 3
   widget::Widget(QWidget *parent)
4
       : QLCDNumber(parent)
 5
6
       , ui(new Ui::Widget)
7
   {
8
       ui->setupUi(this);
       this->isOk = true;
                                  //给定是否展示初始值
9
10
       //设置固定大小
11
       this->setFixedSize(300,150);
12
13
       //改变背景色
14
15
       //定义一个调色板
       QPalette p = this->palette(); //获取页面自带的调色板
16
17
18
       //给调色板设置颜色
       p.setColor(QPalette::Window, Qt::blue); //设置为天蓝色
19
20
       //将该调色板设置到页面中
21
       this->setPalette(p);
22
23
24
       //将页面的头部取消
25
       this->setWindowFlags(Qt::FramelessWindowHint);
26
27
       //设置透明度
28
       this->setWindowOpacity(0.5);
29
       //启动定时器
30
       timerId = startTimer(1000);
31
```

```
//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
33
34
   }
35
   Widget::~Widget()
36
37
       delete ui;
38
39
   }
40
   //定时器事件处理函数
41
   void Widget::timerEvent(QTimerEvent * e)
42
43
       if(e->timerId() == timerId)
44
45
       {
46
          //调取系统时间,显示到页面中去
47
          QTime currentTime = QTime::currentTime();
                                                        //调取
   系统目前时间
48
          //将时间转换为字符串
          QString strTime = currentTime.toString("hh:mm");
49
50
51
          //对冒号是否展示,做出判断
52
53
          if(is0k == true)
          {
54
              strTime[2] = ':';
55
                                    //展示完毕,将判断设为假
              isOk = false;
56
57
          }else
58
           {
59
              strTime[2] = ' ';
60
              isOk = true;
           }
61
62
63
64
          //将时间展示到页面中
          this->display(strTime);
65
       }
66
   }
67
68
69
   void Widget::mousePressEvent(QMouseEvent *e) //鼠标按下事件处
70
   理函数
71
   {
72
       if(e->button() == Qt::LeftButton)
73
       {
                                 //保存鼠标起始位置
74
           startPoint = e->pos();
```

```
}
75
76
77
       if(e->button() == Qt::RightButton)
       {
78
79
          this->close();
                                          //关闭页面
80
       }
  }
81
82
83
   void Widget::mouseMoveEvent(QMouseEvent *e) //鼠标移动事件处理函
   数
   {
84
       QPoint pos = e->globalPos() - startPoint; //计算出页
85
   面移动的起始坐标
86
87
       this->move(pos); //将页面进行移动
88
89
   }
90
```

## 三、绘制事件

```
1
   5> 当有下列情况之一发生时,窗口部件会收到绘制事件,
2
      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
3
      窗口被创建以后第一次显示出来
4
      窗口由隐藏状态转变为可见状态
5
      窗口由最小化状态转变为正常或最大化状态
6
7
      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
8
9
      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
10
     6> 作为 Qwidget类的子类,可以在对该虚函数的覆盖版本中
11
12
      实现诸如显示文本、绘制图形、瀘染图像等操作
     void ShowPicsDlg::paintEvent(QPaintEvent *e)
13
     7> OPainter类是Ot的二维图形引擎,该类具有如下功能
14
     绘制矢量文字
15
     绘制几何图形
16
     绘制响度映射和图像
17
     反走样, 像素混合, 渐变和矢量路径
18
     平移、选择、错切、缩放等线性变换
19
20
     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);
21
     9> OPainter类用于渲染图像的众多成员函数之一
22
```

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

### 3.2 画家类

```
QPainter类 ---> 画家类
1
        void SimpleExampleWidget::paintEvent(QPaintEvent *)
2
3
        {
            QPainter painter(this);
4
5
            painter.setPen(Qt::blue);
            painter.setFont(QFont("Arial", 30));
6
7
            painter.drawText(rect(), Qt::AlignCenter, "Qt");
8
        }
```

#### 案例:

```
1 #ifndef WIDGET_H
 2 #define WIDGET_H
 4 #include <QWidget>
 5 #include<QPaintEvent>
                           //绘制事件
 6 #include<QDebug>
  #include<QPainter>
 7
                           //画家类
   #include<QPen>
                           //画笔类
 8
9
   QT_BEGIN_NAMESPACE
10
   namespace Ui { class Widget; }
11
12
   QT_END_NAMESPACE
13
   class Widget: public QWidget
14
   {
15
16
       Q_OBJECT
17
   public:
18
       widget(Qwidget *parent = nullptr);
19
       ~Widget();
20
21
22
       //重写绘制事件处理函数
       void paintEvent(QPaintEvent *event) override;
23
24
25
26
   private:
```

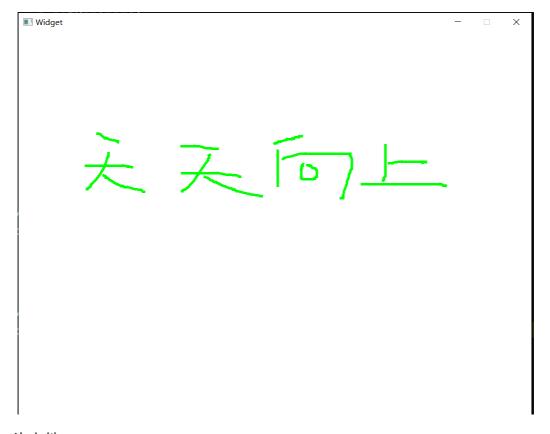
```
28 Ui::Widget *ui;
29 };
30 #endif // WIDGET_H
31
```

```
1 #include "widget.h"
 2
   #include "ui_widget.h"
 3
   Widget::Widget(QWidget *parent)
 4
 5
        : QWidget(parent)
 6
        , ui(new Ui::Widget)
 7
 8
       ui->setupUi(this);
   }
 9
10
   Widget::~Widget()
11
12
   {
13
       delete ui;
   }
14
15
   //重写的绘制事件处理函数
16
   void Widget::paintEvent(QPaintEvent *event)
17
18
   {
       //qDebug()<<"hello world";</pre>
19
20
       //实例化有个画家
21
22
       QPainter painter(this);
23
       //给画家设置画笔
24
25
       painter.setPen(Qt::blue);
26
       //设置画家的字体
27
       painter.setFont(QFont("隶书", 30));
       //绘制适量文字
28
       painter.drawText(this->rect(), Qt::AlignCenter, "华清远
29
    见");
30
31
       //想要绘制一个矩形框
32
       QPen pen;
       //设置画笔的风格
33
34
       pen.setStyle(Qt::DashDotDotLine);
       //设置画笔的粗细
35
       pen.setWidth(10);
36
37
       //设置画笔的颜色
```

```
38
       pen.setColor(QColor(100,100,0));
39
40
       //更新画家的画笔
       painter.setPen(pen);
41
42
       //绘制矩形框
43
       painter.drawRect(20,20, this->width()-40, this-
44
   >height()-40);
45
46
47
   }
48
```

### 四、画图板

#### 1> 效果图



```
#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
   class Widget : public QWidget
14
   {
15
16
       Q_OBJECT
17
   public:
18
       widget(Qwidget *parent = nullptr);
19
       ~Widget();
20
21
       //鼠标移动事件
22
23
      void mouseMoveEvent(QMouseEvent *event) override;
       //鼠标按压事件
24
       void mousePressEvent(QMouseEvent *event) override;
25
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
                          //鼠标起始点坐标
       QPoint startPoint;
34
35
   };
36
   #endif // WIDGET_H
37
```

```
1 #include "widget.h"
   #include "ui_widget.h"
2
 3
   Widget::Widget(QWidget *parent)
4
        : QWidget(parent)
 5
        , ui(new Ui::Widget)
6
7
   {
8
       ui->setupUi(this);
9
       //设置页面大小
10
       this->setFixedSize(1024,768);
11
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
15
       //设置填充颜色
16
17
       pixmap->fill(Qt::white);
                               //用白色填充
18
   }
19
20
   Widget::~Widget()
21
   {
22
       delete ui;
   }
23
24
   //鼠标移动事件
25
   void Widget::mouseMoveEvent(QMouseEvent *event)
26
27
   {
       //实例化一个画家类,以pixmap作为父组件
28
29
       QPainter p(pixmap);
30
31
32
       //定义画笔
33
       QPen pen;
34
       pen.setWidth(5);
35
       pen.setColor(QColor(0,255,0));
36
37
       //给画家设置画笔
38
       p.setPen(pen);
39
40
       //从起始点到鼠标所在位置划线
41
       p.drawLine(startPoint, event->pos());
42
       //更新起始点坐标
43
44
       startPoint = event->pos();
45
46
       //调用绘制事件
47
       update();
48
49
   }
   //鼠标按压事件
50
  void Widget::mousePressEvent(QMouseEvent *event)
51
52
       startPoint = event->pos(); //给起始点初始值
53
54
55
   }
   //绘制事件
56
   void Widget::paintEvent(QPaintEvent *event)
```

```
58 {
59     //定义一个画家类,用来将pixmap绘制到窗口上
60     QPainter painter(this);
61
62     //将绘制好的pixmap画到窗口上
63     painter.drawPixmap(0,0, *pixmap);
64
65 }
66
```

## 五、基于tcp的网络聊天室

#### 服务器端原理图

#### ₡3.1.1 通信流程

- 1 创建套接字服务器 QTcpServer 对象
- ② 通过 QTcpServer 对象设置监听, 即: QTcpServer::listen()
- 3 基于 QTcpServer::newConnection() 信号检测是否有新的客户端连接
- 4 如果有新的客户端连接调用 QTcpSocket \*QTcpServer::nextPendingConnection() 得到通信的套接字对象
- 5 使用通信的套接字对象 QTcpSocket 和客户端进行通信

#### 客户端原理图

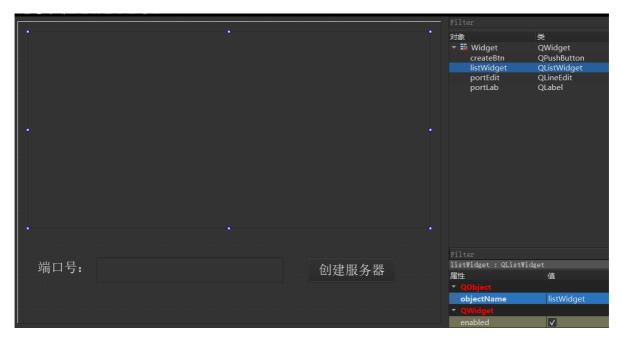
#### ∅3.2 客户端

#### ∅3.2.1 通信流程

- 1 创建通信的套接字类 QTcpSocket 对象
- ② 使用服务器端绑定的 IP 和端口连接服务器 QAbstractSocket::connectToHost()
- ③ 使用 QTcpSocket 对象和服务器进行通信

### 5.1 服务器端

1> ui界面



#### 2> 配置文件

```
1
   QT
             += core gui network
 2
 3
   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
 4
 5
   CONFIG += c++11
 6
   # 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
   # deprecated API in order to know how to port your code away
10
    from it.
11
   DEFINES += QT_DEPRECATED_WARNINGS
12
   # You can also make your code fail to compile if it uses
13
    deprecated APIs.
   # In order to do so, uncomment the following line.
14
   # You can also select to disable deprecated APIs only up to a
15
    certain version of Ot.
   #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000
                                                          #
16
    disables all the APIs deprecated before Qt 6.0.0
17
    SOURCES += \
18
19
        main.cpp \
20
       widget.cpp
21
22
   HEADERS += \
```

```
widget.h

forms += \
    widget.ui

perpendicular training for deployment.

property formula form
```

```
1 #ifndef WIDGET_H
2 #define WIDGET_H
4 #include <QWidget>
5 #include<QTcpServer> //服务器类
6 #include<QTcpSocket>
                             //客户端类
7
   #include<QDebug>
8
9
   QT_BEGIN_NAMESPACE
10 | namespace Ui { class Widget; }
11
   QT_END_NAMESPACE
12
   class Widget: public QWidget
13
   {
14
15
       Q_OBJECT
16
   public:
17
       widget(Qwidget *parent = nullptr);
18
19
       ~Widget();
20
21
       void sendToAll(QByteArray msg); //声明广播函数
22
   private slots:
23
24
       void on_createBtn_clicked();
25
26
       void on_tcpServer_newConnection();
                                              //newConnection
   信号对应的槽函数
27
28
       void on_socket_readyRead();
                                                //readyRead信号
   对应的槽函数
29
   private:
30
```

```
31
       Ui::Widget *ui;
32
33
       //定义服务器的指针,用来指向服务器
       QTcpServer *tcpServer;
34
35
       //定义链表存储连接过来的客户端
36
37
       QList<QTcpSocket *> tcpSockets;
38
39
40
  };
   #endif // WIDGET_H
41
42
43
```

```
1 #include "widget.h"
 2
   #include "ui_widget.h"
 3
 4
   Widget::Widget(QWidget *parent)
 5
       : QWidget(parent)
       , ui(new Ui::Widget)
 6
 7
   {
 8
       ui->setupUi(this);
9
10
       //创建出服务器对象
11
       tcpServer = new QTcpServer(this);
12
   }
13
   Widget::~Widget()
14
15
16
       delete ui;
17
18
   //定义广播函数,将获取到的消息发送给每个客户端
19
   void Widget::sendToAll(QByteArray msg)
20
21
       for(int i=0; i<tcpSockets.length(); i++)</pre>
22
23
       {
           tcpSockets.at(i)->write(msg);
24
25
           //将消息写到套接字中,功能是将服务器消息发送给客户端
26
       }
27
28
   }
29
```

```
30 //创建服务器按钮对应的槽函数
   void Widget::on_createBtn_clicked()
31
32
33
      quint16 port = ui->portEdit->text().toUInt();
34
      //从ui界面中,获取服务器的端口号
35
      //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
36
37
      //将服务器设置为监听状态
38
      if(tcpServer->listen(QHostAddress::Any, port))
39
40
          //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
41
          //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
42
   问
43
          //返回值:成功返回true,失败返回false
44
          qDebug()<<"创建服务器成功";
45
46
          //将断开输入框和创建服务器按钮设为不可用
47
          ui->portEdit->setEnabled(false);
          ui->createBtn->setEnabled(false);
48
49
      }else
50
      {
51
          qDebug()<<"创建服务器失败";
52
      }
53
      //此时服务器一直处于监听状态,如果有客户端连接进来
54
      //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
55
   函数中
56
      //在槽函数中处理相应逻辑
57
      connect(tcpServer, &QTcpServer::newConnection,
             this, &Widget::on_tcpServer_newConnection);
58
      //功能:将服务器发射的newConnection与自定义的槽函数连接
59
60
      //参数1: 信号发射者
      //参数2: 发射的信号
61
      //参数3: 信号接收者
62
      //参数4: 槽函数
63
64
   }
65
   //处理newConnection信号的槽函数的定义
66
   void Widget::on_tcpServer_newConnection()
67
68
69
      //获取最新连接的客户端
70
      QTcpSocket *socket = tcpServer->nextPendingConnection();
71
      //QTcpSocket *nextPendingConnection():获取最新连接的客户端
   socket
```

```
//参数: 无
72
       //返回值:新连接进来的客户端的套接字
73
74
75
       //将该客户端放入客户端链表中
76
       tcpSockets.push_back(socket);
       //链表操作,尾插法
77
78
79
       //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
80
81
       //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
82
       connect(socket, &QTcpSocket::readyRead, this,
83
    &widget::on_socket_readyRead);
       //参数1: 客户端套接字作为信号发射者
84
85
       //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
86
       //参数4: 处理readyRead信号的自定义槽函数
87
88
   }
89
   //处理readyRead信号的槽函数
90
   void Widget::on_socket_readyRead()
91
92
       //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
93
    客户端信息
94
       for(int i=0; i<tcpSockets.size(); i++)</pre>
95
       {
           //判断第i个客户端是否有效,如果无效,则进行删除
96
97
           if(tcpSockets.at(i)->state() == false)
98
           {
99
              //state(): 判断客户端的状态,如果可以则是该客户端,不可用
    为false
100
              tcpSockets.removeAt(i);
101
              //功能: 从链表中移除第i个元素
              //参数:链表的下标,从0开始
102
              //返回值:无
103
104
           }
       }
105
106
       //去判断到低是哪个客户端发送来的消息
107
       for(int i=0; i<tcpSockets.size(); i++)</pre>
108
109
110
           //判断该客户端中是否有数据
111
           if(tcpSockets.at(i)->bytesAvailable())
112
           {
              //qint64 bytesAvailable():判断客户端中是否有数据
113
```

```
//参数: 无
114
              //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
115
    数据可以被读取
116
117
              //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
118
              QByteArray msg = tcpSockets.at(i)->readAll();
119
              //功能: 读取套接字中所有的数据
              //参数: 无
120
121
              //返回值: QByteArray字节数组
122
123
              //将该数据展示到页面中去
124
              ui->listWidget-
    >addItem(QString::fromLocal8Bit(msg));
125
              //页面要展示数据,所需要的是QString类型的数据
126
              //但是,readAll()函数返回的是QByteArray类型的数据
127
              //所以,需要调用QString的成员函数,来将字节数组转换为字符
    串
128
129
              //将该消息广播给所有客户端
130
              sendToAll(msg);
131
132
           }
133
       }
134
135
   }
136
```

#### 5> ui文件

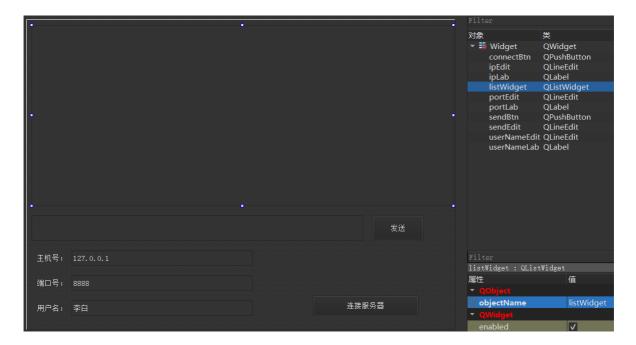
```
<?xml version="1.0" encoding="UTF-8"?>
   <ui version="4.0">
 2
    <class>Widget</class>
 3
     <widget class="Qwidget" name="Widget">
 4
 5
      cproperty name="geometry">
 6
       <rect>
 7
        < x > 0 < / x >
 8
        <y>0</y>
 9
        <width>862</width>
        <height>614</height>
10
11
       </rect>
12
      cproperty name="font">
13
       <font>
14
15
        <pointsize>16</pointsize>
```

```
16
       </font>
17
      property name="windowTitle">
18
19
       <string>Widget</string>
20
      </property>
      <widget class="QListWidget" name="listWidget">
21
22
       property name="geometry">
23
        <rect>
24
         < x > 20 < / x >
25
         <y>20</y>
         <width>821</width>
26
         <height>401</height>
27
28
        </rect>
29
       </property>
30
      </widget>
      <widget class="QLabel" name="portLab">
31
       cproperty name="geometry">
32
33
        <rect>
34
         < x > 40 < / x >
35
         <y>470</y>
         <width>101</width>
36
37
         <height>61</height>
38
        </rect>
39
       </property>
       cproperty name="text">
40
        <string>端口号: </string>
41
42
       </property>
43
      </widget>
      <widget class="QLineEdit" name="portEdit">
44
       cproperty name="geometry">
45
        <rect>
46
47
         < x > 160 < / x >
48
         <y>480</y>
         <width>381</width>
49
         <height>51</height>
50
        </rect>
51
52
       </property>
53
      </widget>
      <widget class="QPushButton" name="createBtn">
54
       cproperty name="geometry">
55
56
        <rect>
57
         < x > 590 < / x >
58
         <y>480</y>
         <width>181</width>
59
         <height>51</height>
60
```

```
61
        </rect>
62
       </property>
63
       property name="text">
        <string>创建服务器</string>
64
65
       </property>
      </widget>
66
     </widget>
67
68
     <resources/>
69
     <connections/>
70
    </ui>
71
```

### 5.2 客户端

#### 1> ui界面



#### 2> 配置文件

```
1
  QT
           += core gui network
2
3
  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5
  CONFIG += c++11
6
  # 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
```

```
10 | # deprecated API in order to know how to port your code away
   from it.
11 DEFINES += QT_DEPRECATED_WARNINGS
12
   # You can also make your code fail to compile if it uses
13
   deprecated APIs.
14 # In order to do so, uncomment the following line.
15 | # You can also select to disable deprecated APIs only up to a
   certain version of Qt.
16 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000
                                                         #
   disables all the APIs deprecated before Qt 6.0.0
17
   SOURCES += \
18
19
       main.cpp \
       widget.cpp
20
21
22 | HEADERS += \
23
       widget.h
24
25 | FORMS += \
26
       widget.ui
27
28 | # Default rules for deployment.
29 qnx: target.path = /tmp/$${TARGET}/bin
30 else: unix:!android: target.path = /opt/$${TARGET}/bin
31 !isEmpty(target.path): INSTALLS += target
32
```

```
1 #ifndef WIDGET_H
2
  #define WIDGET_H
 3
4 #include <QWidget>
5
   #include<QTcpSocket>
   #include<QMessageBox>
6
7
   QT_BEGIN_NAMESPACE
8
9
   namespace Ui { class Widget; }
   QT_END_NAMESPACE
10
11
12
   class Widget : public QWidget
   {
13
14
       Q_OBJECT
15
```

```
16 public:
       widget(Qwidget *parent = nullptr);
17
       ~Widget();
18
19
20
   private slots:
       void on_connectBtn_clicked();
21
22
23
       void on_tcpSocket_connected();
                                          //处理connected信号的
   槽函数
24
25
       void on_tcpSocket_readyRead();
                                     //处理readyRead信号的
   槽函数
26
       void on_sendBtn_clicked();
27
28
29
       void on_tcpSocket_disconnected(); //处理disconnected
   信号的槽函数
30
   private:
31
       Ui::Widget *ui;
32
33
34
       //定义一个客户端指针
       QTcpSocket *tcpSocket;
35
36
      //定义变量接受用户名
37
       QString userName;
38
39
40
  };
41 #endif // WIDGET_H
42
```

```
1 #include "widget.h"
   #include "ui_widget.h"
 2
 3
   widget::Widget(QWidget *parent)
 4
       : QWidget(parent)
 5
 6
       , ui(new Ui::Widget)
 7
   {
8
       ui->setupUi(this);
9
       //初始时,发送信息的编辑框和按钮不可用
10
       ui->sendBtn->setEnabled(false);
11
       ui->sendEdit->setEnabled(false);
12
```

```
13
      //实例化一个客户端
14
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);
      //参数1: 该客户端作为信号发射者
20
21
      //参数2: 要发射的是connected信号
      //参数3: 该页面接受信号
22
      //参数4: 处理该信号的槽函数
23
24
25
26
27
      //在客户端与服务器建立起连接后
      //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
28
      //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
29
   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
      //参数3: 该页面接受到信号
33
34
      //参数4: 处理相关逻辑的槽函数
35
36
      //当客户端与服务器断开连接后,会触发一个disconnected的信号,
37
      //我们可以在该信号对应的槽函数中处理相关逻辑
38
      connect(tcpSocket, &QTcpSocket::disconnected, this,
39
   &Widget::on_tcpSocket_disconnected);
40
41
   Widget::~Widget()
42
43
44
      delete ui:
45
46
47
   }
48
49
   //连接服务器按钮对应的槽函数
   void Widget::on_connectBtn_clicked()
50
51
      if(ui->connectBtn->text() == "连接服务器")
52
53
      {
```

```
54
          //编写连接服务器的逻辑
55
          QString ip = ui->ipEdit->text(); //获取界面ip地址
          quint16 port = ui->portEdit->text().toUInt(); //获
56
   取页面上的端口号
57
58
          //连接到服务器
59
          tcpSocket->connectToHost(ip, port,
   QTcpSocket::ReadWrite, QTcpSocket::AnyIPProtocol);
60
          //功能:将该客户端连接到服务器
61
          //参数1: 要连接的主机ip地址
          //参数2: 要连接的端口号
62
          //参数3:连接模式:读写都可以
63
64
          //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
65
66
          //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
67
68
69
          ui->connectBtn->setText("断开服务器");
70
      }else if(ui->connectBtn->text() == "断开服务器")
71
72
      {
73
          userName = ui->userNameEdit->text(); //获取用户名
74
75
          QString msg = userName +": 离开聊天室";
76
          tcpSocket->write(msg.toLocal8Bit());
77
78
          //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
79
          tcpSocket->disconnectFromHost();
80
          //功能: 断开客户端的连接
          //参数: 无
81
82
          //返回值:无
83
84
85
86
87
          ui->connectBtn->setText("连接服务器");
      }
88
89
90
  }
91
92
   //connected信号对应的槽函数的定义
   void Widget::on_tcpSocket_connected()
93
94
```

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

```
139
        //将编辑信息框中的消息清除
140
141
        ui->sendEdit->clear();
142
    }
143
144
    //disconnected信号对应的槽函数定义
    void Widget::on_tcpSocket_disconnected()
145
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
        //将用户信息设为可用状态
151
        ui->ipEdit->setEnabled(true);
152
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
      cproperty name="geometry">
 6
       <rect>
 7
        < x > 0 < / x >
 8
        < y > 0 < / y >
 9
        <width>854</width>
10
        <height>618</height>
       </rect>
11
12
      </property>
      property name="windowTitle">
13
14
       <string>Widget</string>
15
      </property>
      <widget class="QListWidget" name="listWidget">
16
17
       cproperty name="geometry">
18
        <rect>
19
         < x > 10 < / x >
20
         < y > 10 < / y >
         <width>841</width>
21
22
         <height>361</height>
23
        </rect>
```

```
24
       </property>
25
      </widget>
26
      <widget class="QLineEdit" name="sendEdit">
       property name="geometry">
27
28
        <rect>
29
         < x > 10 < / x >
30
         < y > 390 < / y >
31
         <width>661</width>
         <height>51</height>
32
33
        </rect>
34
       </property>
35
      </widget>
36
      <widget class="QPushButton" name="sendBtn">
37
       cproperty name="geometry">
38
        <rect>
39
         < x > 690 < / x >
40
         < y > 390 < / y >
41
         <width>101</width>
42
         <height>51</height>
43
        </rect>
44
       </property>
45
       property name="text">
46
        <string>发送</string>
47
       </property>
      </widget>
48
      <widget class="QLabel" name="ipLab">
49
       cproperty name="geometry">
50
51
        <rect>
52
         < x > 20 < /x >
53
         <y>460</y>
54
         <width>81</width>
         <height>31</height>
55
56
        </rect>
57
       </property>
58
       cproperty name="text">
59
        <string>主机号: </string>
60
       </property>
61
      </widget>
      <widget class="QLineEdit" name="ipEdit">
62
       cproperty name="geometry">
63
64
        <rect>
65
         < x > 90 < /x >
66
         < y > 460 < / y >
         <width>361</width>
67
         <height>31</height>
68
```

```
69
         </rect>
 70
        </property>
 71
        property name="text">
         <string>127.0.0.1
 72
 73
        </property>
 74
       </widget>
 75
       <widget class="QLineEdit" name="portEdit">
        property name="geometry">
 76
 77
         <rect>
 78
          < x > 90 < /x >
          < y > 510 < / y >
 79
 80
          <width>361</width>
 81
          <height>31</height>
 82
         </rect>
 83
        </property>
 84
        property name="text">
 85
         <string>8888</string>
 86
        </property>
       </widget>
 87
 88
       <widget class="QLabel" name="portLab">
        property name="geometry">
 89
 90
         <rect>
 91
          < x > 20 < /x >
 92
          < y > 510 < / y >
 93
          <width>81</width>
 94
          <height>31</height>
 95
         </rect>
 96
        </property>
 97
        property name="text">
 98
         <string>端口号: </string>
 99
        </property>
100
       </widget>
       <widget class="QLabel" name="userNameLab">
101
        cproperty name="geometry">
102
103
         <rect>
104
          < x > 20 < / x >
105
          < y > 560 < / y >
          <width>81</width>
106
          <height>31</height>
107
108
         </rect>
109
        </property>
        property name="text">
110
111
         <string>用户名: </string>
112
        </property>
       </widget>
113
```

```
<widget class="QLineEdit" name="userNameEdit">
114
115
        cproperty name="geometry">
116
         <rect>
117
          < x > 90 < /x >
118
          <y>560</y>
          <width>361</width>
119
120
          <height>31</height>
121
         </rect>
122
        </property>
123
        cproperty name="text">
124
         <string>李白</string>
125
        </property>
126
       </widget>
       <widget class="QPushButton" name="connectBtn">
127
128
        cproperty name="geometry">
129
         <rect>
130
          < x > 570 < / x >
131
          <y>550</y>
132
          <width>211</width>
133
          <height>41</height>
134
         </rect>
135
        </property>
        cproperty name="text">
136
137
         <string>连接服务器</string>
138
        </property>
139
       </widget>
     </widget>
140
141
     <resources/>
142
     <connections/>
143
    </ui>
144
```

# 作业

实现闹钟功能

是示系统时间,可以是QLable 此处是QLineEdit,允许用户编辑 定时的时间 启动 关闭 按钮 按钮
今天是二狗子结婚,准备好份子钱, 开席开席。
QTextEdit

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