96 TCP服务器功能实现

第二步，实现TcpServer的通信功能。

96.1 实现功能

C:\Users\Minwei\Projects\Qt\TcpServer\tcpserverwindow.h：

```
#ifndef TCPSERVERWINDOW_H
#define TCPSERVERWINDOW_H

#include <QMainWindow>
#include <QLabel>
#include <QTcpServer>
#include <QTcpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class TcpServerWindow; }
QT_END_NAMESPACE

class TcpServerWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_m_actListen_triggered();
    void on_m_actClose_triggered();
    void on_m_actClear_triggered();

    void on_m_btnSend_clicked();

    void on_m_server_newConnection();

    void on_m_socket_readyRead();
    void on_m_socket_disconnected();

    void on_m_socket_stateChanged(
        QAbstractSocket::SocketState socketState);
    void on_m_socket_error(
        QAbstractSocket::SocketError socketError);

private:
    Ui::TcpServerWindow *ui;
    QLabel* m_labListenState;
    QLabel* m_labSocketState;
    QLabel* m_labSocketError;
    QTcpServer* m_server;
    QTcpSocket* m_socket;
};

```

```
#endif // TCPSERVERWINDOW_H
```

C:\Users\Minwei\Projects\Qt\TcpServer\tcpserverwindow.cpp：

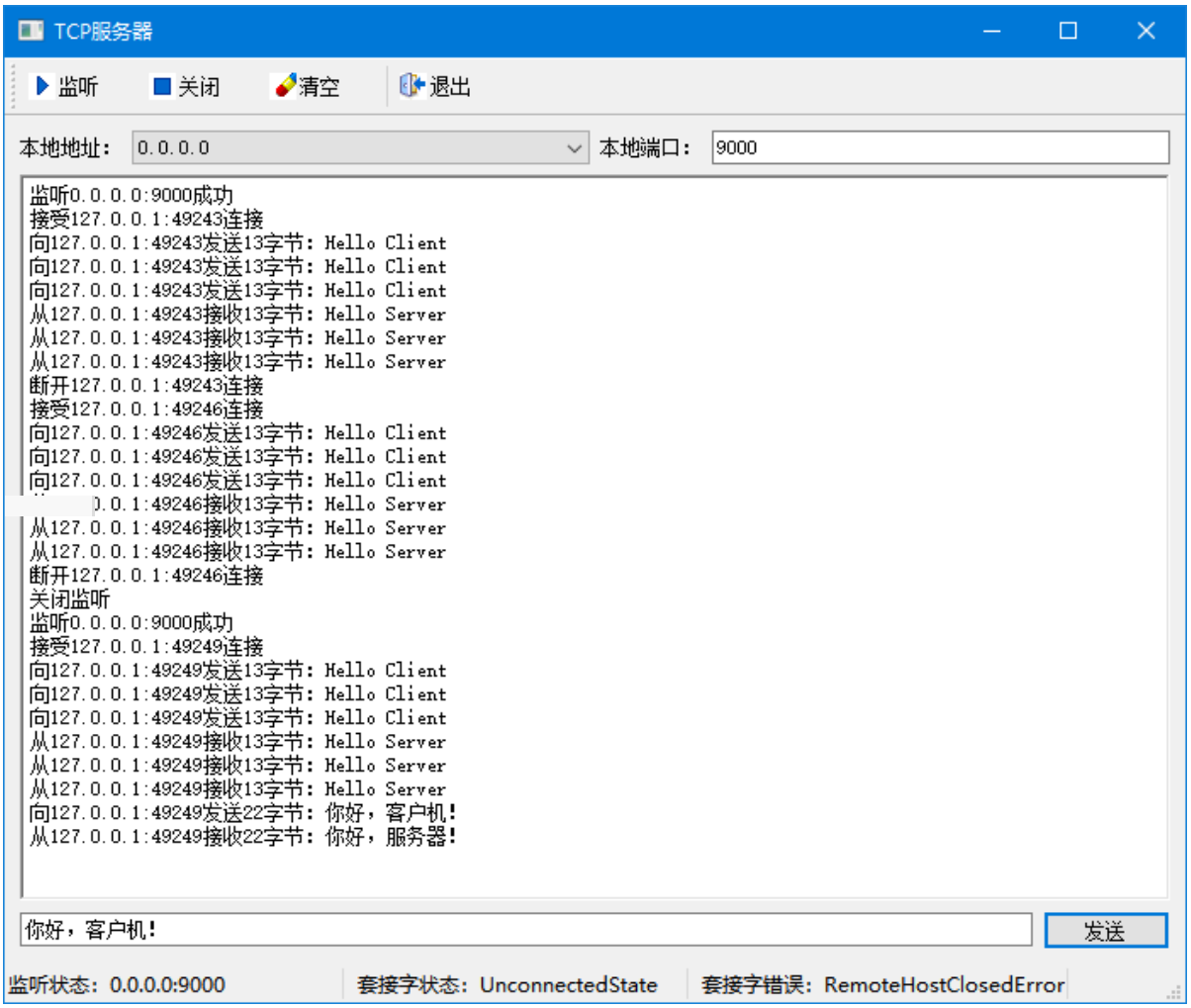
```
#include <QHostInfo>
#include <QMetaEnum>

#include "tcpserverwindow.h"
#include "ui_tcpserverwindow.h"

TcpServerWindow::TcpServerWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::TcpServerWindow)
    , m_labListenState(new QLabel("监听状态：关闭"))
    , m_labSocketState(new QLabel(" 套接字状态："))
    , m_labSocketError(new QLabel(" 套接字错误："))
    , m_server(new QTcpServer(this))
    , m_socket(Q_NULLPTR)
{
    ui->setupUi(this);

    for (QHostAddress address : QHostInfo::fromName(
        QHostInfo::localHostName()).addresses())
        if(address.protocol() == QAbstractSocket::IPv4Protocol)
            ui->m_comboLocalAddr->addItem(address.toString());

    m_labListenState->setMinimumWidth(196);
    ui->m_statusBar->addWidget(m_labListenState);
    m_labSocketState->setMinimumWidth(197);
    ui->m_statusBar->addWidget(m_labSocketState);
    m_labSocketError->setMinimumWidth(197);
    ui->m_statusBar->addWidget(m_labSocketError);

    connect(m_server, SIGNAL(newConnection()),
        this, SLOT(on_m_server_newConnection()));
}

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

void TcpServerWindow::on_m_actListen_triggered()
{
    QHostAddress address(ui->m_comboLocalAddr->currentText());
    quint16 port = ui->m_editLocalPort->text().toUShort();

    if (m_server->listen(address, port))
    {
        ui->m_editOutput->appendPlainText(QString("监听%1:%2成功").
            arg(address.toString()).arg(port));
        m_labListenState->setText(QString("监听状态：%1:%2").
            arg(address.toString()).arg(port));
    }
    else
```

```
        ui->m_editOutput->appendPlainText(QString("监听%1:%2失败").
            arg(address.toString()).arg(port));
}

void TcpServerWindow::on_m_actClose_triggered()
{
    m_server->close();

    ui->m_editOutput->appendPlainText("关闭监听");
    m_labListenState->setText("监听状态：关闭");
}

void TcpServerWindow::on_m_actClear_triggered()
{
    ui->m_editOutput->clear();
}

void TcpServerWindow::on_m_btnSend_clicked()
{
    QByteArray data((ui->m_editSend->text() + "\n").toUtf8());

    qint64 nbytes = m_socket->write(data);

    if (nbytes == -1)
        ui->m_editOutput->appendPlainText("发送失败");
    else
        ui->m_editOutput->appendPlainText(QString("向%1:%2发送%3字节：%4").
            arg(m_socket->peerAddress().toString()).arg(m_socket-
>peerPort()).
            arg(nbytes).arg(QString(data).trimmed()));
}

void TcpServerWindow::on_m_server_newConnection()
{
    if (m_socket != Q_NULLPTR &&
        m_socket->state() == QAbstractSocket::ConnectedState)
        m_socket->disconnectFromHost();

    m_socket = m_server->nextPendingConnection();

    connect(m_socket, SIGNAL(readyRead()),
        this, SLOT(on_m_socket_readyRead()));
    connect(m_socket, SIGNAL(disconnected()),
        this, SLOT(on_m_socket_disconnected()));

    connect(m_socket, SIGNAL(stateChanged(QAbstractSocket::SocketState)),
        this,
SLOT(on_m_socket_stateChanged(QAbstractSocket::SocketState)));
    connect(m_socket, SIGNAL(error(QAbstractSocket::SocketError)),
        this, SLOT(on_m_socket_error(QAbstractSocket::SocketError)));

    ui->m_editOutput->appendPlainText(QString("接受%1:%2连接").
        arg(m_socket->peerAddress().toString()).arg(m_socket->peerPort()));

    ui->m_btnSend->setEnabled(true);
}
```

```
106
107 void TcpServerWindow::on_m_socket_readyRead()
108 {
109     while (m_socket->canReadLine())
110     {
111         QByteArray data = m_socket->readLine();
112
113         ui->m_editOutput->appendPlainText(QString("从%1:%2接收%3字节：%4").
114             arg(m_socket->peerAddress().toString()).arg(m_socket-
>peerPort()).
115             arg(data.size()).arg(QString(data).trimmed()));
116     }
117 }
118
119 void TcpServerWindow::on_m_socket_disconnected()
120 {
121     ui->m_btnSend->setEnabled(false);
122
123     ui->m_editOutput->appendPlainText(QString("断开%1:%2连接").
124         arg(m_socket->peerAddress().toString()).arg(m_socket->peerPort()));
125
126     m_socket->deleteLater();
127     m_socket = Q_NULLPTR;
128 }
129
130 void TcpServerWindow::on_m_socket_stateChanged(
131     QAbstractSocket::SocketState socketState)
132 {
133     m_labSocketState->setText(QString(" 套接字状态：%1").arg(
134         QMetaEnum::fromType<QAbstractSocket::SocketState>().
135         valueToKey(socketState)));
136 }
137
138 void TcpServerWindow::on_m_socket_error(
139     QAbstractSocket::SocketError socketError)
140 {
141     m_labSocketError->setText(QString(" 套接字错误：%1").arg(
142         QMetaEnum::fromType<QAbstractSocket::SocketError>().
143         valueToKey(socketError)));
144 }
```

96.2 测试验证

运行效果如图所示：



96.3 功能局限

这个版本的TCP服务器只能和一个TCP客户机保持连接并通信。当有新客户机向服务器发起连接时，服务器会断开与之前客户机的连接。这主要体现在服务器程序处理QTcpServer对象发射的newConnection信号的槽函数上：

```
...
TcpServerWindow::TcpServerWindow(QWidget *parent)
    ...
{
    ...
    connect(m_server, SIGNAL(newConnection()),
        this, SLOT(on_m_server_newConnection()));
    ...
}
...
void TcpServerWindow::on_m_server_newConnection()
{
    ...
    if (m_socket != Q_NULLPTR &&
        m_socket->state() == QAbstractSocket::ConnectedState)
        m_socket->disconnectFromHost();
    ...
    m_socket = m_server->nextPendingConnection();
    ...
}
...
```