第31篇 (网络) TCP

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Network05

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Network06

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Network07

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Network08

一、服务器端

在服务器端的程序中，我们本地主机的一个端口，这里使用6666，然后关联`newConnection()`信号与自己写的`sendMessage()`槽。就是说一旦有客户端的连接请求，就会执行`sendMessage()`函数，在这个函数里我们发送一个简单的字符串。

1．新建QtGui应用

项目名为`tcpServer`，基类选择`QWidget`，类名为`Widget`。完成后打开项目文件`tcpServer.pro`并添加一行代码：`QT += network`，然后保存该文件。

2．在`widget.ui`的设计区添加一个Label，更改其显示文本为“等待连接”，然后更改其`objectName`为`statusLabel`，用于显示一些状态信息。

3．在`widget.h`文件中做以下更改。

添加头文件：`#include <QtNetWork>`

添加private对象：`QTcpServer *tcpServer;`

添加私有槽：

```
private slots:
void sendMessage();
```

4．在`widget.cpp`文件中进行更改。

在其构造函数中添加代码：

```
tcpServer = new QTcpServer(this);
if(!tcpServer->listen(QHostAddress::LocalHost,6666))
{  //本地主机的6666端口，如果出错就输出错误信息，并关闭
    qDebug() << tcpServer->errorString();
    close();
}
//连接信号和相应槽函数
connect(tcpServer,SIGNAL(newConnection()),this,SLOT(sendMessage()));
```

我们在构造函数中使用`tcpServer`的`listen()`函数进行，然后关联了`newConnection()`和我们自己的`sendMessage()`函数。

下面我们实现`sendMessage()`函数。

```
void Widget::sendMessage()
{
    //用于暂存我们要发送的数据
    QByteArray block;

    //使用数据流写入数据
    QDataStream out(&block,QIODevice::WriteOnly);

    //设置数据流的版本，客户端和服务器端使用的版本要相同
    out.setVersion(QDataStream::Qt_4_6);

    out<<(quint16) 0;
    out<<tr("hello Tcp!!!");
    out.device()->seek(0);
    out<<(quint16) (block.size() - sizeof(quint16));

    //我们获取已经建立的连接的子套接字
    QTcpSocket *clientConnection = tcpServer->nextPendingConnection();

    connect(clientConnection,SIGNAL(disconnected()),clientConnection,
            SLOT(deleteLater()));
    clientConnection->write(block);
    clientConnection->disconnectFromHost();

    //发送数据成功后，显示提示
    ui->statusLabel->setText("send message successful!!!");
}
```

这个是数据发送函数，我们主要介绍两点：

（1）为了保证在客户端能接收到完整的文件，我们都在数据流的最开始写入完整文件的大小信息，这样客户端就可以根据大小信息来判断是否接受到了完整的文件。而在服务器端，在发送数据时就要首先发送实际文件的大小信息，但是，文件的大小一开始是无法预知的，所以这里先使用了`out<<(quint16) 0;`在`block`的开始添加了一个`quint16`大小的空间，也就是两字节的空间，它用于后面放置文件的大小信息。然后`out<<tr("hello Tcp!!!");`输入实际的文件，这里是字符串。当文件输入完成后我们再使用`out.device()->seek(0);`返回到`block`的开始，加入实际的文件大小信息，也就是后面的代码，它是实际文件的大小：`out<<(quint16) (block.size() - sizeof(quint16));`

（2）在服务器端我们可以使用`tcpServer`的`nextPendingConnection()`函数来获取已经建立的连接的Tcp套接字，使用它来完成数据的发送和其它操作。比如这里，我们关联了`disconnected()`信号和`deleteLater()`槽，然后我们发送数据

```
clientConnection->write(block);
```

然后是`clientConnection->disconnectFromHost();`

它表示当发送完成时就会断开连接，这时就会发出`disconnected()`信号，而最后调用`deleteLater()`函数保证在关闭连接后删除该套接字`clientConnection`。

5．这样服务器的程序就完成了，可以先运行一下程序。

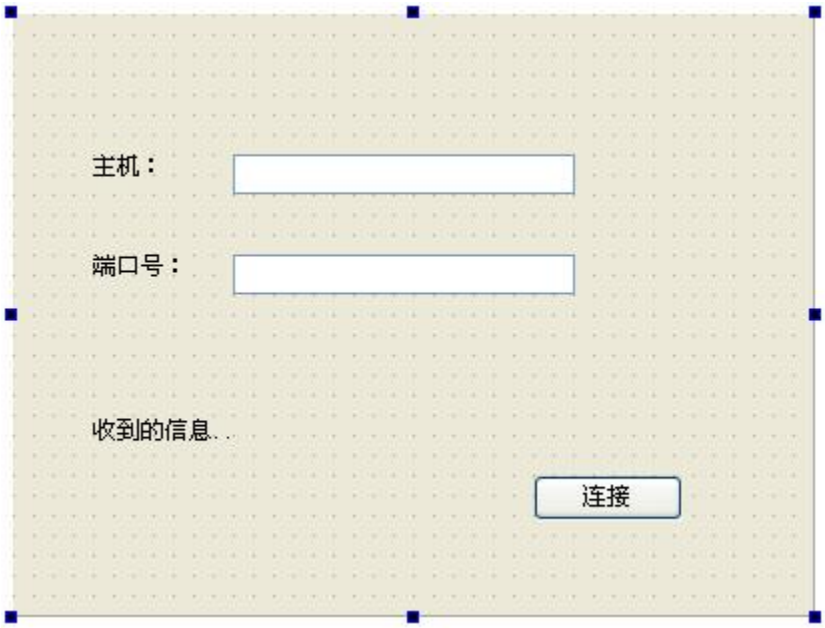
二、客户端

我们在客户端程序中向服务器发送连接请求，当连接成功时接收服务器发送的数据。

1．新建Qt Gui应用，

项目名`tcpClient`，基类选择`QWidget`，类名为`Widget`。完成后打开项目文件`tcpClient.pro`并添加一行代码：`QT += network`，然后保存该文件。

2．我们在`widget.ui`中添加几个标签`Label`和两个`Line Edit`以及一个按钮`Push Button`。设计效果如下图所示。



其中“主机”后的`LineEdit`的`objectName`为`hostLineEdit`，“端口号”后的为`portLineEdit`。

“收到的信息”标签的`objectName`为`messageLabel`。

3．在`widget.h`文件中做更改。

添加头文件：`#include <QtNetwork>`

添加`private`变量：

```
QTcpSocket *tcpSocket;
QString message;  //存放从服务器接收到的字符串
quint16blockSize; //存放文件的大小信息
```

添加私有槽：

```
private slots:
    void newConnect(); //连接服务器
    void readMessage();  //接收数据
void displayError(QAbstractSocket::SocketError);  //显示错误
```

4．在`widget.cpp`文件中做更改。

（1）在构造函数中添加代码：

```
tcpSocket = new QTcpSocket(this);
connect(tcpSocket,SIGNAL(readyRead()),this,SLOT(readMessage()));
connect(tcpSocket,SIGNAL(error(QAbstractSocket::SocketError)),
         this,SLOT(displayError(QAbstractSocket::SocketError)));
```

这里关联了`tcpSocket`的两个信号，当有数据到来时发出`readyRead()`信号，我们执行读取数据的`readMessage()`函数。当出现错误时发出`error()`信号，我们执行`displayError()`槽函数。

(2)实现`newConnect()`函数。

```
void Widget::newConnect()
{
    blockSize = 0; //初始化其为0
tcpSocket->abort(); //取消已有的连接

//连接到主机，这里从界面获取主机地址和端口号
    tcpSocket->connectToHost(ui->hostLineEdit->text(),
                             ui->portLineEdit->text().toInt());
}
```

这个函数实现了连接到服务器，下面会在“连接”按钮的单击事件槽函数中调用这个函数。

（3）实现`readMessage()`函数。

```
void Widget::readMessage()
{
    QDataStream in(tcpSocket);
    in.setVersion(QDataStream::Qt_4_6);
    //设置数据流版本，这里要和服务器端相同
    if(blockSize==0) //如果是刚开始接收数据
    {
       //判断接收的数据是否有两字节，也就是文件的大小信息
       //如果有则保存到blockSize变量中，没有则返回，继续接收数据
       if(tcpSocket->bytesAvailable() < (int)sizeof(quint16)) return;
       in >> blockSize;
    }
    if(tcpSocket->bytesAvailable() < blockSize) return;
    //如果没有得到全部的数据，则返回，继续接收数据
    in >> message;
    //将接收到的数据存放到变量中
    ui->messageLabel->setText(message);
    //显示接收到的数据
}
```

这个函数实现了数据的接收，它与服务器端的发送函数相对应。首先我们要获取文件的大小信息，然后根据文件的大小来判断是否接收到了完整的文件。

(4)实现`displayError()`函数。

```
void Widget::displayError(QAbstractSocket::SocketError)
{
    qDebug() << tcpSocket->errorString(); //输出错误信息
}
```

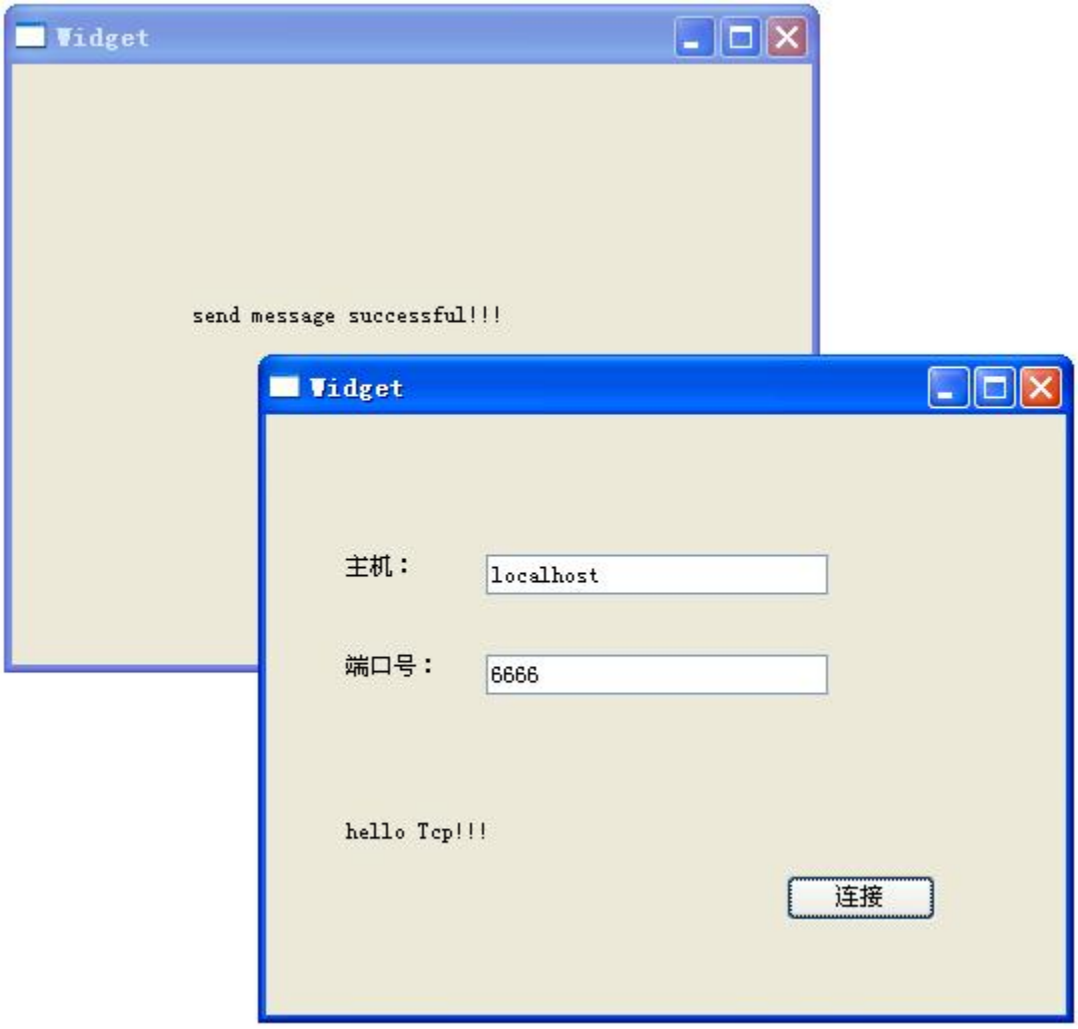
这里简单的实现了错误信息的输出。

（5）我们在`widget.ui`中进入“连接”按钮的单击事件槽函数，然后更改如下。

```
void Widget::on_pushButton_clicked() //连接按钮
{
    newConnect(); //请求连接
}
```

这里直接调用了`newConnect()`函数。

5．我们运行程序，同时运行服务器程序，然后在“主机”后填入“localhost”，在“端口号”后填入“6666”，点击“连接”按钮，效果如下。



可以看到我们正确地接收到了数据。因为服务器端和客户端是在同一台机子上运行的，所以我这里填写了“主机”为“localhost”，如果你在不同的机子上运行，需要在“主机”后填写其正确的IP地址。

三、客户端

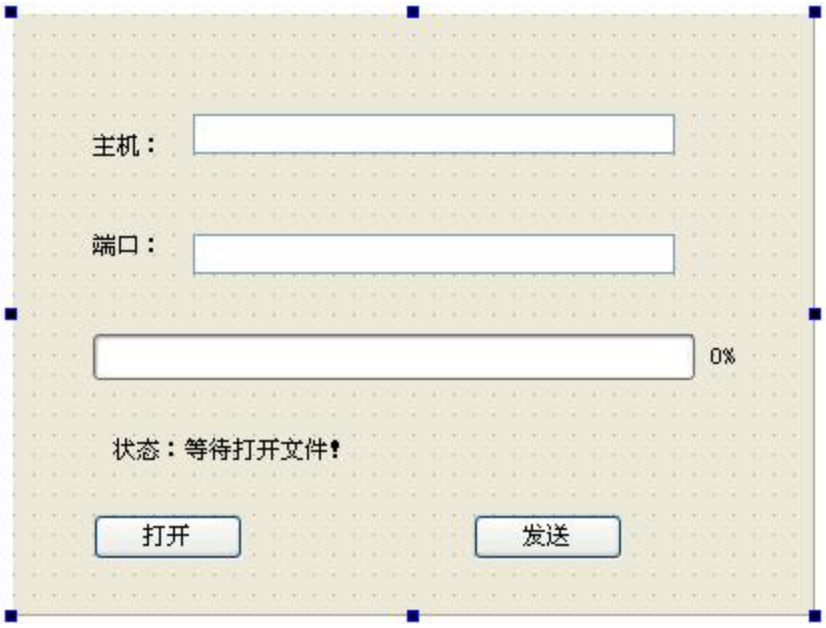
这次先讲解客户端，在客户端里需要与服务器进行连接，一旦连接成功，就会发出`connected()`信号，这时我们就进行文件的发送。

在上一节已经看到，发送数据时先发送了数据的大小信息。这一次，我们要先发送文件的总大小，然后文件名长度，然后是文件名，这三部分合称为文件头结构，最后再发送文件数据。所以在发送函数里就要进行相应的处理，当然，在服务器的接收函数里也要进行相应的处理。对于文件大小，这次使用了`qint64`，它是64位的，可以表示一个很大的文件了。

1．新建QtGui项目

名称为`tcpSender`，基类选择`QWidget`，类名为`Widget`，完成后打开`tcpSender.pro`添加一行代码：`QT += network` 。

2．我们在`widget.ui`文件中将界面设计如下。



这里“主机”后的`Line Edit`的`objectName`为`hostLineEdit` ；“端口”后的`Line Edit`的`objectName`为`portLineEdit` ；下面的`Progress Bar`的`objectName`为`clientProgressBar`，其`value`属性设为0；“状态”`Label`的`objetName`为`clientStatusLabel` ；“打开”按钮的`objectName`为`openButton` ；“发送”按钮的`objectName`为`sendButton`。

3．在`widget.h`文件中进行更改。

（1）添加头文件包含`#include <QtNetwork>`

（2）添加`private`变量：

```
QTcpSocket *tcpClient;
    QFile *localFile;  //要发送的文件
    qint64 totalBytes;  //数据总大小
    qint64 bytesWritten;  //已经发送数据大小
    qint64 bytesToWrite;   //剩余数据大小
    qint64 loadSize;   //每次发送数据的大小
    QString fileName;  //保存文件路径
QByteArray outBlock;  //数据缓冲区，即存放每次要发送的数据
```

（3）添加私有槽函数：

```
private slots:
    void send();  //连接服务器
    void startTransfer();  //发送文件大小等信息
    void updateClientProgress(qint64); //发送数据，更新进度条
    void displayError(QAbstractSocket::SocketError); //显示错误
void openFile();  //打开文件
```

4．在`widget.cpp`文件中进行更改

添加头文件：`#include <QFileDialog>`

（1）在构造函数中添加代码：

```
loadSize = 4*1024;
totalBytes = 0;
bytesWritten = 0;
bytesToWrite = 0;
tcpClient = new QTcpSocket(this);
//当连接服务器成功时，发出connected()信号，我们开始传送文件
connect(tcpClient,SIGNAL(connected()),this,SLOT(startTransfer()));
//当有数据发送成功时，我们更新进度条
connect(tcpClient,SIGNAL(bytesWritten(qint64)),this,
       SLOT(updateClientProgress(qint64)));
connect(tcpClient,SIGNAL(error(QAbstractSocket::SocketError)),this,
       SLOT(displayError(QAbstractSocket::SocketError)));
//开始使”发送“按钮不可用
ui->sendButton->setEnabled(false);
```

我们主要是进行了变量的初始化和几个信号和槽函数的关联。

（2）实现打开文件函数。

```
void Widget::openFile()   //打开文件
{
    fileName = QFileDialog::getOpenFileName(this);
    if(!fileName.isEmpty())
    {
       ui->sendButton->setEnabled(true);
       ui->clientStatusLabel->setText(tr("打开文件 %1 成功！")
                                      .arg(fileName));
    }
}
```

该函数将在下面的“打开”按钮单击事件槽函数中调用。

（3）实现连接函数。

```
void Widget::send()   //连接到服务器，执行发送
{
    ui->sendButton->setEnabled(false);
    bytesWritten = 0;
    //初始化已发送字节为0
    ui->clientStatusLabel->setText(tr("连接中..."));
    tcpClient->connectToHost(ui->hostLineEdit->text(),
                             ui->portLineEdit->text().toInt());//连接
}
```

该函数将在“发送”按钮的单击事件槽函数中调用。

（4）实现文件头结构的发送。

```
void Widget::startTransfer()  //实现文件大小等信息的发送
{
    localFile = new QFile(fileName);
    if(!localFile->open(QFile::ReadOnly))
    {
       qDebug() << "open file error!";
       return;
    }

    //文件总大小
    totalBytes = localFile->size();

    QDataStream sendOut(&outBlock,QIODevice::WriteOnly);
    sendOut.setVersion(QDataStream::Qt_4_6);
QString currentFileName = fileName.right(fileName.size()
- fileName.lastIndexOf('/')-1);

    //依次写入总大小信息空间，文件名大小信息空间，文件名
    sendOut << qint64(0) << qint64(0) << currentFileName;

    //这里的总大小是文件名大小等信息和实际文件大小的总和
    totalBytes += outBlock.size();

    sendOut.device()->seek(0);
    //返回outBolock的开始，用实际的大小信息代替两个qint64(0)空间
    sendOut<<totalBytes<<qint64((outBlock.size() - sizeof(qint64)*2));

    //发送完头数据后剩余数据的大小
    bytesToWrite = totalBytes - tcpClient->write(outBlock);

    ui->clientStatusLabel->setText(tr("已连接"));
```

```
    outBlock.resize(0);
}
```

（5）下面是更新进度条，也就是发送文件数据。

```
//更新进度条，实现文件的传送
void Widget::updateClientProgress(qint64 numBytes)
{
    //已经发送数据的大小
    bytesWritten += (int)numBytes;

    if(bytesToWrite > 0) //如果已经发送了数据
    {
   //每次发送loadSize大小的数据，这里设置为4KB，如果剩余的数据不足4KB，
   //就发送剩余数据的大小
       outBlock = localFile->read(qMin(bytesToWrite,loadSize));

       //发送完一次数据后还剩余数据的大小
       bytesToWrite -= (int)tcpClient->write(outBlock);

       //清空发送缓冲区
       outBlock.resize(0);

    } else {
       localFile->close(); //如果没有发送任何数据，则关闭文件
    }

    //更新进度条
    ui->clientProgressBar->setMaximum(totalBytes);
    ui->clientProgressBar->setValue(bytesWritten);

    if(bytesWritten == totalBytes) //发送完毕
    {
     ui->clientStatusLabel->setText(tr("传送文件 %1 成功")
.arg(fileName));
       localFile->close();
       tcpClient->close();
    }
}
```

（6）实现错误处理函数。

```
void Widget::displayError(QAbstractSocket::SocketError) //显示错误
{
    qDebug() << tcpClient->errorString();
    tcpClient->close();
    ui->clientProgressBar->reset();
    ui->clientStatusLabel->setText(tr("客户端就绪"));
```

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

（7）我们从`widget.ui`中分别进行“打开”按钮和“发送”按钮的单击事件槽函数，然后更改如下。

```
void Widget::on_openButton_clicked() //打开按钮
{
    openFile();
}
void Widget::on_sendButton_clicked() //发送按钮
{
    send();
}
```

5．我们为了使程序中的中文不显示乱码，在`main.cpp`文件中更改。

添加头文件：`#include <QTextCodec>`

在main函数中添加代码：`QTextCodec::setCodecForTr(QTextCodec::codecForName("UTF-8"));`

6．现在可以先运行程序。

7．程序整体思路分析。

我们设计好界面，然后按下“打开”按钮，选择要发送的文件，这时调用了`openFile()`函数。然后点击“发送”按钮，调用`send()`函数，与服务器进行连接。当连接成功时就会发出`connected()`信号，这时就会执行`startTransfer()`函数，进行文件头结构的发送，当发送成功时就会发出`bytesWritten(qint64)`信号，这时执行`updateClientProgress(qint64 numBytes)`进行文件数据的传输和进度条的更新。这里使用了一个`loadSize`变量，我们在构造函数中将其初始化为`4*1024`即4字节，它的作用是，我们将整个大的文件分成很多小的部分进行发送，每部分为4字节。而当连接出现问题时就会发出`error(QAbstractSocket::SocketError)`信号，这时就会执行`displayError()`函数。对于程序中其他细节我们就不再分析，希望大家能自己编程研究一下。

四、服务器端

我们在服务器端进行数据的接收。服务器端程序是很简单的，我们开始进行监听，一旦发现有连接请求就发出`newConnection()`信号，然后我们便接受连接，开始接收数据。

1．新建QtGui应用

名称为`tcpReceiver`，基类选择`QWidget`，类名为`Widget`，完成后打开`tcpReceiver.pro`添加一行代码：`QT += network`。

2．我们更改`widget.ui`文件，设计界面如下。

其中“服务器端”`Label`的`objectName`为`serverStatusLabel`；进度条`ProgressBar`的`objectName`为`serverProgressBar`，设置其`value`属性为0；“开始监听”按钮的`objectName`为`startButton`。

效果如下。

3．更改`widget.h`文件的内容。

（1）添加头文件包含：`#include <QtNetwork>`

（2）添加私有变量：

```
QTcpServer tcpServer;
QTcpSocket *tcpServerConnection;
qint64 totalBytes;  //存放总大小信息
qint64 bytesReceived;  //已收到数据的大小
qint64 fileNameSize;  //文件名的大小信息
QString fileName;   //存放文件名
QFile *localFile;   //本地文件
QByteArray inBlock;   //数据缓冲区
```

（3）添加私有槽函数：

```
private slots:
    void on_startButton_clicked();
    void start();   //开始监听
    void acceptConnection();  //建立连接
void updateServerProgress();  //更新进度条，接收数据

//显示错误
void displayError(QAbstractSocket::SocketError socketError);
```

4．更改`widget.cpp`文件。

（1）在构造函数中添加代码：

```
totalBytes = 0;
    bytesReceived = 0;
fileNameSize = 0;

//当发现新连接时发出newConnection()信号
    connect(&tcpServer,SIGNAL(newConnection()),this,
SLOT(acceptConnection()));
```

（2）实现`start()`函数。

```
void Widget::start() //开始监听
{
    ui->startButton->setEnabled(false);
    bytesReceived =0;
    if(!tcpServer.listen(QHostAddress::LocalHost,6666))
    {
       qDebug() << tcpServer.errorString();
       close();
       return;
    }
    ui->serverStatusLabel->setText(tr("监听"));
}
```

（3）实现接受连接函数。

```
void Widget::acceptConnection()  //接受连接
{
    tcpServerConnection = tcpServer.nextPendingConnection();
connect(tcpServerConnection,SIGNAL(readyRead()),this,
SLOT(updateServerProgress()));
    connect(tcpServerConnection,
```

```
SIGNAL(error(QAbstractSocket::SocketError)),this,
           SLOT(displayError(QAbstractSocket::SocketError)));
    ui->serverStatusLabel->setText(tr("接受连接"));
    tcpServer.close();
}
```

（4）实现更新进度条函数。

```
void Widget::updateServerProgress()  //更新进度条，接收数据
{
   QDataStream in(tcpServerConnection);
   in.setVersion(QDataStream::Qt_4_6);
   if(bytesReceived <= sizeof(qint64)*2)
   { //如果接收到的数据小于16个字节，那么是刚开始接收数据，我们保存到//来的头文件信息
       if((tcpServerConnection->bytesAvailable() >= sizeof(qint64)*2)
           && (fileNameSize == 0))
       { //接收数据总大小信息和文件名大小信息
           in >> totalBytes >> fileNameSize;
           bytesReceived += sizeof(qint64) * 2;
       }
       if((tcpServerConnection->bytesAvailable() >= fileNameSize)
           && (fileNameSize != 0))
       {  //接收文件名，并建立文件
           in >> fileName;
           ui->serverStatusLabel->setText(tr("接收文件 %1 ...")
                                          .arg(fileName));
           bytesReceived += fileNameSize;
           localFile= new QFile(fileName);
           if(!localFile->open(QFile::WriteOnly))
           {
                qDebug() << "open file error!";
                return;
           }
       }
       else return;
   }
   if(bytesReceived < totalBytes)
   {  //如果接收的数据小于总数据，那么写入文件
      bytesReceived += tcpServerConnection->bytesAvailable();
      inBlock= tcpServerConnection->readAll();
      localFile->write(inBlock);
      inBlock.resize(0);
   }
//更新进度条
   ui->serverProgressBar->setMaximum(totalBytes);
   ui->serverProgressBar->setValue(bytesReceived);

   if(bytesReceived == totalBytes)
   { //接收数据完成时
    tcpServerConnection->close();
    localFile->close();
```

```
    ui->startButton->setEnabled(true);
ui->serverStatusLabel->setText(tr("接收文件 %1 成功！")
.arg(fileName));
   }
}
```

（5）错误处理函数。

```
void Widget::displayError(QAbstractSocket::SocketError) //错误处理
{
    qDebug() << tcpServerConnection->errorString();
    tcpServerConnection->close();
    ui->serverProgressBar->reset();
    ui->serverStatusLabel->setText(tr("服务端就绪"));
    ui->startButton->setEnabled(true);
}
```

（6）我们在`widget.ui`中进入“开始监听”按钮的单击事件槽函数，更改如下。

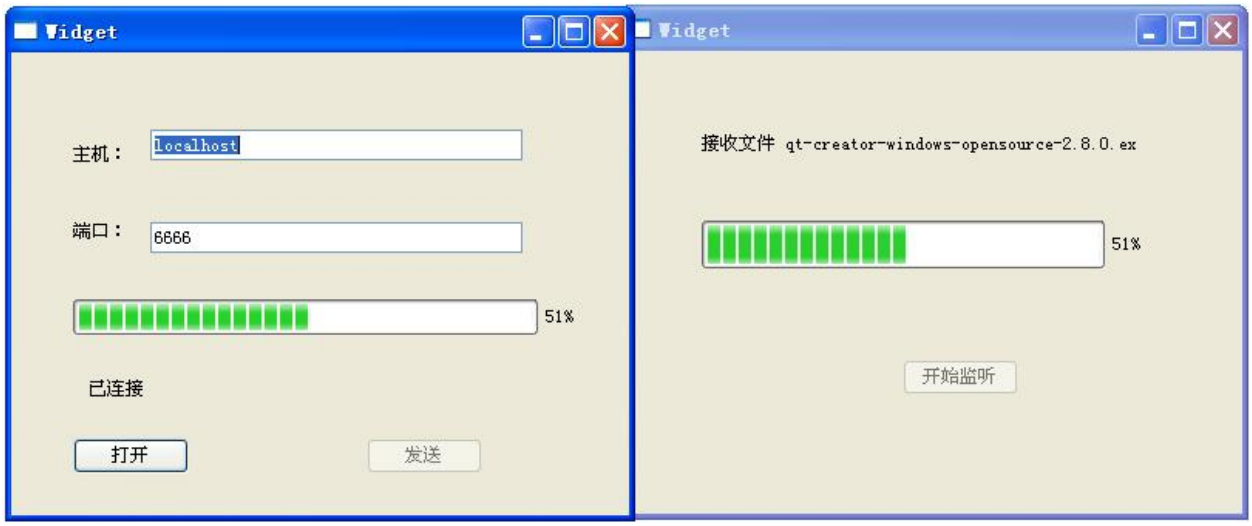
```
void Widget::on_startButton_clicked() //开始监听按钮
{
    start();
}
```

5．我们为了使程序中的中文不显示乱码，在`main.cpp`文件中更改。

添加头文件包含：`#include <QTextCodec>`

在`main`函数中添加代码：`QTextCodec::setCodecForTr(QTextCodec::codecForName("UTF-8"));`

6．运行程序，并同时运行`tcpSender`程序，效果如下。



我们先在服务器端按下“开始监听”按钮，然后在客户端输入主机地址和端口号，然后打开要发送的文件，点击“发送”按钮进行发送。