# Qt网络通信

## 课程主要内容

- IP地址与端口
- 使用网络模块
- TCP协议
- UDP协议

## IP地址

- 用于区分同一网络下不同设备的地址编号
- IPv4
  - 32位地址编号空间
  - 将32位分成4组，每组用十进制表示，组间用 . 分隔
- IPv6
  - 128位地址编号空间
  - 将128位分成8组，每组用十六进制表示，组间用 : 分隔
  - 连续的0用::表示
- 本地主机地址：127.0.0.1（IPv4）或::1（IPv6）

## 端口

- 区分数据应递交给哪个进程进行处理的编号
- 范围：0~65535（16字节）
  - 周知端口：0~1023

    | | |
    |---|---|
    | • FTP | 21 |
    | • WWW服务 | 80 |

  - 注册端口：1024~49151
  - 动态端口：49152~65535

# 套接字（Socket）

- 套接字 = 源IP + 源端口 + 目标IP + 目标端口 + 协议
- 监听套接字
  - 负责处理客户端的连接请求
  - 有特定的源IP和源端口
- 连接套接字
  - 负责处理连接后的数据传输
  - 使用数据流进行读写

## 使用网络模块

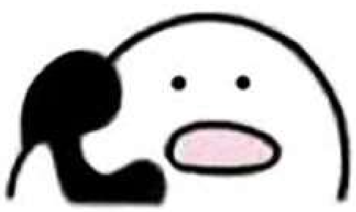
- 包含头文件<QtNetwork>
- 在.pro文件中添加Qt += network

# TCP协议

- 需要连接
  - 三次握手建立连接，四次挥手断开连接

# TCP连接建立

- 客户端发送连接请求，并发送随机信号数据Seq=x
- 服务端发送ACK=x+1和另一随机信号数据Seq=y
- 客户端检验x+1并发送ACK=y+1
- 服务端检验y+1，连接成功



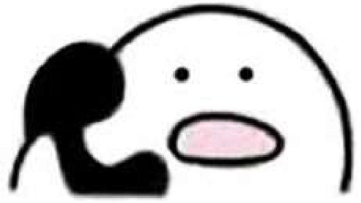
歪？ 听得到吗？

听到了，歪？ 你听得到我吗？

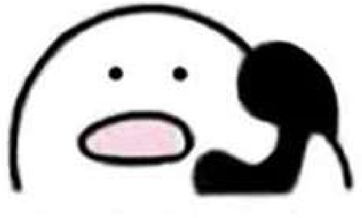
听到了

# TCP连接断开

- 客户端发送结束通知，并发送随机信号数据Seq=x
- 服务端发送ACK=x+1
- 客户端检验x+1
- 服务端发送断开请求，并发送随机信号数据Seq=y
- 客户端发送ACK=y+1，**等待一段时间无回复后**释放端口
- 服务端检验y+1，释放端口



我说完了



哦

---

挂了，掰掰

我也说完了，挂了，掰掰

# TCP协议

- 需要连接
  - 三次握手建立连接，四次握手断开连接
- 数据校验
- 强制答复
  - 接收端需答复发送端数据包接收情况
- 自动重发
  - 重发一定时间内未收到答复的数据包
- 丢弃重复

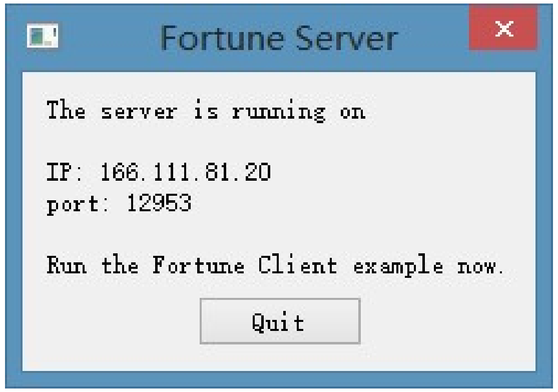
## TCP协议

- 编号排序
  - 发送端对数据包编号，接收端进行排序
- 流量控制
  - 当数据缓冲区空间不足时，通知发送方暂缓传输
- 紧急标志
  - 通知接收端提高该数据包的优先级（而不是按照编号顺序处理）

# TCP服务端

- 服务端类Server

```
class Server : public Qdialog {
   Q_OBJECT
public:
   explicit Server(QWidget *parent = Q_NULLPTR);
private slots:
   void sessionOpened();
   void sendFortune();
private:
   QLabel *statusLabel;
   QTcpServer *tcpServer;
   QStringList fortunes;
   …
};
```



```
The server is running on

IP: 166.111.81.20
port: 12953

Run the Fortune Client example now.
```

# TCP服务端

- Server构造函数

```
Server::Server(QWidget *parent)
    : QDialog(parent)
    , statusLabel(new QLabel)
    , tcpServer(Q_NULLPTR)
{
    …
}
```

## TCP服务端

- 创建QTcpServer，监听指定端口

```
{
    …
    tcpServer = new QTcpServer(this);
    if (!tcpServer->listen()) {
        QMessageBox::critical(this, tr("Fortune Server"), tr("Unable to start the server: %1.").arg(tcpServer->errorString()));
        close();
        return;
    }
    …
}
```

## TCP服务端

- 查找服务端自动监听的端口并显示在界面中

```
{  …
   QString ipAddress;
   QList<QHostAddress> ipAddressesList = QNetworkInterface::allAddresses();
   for (int i = 0; i < ipAddressesList.size(); ++i) {     // use the first non-localhost IPv4
address
      if (ipAddressesList.at(i) != QHostAddress::LocalHost &&
ipAddressesList.at(i).toIPv4Address()) {
         ipAddress = ipAddressesList.at(i).toString(); break; }
   }
   if (ipAddress.isEmpty())     // if we did not find one, use IPv4 localhost
      ipAddress = QHostAddress(QHostAddress::LocalHost).toString();
   statusLabel->setText(tr("The server is running on\n\nIP: %1\nport: %2\n\nRun
the Fortune Client example now.").arg(ipAddress).arg(tcpServer->serverPort()));
   …
}
```

## TCP服务端

- 将一些fortune存入“fortunes”变量

```
{
    …
    fortunes << tr("You've been leading a dog's life. Stay off the furniture.")
            << tr("You've got to think about tomorrow.")
            << tr("You will be surprised by a loud noise.")
            << tr("You will feel hungry again in another hour.")
            << tr("You might have mail.")
            << tr("You cannot kill time without injuring eternity.")
            << tr("Computers are not intelligent. They only think they are.");
    …
}
```

# TCP服务端

- 将控件添加到界面中

```
{
    …
    QPushButton *quitButton = new QPushButton(tr("Quit"));
    quitButton->setAutoDefault(false);
    connect(quitButton, &QAbstractButton::clicked, this, &QWidget::close);
    QHBoxLayout *buttonLayout = new QHBoxLayout;
    buttonLayout->addWidget(quitButton);
    QVBoxLayout *mainLayout = new QVBoxLayout(this);
    mainLayout->addWidget(statusLabel);
    mainLayout->addLayout(buttonLayout);
    …
}
```

# TCP服务端

- 等待客户端连接，处理newConnection信号（处理函数sendFortune）

```
{
    …
    connect(tcpServer, &QTcpServer::newConnection,
this, &Server::sendFortune);
    …
}
```

# TCP服务端

- 处理客户端连接，向客户端发送数据：使用QDataStream向QByteArray中写入数据

```
void Server:: sendFortune() {
    QByteArray block;
    QDataStream out(&block, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_0);
    out << fortunes.at(qrand() % fortunes.size());
    ……
}
```

## TCP服务端

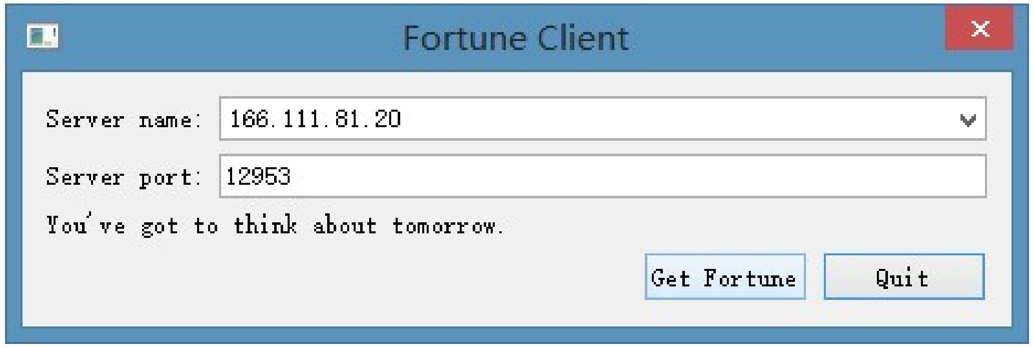
- 处理客户端连接，向客户端发送数据：创建QTcpSocket，使用write函数发送数据，使用disconnectFromHost关闭连接

```
void Server:: sendFortune() {
    ……
    QTcpSocket *clientConnection
       = tcpServer->nextPendingConnection();
    connect(clientConnection, &QAbstractSocket::disconnected,
          clientConnection, &QObject::deleteLater);
    clientConnection->write(block);
    clientConnection->disconnectFromHost();
}
```

# TCP客户端

- 客户端类Client

```
class Client : public QDialog{
    Q_OBJECT
public:
    explicit Client(QWidget *parent = Q_NULLPTR);
private slots:
    void requestNewFortune();
    void readFortune();
    …
};
```



Server name: 166.111.81.20

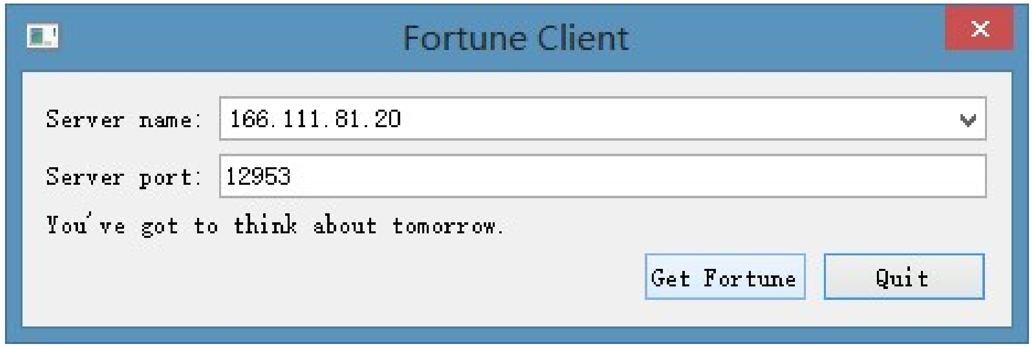
Server port: 12953

You've got to think about tomorrow.

# TCP客户端

- 客户端类Client

```
class Client : public QDialog{
    …
private:
    QComboBox *hostCombo;
    QLineEdit *portLineEdit;
    QLabel *statusLabel;
    QPushButton *getFortuneButton;
    QTcpSocket *tcpSocket;
    QDataStream in;
    QString currentFortune;
};
```



Fortune Client

Server name: 166.111.81.20

Server port: 12953

You've got to think about tomorrow.

# TCP客户端

- Client构造函数

```
Client::Client(QWidget *parent)
    : QDialog(parent)
    , hostCombo(new QComboBox)
    , portLineEdit(new QLineEdit)
    , getFortuneButton(new QPushButton(tr("Get Fortune")))
    , tcpSocket(new QTcpSocket(this))
{
    …
}
```

# TCP客户端

- 将数据流绑定到socket

```
{
    …
    in.setDevice(tcpSocket);
    in.setVersion(QDataStream::Qt_4_0);
    …
}
```

# TCP客户端

- 添加本机的IP

```
{
    …
    // find out IP addresses of this machine
    QList<QHostAddress> ipAddressesList = QNetworkInterface::allAddresses();
    for (int i = 0; i < ipAddressesList.size(); ++i) {      // add non-localhost addresses
        if (!ipAddressesList.at(i).isLoopback())
            hostCombo->addItem(ipAddressesList.at(i).toString());
    }
    for (int i = 0; i < ipAddressesList.size(); ++i) {      // add localhost addresses
        if (ipAddressesList.at(i).isLoopback())
            hostCombo->addItem(ipAddressesList.at(i).toString());
    }
    …
}
```

# TCP客户端

- 设置界面中的控件

```
{
    …
    portLineEdit->setValidator(new QIntValidator(1, 65535, this));
    QLabel *hostLabel = new QLabel(tr("Server &name:"));
    hostLabel->setBuddy(hostCombo);
    QLabel *portLabel = new QLabel(tr("Server &port:"));
    portLabel->setBuddy(portLineEdit);
    statusLabel = new QLabel(tr("This examples requires that you run
the Fortune Server example as well."));
    …
}
```

# TCP客户端

- 设置界面中的控件

```
{
    …
    QPushButton *quitButton = new QPushButton(tr("Quit"));
    QDialogButtonBox *buttonBox = new QDialogButtonBox;
    buttonBox->addButton(getFortuneButton,
QDialogButtonBox::ActionRole);
    buttonBox->addButton(quitButton,
QDialogButtonBox::RejectRole);
    connect(quitButton, &QAbstractButton::clicked, this,
&QWidget::close);
    …
}
```

# TCP客户端

- 单击按钮请求数据（请求函数 requestNewFortune）

```
{
    …
    connect(
            getFortuneButton, &QAbstractButton::clicked,
            this, &Client::requestNewFortune);
    …
}
```

## TCP客户端

- 等待新数据传入，处理readyRead信号（处理函数readFortune）

```
{
    …
    connect(tcpSocket, &QIODevice::readyRead, this,
&Client::readFortune);
    …
}
```

# TCP客户端

- 将控件添加到界面中

```
{
    …
    QGridLayout *mainLayout = new QGridLayout(this);
    mainLayout->addWidget(hostLabel, 0, 0);
    mainLayout->addWidget(hostCombo, 0, 1);
    mainLayout->addWidget(portLabel, 1, 0);
    mainLayout->addWidget(portLineEdit, 1, 1);
    mainLayout->addWidget(statusLabel, 2, 0, 1, 2);
    mainLayout->addWidget(buttonBox, 3, 0, 1, 2);
    …
}
```

## TCP客户端

- 向服务器请求数据

```
void Client::requestNewFortune()
{
   tcpSocket->abort();
   tcpSocket->connectToHost(hostCombo->currentText(), portLineEdit->text().toInt());
}
```

# TCP客户端

- 接收服务器的数据

```
void Client:: receiveFromServer()
{
    in.startTransaction();
    QString nextStr;
    in >> nextStr;
}
```

# TCP实例

- 参见Fortune Server Example和Fortune Client Example

## UDP协议

- 直接读写，无连接
- 不可靠

# UDP发送端

- 新建一个QUdpSocket

```
void Client::initSocket() {
    udpSocket = new QUdpSocket(this);
}
```

## UDP发送端

- 直接使用writeDatagram写数据，发送端口由操作系统分配

```
  void Client::writeSocket() {
      char data[20] = “hello, world”
      udpSocket->writeDatagram(data, 20,
QHostAddress::LocalHost, 7755);
  }
```

## UDP接收端

- 新建一个QUdpSocket，绑定一个端口，创建一个处理信号的槽

```
void Server::initSocket()
 {
    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress::LocalHost, 7755);
    connect(udpSocket, SIGNAL(readyRead()),
            this, SLOT(readPendingDatagrams()));
 }
```

# UDP接收端

- 将QUdpSocket的readyRead信号连接到槽，在槽中用receiveDatagram读数据

```
 void Server::readPendingDatagrams()
  {
     while (udpSocket->hasPendingDatagrams()) {
         QNetworkDatagram datagram = udpSocket-
>receiveDatagram();
         processTheDatagram(datagram);
     }
  }
```

## 课后思考题

- 阅读Qt文档，如果需要用指定端口发送UDP数据包，应该怎样做？
- 断开TCP连接时，为什么客户端要等待一段时间无回复后再释放端口？
- TCP和UDP能否共享一个端口？