

UDP概述

- **UDP** 是**User Datagram Protocol**的简称，中文名是用户数据报协议，是**OSI（Open System Interconnection，开放式系统互联）**参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。**UDP**在**IP**报文的协议号是**17**。
- **UDP**协议与**TCP**协议一样用于处理数据包，在**OSI**模型中，两者都位于传输层，处于**IP**协议的上一层。**UDP**有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。**UDP**用来支持那些需要在计算机之间传输数据的网络应用。包括网络视频会议系统在内的众多的客户/服务器模式的网络应用都需要使用**UDP**协议。**UDP**协议从问世至今已经被使用了很多年，虽然其最初的光彩已经被一些类似协议所掩盖，但即使在今天**UDP**仍然不失为一项非常实用和可行的网络传输层协议。
- 许多应用只支持**UDP**，如：多媒体数据流，不产生任何额外的数据，即使知道有破坏的包也不进行重发。当强调传输性能而不是传输的完整性时，如：音频和多媒体应用，**UDP**是最好的选择。在数据传输时间很短，以至于此前的连接过程成为整个流量主体的情况下，**UDP**也是一个好的选择。
- **UDP（User Datagram Protocol即用户数据报协议）**是一个轻量级的，不可靠的，面向数据报的无连接协议。腾讯**QQ**，其聊天时就是使用**UDP**协议进行消息发送的。就像**QQ**那样，当有很多用户，发送的大部分都是短消息，要求能及时响应，并且对安全性要求不是很高的情况下使用**UDP**协议。

QUdpSocket 类

- 在**Qt**中提供了**QUdpSocket** 类来进行**UDP**数据报（**datagrams**）的发送和接收。这里我们还要了解一个名词**Socket**，也就是常说的“套接字”。**Socket**简单地说，就是一个**IP**地址加一个**port**端口。因为我们要传输数据，就要知道往哪个机子上传送，而**IP**地址确定了一台主机，但是这台机子上可能运行着各种各样的网络程序，我们要往哪个程序中发送呢？这时就要使用一个端口来指定**UDP**程序。所以说，**Socket**指明了数据报传输的路径。

使用方法

- 首先在工程文件中添加库。源文件引用头文件

```
1 QT += network
```

```
2 #include <QUdpSocket>
```

- 使用这个类最常见的方法是使用**bind()**绑定到地址和端口，然后调用**writeDatagram()**和**readDatagram()**来传输数据。如果您想使用标准的**QIODevice**函数**read()**、**readLine()**、**write()**等，您必须首先通过调用**connectToHost()**将套接字直接连接到对等端。
- 每次将数据报写入网络时，套接字都会发出**byteswrite()**信号。如果只想发送数据报，则不需要调用**bind()**。
- **readyRead()**信号在数据报到达时发出。在这种情况下，**hasPendingDatagrams()**返回**true**。调用**pendingDatagramSize()**获取第一个挂起的数据报的大小，并调用**readDatagram()**读取它。
- 注意:当您收到**readyRead()**信号时，应该读取传入的数据报，否则将不会为下一个数据报发出此信号。

C/S（客户端/服务器）编程模型示例



```
1 //服务端sender
2 Widget::Widget(QWidget *parent) :QWidget(parent),
3   sender(new QUdpSocket),
4   pushButton(new QPushButton),
5   textEdit(new QTextEdit)
```

```

6  {
7      connect(pushButton,SIGNAL(clicked()),this,SLOT(pushButton_click
      On()));
8      pushButton->setText("点击发送");
9      resize(200,200);
10     QGridLayout* gridLayout=new QGridLayout;
11     gridLayout->addWidget(textEdit,0,0);
12     gridLayout->addWidget(pushButton,1,0);
13     setLayout(gridLayout);
14 }
15 void Widget::pushButton_clickOn(){
16     //这里定义了一个QByteArray类型的数据报datagram
17     //QByteArray 类似于const char* 是8位字节数组
18     QByteArray datagram=textEdit->toPlainText().toLocal8Bit();
19     //writeDatagram()函数来发送数据报
20     //函数有四个参数，分别是数据报的内容，数据报的大小，主机地址和端口号
21     //这里使用了广播地址QHostAddress::Broadcast，同时给网络中所有的主
    机发送数据报
22     //端口号，它是可以随意指定的，最大为65535
23     sender->writeDatagram(datagram.data(),datagram.size(),
24     QHostAddress::Broadcast,454545);
25 }
26
27 //客户端receiver
28 Widget::Widget(QWidget *parent) :
29     QWidget(parent),
30     receiver(new QUdpSocket(this)),
31     textEdit(new QTextEdit)
32 {
33     //绑定到与服务端相同得端口
34     //QUdpSocket::ShareAddress表明其他服务也可以绑定到这个端口上
35     //receiver发现有数据报到达时就会发出readyRead()信号
36     receiver->bind(454545,QUdpSocket::ShareAddress);
37     connect(receiver,SIGNAL(readyRead()),this,SLOT(processPendingD
    atagram()));
38

```

```

39  resize(600,600);
40  QGridLayout* gridLayout=new QGridLayout;
41  gridLayout->addWidget(textEdit,0,0);
42  setLayout(gridLayout);
43  }
44  void Widget::processPendingDatagram(){
45      //拥有等待的数据报
46      while(receiver->hasPendingDatagrams()){
47          //存放接收的数据报
48          QByteArray datagram;
49          //让datagram的大小为等待处理的数据报的大小，这样才能接收到完整的数据
50          datagram.resize(receiver->pendingDatagramSize());
51          //接收数据报，将其存放到datagram中
52          receiver->readDatagram(datagram.data(),datagram.size());
53          //将QByteArray装换成QString类型
54          textEdit->append(static_cast<QString>(datagram));
55      }
56  }

```

其他使用

- **QUdpSocket**也支持**UDP**多播。使用**joinMulticastGroup()**和**leaveMulticastGroup()**来控制组成员关系，使用**QAbstractSocket::multicastOption**和**QAbstractSocket::MulticastLoopbackOption**来设置**TTL**和**loopback socket**选项。使用**setMulticastInterface()**来控制多播数据报的传出接口，使用**multicastInterface()**来查询它。
- 使用**QUdpSocket**，您还可以使用**connectToHost()**建立到**UDP**服务器的虚拟连接，然后使用**read()**和**write()**来交换数据报，而不需要为每个数据报指定接收方。