

风味鱼

站在巨人的肩膀上

博客园 首页 新随笔 联系 管理 订阅

随笔- 133 文章- 0 评论- 5

昵称： 风味鱼
园龄： 3年5个月
粉丝： 18
关注： 0
[+加关注](#)

界面编程之QT的Socket通信20180730

/*****
*****/

一、linux下的tcp通过程程

其中bind绑定，会固定一个端口， 否则是随机的。

一个链接是由双方的ip和端口组成的，固定端口保证源的不变性，

这样另一端在任何时候访问的目的都是一致的，也可以说这个端口提供了什么服务。

同时绑定后直接操作socket id就可以操作对应的链接了。

/*****
*****/

二、QT下的TCP通过程程

Qt中提供的所有的Socket类都是非阻塞的。

Qt中常用的用于socket通信的套接字类：

- QTcpServer
- 用于TCP/IP通信，作为服务器端套接字使用
- QTcpSocket
- 用于TCP/IP通信，作为客户端套接字使用。
- QUdpSocket
- 用于UDP通信，服务器，客户端均使用此套接字。

1.QT下的服务端

- 1).socket函数变为QTcpServer
- 2).bind ,listen 统一为listen
- 同时没有accept，当有一个链接过来的时候，会产生一个信号： newconnection， 可以从对应的槽函数中取出建立好的套接字(对方的)QTcpSocket

如果成功和对方建立好链接，通信套接字会自动触发connected信号

3).read :

对方发送数据过来，链接的套接字(通信套接字)就会触发(本机的)readyRead信号，需要在对应的槽函数中接收数据

- 4).write,
- 发送数据,对方的(客户端的)套接字(通信套接字)就会触发readyRead信号，需要在对应的槽函数中接收数据

如果对方主动断开连接，对方的(客户端的)套接字(通信套接字)会自动触发disconnected信号

2.QT下的客户端：

<	2019年11月						>
日	一	二	三	四	五	六	
27	28	29	30	31	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
1	2	3	4	5	6	7	

搜索

常用链接

- [我的随笔](#)
- [我的评论](#)
- [我的参与](#)
- [最新评论](#)
- [我的标签](#)

最新随笔

- 1.音视频处理之PS封装的介绍与使用20180928
- 2.界面编程之QT的数据库操作20180801
- 3.界面编程之QT的线程20180731
- 4.界面编程之QT的Socket通信20180730
- 5.界面编程之QT的文件操作20180729
- 6.界面编程之QT绘图和绘图设备20180728
- 7.界面编程之QT的事件20180727
- 8.界面编程之QT窗口系统20180726
- 9.界面编程之QT的信号与槽20180725
- 10.界面编程之QT的基本介绍与使用20180722

我的标签

- [Linux\(22\)](#)
- [设计模式\(17\)](#)
- [音视频\(10\)](#)
- [音视频处理\(10\)](#)
- [界面编程\(9\)](#)
- [流媒体\(9\)](#)
- [QT\(9\)](#)
- [驱动\(6\)](#)
- [C++\(5\)](#)
- [Java\(5\)](#)
- [更多](#)

1).socket函数变为 QTcpSocket

2).connect变为connectToHost()

如果成功和对方建立好链接, 就会自动触发connected信号

3).read :

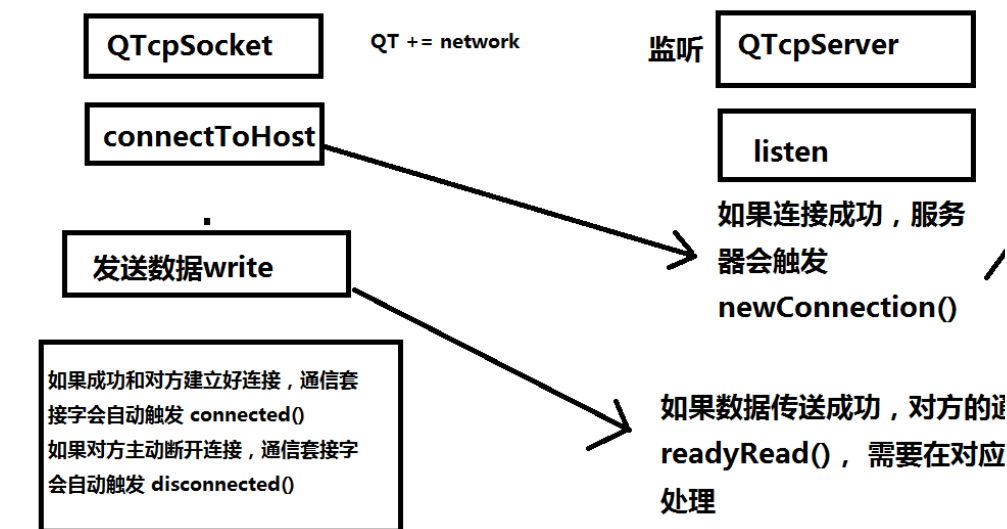
对方发送数据过来, 链接的套接字(通信套接字)就会触发(本机的)readyRead信号, 需要在对应的槽函数中接收数据

4).write,

发送数据,对方的(服务器的)套接字(通信套接字)就会触发readyRead信号, 需要在对应的槽函数中接收数据

如果对方主动断开连接, 就会自动触发disconnected信号

具体见图《QtTCP通信过程》



```

/*****
*****/

```

三、TCP服务器

Qwidget是基类, 比较干净, QMainWindow相对比较多。

如果输入头文件没有提示, 就需要在项目文件中加入对应模块, 同时再编译不运行一下, 让qt可以构建并加载对应的模块。

```

#include <QTcpServer> //监听套接字
#include <QTcpSocket> //通信套接字//对方的(客户端的)套接字(通信套接字)

```

```

//监听套接字, 指定父对象, 让其自动回收空间

```

```

tcpServer = new QTcpServer(this);
tcpServer->listen(QHostAddress::Any, 8888);

```

```

setWindowTitle("服务器: 8888");

```

```

connect(tcpServer, &QTcpServer::newConnection,
 [=]()//信号无参数, 这里也没有参数
{
    //取出建立好连接的套接字
    tcpSocket = tcpServer->nextPendingConnection();
}

```

随笔分类

[Android\(1\)](#)
[C\(7\)](#)
[C++\(5\)](#)
[DigitalPhotoFrame](#)
[Java\(5\)](#)
[Linux\(26\)](#)
[uC/OS-II\(3\)](#)
[Web\(4\)](#)
[程序工具](#)
[界面编程\(9\)](#)
[流媒体协议\(14\)](#)
[汽车电子\(3\)](#)
[驱动基础\(7\)](#)
[设计理念\(18\)](#)
[数据结构与算法\(1\)](#)
[数据库\(4\)](#)
[网络协议\(6\)](#)
[消费电子之智能家居IPC\(3\)](#)
[音视频处理\(24\)](#)

随笔档案

[2019年4月\(1\)](#)
[2019年2月\(11\)](#)
[2019年1月\(5\)](#)
[2018年12月\(1\)](#)
[2018年11月\(1\)](#)
[2018年10月\(2\)](#)
[2018年9月\(1\)](#)
[2018年8月\(1\)](#)
[2018年7月\(10\)](#)
[2018年6月\(3\)](#)
[2018年5月\(1\)](#)
[2018年4月\(2\)](#)
[2018年3月\(4\)](#)
[2018年2月\(3\)](#)
[2018年1月\(1\)](#)
[2017年12月\(1\)](#)
[2017年11月\(1\)](#)
[2017年10月\(1\)](#)
[2017年9月\(10\)](#)
[2017年8月\(6\)](#)
[2017年7月\(17\)](#)
[2017年6月\(5\)](#)
[2017年5月\(1\)](#)
[2017年3月\(5\)](#)
[2017年2月\(6\)](#)
[2017年1月\(1\)](#)
[2016年9月\(1\)](#)
[2016年8月\(3\)](#)
[2016年7月\(11\)](#)
[2016年6月\(8\)](#)
[2016年5月\(9\)](#)

最新评论

1. Re:流媒体协议之RTSP客户端的实现20171014

--13654915480

2. Re:Linux之SSL安全套接字20160704

这篇文章写的很细致, 以前不明白的问题现在明白了, 看了这篇文章后, 我又继续听了这个课程, 加深了对这个技术的理解, 所以把课程推荐给有这方面疑惑的小伙伴, 希望能帮到大家! 一个系统、全面讲解传统加密通信SSL...

--合月

3. Re:网络协议之mDNS20170217

@ 风味鱼楼主, 我的意思是程序进入了 发现服务 这个条件, 在发现局域网内有回应的服务的时候, 如果有设备掉电了, 这个发现服务程序还是存在掉电设备所提供的服务, 该如何处理...

```
//获取对方的IP和端口
QString ip = tcpSocket->peerAddress().toString();

qint16 port = tcpSocket->peerPort();

QString temp = QString("[%1:%2]:成功连接").arg(ip).arg(port);

ui->textEditRead->setText(temp);

//必须放在里面，因为建立好链接才能读，或者说tcpSocket有指向才能操作
connect(tcpSocket, &QTcpSocket::readyRead,
        [=]()
        {
            //从通信套接字中取出内容
            QByteArray array = tcpSocket->readAll();
            ui->textEditRead->append(array);
        }
        );
```

```
void ServerWidget::on_buttonSend_clicked()
{
    if(NULL == tcpSocket)
    {
        return;
    }
    //获取编辑区内容
    QString str = ui->textEditWrite->toPlainText();
    //给对方发送数据， 使用套接字是tcpSocket
    tcpSocket->write( str.toUtf8().data() );

}

void ServerWidget::on_buttonClose_clicked()
{
    if(NULL == tcpSocket)
    {
        return;
    }
    //主动和客户端断开连接
    tcpSocket->disconnectFromHost();
    tcpSocket->close();
    tcpSocket = NULL;
}
```

```
/*
*****
*****
*/
```

四、TCP客户端

可以在项目中添加新文件中选择Qt--->Qt设计师界面类(这个是带ui的)，选择这个后项目会多出一个ui

--Psrion

[4. Re:网络协议之mDNS20170217](#)

@ Psrion试试注册之前，执行注销处理看看...

--风味鱼

[5. Re:网络协议之mDNS20170217](#)

请问，如果一个设备掉电，它所注册的服务又如何处理呢？？注意这里，掉电之前并没有将服务注销，因此掉电的时候如果查询当前的所有服务，会发现之前掉电设备的服务依然存在，如何处理呢？

--Psrion

阅读排行榜

- [1. GSM之AT操作命令详解20160615\(21451\)](#)
- [2. USB驱动之CDC类的介绍与应用20160905\(15142\)](#)
- [3. 界面编程之QT的Socket通信20180730\(9766\)](#)
- [4. 网络协议之mDNS20170217\(9331\)](#)
- [5. Java高级应用之泛型与反射20170627\(5504\)](#)

评论排行榜

- [1. 网络协议之mDNS20170217\(3\)](#)
- [2. 流媒体协议之RTSP客户端的实现20171014\(1\)](#)
- [3. Linux之SSL安全套接字20160704\(1\)](#)

推荐排行榜

- [1. 华为C语言编程规范\(3\)](#)

```
ui->setupUi(this); //显示ui

tcpSocket = NULL;

//分配空间, 指定父对象
tcpSocket = new QTcpSocket(this);

setWindowTitle("客户端");

connect(tcpSocket, &QTcpSocket::connected,
        [=]()
        {
            ui->textEditRead->setText("成功和服务端建立好连接");
        }
        );

//因为tcpSocket已经分配了空间, 有指向, 所以可以放在外面
connect(tcpSocket, &QTcpSocket::readyRead,
        [=]()
        {
            //获取对方发送的内容
            QByteArray array = tcpSocket->readAll();
            //追加到编辑区中
            ui->textEditRead->append(array);
        }
        );

void ClientWidget::on_buttonConnect_clicked()
{
    //获取服务器ip和端口
    QString ip = ui->lineEditIP->text();
    quint16 port = ui->lineEditPort->text().toInt();

    //主动和服务端建立连接
    tcpSocket->connectToHost(QHostAddress(ip), port);
}

void ClientWidget::on_buttonSend_clicked()
{
    //获取编辑框内容
    QString str = ui->textEditWrite->toPlainText();
    //发送数据
    tcpSocket->write( str.toUtf8().data() );
}

void ClientWidget::on_buttonClose_clicked()
{
    //主动和对方断开连接
```

```

tcpSocket->disconnectFromHost();

tcpSocket->close();//这里释放连接, 前面connect的时候会建立连接
}

```

```

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    ServerWidget w;

    w.show();


    ClientWidget w2;

    w2.show();//显示另外一个窗口

    return a.exec();
}

```



上述代码具体见《TCP》



```

1 #ifndef SERVERWIDGET_H
2 #define SERVERWIDGET_H
3
4 #include <QWidget>
5 #include <QTcpServer> //监听套接字
6 #include <QTcpSocket> //通信套接字
7
8 namespace Ui {
9 class ServerWidget;
10 }
11
12 class ServerWidget : public QWidget
13 {
14     Q_OBJECT
15
16 public:
17     explicit ServerWidget(QWidget *parent = 0);
18     ~ServerWidget();
19
20 private slots:
21     void on_buttonSend_clicked();
22
23     void on_buttonClose_clicked();
24
25 private:
26     Ui::ServerWidget *ui;
27
28     QTcpServer *tcpServer; //监听套接字
29     QTcpSocket *tcpSocket; //通信套接字
30
31 };
32
33 #endif // SERVERWIDGET_H

```

```

1 #include "serverwidget.h"
2 #include "ui_serverwidget.h"
3
4 ServerWidget::ServerWidget(QWidget *parent) :
5     QWidget(parent),
6     ui(new Ui::ServerWidget)
7 {
8     ui->setupUi(this);

```

```
9
10 tcpServer = NULL;
11 tcpSocket = NULL;
12
13 //监听套接字, 指定父对象, 让其自动回收空间
14 tcpServer = new QTcpServer(this);
15
16 tcpServer->listen(QHostAddress::Any, 8888);
17
18 setWindowTitle("服务器: 8888");
19
20 connect(tcpServer, &QTcpServer::newConnection,
21         [=] ()
22         {
23             //取出建立好连接的套接字
24             tcpSocket = tcpServer->nextPendingConnection();
25
26             //获取对方的IP和端口
27             QString ip = tcpSocket->peerAddress().toString();
28             quint16 port = tcpSocket->peerPort();
29             QString temp = QString("%1:%2:成功连接").arg(ip).arg(port);
30
31             ui->textEditRead->setText(temp);
32
33             connect(tcpSocket, &QTcpSocket::readyRead,
34                     [=] ()
35                     {
36                         //从通信套接字中取出内容
37                         QByteArray array = tcpSocket->readAll();
38                         ui->textEditRead->append(array);
39                     }
40
41                     );
42
43
44         }
45
46         );
47
48 }
49
50 ServerWidget::~ServerWidget()
51 {
52     delete ui;
53 }
54
55 void ServerWidget::on_buttonSend_clicked()
56 {
57     if(NULL == tcpSocket)
58     {
59         return;
60     }
61     //获取编辑区内容
62     QString str = ui->textEditWrite->toPlainText();
63     //给对方发送数据, 使用套接字是tcpSocket
64     tcpSocket->write( str.toUtf8().data() );
65
66 }
67
68 void ServerWidget::on_buttonClose_clicked()
69 {
70     if(NULL == tcpSocket)
71     {
72         return;
73     }
74
75     //主动和客户端端口连接
76     tcpSocket->disconnectFromHost();
77     tcpSocket->close();
78     tcpSocket = NULL;
79 }
```



```

1 #ifndef CLIENTWIDGET_H
2 #define CLIENTWIDGET_H
3
4 #include <QWidget>
5 #include <QTcpSocket> //通信套接字
6
7 namespace Ui {
8 class ClientWidget;
9 }
10
11 class ClientWidget : public QWidget
12 {
13     Q_OBJECT
14
15 public:
16     explicit ClientWidget(QWidget *parent = 0);
17     ~ClientWidget();
18
19 private slots:
20     void on_buttonConnect_clicked();
21
22     void on_buttonSend_clicked();
23
24     void on_buttonClose_clicked();
25
26 private:
27     Ui::ClientWidget *ui;
28
29     QTcpSocket *tcpSocket; //通信套接字
30 };
31
32 #endif // CLIENTWIDGET_H

```



```

1 #include "clientwidget.h"
2 #include "ui_clientwidget.h"
3 #include <QHostAddress>
4
5 ClientWidget::ClientWidget(QWidget *parent) :
6     QWidget(parent),
7     ui(new Ui::ClientWidget)
8 {
9     ui->setupUi(this);
10
11     tcpSocket = NULL;
12
13     //分配空间, 指定父对象
14     tcpSocket = new QTcpSocket(this);
15
16     setWindowTitle("客户端");
17
18
19     connect(tcpSocket, &QTcpSocket::connected,
20             [=] ()
21             {
22                 ui->textEditRead->setText("成功和服务器建立好连接");
23             }
24             );
25
26     connect(tcpSocket, &QTcpSocket::readyRead,
27             [=] ()
28             {
29                 //获取对方发送的内容
30                 QByteArray array = tcpSocket->readAll();
31                 //追加到编辑区中
32                 ui->textEditRead->append(array);
33             }
34             );
35
36
37 }
38
39 ClientWidget::~~ClientWidget()

```

```

40 {
41     delete ui;
42 }
43
44 void ClientWidget::on_buttonConnect_clicked()
45 {
46     //获取服务器ip和端口
47     QString ip = ui->lineEditIP->text();
48     qint16 port = ui->lineEditPort->text().toInt();
49
50     //主动和服务器建立连接
51     tcpSocket->connectToHost(QHostAddress(ip), port);
52
53 }
54
55 void ClientWidget::on_buttonSend_clicked()
56 {
57     //获取编辑框内容
58     QString str = ui->textEditWrite->toPlainText();
59     //发送数据
60     tcpSocket->write(str.toUtf8().data());
61
62 }
63
64 void ClientWidget::on_buttonClose_clicked()
65 {
66     //主动和对方断开连接
67     tcpSocket->disconnectFromHost();
68     tcpSocket->close();
69 }

```

```

/*****
*****/

```

五、UDP通信过程

使用Qt提供的QUdpSocket进行UDP通信。在UDP方式下，客户端并不与服务器建立连接，它只负责调用发送函数向服务器发送数据。

类似的服务器也不从客户端接收连接，只负责调用接收函数，等待来自客户端的数据的到达。

在UDP通信中，服务器端和客户端的概念已经显得有些淡化，两部分做的工作都大致相同

1. QT下的服务端

socket函数变为QUdpSocket

bind ,还是bind,(固定端口，让别人可以知道往哪里发，客户端也可以绑定)

readDatagram :

对方发送数据过来，套接字就会触发readyRead信号，需要在对应的槽函数中接收数据

writeDatagram,

发送数据,对方的(客户端的)套接字就会触发readyRead信号，需要在对应的槽函数中接收数据

close 还是close

2. QT下的客户端:

socket函数变为 QUdpSocket

readDatagram :

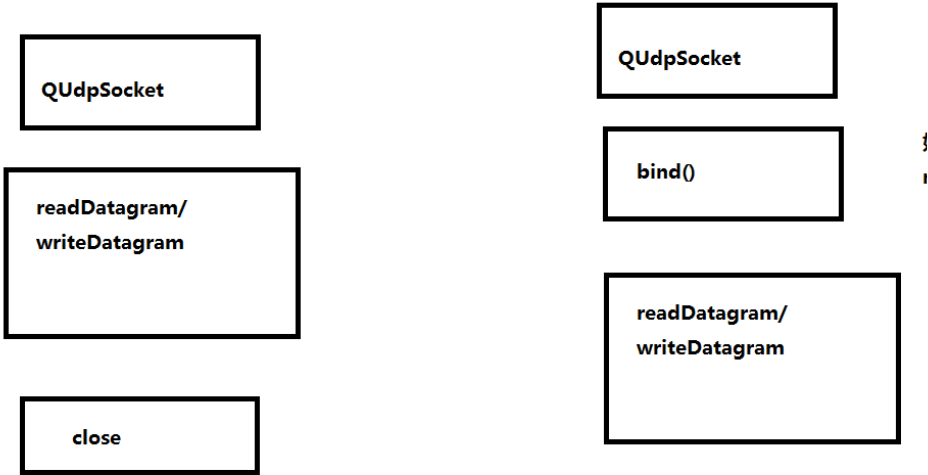
对方发送数据过来，套接字就会触发readyRead信号，需要在对应的槽函数中接收数据

writeDatagram,

发送数据,对方的(客户端的)套接字就会触发readyRead信号，需要在对应的槽函数中接收数据

close 还是close

具体见图《QtUDP通信用途》



```

/*****
*****/

```

六、UDP文本发送

UDP中没有严格的区分服务端和客户端。

关闭按钮是用于关闭窗口的，这主要是由于udp不是面向连接的，没有断开连接的说法。

```
#include <QUdpSocket> //UDP套接字

//分配空间，指定父对象，这是为了让父对象来回收，其实也可以不用指定，自己来回收资源也行
udpSocket = new QUdpSocket(this);

//绑定
udpSocket->bind(8888);

setWindowTitle("服务器端口为： 8888");

//当对方成功发送数据过来
//自动触发 readyRead()
connect(udpSocket, &QUdpSocket::readyRead, this, &Widget::dealMsg);

void Widget::dealMsg()
{
    //读取对方发送的内容
    char buf[1024] = {0};
    QHostAddress cliAddr; //对方地址
    quint16 port; //对方端口
    qint64 len = udpSocket->readDatagram(buf, sizeof(buf), &cliAddr, &port);
    if(len > 0)
```

```

{
    //格式化 [192.68.2.2:8888]aaaa
    QString str = QString("[%1:%2] %3")
        .arg(cliAddr.toString())
        .arg(port)
        .arg(buf);
    //给编辑区设置内容
    ui->textEdit->setText(str);
}
}
//发送按钮
void Widget::on_buttonSend_clicked()
{
    //先获取对方的IP和端口
    QString ip = ui->lineEditIP->text();
    quint16 port = ui->lineEditPort->text().toInt();

    //获取编辑区内容
    QString str = ui->textEdit->toPlainText();

    //给指定的IP发送数据
    udpSocket->writeDatagram(str.toUtf8(), QHostAddress(ip), port);
}

/*****
*****/

```

七、UDP多播组播

1.广播

广播地址：255.255.255.255,在某个局域网上就自动会变为那个局域网的广播地址，如果指定了

是某个局域网的广播地址如：192.168.1.255，则只能在这个局域网192.168.1.x上广播。

只要是网段是一样的，对应的端口就都会收到。

比如广播地址：255.255.255.255,端口8999，则其他同网段中的端口8999就会收到。

2.组播

总是广播容易造成网络阻塞，所以需要组播了，另外，

我们再用广播发送消息的时候会发送给所有用户，但是有些用户是不想接受消息的，这时候我们就应该使用组播，

接收方只有先注册到组播地址中才能收到组播消息，否则则接受不到消息。另外组播是在Internet中使用的。

组播地址属于D类地址，D类地址又分出其他的，关于组播地址的分类：

224.0.0.0 ~ 224.0.0.255为预留的组播地址（永久组地址），地址224.0.0.0保留不做分配，其它地址供路由协议使用；

224.0.1.0 ~ 224.0.1.255是公用组播地址，可以用于Internet；

224.0.2.0 ~ 238.255.255.255为用户可用的组播地址（临时组地址），全网范围内有效；

239.0.0.0 ~ 239.255.255.255为本地管理组播地址，仅在特定的本地范围内有效。

在使用QUdpSocket类的writeDatagram()函数发送数据的时候，其中第二个参数host应该指定为组播地址，

注册加入到组播地址需要使用QUdpSocket类的成员函数：

```
bool joinMulticastGroup(const QHostAddress & groupAddress)
```

现实生活中的qq群，拉在一起这种的，用的就是组播

绑定后加入某个组播，在组播内你发组成员就都能收到，其他组成员发你也会收到

//绑定

//udpSocket->bind(8888);//使用组播只能使用(绑定)ipv4的ip，不能使用任意的ip，所以这里注释掉

udpSocket->bind(QHostAddress::AnyIPv4, 8888);//所以这里就要指定为ipv4

//加入某个组播 //广播不需要加入的操作就直接能发能收

//组播地址是D类地址

udpSocket->joinMulticastGroup(QHostAddress("224.0.0.2"));//加入后，其他人就可以向这个ip以及绑定的端口发送数据了

//udpSocket->leaveMulticastGroup(QHostAddress("224.0.0.2")); //退出组播

上述代码具体见《UDP》

```

1 #ifndef WIDGET_H
2 #define WIDGET_H
3
4 #include <QWidget>
5 #include <QUdpSocket> //UDP套接字
6
7 namespace Ui {
8 class Widget;
9 }
10
11 class Widget : public QWidget
12 {
13     Q_OBJECT
14
15 public:
16     explicit Widget(QWidget *parent = 0);
17     ~Widget();
18
19     void dealMsg(); //槽函数，处理对方发过来的数据
20
21 private slots:
22     void on_buttonSend_clicked();
23
24 private:
25     Ui::Widget *ui;
26
27     QUdpSocket *udpSocket; //UDP套接字
28 };
29
30 #endif // WIDGET_H

```

```

1 #include "widget.h"
2 #include "ui_widget.h"
3 #include <QHostAddress>
4
5 Widget::Widget(QWidget *parent) :
6     QWidget(parent),
7     ui(new Ui::Widget)
8 {
9     ui->setupUi(this);
10
11     //分配空间，指定父对象
12     udpSocket = new QUdpSocket(this);
13
14     //绑定
15     //udpSocket->bind(8888);
16     udpSocket->bind(QHostAddress::AnyIPv4, 8888);

```

```

17
18 //加入某个组播
19 //组播地址是D类地址
20 udpSocket->joinMulticastGroup( QHostAddress("224.0.0.2") );
21 //udpSocket->leaveMulticastGroup(); //退出组播
22
23 setWindowTitle("服务器端口为: 8888");
24
25 //当对方成功发送数据过来
26 //自动触发 readyRead()
27 connect(udpSocket, &QUdpSocket::readyRead, this, &Widget::dealMsg);
28 }
29
30 void Widget::dealMsg()
31 {
32     //读取对方发送的内容
33     char buf[1024] = {0};
34     QHostAddress cliAddr; //对方地址
35     quint16 port; //对方端口
36     qint64 len = udpSocket->readDatagram(buf, sizeof(buf), &cliAddr, &port);
37     if(len > 0)
38     {
39         //格式化 [192.68.2.2:8888]aaaa
40         QString str = QString("[%1:%2] %3")
41             .arg(cliAddr.toString())
42             .arg(port)
43             .arg(buf);
44         //给编辑区设置内容
45         ui->textEdit->setText(str);
46     }
47
48
49 }
50
51 Widget::~Widget()
52 {
53     delete ui;
54 }
55
56 //发送按钮
57 void Widget::on_buttonSend_clicked()
58 {
59     //先获取对方的IP和端口
60     QString ip = ui->lineEditIP->text();
61     quint16 port = ui->lineEditPort->text().toInt();
62
63     //获取编辑区内容
64     QString str = ui->textEdit->toPlainText();
65
66     //给指定的IP发送数据
67     udpSocket->writeDatagram(str.toUtf8(), QHostAddress(ip), port);
68
69
70 }

```



```

/*****
*****/

```

八、QTimer定时器的使用

QTimer 定时器对象，相对于那个事件的定时器好用多了。多个定时器创建多个对象即可

```
#include <QTimer> //定时器对象
```

定时器对象里面有个timeout的信号，当设置的定时时间到了的时候就会发出这样的一个信号。

当然如果停止了这个定时器就不会发送。

```
myTimer = new QTimer(this);
```

```
i = 0;
```

```
connect(myTimer, &QTimer::timeout,
        [=]()
        {
            i++;
            ui->lcdNumber->display(i);
        }
        );

void Widget::on_buttonStart_clicked()
{
    //启动定时器
    //时间间隔为100ms
    //每隔100ms,定时器myTimer内部自动触发timeout()信号
    //如果定时器没有激活, 才启动
    if(myTimer->isActive() == false)
    {
        myTimer->start(100);
    }
}

void Widget::on_buttonStop_clicked()
{
    if(true == myTimer->isActive())
    {
        myTimer->stop();
        i = 0;
    }
}
```

上述代码具体见《QTimer》

```
1 #ifndef WIDGET_H
2 #define WIDGET_H
3
4 #include <QWidget>
5 #include <QTimer> //定时器对象
6
7 namespace Ui {
8 class Widget;
9 }
10
11 class Widget : public QWidget
12 {
13     Q_OBJECT
14
15 public:
16     explicit Widget(QWidget *parent = 0);
17     ~Widget();
18
19 private slots:
20     void on_buttonStart_clicked();
21
22     void on_buttonStop_clicked();
23
24 private:
25     Ui::Widget *ui;
26
27     QTimer *myTimer; //定时器对象
28     int i;
```

```

29 };
30
31 #endif // WIDGET_H

```

```

1 #include "widget.h"
2 #include "ui_widget.h"
3
4 Widget::Widget(QWidget *parent) :
5     QWidget(parent),
6     ui(new Ui::Widget)
7 {
8     ui->setupUi(this);
9
10    myTimer = new QTimer(this);
11    i = 0;
12
13    connect(myTimer, &QTimer::timeout,
14            [=] ()
15            {
16                i++;
17                ui->lcdNumber->display(i);
18            }
19
20    );
21
22
23 }
24
25 Widget::~Widget()
26 {
27     delete ui;
28 }
29
30 void Widget::on_buttonStart_clicked()
31 {
32     //启动定时器
33     //时间间隔为100ms
34     //每隔100ms,定时器myTimer自动触发timeout()
35     //如果定时器没有激活,才启动
36     if(myTimer->isActive() == false)
37     {
38         myTimer->start(100);
39     }
40
41 }
42
43 void Widget::on_buttonStop_clicked()
44 {
45     if(true == myTimer->isActive())
46     {
47         myTimer->stop();
48         i = 0;
49     }
50 }

```

```

/*****
*****/

```

九、TCP传文件流程图

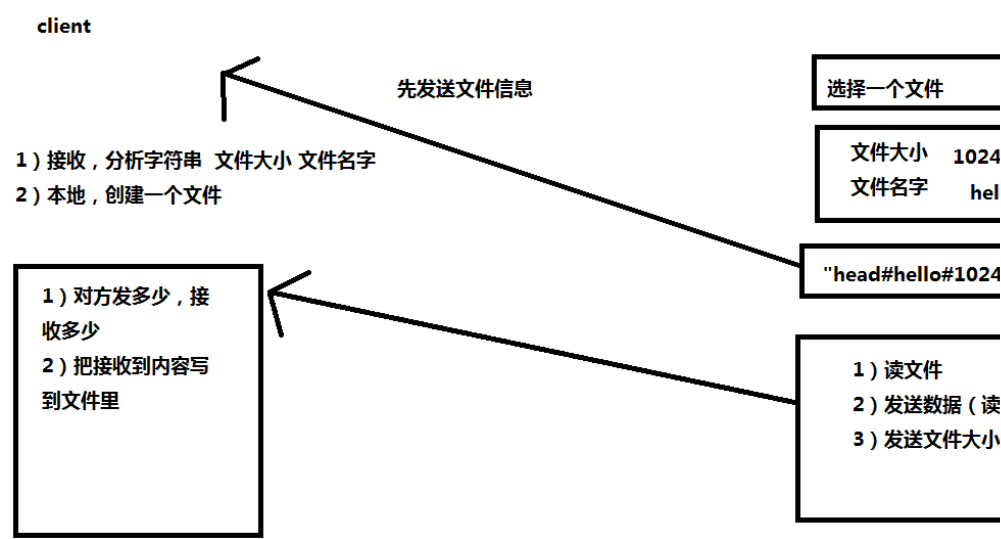
tcp中当两包数据发送间隔很短的时候，接收的时候就会出现两个包粘在一起的情况，也就是粘包。

比如简单的解决方法是控制发送间隔，使用定时器延时(图形界面不要用sleep除非开线程)让不能粘在一起的包分开。

当然也可以通过在数据包中增加包头，包长，包校验，包尾等信息来保证每一包数据的准确性。

还有一种办法是，不在乎粘的数据(比如文件数据)放在一个链接里，需要区分出来的数据(比如命令或者信息数据)放在另一个tcp链接里，

具体见图《TCP传文件流程图》



```
/*  
*****  
*****  
*/
```

十、TCP传文件服务器

```
.....  
//两个按钮都不能按，按钮颜色变灰并且不能按  
ui->buttonFile->setEnabled(false);  
ui->buttonSend->setEnabled(false);  
.....  
//成功连接后，才能按选择文件  
ui->buttonFile->setEnabled(true);  
.....  
connect(&timer, &QTimer::timeout,  
        [=]()  
        {  
            //关闭定时器  
            timer.stop();  
  
            //发送文件  
            sendData();  
        }  
        );  
.....  
//选择文件的按钮  
void ServerWidget::on_buttonFile_clicked()  
{  
    QString filePath = QFileDialog::getOpenFileName(this, "open", "../");  
    if(false == filePath.isEmpty()) //如果选择文件路径有效  
    {  
        fileName.clear();  
        fileSize = 0;  
    }  
}
```

```
//获取文件信息
QFileInfo info(filePath);

fileName = info.fileName(); //获取文件名字
fileSize = info.size(); //获取文件大小


sendSize = 0; //发送文件的大小


//只读方式打开文件
//指定文件的名字
file.setFileName(filePath);


//打开文件
bool isOk = file.open(QIODevice::ReadOnly);
if(false == isOk)
{
    qDebug() << "只读方式打开文件失败 106";
}


//提示打开文件的路径
ui->textEdit->append(filePath);


ui->buttonFile->setEnabled(false);
ui->buttonSend->setEnabled(true);
}
else
{
    qDebug() << "选择文件路径出错 118";
}
}
//发送文件按钮
void ServerWidget::on_buttonSend_clicked()
{
    ui->buttonSend->setEnabled(false);


    //先发送文件头信息 文件名##文件大小
    QString head = QString("%1##%2").arg(fileName).arg(fileSize);
    //发送头部信息
    qint64 len = tcpSocket->write( head.toUtf8() );
    if(len > 0)//头部信息发送成功
    {
        //发送真正的文件信息
        //防止TCP黏包
        //需要通过定时器延时 20 ms
        timer.start(20);
    }
    else
    {
        qDebug() << "头部信息发送失败 142";
    }
}
```



```

        file.close();

        ui->buttonFile->setEnabled(true);
        ui->buttonSend->setEnabled(false);
    }
}

void ServerWidget::sendData()
{
    ui->textEdit->append("正在发送文件.....");
    qint64 len = 0;
    do
    {
        //每次发送数据的大小
        char buf[4*1024] = {0};
        len = 0;

        //往文件中读数据
        len = file.read(buf, sizeof(buf));
        //发送数据，读多少，发多少
        len = tcpSocket->write(buf, len);

        //发送的数据需要累积
        sendSize += len;
    }while(len > 0);

    //  //是否发送文件完毕
    //  if(sendSize == fileSize)
    //  {
    //      ui->textEdit->append("文件发送完毕");
    //      file.close();

    //      //把客户端端口
    //      tcpSocket->disconnectFromHost();
    //      tcpSocket->close();
    //  }

}

/*****
*****/

```

十一、TCP传文件客户端

```

        connect(tcpSocket, &QTcpSocket::readyRead,
[=]()
{
    //取出接收的内容
    QByteArray buf = tcpSocket->readAll();

    if(true == isStart)

```

```
{//接收头

isStart = false;

//解析头部信息 QString buf = "hello##1024"

//   QString str = "hello##1024#mike";
//   str.section("##", 0, 0);/"##"分段符号, 0第一段开始, 0第一段结束, 所以取出来是hello


//初始化
//文件名
fileName = QString(buf).section("##", 0, 0);
//文件大小
fileSize = QString(buf).section("##", 1, 1).toInt();
recvSize = 0; //已经接收文件大小


//打开文件
//关联文件名字
file.setFileName(fileName);


//只写方式方式, 打开文件
bool isOk = file.open(QIODevice::WriteOnly);
if(false == isOk)
{
    qDebug() << "WriteOnly error 49";


    tcpSocket->disconnectFromHost(); //断开连接
    tcpSocket->close(); //关闭套接字


    return; //如果打开文件失败, 中断函数
}


//弹出对话框, 显示接收文件的信息
QString str = QString("接收的文件: [%1: %2kb]").arg(fileName).arg(fileSize/1024);
QMessageBox::information(this, "文件信息", str);


//设置进度条
ui->progressBar->setMinimum(0); //最小值
ui->progressBar->setMaximum(fileSize/1024); //最大值
ui->progressBar->setValue(0); //当前值
}

else //文件信息
{
    qint64 len = file.write(buf);
    if(len > 0) //接收数据大于0
    {
        recvSize += len; //累计接收大小
        qDebug() << len;
    }


//更新进度条
```

```

ui->progressBar->setValue(recvSize/1024);

if(recvSize == fileSize) //文件接收完毕
{

    //先给服务发送(接收文件完成的信息)
    tcpSocket->write("file done");

    QMessageBox::information(this, "完成", "文件接收完成");
    file.close(); //关闭文件
    //断开连接
    tcpSocket->disconnectFromHost();
    tcpSocket->close();
}
}
}
);

```

```

/*****
*****/

```

十二、TCP传文件进度条和黏包

//注意设置进度条使用除以1024的方法，不然太大，因为有可能文件太大，而进度条的那个值是int的

.....

//设置进度条

ui->progressBar->setMinimum(0); //最小值

ui->progressBar->setMaximum(fileSize/1024); //最大值//注意使用除以1024的方法，不然太大

ui->progressBar->setValue(0); //当前值

.....

//更新进度条

ui->progressBar->setValue(recvSize/1024); //注意使用除以1024的方法，不然太大

.....

上述代码具体见《TCPFile》

```

1 #include "serverwidget.h"
2 #include <QApplication>
3 #include "clientwidget.h"
4
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     ServerWidget w;
9     w.show();
10
11     ClientWidget w2;
12     w2.show();
13
14     return a.exec();
15 }

```

```
1 #ifndef SERVERWIDGET_H
2 #define SERVERWIDGET_H
3
4 #include <QWidget>
5 #include <QTcpServer> //监听套接字
6 #include <QTcpSocket> //通信套接字
7 #include <QFile>
8 #include <QTimer>
9
10 namespace Ui {
11 class ServerWidget;
12 }
13
14 class ServerWidget : public QWidget
15 {
16     Q_OBJECT
17
18 public:
19     explicit ServerWidget(QWidget *parent = 0);
20     ~ServerWidget();
21
22     void sendData(); //发送文件数据
23
24 private slots:
25     void on_buttonFile_clicked();
26
27     void on_buttonSend_clicked();
28
29 private:
30     Ui::ServerWidget *ui;
31
32     QTcpServer *tcpServer; //监听套接字
33     QTcpSocket *tcpSocket; //通信套接字
34
35     QFile file; //文件对象
36     QString fileName; //文件名字
37     qint64 fileSize; //文件大小
38     qint64 sendSize; //已经发送文件的大小
39
40     QTimer timer; //定时器
41
42
43
44
45
46 };
47
48 #endif // SERVERWIDGET_H
```



```
1 #include "serverwidget.h"
2 #include "ui_serverwidget.h"
3 #include <QFileDialog>
4 #include <QDebug>
5 #include <QFileInfo>
6
7 ServerWidget::ServerWidget(QWidget *parent) :
8     QWidget(parent),
9     ui(new Ui::ServerWidget)
10 {
11     ui->setupUi(this);
12
13     //监听套接字
14     tcpServer = new QTcpServer(this);
15
16     //监听
17     tcpServer->listen(QHostAddress::Any, 8888);
18     setWindowTitle("服务器端口为: 8888");
19
20     //两个按钮都不能按
21     ui->buttonFile->setEnabled(false);
22     ui->buttonSend->setEnabled(false);
23 }
```



```
24 //如果客户端成功和服务器连接
25 //tcpServer会自动触发 newConnection()
26 connect(tcpServer, &QTcpServer::newConnection,
27 [=] ()
28 {
29     //取出建立好连接的套接字
30     tcpSocket = tcpServer->nextPendingConnection();
31     //获取对方的ip和端口
32     QString ip = tcpSocket->peerAddress().toString();
33     quint16 port = tcpSocket->peerPort();
34
35     QString str = QString("[%1:%2] 成功连接").arg(ip).arg(port);
36     ui->textEdit->setText(str); //显示到编辑区
37
38     //成功连接后, 才能按选择文件
39     ui->buttonFile->setEnabled(true);
40
41     connect(tcpSocket, &QTcpSocket::readyRead,
42 [=] ()
43 {
44     //取客户端的信息
45     QByteArray buf = tcpSocket->readAll();
46     if(QString(buf) == "file done")
47     { //文件接收完毕
48         ui->textEdit->append("文件发送完毕");
49         file.close();
50
51         //断开客户端端口
52         tcpSocket->disconnectFromHost();
53         tcpSocket->close();
54     }
55
56     }
57
58     );
59
60 }
61 );
62
63 connect(&timer, &QTimer::timeout,
64 [=] ()
65 {
66     //关闭定时器
67     timer.stop();
68
69     //发送文件
70     sendData();
71 }
72
73 );
74
75 }
76
77 ServerWidget::~ServerWidget()
78 {
79     delete ui;
80 }
81
82 //选择文件的按钮
83 void ServerWidget::on_buttonFile_clicked()
84 {
85     QString filePath = QFileDialog::getOpenFileName(this, "open", "../");
86     if(false == filePath.isEmpty()) //如果选择文件路径有效
87     {
88         fileName.clear();
89         fileSize = 0;
90
91         //获取文件信息
92         QFile::Info info(filePath);
93         fileName = info.fileName(); //获取文件名字
94         fileSize = info.size(); //获取文件大小
95
96         sendSize = 0; //发送文件的大小
97
98         //只读方式打开文件
99         //指定文件的名字
100         file.setFileName(filePath);
```

```
101
102     //打开文件
103     bool isOk = file.open(QIODevice::ReadOnly);
104     if(false == isOk)
105     {
106         qDebug() << "只读方式打开文件失败 106";
107     }
108
109     //提示打开文件的路径
110     ui->textEdit->append(filePath);
111
112     ui->buttonFile->setEnabled(false);
113     ui->buttonSend->setEnabled(true);
114
115 }
116 else
117 {
118     qDebug() << "选择文件路径出错 118";
119 }
120
121 }
122 //发送文件按钮
123 void ServerWidget::on_buttonSend_clicked()
124 {
125     ui->buttonSend->setEnabled(false);
126
127     //先发送文件头信息 文件名##文件大小
128     QString head = QString("%1##%2").arg(fileName).arg(fileSize);
129     //发送头部信息
130     qint64 len = tcpSocket->write(head.toUtf8());
131     if(len > 0) //头部信息发送成功
132     {
133         //发送真正的文件信息
134         //防止TCP黏包
135         //需要通过定时器延时 20 ms
136         timer.start(20);
137
138     }
139     else
140     {
141         {
142             qDebug() << "头部信息发送失败 142";
143             file.close();
144             ui->buttonFile->setEnabled(true);
145             ui->buttonSend->setEnabled(false);
146         }
147     }
148
149 void ServerWidget::sendData()
150 {
151     ui->textEdit->append("正在发送文件.....");
152     qint64 len = 0;
153     do
154     {
155         //每次发送数据的大小
156         char buf[4*1024] = {0};
157         len = 0;
158
159         //往文件中读数据
160         len = file.read(buf, sizeof(buf));
161         //发送数据, 读多少, 发多少
162         len = tcpSocket->write(buf, len);
163
164         //发送的数据需要累积
165         sendSize += len;
166
167     }while(len > 0);
168
169
170 // //是否发送文件完毕
171 // if(sendSize == fileSize)
172 // {
173 //     ui->textEdit->append("文件发送完毕");
174 //     file.close();
175 //
176 //     //把客户端端口
177 //     tcpSocket->disconnectFromHost();
```

```

178 //      tcpSocket->close();
179 //      }
180
181
182 }

```



```

1 #ifndef CLIENTWIDGET_H
2 #define CLIENTWIDGET_H
3
4 #include <QWidget>
5 #include <QTcpSocket>
6 #include <QFile>
7
8 namespace Ui {
9 class ClientWidget;
10 }
11
12 class ClientWidget : public QWidget
13 {
14     Q_OBJECT
15
16 public:
17     explicit ClientWidget(QWidget *parent = 0);
18     ~ClientWidget();
19
20 private slots:
21     void on_buttonConnect_clicked();
22
23 private:
24     Ui::ClientWidget *ui;
25
26     QTcpSocket *tcpSocket;
27
28     QFile file; //文件对象
29     QString fileName; //文件名字
30     qint64 fileSize; //文件大小
31     qint64 recvSize; //已经接收文件的大小
32
33     bool isStart; //标志位, 是否为头部信息
34 };
35
36 #endif // CLIENTWIDGET_H

```



```

1 #include "clientwidget.h"
2 #include "ui_clientwidget.h"
3 #include <QDebug>
4 #include <QMessageBox>
5 #include <QHostAddress>
6
7 ClientWidget::ClientWidget(QWidget *parent) :
8     QWidget(parent),
9     ui(new Ui::ClientWidget)
10 {
11     ui->setupUi(this);
12
13     tcpSocket = new QTcpSocket(this);
14
15     isStart = true;
16
17     ui->progressBar->setValue(0); //当前值
18
19     setWindowTitle("客户端");
20
21     connect(tcpSocket, &QTcpSocket::readyRead,
22         [=] ()
23         {
24             //取出接收的内容
25             QByteArray buf = tcpSocket->readAll();

```

```
26
27     if(true == isStart)
28     { //接收头
29         isStart = false;
30         //解析头部信息 QString buf = "hello##1024"
31         //             QString str = "hello##1024#mike";
32         //             str.section("##", 0, 0)
33
34         //初始化
35         //文件名
36         fileName = QString(buf).section("##", 0, 0);
37         //文件大小
38         fileSize = QString(buf).section("##", 1, 1).toInt();
39         recvSize = 0; //已经接收文件大小
40
41         //打开文件
42         //关联文件名字
43         file.setFileName(fileName);
44
45         //只写方式方式, 打开文件
46         bool isOk = file.open(QIODevice::WriteOnly);
47         if(false == isOk)
48         {
49             qDebug() << "WriteOnly error 49";
50
51             tcpSocket->disconnectFromHost(); //断开连接
52             tcpSocket->close(); //关闭套接字
53
54             return; //如果打开文件失败, 中断函数
55         }
56
57         //弹出对话框, 显示接收文件的信息
58         QString str = QString("接收的文件: [%1:
%2kb]").arg(fileName).arg(fileSize/1024);
59         QMessageBox::information(this, "文件信息", str);
60
61         //设置进度条
62         ui->progressBar->setMinimum(0); //最小值
63         ui->progressBar->setMaximum(fileSize/1024); //最大值
64         ui->progressBar->setValue(0); //当前值
65
66     }
67     else //文件信息
68     {
69         qint64 len = file.write(buf);
70         if(len > 0) //接收数据大于0
71         {
72             recvSize += len; //累计接收大小
73             qDebug() << len;
74         }
75
76         //更新进度条
77         ui->progressBar->setValue(recvSize/1024);
78
79         if(recvSize == fileSize) //文件接收完毕
80         {
81
82             //先给服务发送 (接收文件完成的信息)
83             tcpSocket->write("file done");
84
85             QMessageBox::information(this, "完成", "文件接收完成");
86             file.close(); //关闭文件
87             //断开连接
88             tcpSocket->disconnectFromHost();
89             tcpSocket->close();
90
91         }
92     }
93
94 }
95
96 );
97
98 }
99
100 ClientWidget::~ClientWidget()
101 {
```



```
102     delete ui;
103 }
104
105 void ClientWidget::on_buttonConnect_clicked()
106 {
107     //获取服务器的ip和端口
108     QString ip = ui->lineEditIP->text();
109     quint16 port = ui->lineEditPort->text().toInt();
110
111     //主动和服务器连接
112     tcpSocket->connectToHost(QHostAddress(ip), port);
113
114     isStart = true;
115
116     //设置进度条
117     ui->progressBar->setValue(0);
118 }
```



分类: [界面编程](#)

标签: [界面编程](#), [QT](#)

好文要顶

关注我

收藏该文



[风味鱼](#)

[关注 - 0](#)

[粉丝 - 18](#)

[+加关注](#)

0

0

« 上一篇: [界面编程之QT的文件操作20180729](#)

» 下一篇: [界面编程之QT的线程20180731](#)

posted @ 2018-07-28 17:55 [风味鱼](#) 阅读(9766) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#) 网站首页。

【推荐】腾讯云海外1核2G云服务器低至2折，半价续费券限量免费领取！

【活动】京东云服务器_云主机低于1折，低价高性能产品备战双11

【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

【培训】马士兵老师强势回归！Java线下课程全免费，双十一大促！

【推荐】天翼云双十一翼降到底，云主机11.11元起，抽奖送大礼

【福利】个推四大热门移动开发SDK全部免费用一年，限时抢！

【推荐】流程自动化专家UiBot，全套体系化教程成就高薪RPA工程师

京东云 | 腾讯云 | 阿里云 | 华为云 | 百度云 | 腾讯云 | 阿里云 | 华为云 | 百度云 | 腾讯云 | 阿里云 | 华为云 | 百度云

云主机
最低 1 折

每人限购三台

立即秒杀

相关博文:

- [QT之TCP通信](#)
 - [Qt网络应用----socket通信例子](#)
 - [Qt开发UDP](#)
 - [QT5 网络通讯](#)
 - [php的socket通信](#)
- » 更多推荐...

腾讯云 | 11.11 智慧上云

爆品限时购
云服务器 1核2G 首年 88元

最新 IT 新闻:

- [佰才邦宣布完成C1轮融资，C轮6亿规模将成国内5G领域单笔最大融资](#)

- [朱文佳成为今日头条新任CEO 此前消息称已向张一鸣直接汇报](#)
 - [密码管理公司1Password获2亿美元A轮融资，Accel领投](#)
 - [世卫组织启动胰岛素预认证应对糖尿病](#)
 - [深海沉船里，物理学家最爱的不是宝藏，而是这种常见金属？！](#)
- » [更多新闻...](#)

历史上的今天：

2018-07-28 [界面编程之QT的文件操作20180729](#)

2018-07-28 [界面编程之QT绘图和绘图设备20180728](#)

2017-07-28 [设计模式之门面模式20170728](#)

Copyright © 2019 风味鱼
Powered by .NET Core 3.0.0 on Linux