新范式网络调试助手软件

源程序文档

青岛新范式信息技术有限公司

#include "toolkit.h"

#include "setting.h"

#include "baseform.h"

#include <QShortcut>

#include <QWidget>

#include <QLabel>

#include <QToolButton>

#include <QLineEdit>

#include <QTreeWidget>

#include <QComboBox>

#include <QListWidget>

#include <QPlainTextEdit>

#define PROP\_EDIT "edit"

#define PROP\_DIRT "dirt"

#define PROP\_TARG "targ"

BaseForm::BaseForm(QWidget\* p, Qt::WindowFlags f)

:QWidget(p, f),m\_cntRecv(0),m\_cntSend(0),m\_labRecv(0),m\_labSend(0),m\_cnlist(0)

{

}

BaseForm::~BaseForm()

{

}

bool BaseForm::init()

{

if (!initForm() || !initHotkeys())

return false;

initConfig();

m\_logger.setProperty(SET\_SEC\_DIR, property(SET\_SEC\_DIR).toString());

return true;

}

void BaseForm::initCounter(QLabel\* r, QLabel\* s)

{

m\_labRecv = r;

m\_labSend = s;

}

void BaseForm::initLogger(QCheckBox\* w, QToolButton\* c, QTreeWidget\* o, QPlainTextEdit\* d)

{

m\_logger.init(o, w, d);

connect(c, SIGNAL(released()), this, SLOT(clear()));

connect(&m\_logger, SIGNAL(clearLog()), this, SLOT(clear()));

bindFocus(o, Qt::Key\_F3);

QShortcut\* wr = new QShortcut(QKeySequence(Qt::CTRL + Qt::Key\_W), this);

QShortcut\* cl = new QShortcut(QKeySequence(Qt::CTRL + Qt::Key\_D), this);

QShortcut\* sl = new QShortcut(QKeySequence(Qt::Key\_F4), this);

sl->setProperty(PROP\_TARG, qVariantFromValue((void\*)d));

connect(wr, SIGNAL(activated()), w, SLOT(click()));

connect(sl, SIGNAL(activated()), this, SLOT(hotOutput()));

connect(cl, SIGNAL(activated()), this, SLOT(clear()));

connect(this, SIGNAL(output(const QString&)), &m\_logger, SLOT(output(const QString&)));

connect(this, SIGNAL(output(const QString&, const char\*, quint32)), &m\_logger, SLOT(output(const QString&, const char\*, quint32)));

}

void BaseForm::initLister(QToolButton\* a, QToolButton\* k, QListWidget\* l)

{

m\_cnlist = l;

QShortcut\* sk = new QShortcut(QKeySequence(Qt::CTRL + Qt::Key\_K), this);

QShortcut\* sa = new QShortcut(QKeySequence(Qt::ALT + Qt::Key\_A), this);

connect(sk, SIGNAL(activated()), this, SLOT(kill()));

connect(sa, SIGNAL(activated()), m\_cnlist, SLOT(selectAll()));

connect(k, SIGNAL(released()), this, SLOT(kill()));

connect(a, SIGNAL(released()), m\_cnlist, SLOT(selectAll()));

bindFocus(m\_cnlist, Qt::Key\_F2);

}

void BaseForm::bindBuffer(qint32 id, QLineEdit\* e, QToolButton\* s, QComboBox\* d)

{

s->setProperty(PROP\_EDIT, qVariantFromValue((void\*)e));

s->setProperty(PROP\_DIRT, qVariantFromValue((void\*)d));

connect(s, SIGNAL(released()), this, SLOT(send()));

bindClick(s, Qt::Key\_0 + id + Qt::CTRL);

bindFocus(e, Qt::Key\_0 + id + Qt::ALT);

bindFocus(d, Qt::Key\_0 + id + Qt::CTRL + Qt::SHIFT);

}

void BaseForm::bindFocus(QWidget\* w, qint32 k)

{

QShortcut\* s = new QShortcut(QKeySequence(k), this);

s->setProperty(PROP\_TARG, qVariantFromValue((void\*)w));

connect(s, SIGNAL(activated()), this, SLOT(focus()));

}

void BaseForm::bindClick(QAbstractButton\* b, qint32 k)

{

QShortcut\* s = new QShortcut(QKeySequence(k), this);

connect(s, SIGNAL(activated()), b, SLOT(click()));

}

void BaseForm::bindSelect(QComboBox\* b, qint32 i, qint32 k)

{

QShortcut\* s = new QShortcut(QKeySequence(k), this);

s->setProperty(PROP\_TARG, qVariantFromValue((void\*)b));

s->setObjectName(QString::number(i));

connect(s, SIGNAL(activated()), this, SLOT(select()));

}

void BaseForm::focus()

{

QWidget\* w = (QWidget\*)sender()->property(PROP\_TARG).value<void\*>();

if (w) w->setFocus(Qt::TabFocusReason);

}

void BaseForm::hotOutput()

{

QPlainTextEdit\* t = (QPlainTextEdit\*)sender()->property(PROP\_TARG).value<void\*>();

if (t)

{

t->setFocus(Qt::TabFocusReason);

t->selectAll();

}

}

void BaseForm::select()

{

QComboBox\* b = (QComboBox\*)sender()->property(PROP\_TARG).value<void\*>();

if (b && b->isEnabled())

{

qint32 i = sender()->objectName().toInt();

if (i < 0)

{

i = b->currentIndex() + 1;

if (i >= b->count()) i = 0;

}

b->setCurrentIndex(i);

}

}

void BaseForm::countRecv(qint32 bytes)

{

if (bytes < 0)

m\_cntRecv = 0;

else

m\_cntRecv += bytes;

m\_labRecv->setText(QString::number(m\_cntRecv));

}

void BaseForm::countSend(qint32 bytes)

{

if (bytes < 0)

m\_cntSend = 0;

else

m\_cntSend += bytes;

m\_labSend->setText(QString::number(m\_cntSend));

}

void BaseForm::send()

{

QLineEdit\* e = (QLineEdit\*)sender()->property(PROP\_EDIT).value<void\*>();

QComboBox\* d = (QComboBox\*)sender()->property(PROP\_DIRT).value<void\*>();

if (e)

send(e->text(), (d?d->currentText():""));

}

void BaseForm::clear()

{

m\_logger.clear();

lock();

countRecv(-1);

countSend(-1);

unlock();

}

void BaseForm::kill()

{

if (lock(1000))

{

QStringList list;

listerSelected(list);

kill(list);

unlock();

}

}

void BaseForm::listerSelected(QStringList& output)

{

qint32 i = m\_cnlist->count();

while (i--)

{

QListWidgetItem\* itm = m\_cnlist->item(i);

if (itm && itm->isSelected())

output << itm->text();

}

}

void BaseForm::listerAdd(const QString& caption)

{

listerRemove(caption);

m\_cnlist->addItem(caption);

}

void BaseForm::listerRemove(const QString& caption)

{

qint32 i = m\_cnlist->count();

while (i--)

{

QListWidgetItem\* itm = m\_cnlist->item(i);

if (itm && itm->text()==caption)

delete m\_cnlist->takeItem(i);

}

}

#include "bluetoothmodel.h"

BlueToothModel::BlueToothModel(QObject \*parent) : QObject(parent) {

discoveryAgent = new QBluetoothDeviceDiscoveryAgent();

discoveryAgent->setLowEnergyDiscoveryTimeout(5000);

connect(discoveryAgent, &QBluetoothDeviceDiscoveryAgent::deviceDiscovered,

this, &BlueToothModel::addDevice);

connect(discoveryAgent,

QOverload<QBluetoothDeviceDiscoveryAgent::Error>::of(&QBluetoothDeviceDiscoveryAgent::error),

this, &BlueToothModel::deviceScanError);

connect(discoveryAgent, &QBluetoothDeviceDiscoveryAgent::finished, this, &BlueToothModel::deviceScanFinished);

setUpdate("Search");

qDebug("init bluetooth");

}

BlueToothModel::~BlueToothModel(){

delete discoveryAgent;

delete controller;

qDeleteAll(devices);

qDeleteAll(m\_services);

qDeleteAll(m\_characteristics);

devices.clear();

m\_services.clear();

m\_characteristics.clear();

}

void BlueToothModel::setUpdate(const QString &message) {

}

void BlueToothModel::startDeviceDiscovery()

{

qDeleteAll(devices);

devices.clear();

emit devicesUpdated();

qDebug("start scan");

setUpdate("Scanning for devices ...");

discoveryAgent->start(QBluetoothDeviceDiscoveryAgent::LowEnergyMethod);

if (discoveryAgent->isActive()) {

m\_deviceScanState = true;

Q\_EMIT stateChanged();

}

}

void BlueToothModel::addDevice(const QBluetoothDeviceInfo &info) {

qDebug("name %s",info.name().toStdString().data());

qDebug("add %s",info.address().toString().toStdString().data());

qDebug("uuid %s",info.deviceUuid().toString().toStdString().data());

}

void BlueToothModel::deviceScanFinished() {

qDebug("%s","finished");

}

void BlueToothModel::deviceScanError(QBluetoothDeviceDiscoveryAgent::Error error) {

qDebug("Error %d",error);

}

#include "toolkit.h"

#include "setting.h"

#include "clientskt.h"

#include "clientform.h"

#define SET\_SEC\_CLT "client"

#define SET\_KEY\_CLT "/client"

#define SET\_KEY\_CMBIP "/ip"

#define SET\_KEY\_CMBPT "/port"

#define SET\_VAL\_LGCLT "log\_client"

ClientForm::ClientForm(QWidget\* p, Qt::WindowFlags f)

:BaseForm(p, f),m\_client(0)

{

m\_ui.setupUi(this);

}

ClientForm::~ClientForm()

{

unplug();

saveConfig();

}

bool ClientForm::initForm()

{

initCounter(m\_ui.labRecv, m\_ui.labSend);

initLogger(m\_ui.chkLog, m\_ui.btnClear, m\_ui.treeOutput, m\_ui.txtOutput);

bindBuffer(0, m\_ui.edtBuf0, m\_ui.btnSend0, 0);

bindBuffer(1, m\_ui.edtBuf1, m\_ui.btnSend1, 0);

bindBuffer(2, m\_ui.edtBuf2, m\_ui.btnSend2, 0);

bindBuffer(3, m\_ui.edtBuf3, m\_ui.btnSend3, 0);

connect(m\_ui.btnTcp, SIGNAL(clicked(bool)), this, SLOT(trigger(bool)));

connect(m\_ui.btnUdp, SIGNAL(clicked(bool)), this, SLOT(trigger(bool)));

return true;

}

void ClientForm::initConfig()

{

QString ssc(SET\_SEC\_CLT);

Setting::lord(ssc+SET\_KEY\_CMBIP, SET\_PFX\_CMBITM, \*m\_ui.cmbAddr);

Setting::lord(ssc+SET\_KEY\_CMBPT, SET\_PFX\_CMBITM, \*m\_ui.cmbPort);

QString skl(SET\_SEC\_DIR); skl += SET\_KEY\_LOG;

skl = Setting::get(skl, SET\_KEY\_CLT, SET\_VAL\_LGCLT);

setProperty(SET\_SEC\_DIR, skl);

TK::initNetworkInterfaces(m\_ui.cmbAddr);

}

void ClientForm::saveConfig()

{

QString ssc(SET\_SEC\_CLT);

Setting::save(ssc+SET\_KEY\_CMBIP, SET\_PFX\_CMBITM, \*m\_ui.cmbAddr);

Setting::save(ssc+SET\_KEY\_CMBPT, SET\_PFX\_CMBITM, \*m\_ui.cmbPort);

QString skl(SET\_SEC\_DIR); skl += SET\_KEY\_LOG;

Setting::set(skl, SET\_KEY\_CLT, property(SET\_SEC\_DIR).toString());

}

bool ClientForm::initHotkeys()

{

bindFocus(m\_ui.cmbAddr, Qt::Key\_Escape);

bindClick(m\_ui.btnTcp, Qt::CTRL + Qt::Key\_T);

bindClick(m\_ui.btnUdp, Qt::CTRL + Qt::Key\_U);

return true;

}

void ClientForm::unplug()

{

ClientSkt\* client = 0;

if (lock())

{

client = m\_client;

m\_client = NULL;

unlock();

}

if (client)

client->unplug();

}

void ClientForm::unpluged()

{

ClientSkt\* c = qobject\_cast<ClientSkt\*>(sender());

if (!c) return;

if (lock())

{

if (c==m\_client)

m\_client = NULL;

unlock();

}

if (c->name().contains(TK::socketTypeName(true)))

TK::resetPushBox(m\_ui.btnTcp);

else

TK::resetPushBox(m\_ui.btnUdp);

c->disconnect(this);

c->deleteLater();

}

void ClientForm::trigger(bool checked)

{

unplug();

m\_ui.cmbAddr->setDisabled(checked);

m\_ui.cmbPort->setDisabled(checked);

QToolButton\* btn = qobject\_cast<QToolButton\*>(sender());

if (checked && !plug(btn==m\_ui.btnTcp))

btn->click();

}

bool ClientForm::plug(bool istcp)

{

ClientSkt\* skt = 0;

lock();

if (m\_client)

{

m\_client->disconnect(this);

m\_client->deleteLater();

m\_client = NULL;

}

IPAddr addr;

bool res = TK::popIPAddr(m\_ui.cmbAddr, m\_ui.cmbPort, addr);

if (res)

{

if (istcp)

m\_client = new ClientSktTcp();

else

m\_client = new ClientSktUdp();

if (!m\_client)

{

res = false;

}

else

{

connect(m\_client, SIGNAL(unpluged()), this, SLOT(unpluged()));

connect(m\_client, SIGNAL(message(const QString&)), this, SIGNAL(output(const QString&)));

connect(m\_client, SIGNAL(dumpbin(const QString&,const char\*,quint32)), this, SIGNAL(output(const QString&,const char\*,quint32)));

connect(m\_client, SIGNAL(countRecv(qint32)), this, SLOT(countRecv(qint32)));

connect(m\_client, SIGNAL(countSend(qint32)), this, SLOT(countSend(qint32)));

skt = m\_client;

}

}

unlock();

if (res && skt)

{

res = skt->plug(addr.ip, addr.port);

if (res)

TK::pushIPAddr(m\_ui.cmbAddr, m\_ui.cmbPort);

}

return res;

}

void ClientForm::send(const QString& data, const QString&)

{

if (m\_client && lock(1000))

{

m\_client->send(data);

unlock();

}

}

//

// Created by saka on 2019-02-28.

//

#include <QHostAddress>

#include <QNetworkInterface>

#include <QTime>

#include <QDataStream>

#include "ClientModel.h"

#include "toolkit.h"

ClientModel::ClientModel(QObject \*parent) : QmlLogModel(parent)

{

}

void ClientModel::getAddr() {

QList<QHostAddress> lst = QNetworkInterface::allAddresses();

foreach (QHostAddress a, lst) {

if (QAbstractSocket::IPv4Protocol == a.protocol()) {

emit appendLocalAddr(a.toString());

}

}

}

void ClientModel::toggleConnect(bool checked, QString addr, QString port) {

auto r\_port = quint16(port.toUInt());

if (checked) {

openClient(addr, r\_port);

} else {

closeClient();

}

}

void ClientModel::send(const QString &data) {

QString err;

QByteArray bin;

if (!TK::ascii2bin(data, bin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

qDebug()<<"UDP model send data "<<data;

sendToDst(bin);

}

void ClientModel::sendWithHeader(const QString header, const qint32 lengthSize, bool bigEndian, const QString &data)

{

QString err;

QByteArray dataBin;

QByteArray headerBin;

if (!TK::ascii2bin(header, headerBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

if (!TK::ascii2bin(data, dataBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

QByteArray length;

length.resize(lengthSize);

QDataStream sendData(&length,QIODevice::WriteOnly);

sendData.setByteOrder(bigEndian? QDataStream::BigEndian: QDataStream::LittleEndian);

switch (lengthSize) {

case 0:

break;

case 2:

sendData<<quint16(dataBin.size());

break;

case 4:

sendData<<quint32(dataBin.size());

break;

}

dataBin.prepend(length);

dataBin.prepend(headerBin);

sendToDst(dataBin);

}

void ClientModel::sendMessageData(SendMessageData \*data)

{

if(data->plainText()){

qDebug()<<"纯文本发送";

sendToDst(data->content().toUtf8());

return;

}else {

qDebug()<<"非纯文本发送";

}

if(!data->withHeader()){

send(data->content());

return;

}

QString err;

QByteArray dataBin;

QByteArray headerBin;

if (!TK::ascii2bin(data->header(), headerBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

if (!TK::ascii2bin(data->getTargetMsg(), dataBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

QByteArray length;

length.resize(data->lengthSize());

QDataStream sendData(&length,QIODevice::WriteOnly);

sendData.setByteOrder(data->endian()? QDataStream::BigEndian: QDataStream::LittleEndian);

switch (data->lengthSize()) {

case 0:

break;

case 2:

sendData<<quint16(dataBin.size());

break;

case 4:

sendData<<quint32(dataBin.size());

break;

}

dataBin.prepend(length);

dataBin.prepend(headerBin);

sendToDst(dataBin);

}

bool ClientModel::closeClient() {

emit connClose(host);

close();

return true;

}

#include <QTcpSocket>

#include "toolkit.h"

#include "clientskt.h"

#define MAXBUFFER 1024\*1024

ClientSkt::ClientSkt(QObject \*parent)

: QObject(parent),m\_port(0)

{

}

ClientSkt::~ClientSkt()

{

}

bool ClientSkt::plug(const QHostAddress& ip, quint16 port)

{

m\_ip = ip;

m\_port = port;

m\_error.clear();

return open();

}

void ClientSkt::unplug()

{

close();

emit unpluged();

}

void ClientSkt::setError(const QString& err)

{

m\_error = err;

}

void ClientSkt::recordRecv(qint32 bytes)

{

emit countRecv(bytes);

}

void ClientSkt::recordSend(qint32 bytes)

{

emit countSend(bytes);

}

void ClientSkt::send(const QString& data)

{

QString err;

QByteArray bin;

if (!TK::ascii2bin(data, bin, err))

{

show("bad data format to send: "+err);

return;

}

send(bin);

}

void ClientSkt::dump(const char\* buf, qint32 len, bool isSend)

{

emit dumpbin(QString("DAT %1").arg(isSend?"<---":"--->"), buf, (quint32)len);

}

void ClientSkt::show(const QString& msg)

{

emit message(msg);

}

ClientSktTcp::ClientSktTcp(QObject \*parent)

:ClientSkt(parent)

{

}

ClientSktTcp::~ClientSktTcp()

{

}

bool ClientSktTcp::open()

{

connect(&m\_socket, SIGNAL(readyRead()), this, SLOT(newData()));

connect(&m\_socket, SIGNAL(disconnected()), this, SLOT(closed()));

connect(&m\_socket, SIGNAL(connected()), this, SLOT(asynConn()));

connect(&m\_socket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

m\_socket.connectToHost(addr(), port());

return true;

}

void ClientSktTcp::close()

{

m\_socket.close();

m\_socket.disconnect(this);

}

void ClientSktTcp::error()

{

QTcpSocket\* s = qobject\_cast<QTcpSocket\*>(sender());

show(QString("TCP socket error %1, %2").arg(s->error()).arg(s->errorString()));

unplug();

}

void ClientSktTcp::asynConn()

{

show(QString("TCP connection to %1:%2 opened!")

.arg(addr().toString()).arg(port()));

}

void ClientSktTcp::closed()

{

show(QString("TCP connection closed!"));

}

void ClientSktTcp::newData()

{

QTcpSocket\* s = qobject\_cast<QTcpSocket\*>(sender());

if (!s) return;

qint64 bufLen = s->bytesAvailable();

char\* buf = TK::createBuffer(bufLen, MAXBUFFER);

if (!buf) return;

qint64 readLen = 0;

qint64 ioLen = s->read(buf, bufLen);

while (ioLen > 0)

{

readLen += ioLen;

ioLen = s->read(buf+readLen, bufLen-readLen);

}

if (ioLen >= 0)

{

recordRecv(readLen);

dump(buf, readLen, false);

}

TK::releaseBuffer(buf);

}

void ClientSktTcp::send(const QByteArray& bin)

{

const char \* src = bin.constData();

qint64 srcLen = bin.length();

qint64 writeLen = 0;

qint64 ioLen = m\_socket.write(src, srcLen);

while (ioLen > 0)

{

writeLen += ioLen;

ioLen = m\_socket.write(src+writeLen, srcLen-writeLen);

}

if (writeLen != srcLen)

{

show(QString("failed to send data to %1:%2 [%3]")

.arg(addr().toString()).arg(port()).arg(writeLen));

return;

}

recordSend(writeLen);

dump(src, srcLen, true);

}

ClientSktUdp::ClientSktUdp(QObject \*parent)

:ClientSkt(parent)

{

}

ClientSktUdp::~ClientSktUdp()

{

}

void ClientSktUdp::asynConn()

{

show(QString("UDP channel to %1:%2 opened!")

.arg(addr().toString()).arg(port()));

}

void ClientSktUdp::closed()

{

show(QString("UDP channel closed!"));

}

void ClientSktUdp::close()

{

m\_socket.close();

m\_socket.disconnect(this);

}

void ClientSktUdp::error()

{

QUdpSocket\* s = qobject\_cast<QUdpSocket\*>(sender());

show(QString("UDP socket error %1, %2").arg(s->error()).arg(s->errorString()));

unplug();

}

bool ClientSktUdp::open()

{

connect(&m\_socket, SIGNAL(readyRead()), this, SLOT(newData()));

connect(&m\_socket, SIGNAL(disconnected()), this, SLOT(closed()));

connect(&m\_socket, SIGNAL(connected()), this, SLOT(asynConn()));

connect(&m\_socket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

m\_socket.connectToHost(addr(), port());

return true;

}

void ClientSktUdp::newData()

{

QUdpSocket\* s = qobject\_cast<QUdpSocket\*>(sender());

if (!s) return;

qint64 bufLen = s->bytesAvailable();

char\* buf = TK::createBuffer(bufLen, MAXBUFFER);

if (!buf) return;

qint64 readLen = 0;

qint64 ioLen = s->read(buf, bufLen);

while (ioLen > 0)

{

readLen += ioLen;

ioLen = s->read(buf+readLen, bufLen-readLen);

}

if (ioLen >= 0)

{

recordRecv(readLen);

dump(buf, readLen, false);

}

TK::releaseBuffer(buf);

}

void ClientSktUdp::send(const QByteArray& bin)

{

const char \* src = bin.constData();

qint64 srcLen = bin.length();

qint64 writeLen = 0;

qint64 ioLen = m\_socket.write(src, srcLen);

while (ioLen > 0)

{

writeLen += ioLen;

ioLen = (writeLen >= srcLen) ? 0 :

m\_socket.write(src+writeLen, srcLen-writeLen);

}

if (writeLen != srcLen)

{

show(QString("failed to send data to %1:%2 [%3]")

.arg(addr().toString()).arg(port()).arg(writeLen));

return;

}

recordSend(writeLen);

dump(src, srcLen, true);

}

#include "HttpManager.h"

#include "Util.h"

HttpManager::HttpManager(QObject \*parent) : QObject(parent)

{

}

void HttpManager::checkVersion()

{

#ifdef QT\_NO\_DEBUG

QNetworkRequest request;

QNetworkAccessManager\* naManager = new QNetworkAccessManager(this);

QMetaObject::Connection connRet = QObject::connect(naManager, SIGNAL(finished(QNetworkReply\*)), this, SLOT(requestFinished(QNetworkReply\*)));

Q\_ASSERT(connRet);

QUrl url=QUrl("http://www.rangaofei.cn/ssokit/api/version");

QUrlQuery query;

request.setUrl(url);

request.setRawHeader(QByteArray("uuid"),Util::getUUID().toUtf8());

request.setRawHeader(QByteArray("version"),Config::getVersionName().toUtf8());

request.setRawHeader(QByteArray("sys-type"),Util::getSystemType().toUtf8());

request.setRawHeader(QByteArray("sys-version"),Util::getSystemVersion().toUtf8());

naManager->get(request);

#else

qDebug() << "debug mode,will not update";

#endif

}

void HttpManager::requestFinished(QNetworkReply \*reply)

{

// 获取http状态码

QVariant statusCode = reply->attribute(QNetworkRequest::HttpStatusCodeAttribute);

if(statusCode.isValid())

qDebug() << "status code=" << statusCode.toInt();

QVariant reason = reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).toString();

if(reason.isValid())

qDebug() << "reason=" << reason.toString();

QNetworkReply::NetworkError err = reply->error();

if(err != QNetworkReply::NoError) {

qDebug() << "Failed: " << reply->errorString();

}

else {

QJsonParseError json\_error;

QJsonDocument jsonDoc(QJsonDocument::fromJson(reply->readAll(), &json\_error));

if(json\_error.error != QJsonParseError::NoError)

{

qDebug() << "json error!";

return;

}

if(!jsonDoc.isObject()){

return;

}

QJsonObject wrapperObj=jsonDoc.object();

int code = wrapperObj.value("code").toInt();

if(code!=200){

qDebug()<<"error::"<<wrapperObj.value("msg").toString();

return;

}

QJsonObject rootObj=wrapperObj.value("data").toObject();

int version=rootObj.value("version\_code").toInt();

if(version<=Config::getVersionCode()){

return;

}

int32\_t sys=Config::getSystem();

QString url="";

switch (sys) {

case 0:

break;

case 1:

url=rootObj.value("win\_url").toString();

break;

case 2:

url=rootObj.value("linux\_url").toString();

break;

case 3:

url=rootObj.value("mac\_url").toString();

break;

}

emit showUpdateVersion(

rootObj.value("version\_name").toString(),

rootObj.value("content\_cn").toString(),

rootObj.value("content\_en").toString(),

rootObj.value("release\_date").toString(),

url,

rootObj.value("force\_update").toBool()

);

}

}

#include "itemport.h"

ItemPort::ItemPort(QObject \*parent) : QObject(parent)

{

}

ItemPort::ItemPort(QString port):m\_port(port)

{

}

QString ItemPort::port() const

{

return m\_port;

}

#include "JsonFormat.h"

#include <QDebug>

JsonFormat::JsonFormat(QObject \*parent)

{

Q\_UNUSED(parent)

}

JsonFormat::~JsonFormat()

{

if(m\_jsonDocument!=nullptr){

delete m\_jsonDocument;

m\_jsonDocument=nullptr;

}

}

JsonModel \*JsonFormat::jsonModel(){

return m\_jsonModel;

}

void JsonFormat::setJsonModel(JsonModel \*jsonModel){

this->m\_jsonModel=jsonModel;

}

/\*\*

\* @brief JsonFormat::checkJonsStr

\* @param data

\* 检查是否可以格式化json

\*/

void JsonFormat::checkJonsStr(const QVariant data)

{

QString jsonStr=data.toString();

qDebug()<<jsonStr;

const QByteArray jsonArray=jsonStr.toUtf8();

m\_jsonDocument=new QJsonDocument(QJsonDocument::fromJson(jsonArray,&m\_error));

if(m\_error.error != QJsonParseError::NoError)

{

emit formattedError(m\_error.errorString()+":"+QString::number( m\_error.offset));

return;

}

QByteArray result=m\_jsonDocument->toJson(QJsonDocument::Indented);

emit formattedJson(QString(result));

emit formattedError("");

delete m\_jsonDocument;

m\_jsonDocument=nullptr;

}

void JsonFormat::convertJsonToTreeModel(const QVariant data)

{

QString jsonStr=data.toString();

qDebug()<<jsonStr;

if(jsonStr==nullptr||jsonStr.isNull()||jsonStr.isEmpty()){

return;

}

const QByteArray jsonArray=jsonStr.toUtf8();

QJsonDocument \*jsonDoc=new QJsonDocument(QJsonDocument::fromJson(jsonArray));

if(m\_error.error!=QJsonParseError::NoError)

{

qDebug() << "json error!"<<m\_error.errorString();

qDebug()<<m\_error.offset;

return;

}

if(jsonDoc==nullptr||jsonDoc->isEmpty()||jsonDoc->isNull()){

qDebug()<<"jsondoc is empty";

return;

}

m\_jsonModel=new JsonModel();

m\_jsonModel->convertJsonToTree(jsonDoc);

emit jsonModelChanged();

}

JsonModel::JsonModel(QObject \*parent)

{

Q\_UNUSED(parent)

}

JsonModel::~JsonModel()

{

delete rootItem;

}

int JsonModel::columnCount(const QModelIndex &parent) const

{

Q\_UNUSED(parent)

return 1;

}

QVariant JsonModel::data(const QModelIndex &index, int role) const

{

if (!index.isValid())

{

qDebug()<<"index inValid";

return QVariant();

}

TreeItem \*item = static\_cast<TreeItem\*>(index.internalPointer());

JsonCustomProperty\* prop=item->property();

switch (role) {

case JsonType::KEY:

// return QVariant::fromValue(prop);

return item->itemData();

case JsonType::VALUE:

return "value";

}

return QVariant();

}

Qt::ItemFlags JsonModel::flags(const QModelIndex &index) const

{

if (!index.isValid())

return Qt::NoItemFlags;

return QAbstractItemModel::flags(index);

}

void JsonModel::convertJsonToTree(QJsonDocument \*doc)

{

rootItem=new TreeItem();

rootItem->setData("JSON");

rootItem->setProperty("JSON","",0);

if(doc->isObject()){

QJsonObject jo=doc->object();

parseJsonObject(rootItem,&jo);

}else if(doc->isArray()){

QJsonArray ja=doc->array();

parseJsonArray(rootItem,&ja);

}

}

void JsonModel::parseJsonObject(TreeItem \*parentItem, QJsonObject \*jsonValue)

{

QStringList keyList=jsonValue->keys();

int count=keyList.size();

for(int i=0;i<count;i++){

QString key=keyList.at(i);

QJsonValue jv = jsonValue->value(key);

TreeItem \* childItem=new TreeItem(rootItem);

if(jv==QJsonValue::Undefined){

childItem->setData(key+" : "+"null");

childItem->setProperty(key,"",0);

}else if(jv.isBool()){

childItem->setData(key+" : "+(jv.toBool()?"true":"false"));

childItem->setProperty(key,(jv.toBool()?"true":"false"),1);

}else if(jv.isString()){

childItem->setData(key+" : "+jv.toString());

childItem->setProperty(key,jv.toString(),3);

}else if(jv.isDouble())

{

childItem->setData(key+" : "+QString::number(jv.toDouble()));

childItem->setProperty(key,QString::number(jv.toDouble()),4);

}

else if (jv.isObject()) {

childItem->setData(key);

childItem->setProperty(key,"",5);

QJsonObject jo=jv.toObject();

parseJsonObject(childItem,&jo);

}else if (jv.isArray()) {

childItem->setData(key);

QJsonArray ja=jv.toArray();

childItem->setProperty(key,"",6);

parseJsonArray(childItem,&ja);

}else {

childItem->setData(key+":"+jv.toString("null"));

childItem->setProperty(key,"",7);

}

parentItem->appendChild(childItem);

}

}

void JsonModel::parseJsonArray(TreeItem \*parentItem, QJsonArray \*jsonValue)

{

int count=jsonValue->size();

for(int i=0;i<count;i++){

QJsonValue jv=jsonValue->at(i);

if(jv==QJsonValue::Undefined){

continue;

}

TreeItem \* childItem=new TreeItem(rootItem);

if(jv.isBool())

{

childItem->setData(QString::number(i)+" : "+(jv.toBool()?"true":"false"));

childItem->setProperty(QString::number(i),jv.toBool()?"true":"false",1);

}

else if(jv.isString()){

childItem->setData(QString::number(i)+" : "+jv.toString());

childItem->setProperty(QString::number(i),jv.toString(),2);

}

else if (jv.isDouble()) {

childItem->setData(QString::number(i)+" : "+QString::number(jv.toDouble()));

childItem->setProperty(QString::number(i),QString::number(jv.toDouble()),3);

}

else if(jv.isObject()){

childItem->setData(QString::number(i)+" : ");

childItem->setProperty(QString::number(i),"",4);

QJsonObject jo=jv.toObject();

parseJsonObject(childItem,&jo);

}else if(jv.isArray()){

childItem->setProperty(QString::number(i),"",5);

QJsonArray ja=jv.toArray();

childItem->setData(QString::number(i)+" : ");

childItem->setProperty(QString::number(i),"",6);

parseJsonArray(childItem,&ja);

}

parentItem->appendChild(childItem);

}

}

QModelIndex JsonModel::index(int row, int column, const QModelIndex &parent) const

{

if (!hasIndex(row, column, parent))

return QModelIndex();

TreeItem \*parentItem;

if (!parent.isValid())

parentItem = rootItem;

else

parentItem = static\_cast<TreeItem\*>(parent.internalPointer());

TreeItem \*childItem = parentItem->child(row);

if (childItem)

return createIndex(row, column, childItem);

return QModelIndex();

}

QModelIndex JsonModel::parent(const QModelIndex &index) const

{

if (!index.isValid())

return QModelIndex();

TreeItem \*childItem = static\_cast<TreeItem\*>(index.internalPointer());

TreeItem \*parentItem = childItem->parentItem();

if (parentItem == rootItem)

return QModelIndex();

return createIndex(parentItem->row(), 0, parentItem);

}

int JsonModel::rowCount(const QModelIndex &parent) const

{

TreeItem \*parentItem;

if (!parent.isValid())

parentItem = rootItem;

else

parentItem = static\_cast<TreeItem\*>(parent.internalPointer());

return parentItem->childCount();

}

QHash<int, QByteArray> JsonModel::roleNames() const

{

QHash<int, QByteArray> d;

d[JsonType::KEY] = "key";

d[JsonType::VALUE] = "value";

return d;

}

#include <QDate>

#include <QDir>

#include <QTextStream>

#include <QKeySequence>

#include <QApplication>

#include <QClipboard>

#include <QTreeWidget>

#include <QPlainTextEdit>

#include "toolkit.h"

#include "setting.h"

#include "Logger.h"

#include "SettingTool.h"

#define SET\_MAX\_LOGITM 100

#define SET\_MAX\_LOGTRM 30

Logger::Logger(QObject \*parent)

: QObject(parent),m\_chkWrite(nullptr),m\_treeOut(nullptr),m\_textOut(nullptr)

{

}

Logger::~Logger()

{

m\_file.close();

}

void Logger::syncOutput()

{

QList<QTreeWidgetItem\*> list = m\_treeOut->selectedItems();

if (!list.isEmpty())

m\_textOut->setPlainText(list.first()->text(0));

else

m\_textOut->clear();

}

void Logger::ctxmenu(const QPoint& pos)

{

if (sender() == (QObject\*)m\_treeOut)

m\_cmlog.exec(m\_treeOut->mapToGlobal(pos));

else

m\_cmtxt.exec(m\_textOut->mapToGlobal(pos));

}

void Logger::copy()

{

if (sender() == (QObject\*)m\_treeOut)

{

QList<QTreeWidgetItem\*> list = m\_treeOut->selectedItems();

if (!list.isEmpty())

QApplication::clipboard()->setText(list.first()->text(0));

}

else

{

m\_textOut->copy();

}

}

void Logger::writeLogFile(const QString& info)

{

if (!m\_chkWrite->isChecked())

return;

m\_file.close();

m\_file.setFileName(getLogFileName());

if (m\_file.open(QIODevice::Append|

QIODevice::WriteOnly|

QIODevice::Text))

{

QByteArray a(info.toUtf8());

const char\* d = a.data();

for (int n=a.size(); n>0;)

{

int w = m\_file.write(d, n);

d += w;

n -= w;

}

m\_file.close();

}

}

void Logger::clear()

{

m\_treeOut->clear();

m\_textOut->clear();

}

void Logger::output(const QString& info)

{

output("MSG", info);

}

void Logger::output(const char\* buf, uint len)

{

output("DAT", buf, len);

}

void Logger::pack()

{

if (m\_treeOut->topLevelItemCount() > SET\_MAX\_LOGITM)

m\_treeOut->model()->removeRows(0, SET\_MAX\_LOGTRM);

m\_treeOut->scrollToBottom();

}

QTreeWidgetItem\* Logger::appendLogEntry(QTreeWidgetItem\* p, const QString& t)

{

QTreeWidgetItem\* res = new QTreeWidgetItem(p);

if (res)

{

res->setText(0, t);

if (p)

{

p->addChild(res);

}

else

{

m\_treeOut->addTopLevelItem(res);

m\_textOut->setPlainText(t);

}

}

return res;

}

void Logger::output(const QString& title, const QString& info)

{

QTreeWidgetItem\* it = new QTreeWidgetItem(0);

if (!it) return;

QString lab(QTime::currentTime().toString("HH:mm:ss "));

lab += title;

lab += ' ';

lab += info;

appendLogEntry(0, lab);

pack();

lab += '\n';

lab += '\n';

writeLogFile(lab);

}

void Logger::output(const QString& title, const char\* buf, quint32 len)

{

QString lab(QTime::currentTime().toString("HH:mm:ss "));

QTextStream out(&lab);

out << title

<< " <" << len << "> "

<< TK::bin2ascii(buf, len);

QString hex = TK::bin2hex(buf, len);

QTreeWidgetItem\* it = appendLogEntry(0, lab);

if (it)

{

appendLogEntry(it, hex);

pack();

}

out << '\n' << hex << '\n' << '\n';

writeLogFile(lab);

}

#include "LogMessageModel.h"

#include "toolkit.h"

#include <QDebug>

#include <bitset>

LogMessageList::LogMessageList(QObject \*parent)

{

Q\_UNUSED(parent)

}

LogMessageList::~LogMessageList()

{

}

void LogMessageList::setCurrentItem(int index)

{

this->m\_currentData=m\_dataList[index];

}

void LogMessageList::addData(LogMessageModel \*model)

{

beginInsertRows(QModelIndex(), rowCount(), rowCount());

m\_dataList.append(model);

endInsertRows();

}

void LogMessageList::clearData()

{

beginRemoveRows(QModelIndex(), 0, m\_dataList.size()-1);

qDeleteAll(m\_dataList);

m\_dataList.clear();

endRemoveRows();

}

void LogMessageList::clearRecvData()

{

int length=m\_dataList.size();

if(length<=0){

return;

}

for(int i=length-1;i>=0;i--){

if(m\_dataList.at(i)->isRev()){

beginRemoveRows(QModelIndex(),i,i);

m\_dataList.removeAt(i);

endRemoveRows();

}

}

}

void LogMessageList::clearSendData()

{

int length=m\_dataList.size();

if(length<=0){

return;

}

for(int i=length-1;i>=0;i--){

if(!m\_dataList.at(i)->isRev()){

beginRemoveRows(QModelIndex(),i,1);

m\_dataList.removeAt(i);

endRemoveRows();

}

}

}

LogMessageModel\* LogMessageList::get(int index)

{

if(index<0||index>=m\_dataList.size()){

return nullptr;

}

return m\_dataList[index];

}

int LogMessageList::rowCount(const QModelIndex &parent) const

{

Q\_UNUSED(parent)

return m\_dataList.size();

}

QVariant LogMessageList::data(const QModelIndex &index, int role) const

{

if (index.row() < 0 || index.row() >= m\_dataList.size())

{

return QVariant();

}

LogMessageModel\* d = m\_dataList[index.row()];

qDebug()<<"now is "<<d->buf();

if (role == Datatype::time)

{

return d->time();

}

else if (role == Datatype::isRev)

{

return d->isRev();

}

else if (role == Datatype::host)

{

return d->host();

}

else if (role == Datatype::ascData) {

QString ascData = TK::bin2ascii(d->buf(), static\_cast<uint>(d->length()));

return ascData;

}

else if (role == Datatype::hexData) {

QString hexData = TK::bin2hex(d->buf(),static\_cast<uint>(d->length()));

return hexData;

}

else if (role == Datatype::textData) {

return QString(d->buf());

}

else if (role == Datatype::length) {

return d->length();

}

else if (role == Datatype::binData) {

// QString binData="0x"+d->byteArray().toHex(' ').toUpper();

// QStringList list=binData.split(' ');

// return list.join(" 0x");

return "123";

}

return QVariant();

}

QHash<int, QByteArray> LogMessageList::roleNames() const

{

QHash<int, QByteArray> d;

d[Datatype::time] = "time";

d[Datatype::isRev] = "isRev";

d[Datatype::host] = "host";

d[Datatype::ascData]="ascData";

d[Datatype::hexData] = "hexData";

d[Datatype::textData] ="textData";

d[Datatype::length] = "length";

d[Datatype::binData] = "binData";

return d;

}

#include "logtreemodel.h"

LogTreeModel::LogTreeModel(QObject \*parent) : QAbstractItemModel(parent) {}

LogTreeModel::~LogTreeModel() {

}

QModelIndex LogTreeModel::index(int row, int column, const QModelIndex &parent) const {

return createIndex(row, column);

}

QModelIndex LogTreeModel::parent(const QModelIndex &child) const {

return QModelIndex();

}

int LogTreeModel::rowCount(const QModelIndex &parent) const {

// 根节点下有2行，其它行下没有

if (parent.row() == -1)

{

return 2;

}

return 0;

}

int LogTreeModel::columnCount(const QModelIndex &parent) const {

return 2;

}

QVariant LogTreeModel::data(const QModelIndex &index, int role) const {

return QVariant();

}

//

// Created by 冉高飞 on 2019-02-20.

//

#include <QQmlApplicationEngine>

#include <QWindow>

#include <QApplication>

#include <QTranslator>

#include <QLibraryInfo>

#include <QThread>

#include <QHostInfo>

#include <QNetworkInterface>

#include "bluetoothmodel.h"

#include "TcpServerModel.h"

#include "UdpServerModel.h"

#include "notepadmodel.h"

#include "TcpClientModel.h"

#include "UdpClientModel.h"

#include "SettingTool.h"

#include "TranslatorTool.h"

#include "HttpManager.h"

#include "sqlitetool.h"

#include "SendMessageData.h"

#include "LogMessageModel.h"

#include "SoundManager.h"

#include "JsonFormat.h"

/\*\*

\* @brief initSettings

\* 初始化配置文件信息

\*/

void initSettings(){

QCoreApplication::setOrganizationName("rangaofei");

QCoreApplication::setOrganizationDomain("rangaofei.cn");

QCoreApplication::setApplicationName("SSokit");

}

/\*\*

\* @brief registerQml

\* 注册所有的qml文件

\*/

void registerQml(){

qmlRegisterType<NotepadModel>("src.notepadmodel", 1, 0, "NotepadModel");

qmlRegisterType<BlueToothModel>("src.bluetoothmodel", 1, 0, "BlueToothModel");

qmlRegisterType<TcpServerModel>("src.tcpservermodel", 1, 0, "TcpServerModel");

qmlRegisterType<UdpServerModel>("src.udpservermodel", 1, 0, "UdpServerModel");

qmlRegisterType<TcpClientModel>("src.tcpclientmodel", 1, 0, "TcpClientModel");

qmlRegisterType<UdpClientModel>("src.udpclientmodel", 1, 0, "UdpClientModel");

qmlRegisterType<SendMessageData>("src.sendmessagedata",1,0,"SendMessageData");

qmlRegisterType<JsonFormat>("src.jsonformat",1,0,"JsonFormat");

qmlRegisterUncreatableType<LogMessageModel>("src.logmessagedata",1,0,"LogMessageData","Reference only");

qmlRegisterType<NoteBook>("src.notebook",1,0,"NoteBook");

qmlRegisterSingletonType<SQLiteTool>("src.sqlitetool",1,0,"SqliteTool",[](QQmlEngine \*engine, QJSEngine \*scriptEngine) -> QObject \* {

Q\_UNUSED(engine)

Q\_UNUSED(scriptEngine)

SQLiteTool \*sqliteTool = new SQLiteTool();

return sqliteTool;

});

qmlRegisterType<ItemPort>("src.itemport", 1, 0, "ItemPort");

qmlRegisterSingletonType(QUrl("qrc:/assets/language/Strings.qml"), "src.strings", 1, 0, "Strings");

qmlRegisterSingletonType(QUrl("qrc:/assets/Colors.qml"),"src.colors",1,0,"Colors");

qmlRegisterSingletonType<SettingTool>("src.settingtool",1,0,"SettingTool",

[](QQmlEngine \*engine, QJSEngine \*scriptEngine) -> QObject \* {

Q\_UNUSED(engine)

Q\_UNUSED(scriptEngine)

return SettingTool::getInstance();

});

qmlRegisterSingletonType<HttpManager>("src.httpmanager",1,0,"HttpManager", [](QQmlEngine \*engine, QJSEngine \*scriptEngine) -> QObject \* {

Q\_UNUSED(engine)

Q\_UNUSED(scriptEngine)

HttpManager \*httpManager = new HttpManager();

return httpManager;

});

qmlRegisterSingletonType<Config>("src.config",1,0,"Config", [](QQmlEngine \*engine, QJSEngine \*scriptEngine) -> QObject \* {

Q\_UNUSED(engine)

Q\_UNUSED(scriptEngine)

Config \*config = new Config();

return config;

});

qmlRegisterSingletonType<SoundManager>("src.soundmanager",1,0,"SoundManager", [](QQmlEngine \*engine, QJSEngine \*scriptEngine) -> QObject \* {

Q\_UNUSED(engine)

Q\_UNUSED(scriptEngine)

SoundManager \*manager = new SoundManager();

return manager;

});

qmlRegisterUncreatableType<TreeItem>("src.treeitem",1,0,"TreeItem","Reference only");

}

/\*\*

\* 程序入口

\*/

int main(int argc, char \*argv[]) {

QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);

QApplication app(argc, argv);

initSettings();

TranslatorTool translatorTool;

translatorTool.initLanguage();

registerQml();

QQmlApplicationEngine engine;

engine.load(QUrl(QStringLiteral("qrc:/qml/SSokit.qml")));

if (engine.rootObjects().isEmpty())

return -1;

return QApplication::exec();

}

#include "toolkit.h"

#include "setting.h"

#include "notepadform.h"

#include <QFile>

#include <QTextStream>

#include <QVBoxLayout>

#include <QShortcut>

#include <QTabWidget>

#define SET\_NP\_FILE "sokit.txt"

NotepadForm::NotepadForm(QWidget\* p, Qt::WindowFlags f)

:QWidget(p, f)

{

setupUi();

}

NotepadForm::~NotepadForm()

{

uninit();

}

void NotepadForm::setupUi()

{

resize(680, 450);

QVBoxLayout\* lay = new QVBoxLayout(this);

lay->setSpacing(5);

lay->setContentsMargins(5, 5, 5, 5);

m\_board = new QPlainTextEdit(this);

m\_board->setAcceptDrops(false);

m\_board->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

m\_board->setTabChangesFocus(false);

m\_board->setUndoRedoEnabled(true);

m\_board->setTextInteractionFlags(Qt::TextEditorInteraction);

lay->addWidget(m\_board);

setWindowTitle(tr("NotepadModel"));

QMetaObject::connectSlotsByName(this);

QShortcut\* n = new QShortcut(QKeySequence(Qt::Key\_Tab+Qt::CTRL), this);

QShortcut\* l = new QShortcut(QKeySequence(Qt::Key\_Tab+Qt::SHIFT), this);

n->setObjectName("n");

l->setObjectName("l");

connect(n, SIGNAL(activated()), this, SLOT(jumptab()));

connect(l, SIGNAL(activated()), this, SLOT(jumptab()));

}

bool NotepadForm::init()

{

QFile file(Setting::path() + "/" + SET\_NP\_FILE);

if (file.open(QIODevice::ReadOnly|QIODevice::Text))

{

QTextStream str(&file);

m\_board->setPlainText(str.readAll());

file.close();

}

return true;

}

void NotepadForm::uninit()

{

QFile file(Setting::path() + "/" + SET\_NP\_FILE);

if (file.open(QIODevice::WriteOnly|QIODevice::Text|QIODevice::Truncate))

{

QTextStream str(&file);

str << m\_board->toPlainText();

str.flush();

file.close();

}

}

void NotepadForm::jumptab()

{

qDebug("tab key");

if (parent())

{

QTabWidget\* p = qobject\_cast<QTabWidget\*>(parent()->parent());

if (p)

{

qint32 max = p->count();

qint32 cur = p->indexOf(this);

if (sender()->objectName().startsWith('n')) {

if (++cur >= max) cur = 0;

} else {

if (--cur < 0) cur = (max>0) ? (max-1) : 0;

}

p->setCurrentIndex(cur);

}

}

}

#include "notepadmodel.h"

NotepadModel::NotepadModel(QObject \*parent) : QObject(parent) {

}

QString NotepadModel::placeHolder() {

return "123";

}

void NotepadModel::setPlaceHolder(const QString &data) {

m\_placeHolder = data;

}

#include "QmlLogModel.h"

QmlLogModel::QmlLogModel(QObject \*parent) : QObject(parent),m\_revCount(0),m\_senCount(0)

{

m\_dataList=new LogMessageList;

}

QmlLogModel::~QmlLogModel()

{

}

/\*\*\*

\* 接收的信息数量

\* @brief QmlLogModel::revCount

\* @return

\*/

qint64 QmlLogModel::revCount()

{

return m\_revCount;

}

/\*\*\*

\* 设置接收信息数量

\* @brief QmlLogModel::setRevCount

\* @param count

\*/

void QmlLogModel::setRevCount(qint64 count)

{

this->m\_revCount=count;

}

qint64 QmlLogModel::senCount()

{

return m\_senCount;

}

void QmlLogModel::setSenCount(qint64 count)

{

this->m\_senCount=count;

}

LogMessageList \*QmlLogModel::dataList()

{

return m\_dataList;

}

void QmlLogModel::setDataList(LogMessageList \*dataList)

{

this->m\_dataList=dataList;

}

void QmlLogModel::dumpLogMsg(bool rev, QString &host,const char\* buf, qint64 length)

{

LogMessageModel\* data=new LogMessageModel;

data->setIsRev(rev);

data->setHost(host);

data->setBuf(buf);

data->setLength(length);

dumpLogMsg(data);

}

void QmlLogModel::dumpLogMsg(LogMessageModel\* data) {

m\_dataList->addData(data);

if(data->isRev()){

++m\_revCount;

emit revCountChanged();

}else {

++m\_senCount;

emit senCountChanged();

}

emit setCurrentIndex(m\_dataList->rowCount()-1);

}

void QmlLogModel::clearRevCount(bool clearData)

{

this->m\_revCount=0;

emit revCountChanged();

if(clearData){

this->m\_dataList->clearRecvData();

emit dataListChanged();

}

}

void QmlLogModel::clearSenCount(bool clearData)

{

this->m\_senCount=0;

emit senCountChanged();

if(clearData){

this->m\_dataList->clearSendData();

emit dataListChanged();

}

}

void QmlLogModel::clearAll()

{

clearRevCount(false);

clearSenCount(false);

this->m\_dataList->clearData();

emit dataListChanged();

}

void QmlLogModel::clearData(QmlLogModel::ClearType type)

{

switch (type) {

case ALL:

this->m\_dataList->clearData();

break;

case RECEIVE:

this->m\_dataList->clearRecvData();

break;

case SEND:

this->m\_dataList->clearSendData();

break;

}

}

#include "ReceiveMessageData.h"

ReceiveMessageData::ReceiveMessageData(QObject \*parent) : QObject(parent)

{

}

#include "SendMessageData.h"

SendMessageData::SendMessageData(QObject \*parent) : QObject(parent)

{

}

SendMessageData::~SendMessageData()

{

}

const QString SendMessageData::content()

{

return this->m\_content;

}

void SendMessageData::setContent(QString content)

{

this->m\_content=content;

}

const QString SendMessageData::header()

{

return this->m\_header;

}

void SendMessageData::setHeader(QString header)

{

this->m\_header=header;

}

const QString SendMessageData::footer()

{

return this->m\_footer;

}

void SendMessageData::setFooter(QString footer)

{

this->m\_footer=footer;

}

bool SendMessageData::withHeader()

{

return m\_withHeader;

}

void SendMessageData::setWithHeader(bool withHeader)

{

this->m\_withHeader=withHeader;

}

int SendMessageData::lengthSize()

{

return this->m\_lengthSize;

}

void SendMessageData::setLengthSize(int lengthSize)

{

this->m\_lengthSize=lengthSize;

}

bool SendMessageData::endian()

{

return this->m\_endian;

}

void SendMessageData::setEndian(bool endian)

{

this->m\_endian=endian;

}

bool SendMessageData::plainText()

{

return this->m\_plainText;

}

void SendMessageData::setPlainText(bool plainText)

{

this->m\_plainText=plainText;

}

QString SendMessageData::getTargetMsg()

{

return this->content()+this->m\_footer;

}

#include "toolkit.h"

#include "setting.h"

#include "serverform.h"

#include <QShortcut>

#define SET\_SEC\_SVR "server"

#define SET\_KEY\_SVR "/server"

#define SET\_KEY\_CMBTA "/tip"

#define SET\_KEY\_CMBUA "/uip"

#define SET\_KEY\_CMBTP "/tport"

#define SET\_KEY\_CMBUP "/uport"

#define SET\_VAL\_LGSVR "log\_server"

ServerForm::ServerForm(QWidget \*parent, Qt::WindowFlags flags)

: BaseForm(parent, flags)

{

m\_ui.setupUi(this);

}

ServerForm::~ServerForm()

{

if (lock(1000))

{

m\_tcp.disconnect(this);

m\_udp.disconnect(this);

m\_tcp.stop();

m\_udp.stop();

unlock();

}

saveConfig();

}

void ServerForm::initConfig()

{

QString sss(SET\_SEC\_SVR);

Setting::lord(sss+SET\_KEY\_CMBTA, SET\_PFX\_CMBITM, \*m\_ui.cmbTcpAddr, false);

Setting::lord(sss+SET\_KEY\_CMBUA, SET\_PFX\_CMBITM, \*m\_ui.cmbUdpAddr, false);

Setting::lord(sss+SET\_KEY\_CMBTP, SET\_PFX\_CMBITM, \*m\_ui.cmbTcpPort);

Setting::lord(sss+SET\_KEY\_CMBUP, SET\_PFX\_CMBITM, \*m\_ui.cmbUdpPort);

QString skl(SET\_SEC\_DIR); skl += SET\_KEY\_LOG;

skl = Setting::get(skl, SET\_KEY\_SVR, SET\_VAL\_LGSVR);

setProperty(SET\_SEC\_DIR, skl);

TK::initNetworkInterfaces(m\_ui.cmbTcpAddr, true);

TK::initNetworkInterfaces(m\_ui.cmbUdpAddr, true);

}

void ServerForm::saveConfig()

{

QString sss(SET\_SEC\_SVR);

Setting::save(sss+SET\_KEY\_CMBTA, SET\_PFX\_CMBITM, \*m\_ui.cmbTcpAddr, false);

Setting::save(sss+SET\_KEY\_CMBUA, SET\_PFX\_CMBITM, \*m\_ui.cmbUdpAddr, false);

Setting::save(sss+SET\_KEY\_CMBTP, SET\_PFX\_CMBITM, \*m\_ui.cmbTcpPort);

Setting::save(sss+SET\_KEY\_CMBUP, SET\_PFX\_CMBITM, \*m\_ui.cmbUdpPort);

QString skl(SET\_SEC\_DIR); skl += SET\_KEY\_LOG;

Setting::set(skl, SET\_KEY\_SVR, property(SET\_SEC\_DIR).toString());

}

bool ServerForm::initForm()

{

initCounter(m\_ui.labRecv, m\_ui.labSend);

initLogger(m\_ui.chkLog, m\_ui.btnClear, m\_ui.treeOutput, m\_ui.txtOutput);

initLister(m\_ui.btnConnAll, m\_ui.btnConnDel, m\_ui.lstConn);

bindBuffer(1, m\_ui.edtBuf1, m\_ui.btnSend1, 0);

bindBuffer(2, m\_ui.edtBuf2, m\_ui.btnSend2, 0);

bindBuffer(3, m\_ui.edtBuf3, m\_ui.btnSend3, 0);

connect(m\_ui.btnTcp, SIGNAL(clicked(bool)), this, SLOT(trigger(bool)));

connect(m\_ui.btnUdp, SIGNAL(clicked(bool)), this, SLOT(trigger(bool)));

connect(&m\_tcp, SIGNAL(connOpen(const QString&)), this, SLOT(listerAdd(const QString&)));

connect(&m\_tcp, SIGNAL(connClose(const QString&)), this, SLOT(listerRemove(const QString&)));

connect(&m\_tcp, SIGNAL(message(const QString&)), this, SIGNAL(output(const QString&)));

connect(&m\_tcp, SIGNAL(dumpbin(const QString&,const char\*,quint32)), this, SIGNAL(output(const QString&,const char\*,quint32)));

connect(&m\_tcp, SIGNAL(countRecv(qint32)), this, SLOT(countRecv(qint32)));

connect(&m\_tcp, SIGNAL(countSend(qint32)), this, SLOT(countSend(qint32)));

connect(&m\_udp, SIGNAL(connOpen(const QString&)), this, SLOT(listerAdd(const QString&)));

connect(&m\_udp, SIGNAL(connClose(const QString&)), this, SLOT(listerRemove(const QString&)));

connect(&m\_udp, SIGNAL(message(const QString&)), this, SIGNAL(output(const QString&)));

connect(&m\_udp, SIGNAL(dumpbin(const QString&,const char\*,quint32)), this, SIGNAL(output(const QString&,const char\*,quint32)));

connect(&m\_udp, SIGNAL(countRecv(qint32)), this, SLOT(countRecv(qint32)));

connect(&m\_udp, SIGNAL(countSend(qint32)), this, SLOT(countSend(qint32)));

return true;

}

bool ServerForm::initHotkeys()

{

bindFocus(m\_ui.cmbTcpAddr, Qt::Key\_Escape);

bindClick(m\_ui.btnTcp, Qt::CTRL + Qt::Key\_T);

bindClick(m\_ui.btnUdp, Qt::CTRL + Qt::Key\_U);

return true;

}

void ServerForm::kill(QStringList& list)

{

QString tcpname(TK::socketTypeName(true));

while (!list.isEmpty())

{

QString key = list.takeFirst();

if (key.contains(tcpname))

m\_tcp.kill(key);

else

m\_udp.kill(key);

}

}

void ServerForm::trigger(bool start)

{

QToolButton\* btnTrigger = qobject\_cast<QToolButton\*>(sender());

if (!btnTrigger) return;

bool istcp = (btnTrigger==m\_ui.btnTcp);

QComboBox\* cbAddr = istcp ? m\_ui.cmbTcpAddr : m\_ui.cmbUdpAddr;

QComboBox\* cbPort = istcp ? m\_ui.cmbTcpPort : m\_ui.cmbUdpPort;

ServerSkt\* server = istcp ? (ServerSkt\*)&m\_tcp : (ServerSkt\*)&m\_udp;

IPAddr addr;

if (start)

start = TK::popIPAddr(cbAddr, cbPort, addr);

lock();

if (start)

start = server->start(addr.ip, addr.port);

else

server->stop();

unlock();

cbAddr->setDisabled(start);

cbPort->setDisabled(start);

if (start)

TK::pushIPAddr(0, cbPort);

else

TK::resetPushBox(btnTrigger);

}

void ServerForm::send(const QString& data, const QString&)

{

QStringList list;

if (lock(1000))

{

listerSelected(list);

unlock();

}

QString tcpname(TK::socketTypeName(true));

while (!list.isEmpty())

{

QString key = list.takeFirst();

ServerSkt\* server = key.contains(tcpname)

? (ServerSkt\*)&m\_tcp : (ServerSkt\*)&m\_udp;

server->send(key, data);

}

}

#include <QNetworkInterface>

#include <QDataStream>

#include "ServerModel.h"

#include "toolkit.h"

ServerModel::ServerModel(QObject \*parent) : QmlLogModel(parent)

{

}

ServerModel::~ServerModel() = default;

void ServerModel::initConfig() {

}

void ServerModel::saveConfig() {

}

void ServerModel::kill(QStringList &list) {

Q\_UNUSED(list)

}

void ServerModel::getAddr() {

QList<QHostAddress> lst = QNetworkInterface::allAddresses();

foreach (QHostAddress a, lst) {

if (QAbstractSocket::IPv4Protocol == a.protocol()) {

emit appendLocalAddr(a.toString());

}

}

}

void ServerModel::toggleConnect(bool checked, QString addr, QString port) {

auto r\_port = quint16(port.toUInt());

if (checked) {

m\_conns.clear();

openServer(addr, r\_port);

} else {

closeServer();

}

}

bool ServerModel::closeServer() {

OBJMAP::const\_iterator i;

for (i = m\_conns.constBegin(); i != m\_conns.constEnd(); ++i)

{

QString k = i.key();

void\* v = i.value();

if (close(v))

emit connClose(k);

}

m\_conns.clear();

close();

emit sendErrMsg(CLOSE\_ERR,"Close Server",false);

return true;

}

void ServerModel::getKeys(QStringList &res) {

res = m\_conns.keys();

}

void ServerModel::setCookie(const QString &k, void \*v) {

void \*o = m\_conns.value(k);

if (o) {

if (close(o))

emit connClose(k);

}

m\_conns.insert(k, v);

emit appendConnAddr(k);

}

void ServerModel::unsetCookie(const QString &k) {

m\_conns.remove(k);

emit connClose(k);

}

void \*ServerModel::getCookie(const QString &k) {

return m\_conns.value(k);

}

void ServerModel::kill(const QString &key) {

void \*v = m\_conns.value(key);

if (v) {

if (close(v))

unsetCookie(key);

} else {

unsetCookie(key);

}

}

void ServerModel::send(const QString &key, const QString &data) {

void \*v = m\_conns.value(key);

if (v) {

QString err;

QByteArray bin;

if (!TK::ascii2bin(data, bin, err))

qDebug() << ("bad data format to send: " + err);

else

qDebug()<<bin<<"==bin";

sendToDst(v, bin);

}

}

void ServerModel::sendWithHeader(const QString &key,const QString header, const qint32 lengthSize, bool bigEndian, const QString &data)

{

void \*v = m\_conns.value(key);

if(!v){

return;

}

QString err;

QByteArray dataBin;

QByteArray headerBin;

if (!TK::ascii2bin(header, headerBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

if (!TK::ascii2bin(data, dataBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

QByteArray length;

length.resize(lengthSize);

QDataStream sendData(&length,QIODevice::WriteOnly);

sendData.setByteOrder(bigEndian? QDataStream::BigEndian: QDataStream::LittleEndian);

switch (lengthSize) {

case 0:

break;

case 2:

sendData<<quint16(dataBin.size());

break;

case 4:

sendData<<quint32(dataBin.size());

break;

}

dataBin.prepend(length);

dataBin.prepend(headerBin);

sendToDst(v,dataBin);

}

void ServerModel::sendMessageData(const QString &key,SendMessageData \*data)

{

if(!data->withHeader()){

send(key,data->content());

return;

}

void \*v = m\_conns.value(key);

if(!v){

return;

}

QString err;

QByteArray dataBin;

QByteArray headerBin;

if (!TK::ascii2bin(data->header(), headerBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

if (!TK::ascii2bin(data->getTargetMsg(), dataBin, err)) {

qDebug() << ("bad data format to send: " + err);

return;

}

QByteArray length;

length.resize(data->lengthSize());

QDataStream sendData(&length,QIODevice::WriteOnly);

sendData.setByteOrder(data->endian()? QDataStream::BigEndian: QDataStream::LittleEndian);

switch (data->lengthSize()) {

case 0:

break;

case 2:

sendData<<quint16(dataBin.size());

break;

case 4:

sendData<<quint32(dataBin.size());

break;

}

dataBin.prepend(length);

dataBin.prepend(headerBin);

sendToDst(v,dataBin);

}

#include <QTcpSocket>

#include <QStringList>

#include "toolkit.h"

#include "serverskt.h"

#define PROP\_CONN "CONN"

#define MAXBUFFER 1024\*1024

ServerSkt::ServerSkt(QObject \*parent)

: QObject(parent),m\_port(0)

{

m\_started = false;

}

ServerSkt::~ServerSkt()

{

}

bool ServerSkt::start(const QHostAddress& ip, quint16 port)

{

m\_ip = ip;

m\_port = port;

m\_conns.clear();

m\_error.clear();

m\_started = open();

QString msg("start %1 server %2!");

if (!m\_started)

{

msg = msg.arg(name(),"failed");

if (!m\_error.isEmpty())

{ msg += " error:[";

msg += m\_error;

msg += "].";

}

}

else

{

msg = msg.arg(name(),"successfully");

}

show(msg);

return m\_started;

}

void ServerSkt::stop()

{

if (!m\_started)

return;

OBJMAP::const\_iterator i;

for (i = m\_conns.constBegin(); i != m\_conns.constEnd(); ++i)

{

QString k = i.key();

void\* v = i.value();

if (close(v))

emit connClose(k);

}

m\_conns.clear();

close();

show(QString("stop %1 server!").arg(name()));

m\_started = false;

}

void ServerSkt::setError(const QString& err)

{

m\_error = err;

}

void ServerSkt::recordRecv(qint32 bytes)

{

emit countRecv(bytes);

}

void ServerSkt::recordSend(qint32 bytes)

{

emit countSend(bytes);

}

void ServerSkt::getKeys(QStringList& res)

{

res = m\_conns.keys();

}

void ServerSkt::setCookie(const QString& k, void\* v)

{

void\* o = m\_conns.value(k);

if (o)

{

if (close(o))

emit connClose(k);

}

m\_conns.insert(k, v);

emit connOpen(k);

}

void ServerSkt::unsetCookie(const QString& k)

{

m\_conns.remove(k);

emit connClose(k);

}

void\* ServerSkt::getCookie(const QString& k)

{

return m\_conns.value(k);

}

void ServerSkt::kill(const QString& key)

{

void\* v = m\_conns.value(key);

if (v)

{

if (close(v))

unsetCookie(key);

}

else

{

unsetCookie(key);

}

}

void ServerSkt::send(const QString& key, const QString& data)

{

void\* v = m\_conns.value(key);

if (v)

{

QString err;

QByteArray bin;

if (!TK::ascii2bin(data, bin, err))

show("bad data format to send: "+err);

else

send(v, bin);

}

}

void ServerSkt::dump(const char\* buf, qint32 len, bool isSend, const QString& key)

{

emit dumpbin(QString("DAT %1 %2").arg(isSend?"<---":"--->",key), buf, (quint32)len);

}

void ServerSkt::show(const QString& msg)

{

emit message(msg);

}

ServerSktTcp::ServerSktTcp(QObject \*parent)

:ServerSkt(parent)

{

}

ServerSktTcp::~ServerSktTcp()

{

m\_server.disconnect(this);

stop();

}

bool ServerSktTcp::open()

{

if (m\_server.listen(addr(), port()))

{

connect(&m\_server, SIGNAL(newConnection()), this, SLOT(newConnection()));

return true;

}

else

{

setError(QString("%1, %2").arg(m\_server.serverError()).arg(m\_server.errorString()));

}

return false;

}

bool ServerSktTcp::close(void\* cookie)

{

Conn\* conn = (Conn\*)cookie;

if (conn->client)

conn->client->disconnect(this);

delete conn->client;

delete conn;

return true;

}

void ServerSktTcp::close(QObject\* obj)

{

Conn\* conn = (Conn\*)obj->property(PROP\_CONN).value<void\*>();

if (!conn) return;

unsetCookie(conn->key);

delete conn;

}

void ServerSktTcp::error()

{

QTcpSocket\* s = qobject\_cast<QTcpSocket\*>(sender());

show(QString("TCP socket error %1, %2").arg(s->error()).arg(s->errorString()));

s->deleteLater();

}

void ServerSktTcp::close()

{

m\_server.close();

m\_server.disconnect(this);

}

void ServerSktTcp::newConnection()

{

QTcpServer\* server = qobject\_cast<QTcpServer\*>(sender());

if (!server) return;

QTcpSocket\* client = server->nextPendingConnection();

while (client)

{

Conn\* conn = new Conn;

if (!conn)

{

client->deleteLater();

}

else

{

client->setProperty(PROP\_CONN, qVariantFromValue((void\*)conn));

conn->client = client;

conn->key = TK::ipstr(client->peerAddress(),client->peerPort(), true);

connect(client, SIGNAL(readyRead()), this, SLOT(newData()));

connect(client, SIGNAL(destroyed(QObject\*)), this, SLOT(close(QObject\*)));

connect(client, SIGNAL(disconnected()), client, SLOT(deleteLater()));

connect(client, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

setCookie(conn->key, conn);

}

client = server->nextPendingConnection();

}

}

void ServerSktTcp::newData()

{

QTcpSocket\* client = qobject\_cast<QTcpSocket\*>(sender());

if (!client) return;

Conn\* conn = (Conn\*)client->property(PROP\_CONN).value<void\*>();

if (!conn) return;

qint64 bufLen = client->bytesAvailable();

char\* buf = TK::createBuffer(bufLen, MAXBUFFER);

if (!buf) return;

qint64 readLen = 0;

qint64 ioLen = client->read(buf, bufLen);

while (ioLen > 0)

{

readLen += ioLen;

ioLen = client->read(buf+readLen, bufLen-readLen);

}

if (ioLen >= 0)

{

recordRecv(readLen);

dump(buf, readLen, false, conn->key);

}

TK::releaseBuffer(buf);

}

void ServerSktTcp::send(void\* cookie, const QByteArray& bin)

{

Conn\* conn = (Conn\*)cookie;

const char \* src = bin.constData();

qint64 srcLen = bin.length();

qint64 writeLen = 0;

qint64 ioLen = conn->client->write(src, srcLen);

while (ioLen > 0)

{

writeLen += ioLen;

ioLen = conn->client->write(src+writeLen, srcLen-writeLen);

}

if (writeLen != srcLen)

{

show(QString("failed to send data to %1:%2 [%3]")

.arg(addr().toString()).arg(port()).arg(writeLen));

return;

}

recordSend(writeLen);

dump(src, srcLen, true, conn->key);

}

ServerSktUdp::ServerSktUdp(QObject \*parent)

:ServerSkt(parent)

{

}

ServerSktUdp::~ServerSktUdp()

{

m\_server.disconnect(this);

stop();

}

void ServerSktUdp::error()

{

QUdpSocket\* s = qobject\_cast<QUdpSocket\*>(sender());

show(QString("UDP socket error %1, %2").arg(s->error()).arg(s->errorString()));

}

bool ServerSktUdp::open()

{

if (m\_server.bind(addr(), port(), QUdpSocket::ShareAddress))

{

connect(&m\_server, SIGNAL(readyRead()), this, SLOT(newData()));

connect(&m\_server, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

connect(&m\_timer, SIGNAL(timeout()), this, SLOT(check()));

m\_timer.start(2000);

return true;

}

else

{

setError(QString("%1, %2").arg(m\_server.error()).arg(m\_server.errorString()));

}

return false;

}

bool ServerSktUdp::close(void\* cookie)

{

delete (Conn\*)cookie;

return true;

}

void ServerSktUdp::close()

{

m\_timer.disconnect(this);

m\_timer.stop();

m\_server.close();

m\_server.disconnect(this);

}

void ServerSktUdp::newData()

{

QUdpSocket\* s = qobject\_cast<QUdpSocket\*>(sender());

if (!s) return;

qint64 bufLen = s->pendingDatagramSize();

char\* buf = TK::createBuffer(bufLen, MAXBUFFER);

if (!buf) return;

QHostAddress addr; quint16 port(0);

qint64 readLen = 0;

qint64 ioLen = s->readDatagram(buf, bufLen, &addr, &port);

if (ioLen >= 0)

{

Conn\* conn = (Conn\*)getCookie(TK::ipstr(addr, port, false));

if (!conn)

{

conn = new Conn;

if (conn)

{

conn->key = TK::ipstr(addr, port, false);

conn->addr = addr;

conn->port = port;

setCookie(conn->key, conn);

}

}

if (conn)

{

recordRecv(readLen);

conn->stamp = QDateTime::currentDateTime();

dump(buf, readLen, false, conn->key);

}

}

TK::releaseBuffer(buf);

}

void ServerSktUdp::send(void\* cookie, const QByteArray& bin)

{

Conn\* conn = (Conn\*)cookie;

const char\* src = bin.constData();

qint64 srcLen = bin.length();

qint64 writeLen = 0;

qint64 ioLen = m\_server.writeDatagram(src, srcLen, conn->addr, conn->port);

while (ioLen > 0)

{

writeLen += ioLen;

ioLen = (writeLen >= srcLen) ? 0 :

m\_server.writeDatagram(src+writeLen, srcLen-writeLen, conn->addr, conn->port);

}

if (writeLen != srcLen)

{

show(QString("failed to send data to %1:%2 [%3]")

.arg(conn->addr.toString()).arg(conn->port).arg(writeLen));

return;

}

recordSend(writeLen);

dump(src, srcLen, true, conn->key);

}

void ServerSktUdp::check()

{

QStringList list;

getKeys(list);

while (!list.isEmpty())

{

QString k = list.takeFirst();

void\* c = getCookie(k);

if (c && (((Conn\*)c)->stamp.addSecs(120) < QDateTime::currentDateTime()))

{

close(c);

unsetCookie(k);

}

}

}

#include "SettingTool.h"

SettingTool::SettingTool(QObject \*parent) : QObject(parent)

{

settings=new QSettings;

}

SettingTool::~SettingTool()

{

if( settings!=nullptr){

delete SettingTool::settings;

settings=nullptr;

}

}

void SettingTool::setLaunched()

{

settings->setValue(LAUNCHED,true);

flush();

}

bool SettingTool::isFirstLaunched()

{

return settings->contains(LAUNCHED);

}

void SettingTool::setWindowWidth(int w)

{

settings->setValue(WINDOW\_W,w);

flush();

}

int SettingTool::getWindowWidth()

{

return settings->value(WINDOW\_W,WINDOW\_DEFAULT\_W).toInt();

}

void SettingTool::setWindowHeight(int h)

{

settings->setValue(WINDOW\_H,h);

flush();

}

int SettingTool::getWindowHeight()

{

return settings->value(WINDOW\_H,WINDOW\_DEFAULT\_H).toInt();

}

void SettingTool::setWindowSize(int w, int h)

{

if(!getRememberWindowSize()){

return;

}

settings->setValue(WINDOW\_W,w);

settings->setValue(WINDOW\_H,h);

flush();

}

void SettingTool::setWindowPosition(int x, int y)

{

if(!getRememberWindowPos()){

return;

}

settings->setValue(POSITION\_X,x);

settings->setValue(POSITION\_Y,y);

flush();

}

int SettingTool::getWindowPositionX()

{

return settings->value(POSITION\_X,100).toInt();

}

int SettingTool::getWindowPositionY()

{

return settings->value(POSITION\_Y,100).toInt();

}

void SettingTool::setLanguage(QString type)

{

settings->setValue(LANGUAGE,type);

flush();

}

int SettingTool::getLanguage()

{

return settings->value(LANGUAGE,0).toInt();

}

/\*1是tcpserver,2是tcpclient,3是udpserver,4是udpclient\*/

QList<QString> SettingTool::getPorts(int viewType)

{

int size=0;

switch(viewType){

case 1:

size = settings->beginReadArray(TCP\_SERVER\_PORTS);

break;

case 2:

size=settings->beginReadArray(TCP\_CLIENT\_PORTS);

break;

case 3:

size=settings->beginReadArray(UDP\_SERVER\_PORTS);

break;

case 4:

size=settings->beginReadArray(UDP\_CLIENT\_PORTS);

break;

}

QList<QString> result;

for (int i = 0; i < size; ++i) {

settings->setArrayIndex(i);

QString port=settings->value(PORT).toString();

result.append(port);

}

settings->endArray();

flush();

return result;

}

void SettingTool::savePorts(int viewType,QList<QString> ports)

{

switch (viewType) {

case 1:

settings->beginWriteArray(TCP\_SERVER\_PORTS);

break;

case 2:

settings->beginWriteArray(TCP\_CLIENT\_PORTS);

break;

case 3:

settings->beginWriteArray(UDP\_SERVER\_PORTS);

break;

case 4:

settings->beginWriteArray(UDP\_CLIENT\_PORTS);

break;

}

qDebug()<<"size of ports::"<<ports.size();

for (int i = 0; i < ports.size(); ++i) {

settings->setArrayIndex(i);

qDebug()<<ports.at(i);

settings->setValue(PORT,ports.at(i));

}

settings->endArray();

flush();

}

void SettingTool::flush()

{

settings->sync();

}

void SettingTool::setShowHeader(bool show)

{

settings->setValue(SHOW\_HEADER,show);

flush();

}

bool SettingTool::getShowHeader()

{

return settings->value(SHOW\_HEADER,false).toBool();

}

void SettingTool::setRememberWindowSize(bool remember)

{

settings->setValue(REMEMBER\_SIZE,remember);

}

bool SettingTool::getRememberWindowSize()

{

return settings->value(REMEMBER\_SIZE,false).toBool();

}

void SettingTool::setRememberWindowPos(bool remember)

{

settings->setValue(REMEMBER\_POSITION,remember);

}

bool SettingTool::getRememberWindowPos()

{

return settings->value(REMEMBER\_POSITION,false).toBool();

}

void SettingTool::setShowSendClear(bool clear)

{

qDebug()<<"setShowSendClear "<<clear;

settings->setValue(SHOW\_SEND\_CLEAR,clear);

}

bool SettingTool::getShowSendClear()

{

return settings->value(SHOW\_SEND\_CLEAR,false).toBool();

}

void SettingTool::setEnableLogHover(bool enable)

{

settings->setValue(LOG\_ENABLE\_HOVER,enable);

}

bool SettingTool::getEnableLogHover()

{

return settings->value(LOG\_ENABLE\_HOVER,false).toBool();

}

void SettingTool::setEnableSendSound(bool enable)

{

settings->setValue(PLAY\_SEND\_SOUND,enable);

}

bool SettingTool::getEnableSendSound()

{

return settings->value(PLAY\_SEND\_SOUND,false).toBool();

}

void SettingTool::setEnableReceiveSound(bool enable)

{

settings->setValue(PLAY\_RECEIVE\_SOUND,enable);

}

bool SettingTool::getEnableReceiveSound()

{

return settings->value(PLAY\_RECEIVE\_SOUND,false).toBool();

}

void SettingTool::setEnableSysSound(bool enable)

{

settings->setValue(PLAY\_SYSTEM\_SOUND,enable);

}

bool SettingTool::getEnableSysSound()

{

return settings->value(PLAY\_SYSTEM\_SOUND,false).toBool();

}

#include <QTabWidget>

#include <QShortcut>

#include <QFontDatabase>

#include "toolkit.h"

#include "setting.h"

#include "clientform.h"

#include "serverform.h"

#include "transferform.h"

#include "notepadform.h"

#include "sokit.h"

#define SET\_KEY\_FTNM "/font/name"

#define SET\_KEY\_FTSZ "/font/size"

#define SET\_KEY\_LANG "/lang"

#define SET\_VAL\_LANG "sokit"

#define SET\_VAL\_LANX ".lan"

Sokit::Sokit(int &argc, char \*\*argv)

: QApplication(argc, argv) {

}

Sokit::~Sokit() {

}

void Sokit::show() {

m\_wnd.show();

}

void Sokit::close() {

m\_wnd.close();

}

void Sokit::initDefaultActionsName() {

translate("QLineEdit", "&Undo");

translate("QLineEdit", "&Redo");

translate("QLineEdit", "Cu&t");

translate("QLineEdit", "&Copy");

translate("QLineEdit", "&Paste");

translate("QLineEdit", "Delete");

translate("QLineEdit", "Select All");

translate("QTextControl", "&Undo");

translate("QTextControl", "&Redo");

translate("QTextControl", "Cu&t");

translate("QTextControl", "&Copy");

translate("QTextControl", "&Paste");

translate("QTextControl", "Delete");

translate("QTextControl", "Select All");

}

bool Sokit::initTranslator() {

QString file = Setting::get(SET\_SEC\_CFG, SET\_KEY\_LANG, SET\_VAL\_LANG);

QStringList paths;

paths << "."

<< "../share/" SET\_APP\_NAME

<< "../share/apps/" SET\_APP\_NAME

<< Setting::path();

foreach (QString p, paths) {

if (m\_trans.load(file, p, "", SET\_VAL\_LANX)) {

installTranslator(&m\_trans);

Setting::set(SET\_SEC\_CFG, SET\_KEY\_LANG, file);

break;

}

}

return true;

}

void Sokit::initFont() {

QFontDatabase db;

QStringList fs = db.families();

QFont font;

int match = 0;

QString family = Setting::get(SET\_SEC\_CFG, SET\_KEY\_FTNM, "").trimmed();

QString size = Setting::get(SET\_SEC\_CFG, SET\_KEY\_FTSZ, "").trimmed();

if (family.isEmpty() || fs.filter(family).isEmpty()) {

QStringList defs = translate("Sokit", "font").split(";", QString::SkipEmptyParts);

foreach (QString d, defs) {

family = d.section(',', 0, 0).trimmed();

size = d.section(',', 1, 1).trimmed();

if (!family.isEmpty() && !fs.filter(family).isEmpty()) {

match = 2;

break;

}

}

} else {

match = 1;

}

if (match > 0) {

font.setFamily(family);

if (db.isSmoothlyScalable(family))

font.setStyleStrategy(

(QFont::StyleStrategy) (QFont::PreferAntialias | QFont::PreferOutline | QFont::PreferQuality));

int nsize = size.toInt();

if (nsize > 0 && nsize < 20)

font.setPointSize(nsize);

setFont(font);

if (match > 1) {

Setting::set(SET\_SEC\_CFG, SET\_KEY\_FTNM, family);

Setting::set(SET\_SEC\_CFG, SET\_KEY\_FTSZ, size);

}

}

}

bool Sokit::initUI() {

initTranslator();

initFont();

QShortcut \*k = new QShortcut(QKeySequence(Qt::Key\_F1), &m\_wnd);

QShortcut \*t = new QShortcut(QKeySequence(Qt::Key\_F10), &m\_wnd);

connect(t, SIGNAL(activated()), this, SLOT(ontop()));

m\_wnd.setWindowTitle(translate("Sokit", "sokit -- F1 for help"));

m\_wnd.setWindowIcon(QIcon(":/sokit.png"));

QWidget \*pnl = new QWidget(&m\_wnd);

m\_wnd.setCentralWidget(pnl);

BaseForm \*server = new ServerForm();

BaseForm \*transf = new TransferForm();

BaseForm \*client = new ClientForm();

NotepadForm \*npd = new NotepadForm();

QTabWidget \*tab = new QTabWidget(pnl);

tab->addTab(server, server->windowTitle());

tab->addTab(transf, transf->windowTitle());

tab->addTab(client, client->windowTitle());

tab->addTab(npd, npd->windowTitle());

tab->setCurrentIndex(0);

QLayout \*lay = new QVBoxLayout(pnl);

lay->setSpacing(5);

lay->setContentsMargins(5, 5, 5, 5);

lay->addWidget(tab);

return server->init() && transf->init() &&

client->init() && npd->init();

}

void Sokit::ontop() {

Qt::WindowFlags f = m\_wnd.windowFlags();

if (f & Qt::WindowStaysOnTopHint)

f &= ~Qt::WindowStaysOnTopHint;

else

f |= Qt::WindowStaysOnTopHint;

m\_wnd.setWindowFlags(f);

m\_wnd.show();

}

#include "SoundManager.h"

SoundManager::SoundManager(QObject \*parent) : QObject(parent)

{

}

SoundManager::~SoundManager()

{

}

void SoundManager::playSound(int type)

{

Q\_UNUSED(type)

}

void SoundManager::playSlideOpen()

{

if(SettingTool::getInstance()->getEnableSysSound())

{

QSound::play(WAV\_SLIDE\_OPEN);

}

}

void SoundManager::playSlideClose()

{

if(SettingTool::getInstance()->getEnableSysSound())

{

QSound::play(WAV\_SLIDE\_CLOSE);

}

}

void SoundManager::playReceive()

{

if(SettingTool::getInstance()->getEnableReceiveSound())

{

QSound::play(WAV\_RECEIVE);

}

}

void SoundManager::playSend()

{

if(SettingTool::getInstance()->getEnableSendSound())

{

QSound::play(WAV\_SEND);

}

}

#include <QTcpSocket>

#include "SoundManager.h"

#include "TcpServerModel.h"

#include "toolkit.h"

TcpServerModel::TcpServerModel() {

}

TcpServerModel::~TcpServerModel() {

closeServer();

}

bool TcpServerModel::openServer(QString &addr, quint16 port) {

QHostAddress hostAddress(addr);

if (m\_tcp\_server.listen(hostAddress, port)) {

connect(&m\_tcp\_server, SIGNAL(newConnection()), this, SLOT(newConnection()));

qDebug() << "开启成功";

emit sendErrMsg(START\_ERR, QString("start success"), false);

return true;

} else {

qDebug() << (QString("%1, %2").arg(m\_tcp\_server.serverError()).arg(m\_tcp\_server.errorString()));

emit sendErrMsg(START\_ERR, QString("%1").arg(m\_tcp\_server.errorString()), true);

return false;

}

}

bool TcpServerModel::close() {

qDebug()<<"关闭 tcp 服务器";

m\_tcp\_server.close();

m\_tcp\_server.disconnect(this);

if (m\_tcp\_server.errorString() != nullptr)

qDebug() << "关闭失败" << m\_tcp\_server.errorString();

return true;

}

bool TcpServerModel::close(void \*cookie) {

Conn \*conn = (Conn \*) cookie;

if (conn->client)

conn->client->disconnect(this);

delete conn->client;

delete conn;

return true;

}

void TcpServerModel::sendToDst(void \*cookie, const QByteArray &bin) {

Conn \*conn = (Conn \*) cookie;

qint64 srcLen = bin.length();

char \* src=TK::createBuffer(srcLen,MAXBUFFER);

qDebug()<<bin<<"+++++"<<src;

memcpy(src,bin.data(),srcLen);

qint64 writeLen = 0;

qint64 ioLen = conn->client->write(src, srcLen);

while (ioLen > 0) {

writeLen += ioLen;

ioLen = conn->client->write(src + writeLen, srcLen - writeLen);

}

if (writeLen != srcLen) {

// show(QString("failed to send data to %1:%2 [%3]")

// .arg(addr().toString()).arg(port()).arg(writeLen));

emit sendErrMsg(SEND\_ERR, "failed to send data to", false);

return;

}

// recordSend(writeLen);

// dump(src, srcLen, true, conn->key);

SoundManager::playSend();

dumpLogMsg(false, conn->key, src, writeLen);

}

void TcpServerModel::error() {

QTcpSocket \*s = qobject\_cast<QTcpSocket \*>(sender());

emit sendErrMsg(RECV\_ERR, QString("TCP socket error %1, %2").arg(s->error()).arg(s->errorString()), true);

s->deleteLater();

}

void TcpServerModel::newConnection() {

QTcpServer \*server = qobject\_cast<QTcpServer \*>(sender());

if (!server) return;

QTcpSocket \*client = server->nextPendingConnection();

while (client) {

Conn \*conn = new Conn;

if (!conn) {

client->deleteLater();

} else {

client->setProperty(PROP\_CONN, qVariantFromValue((void \*) conn));

conn->client = client;

conn->key = TK::ipstr(client->peerAddress(), client->peerPort());

connect(client, SIGNAL(readyRead()), this, SLOT(newData()));

connect(client, SIGNAL(destroyed(QObject \* )), this, SLOT(close(QObject \* )));

connect(client, SIGNAL(disconnected()), client, SLOT(deleteLater()));

connect(client, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

setCookie(conn->key, conn);

}

client = server->nextPendingConnection();

}

}

void TcpServerModel::close(QObject \*obj) {

Conn \*conn = (Conn \*) obj->property(PROP\_CONN).value<void \*>();

if (!conn) return;

unsetCookie(conn->key);

delete conn;

}

void TcpServerModel::newData() {

QTcpSocket \*client = qobject\_cast<QTcpSocket \*>(sender());

if (!client) return;

Conn \*conn = (Conn \*) client->property(PROP\_CONN).value<void \*>();

if (!conn) return;

qint64 bufLen = client->bytesAvailable();

char \*buf = TK::createBuffer(bufLen, MAXBUFFER);

if (!buf) return;

qint64 readLen = 0;

qint64 ioLen = client->read(buf, bufLen);

while (ioLen > 0) {

readLen += ioLen;

ioLen = client->read(buf + readLen, bufLen - readLen);

}

if (ioLen >= 0) {

// recordRecv(readLen);

// dump(buf, readLen, false, conn->key);

SoundManager::playReceive();

dumpLogMsg(true, conn->key, buf, readLen);

}

// TK::releaseBuffer(buf);

}

#include <QNetworkInterface>

#include <QAbstractSocket>

#include <QCoreApplication>

#include <QAction>

#include "toolkit.h"

const char\* TK::socketTypeName(bool tcp)

{

return tcp ? "TCP" : "UDP";

}

const char\* TK::socketTypeName(QAbstractSocket\* skt)

{

return (skt->socketType()==QAbstractSocket::TcpSocket) ? "TCP" : "UDP";

}

QString TK::ipstr(const QHostAddress& addr, quint16 port)

{

return addr.toString() + ':' + QString::number(port);

}

QString TK::ipstr(const QHostAddress& addr, quint16 port, bool tcp)

{

return QString("%1 %2").arg((tcp?"[TCP]":"[UDP]"),ipstr(addr,port));

}

const char\* TK::hextab = "0123456789ABCDEF";

#define TOHEXSTR(v, c, tab) \

if (v) { \

\*c++ = tab[(v>>4)&0xF]; \

\*c++ = tab[v & 0xF]; \

}

QString TK::ascii2hex(const QString& src, QVector<uint>& posmap, uint& count)

{

posmap = src.toUcs4();

count = 0;

uint val;

char ch[16];

QString res;

PARSE\_STA status = OUT;

int bound = 0;

int hexpos = 0;

int len = posmap.count();

for (int i=0; i<len; ++i)

{

char\* c = ch;

unsigned char t;

uint& ov = posmap[i];

val = ov;

ov = hexpos;

if ('[' == val)

{

switch (status)

{

case OUT: status = OUT2IN; break;

case OUT2IN: status = OUT; break;

case IN: status = ERR; break;

case ERR: break;

}

if (OUT2IN == status)

continue;

}

else

if (']' == val)

{

if (OUT != status)

{

if (bound & 1) {

status = ERR;

} else {

status = OUT;

continue;

}

}

}

else

if (OUT != status)

{

if ((val >= '0' && val <='9')||

(val >= 'A' && val <='F'))

{

status = IN;

}

else

if (val >= 'a' && val <='f')

{

val = 'A' + (val - 'a');

status = IN;

}

else

if (val == ' ')

{

status = IN;

continue;

}

else

{

status = ERR;

}

}

if (OUT == status)

{

if (bound & 1)

{

status = ERR;

}

else

{

t = val >> 24 & 0xFF;

TOHEXSTR(t,c,hextab)

t = val >> 16 & 0xFF;

TOHEXSTR(t,c,hextab)

t = val >> 8 & 0xFF;

TOHEXSTR(t,c,hextab)

t = val & 0xFF;

TOHEXSTR(t,c,hextab)

}

bound = 0;

}

else

if (IN == status)

{

bound += 1;

\*c++ = (char)val;

}

if (ERR == status)

{

res += QString("ERROR[%1]").arg(i);

hexpos = 0;

while (i < len)

posmap[i++] = 0;

break;

}

count += (c-ch);

if (0 == (bound&1))

\*c++ = ' ';

\*c = 0;

res += ch;

hexpos = res.length();

}

posmap.append(hexpos);

count /= 2;

return res;

}

#undef TOHEXSTR

QString TK::bin2hex(const char\* buf, uint len)

{

char\* tmp = new char[len \* 3 + 3];

const char\* s = buf;

const char\* e = buf + len;

char\* d = tmp;

if (s && d)

{

\*d++ = '[';

while (s < e)

{

char c = \*s++;

\*d++ = hextab[(c>>4)&0xF];

\*d++ = hextab[c & 0xF];

\*d++ = ' ';

}

\*d++ = ']';

\*d = 0;

}

QString res(tmp);

delete[] tmp;

return res;

}

QString TK::bin2ascii(const char\* buf, uint len)

{

char\* tmp = new char[len + 1];

const char\* s = buf;

const char\* e = buf + len;

char\* d = tmp;

if (s && d)

{

while (s < e)

{

char c = \*s++;

\*d++ = isprint((uchar)c) ? c : '.';

}

\*d = 0;

}

QString res(tmp);

delete[] tmp;

return res;

}

bool TK::ascii2bin(const QString& src, QByteArray& dst, QString& err)

{

dst.clear();

err.clear();

QDataStream os(&dst, QIODevice::WriteOnly);

QVector<uint> lst = src.toUcs4();

PARSE\_STA status = OUT;

int bound = 0;

quint8 hexval = 0;

int len = lst.count();

for (int i=0; i<len; ++i)

{

uint val = lst.at(i);

if ('[' == val)

{

switch (status)

{

case OUT: status = OUT2IN; break;

case OUT2IN: status = OUT; break;

case IN: status = ERR; break;

case ERR: break;

}

if (OUT2IN == status)

continue;

}

else

if (']' == val)

{

if (OUT !=status)

{

if (bound & 1) {

status = ERR;

} else {

status = OUT;

continue;

}

}

}

else

if (OUT != status)

{

if (val >= '0' && val <='9')

{

val -= '0';

status = IN;

}

else

if (val >= 'A' && val <='F')

{

val -= 'A';

val += 10;

status = IN;

}

else

if (val >= 'a' && val <='f')

{

val -= 'a';

val += 10;

status = IN;

}

else

if (val == ' ')

{

status = IN;

continue;

}

else

{

status = ERR;

}

}

if (OUT == status)

{

if (bound & 1)

{

status = ERR;

}

else

{

if (val >> 16)

os << (quint32)val;

else

if (val >> 8)

os << (quint16)val;

else

os << (quint8)val;

}

bound = 0;

hexval = 0;

}

else

if (IN == status)

{

if (bound & 1)

os << (quint8)(((quint8)(val&0xF)) | (hexval<<4));

else

hexval = (quint8)(val & 0xF);

++bound;

}

if (ERR == status)

{

dst.clear();

err.append(QString("ERROR[%1]").arg(i));

break;

}

}

return (ERR != status);

}

char\* TK::createBuffer(qint64& cap, qint64 limit)

{

if (cap < 0 || cap > limit)

return nullptr;

if (0 == cap)

cap = 1;

return new char[cap];

}

void TK::releaseBuffer(char\*& buf)

{

delete buf;

buf = nullptr;

}

#include "toolkit.h"

#include "setting.h"

#include "transferskt.h"

#include "transferform.h"

#include <QShortcut>

#define SET\_SEC\_TRANS "transfer"

#define SET\_KEY\_TRANS "/transfer"

#define SET\_KEY\_CMBSA "/srcip"

#define SET\_KEY\_CMBDA "/dstip"

#define SET\_KEY\_CMBSP "/srcport"

#define SET\_KEY\_CMBDP "/dstport"

#define SET\_VAL\_LGTAN "log\_transf"

#define PROP\_EDIT "edit"

#define PROP\_DIRT "dirt"

TransferForm::TransferForm(QWidget \*parent, Qt::WindowFlags flags)

: BaseForm(parent, flags),m\_server(0)

{

m\_ui.setupUi(this);

}

TransferForm::~TransferForm()

{

if (m\_server && lock(1000))

{

m\_server->disconnect(this);

delete m\_server;

m\_server = NULL;

unlock();

}

saveConfig();

}

void TransferForm::initConfig()

{

QString sst(SET\_SEC\_TRANS);

Setting::lord(sst+SET\_KEY\_CMBSA, SET\_PFX\_CMBITM, \*m\_ui.cmbSrcAddr, false);

Setting::lord(sst+SET\_KEY\_CMBDA, SET\_PFX\_CMBITM, \*m\_ui.cmbDstAddr);

Setting::lord(sst+SET\_KEY\_CMBSP, SET\_PFX\_CMBITM, \*m\_ui.cmbSrcPort);

Setting::lord(sst+SET\_KEY\_CMBDP, SET\_PFX\_CMBITM, \*m\_ui.cmbDstPort);

QString skl(SET\_SEC\_DIR); skl += SET\_KEY\_LOG;

skl = Setting::get(skl, SET\_KEY\_TRANS, SET\_VAL\_LGTAN);

setProperty(SET\_SEC\_DIR, skl);

TK::initNetworkInterfaces(m\_ui.cmbSrcAddr, true);

TK::initNetworkInterfaces(m\_ui.cmbDstAddr);

}

void TransferForm::saveConfig()

{

QString sst(SET\_SEC\_TRANS);

Setting::save(sst+SET\_KEY\_CMBSA, SET\_PFX\_CMBITM, \*m\_ui.cmbSrcAddr, false);

Setting::save(sst+SET\_KEY\_CMBDA, SET\_PFX\_CMBITM, \*m\_ui.cmbDstAddr);

Setting::save(sst+SET\_KEY\_CMBSP, SET\_PFX\_CMBITM, \*m\_ui.cmbSrcPort);

Setting::save(sst+SET\_KEY\_CMBDP, SET\_PFX\_CMBITM, \*m\_ui.cmbDstPort);

QString skl(SET\_SEC\_DIR); skl += SET\_KEY\_LOG;

Setting::set(skl, SET\_KEY\_TRANS, property(SET\_SEC\_DIR).toString());

}

bool TransferForm::initForm()

{

initCounter(m\_ui.labRecv, m\_ui.labSend);

initLogger(m\_ui.chkLog, m\_ui.btnClear, m\_ui.treeOutput, m\_ui.txtOutput);

initLister(m\_ui.btnConnAll, m\_ui.btnConnDel, m\_ui.lstConn);

bindBuffer(1, m\_ui.edtBuf1, m\_ui.btnSend1, m\_ui.cmbDir1);

bindBuffer(2, m\_ui.edtBuf2, m\_ui.btnSend2, m\_ui.cmbDir2);

bindBuffer(3, m\_ui.edtBuf3, m\_ui.btnSend3, m\_ui.cmbDir3);

bindSelect(m\_ui.cmbDir1, -1, Qt::Key\_F5);

bindSelect(m\_ui.cmbDir2, -1, Qt::Key\_F6);

bindSelect(m\_ui.cmbDir3, -1, Qt::Key\_F7);

connect(m\_ui.btnTrigger, SIGNAL(clicked(bool)), this, SLOT(trigger(bool)));

return true;

}

bool TransferForm::initHotkeys()

{

bindFocus(m\_ui.cmbSrcAddr, Qt::Key\_Escape);

bindClick(m\_ui.btnTrigger, Qt::CTRL + Qt::Key\_S);

bindSelect(m\_ui.cmbType, 0, Qt::CTRL + Qt::Key\_T);

bindSelect(m\_ui.cmbType, 1, Qt::CTRL + Qt::Key\_U);

return true;

}

void TransferForm::kill(QStringList& list)

{

if (m\_server)

{

while (!list.isEmpty())

m\_server->kill(list.takeFirst());

}

}

void TransferForm::trigger(bool start)

{

if (lock(1000))

{

if (m\_server)

{

m\_server->stop();

m\_server->disconnect(this);

delete m\_server;

m\_server = NULL;

}

IPAddr sa, da;

if (start)

{

start = TK::popIPAddr(m\_ui.cmbSrcAddr, m\_ui.cmbSrcPort, sa) &&

TK::popIPAddr(m\_ui.cmbDstAddr, m\_ui.cmbDstPort, da);

}

if (start)

{

QString type = m\_ui.cmbType->currentText();

if (type.contains(TK::socketTypeName(true)))

m\_server = new TransferSktTcp(this);

else

m\_server = new TransferSktUdp(this);

if (m\_server)

{

connect(m\_server, SIGNAL(connOpen(const QString&)), this, SLOT(listerAdd(const QString&)));

connect(m\_server, SIGNAL(connClose(const QString&)), this, SLOT(listerRemove(const QString&)));

connect(m\_server, SIGNAL(message(const QString&)), this, SIGNAL(output(const QString&)));

connect(m\_server, SIGNAL(dumpbin(const QString&,const char\*,quint32)), this, SIGNAL(output(const QString&,const char\*,quint32)));

connect(m\_server, SIGNAL(countRecv(qint32)), this, SLOT(countRecv(qint32)));

connect(m\_server, SIGNAL(countSend(qint32)), this, SLOT(countSend(qint32)));

connect(m\_server, SIGNAL(stopped()), this, SLOT(stop()));

start = m\_server->start(sa.ip, sa.port, da.ip, da.port);

if (!start)

{

m\_server->disconnect(this);

delete m\_server;

m\_server = NULL;

}

}

else

{

start = false;

}

}

unlock();

}

m\_ui.cmbSrcAddr->setDisabled(start);

m\_ui.cmbSrcPort->setDisabled(start);

m\_ui.cmbDstAddr->setDisabled(start);

m\_ui.cmbDstPort->setDisabled(start);

m\_ui.cmbType->setDisabled(start);

if (start)

{

TK::pushIPAddr(m\_ui.cmbSrcPort, m\_ui.cmbSrcAddr);

TK::pushIPAddr(0, m\_ui.cmbDstPort);

}

else

{

TK::resetPushBox(m\_ui.btnTrigger);

}

}

void TransferForm::stop()

{

trigger(false);

}

void TransferForm::send(const QString& data, const QString& dir)

{

bool s2d = dir.startsWith('S');

if (m\_server &&lock(1000))

{

QStringList list;

listerSelected(list);

while (!list.isEmpty())

m\_server->send(list.takeFirst(), s2d, data);

unlock();

}

}

#include <QTcpSocket>

#include <QStringList>

#include <QMutexLocker>

#include "toolkit.h"

#include "transferskt.h"

#define PROP\_CONN "CONN"

#define MAXBUFFER 1024\*1024

TransferSkt::TransferSkt(QObject \*parent)

: QObject(parent),m\_spt(0),m\_dpt(0)

{

}

TransferSkt::~TransferSkt()

{

}

bool TransferSkt::start(const QHostAddress& sip, quint16 spt, const QHostAddress& dip, quint16 dpt)

{

m\_sip = sip;

m\_dip = dip;

m\_spt = spt;

m\_dpt = dpt;

m\_conns.clear();

m\_error.clear();

bool res = open();

QString msg("start %1 transfer server %2!");

if (!res)

{

msg = msg.arg(name(),"failed");

if (!m\_error.isEmpty())

{ msg += " error:[";

msg += m\_error;

msg += "].";

}

}

else

{

msg = msg.arg(name(),"successfully");

}

show(msg);

return res;

}

void TransferSkt::stop()

{

OBJMAP::const\_iterator i;

for (i = m\_conns.constBegin(); i != m\_conns.constEnd(); ++i)

{

QString k = i.key();

void\* v = i.value();

if (close(v))

emit connClose(k);

}

m\_conns.clear();

close();

show(QString("stop %1 transfer server!").arg(name()));

}

void TransferSkt::setError(const QString& err)

{

m\_error = err;

}

void TransferSkt::recordRecv(qint32 bytes)

{

emit countRecv(bytes);

}

void TransferSkt::recordSend(qint32 bytes)

{

emit countSend(bytes);

}

void TransferSkt::getKeys(QStringList& res)

{

res = m\_conns.keys();

}

void TransferSkt::setCookie(const QString& k, void\* v)

{

void\* o = m\_conns.value(k);

if (o)

{

if (close(o))

emit connClose(k);

}

m\_conns.insert(k, v);

emit connOpen(k);

}

void TransferSkt::unsetCookie(const QString& k)

{

m\_conns.remove(k);

emit connClose(k);

}

void\* TransferSkt::getCookie(const QString& k)

{

return m\_conns.value(k);

}

void TransferSkt::kill(const QString& key)

{

void\* v = m\_conns.value(key);

if (v)

{

if (close(v))

unsetCookie(key);

}

else

{

unsetCookie(key);

}

}

void TransferSkt::send(const QString& key, bool s2d, const QString& data)

{

void\* v = m\_conns.value(key);

if (v)

{

QString err;

QByteArray bin;

if (!TK::ascii2bin(data, bin, err))

show("bad data format to send: "+err);

else

send(v, s2d, bin);

}

}

void TransferSkt::dump(const char\* buf, qint32 len, TransferSkt::DIR dir, const QString& key)

{

QString title("TRN");

switch (dir)

{

case TS2D: title += " -->> "; break;

case TD2S: title += " <<-- "; break;

case SS2D: title += " --+> "; break;

case SD2S: title += " <+-- "; break;

default: title += " ???? "; break;

}

title += key;

emit dumpbin(title, buf, (quint32)len);

}

void TransferSkt::show(const QString& msg)

{

emit message(msg);

}

TransferSktTcp::TransferSktTcp(QObject \*parent)

:TransferSkt(parent)

{

}

TransferSktTcp::~TransferSktTcp()

{

m\_server.disconnect(this);

}

bool TransferSktTcp::open()

{

if (m\_server.listen(srcAddr(), srcPort()))

{

connect(&m\_server, SIGNAL(newConnection()), this, SLOT(newConnection()));

return true;

}

else

{

setError(QString("%1, %2").arg(m\_server.serverError()).arg(m\_server.errorString()));

}

return false;

}

bool TransferSktTcp::close(void\* cookie)

{

Conn\* conn = (Conn\*)cookie;

if (conn->src)

conn->src->disconnect(this);

if (conn->dst)

conn->dst->disconnect(this);

delete conn->src;

delete conn->dst;

delete conn;

return true;

}

void TransferSktTcp::close(QObject\* obj)

{

QMutexLocker locker(&m\_door);

Conn\* conn = (Conn\*)obj->property(PROP\_CONN).value<void\*>();

if (!conn) return;

if (conn->src)

{

conn->src->disconnect(this);

if (obj == conn->dst)

conn->src->deleteLater();

}

if (conn->dst)

{

conn->dst->disconnect(this);

if (obj == conn->src)

conn->dst->deleteLater();

}

unsetCookie(conn->key);

delete conn;

}

void TransferSktTcp::error()

{

QTcpSocket\* s = qobject\_cast<QTcpSocket\*>(sender());

show(QString("TCP socket error %1, %2").arg(s->error()).arg(s->errorString()));

s->deleteLater();

}

void TransferSktTcp::close()

{

m\_server.close();

m\_server.disconnect(this);

}

void TransferSktTcp::newConnection()

{

QTcpServer\* svr = qobject\_cast<QTcpServer\*>(sender());

if (!svr) return;

QTcpSocket\* src = svr->nextPendingConnection();

while (src)

{

Conn\* conn = new Conn;

if (!conn)

{

src->deleteLater();

}

else

{

QTcpSocket\* dst = new QTcpSocket();

if (!dst)

{

delete conn;

src->deleteLater();

}

else

{

src->setProperty(PROP\_CONN, qVariantFromValue((void\*)conn));

dst->setProperty(PROP\_CONN, qVariantFromValue((void\*)conn));

conn->src = src;

conn->dst = dst;

conn->key = TK::ipstr(src->peerAddress(),src->peerPort());

connect(src, SIGNAL(readyRead()), this, SLOT(newData()));

connect(src, SIGNAL(destroyed(QObject\*)), this, SLOT(close(QObject\*)));

connect(src, SIGNAL(disconnected()), src, SLOT(deleteLater()));

connect(src, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

connect(dst, SIGNAL(readyRead()), this, SLOT(newData()));

connect(dst, SIGNAL(destroyed(QObject\*)), this, SLOT(close(QObject\*)));

connect(dst, SIGNAL(disconnected()), dst, SLOT(deleteLater()));

connect(dst, SIGNAL(connected()), this, SLOT(asynConnection()));

connect(dst, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

dst->connectToHost(dstAddr(), dstPort());

setCookie(conn->key, conn);

}

}

src = svr->nextPendingConnection();

}

}

void TransferSktTcp::asynConnection()

{

QTcpSocket\* s = qobject\_cast<QTcpSocket\*>(sender());

if (!s) return;

Conn\* conn = (Conn\*)s->property(PROP\_CONN).value<void\*>();

if (!conn) return;

show(QString("connection %1 to %2:%3 opened!")

.arg(conn->key, s->peerName()).arg(s->peerPort()));

}

void TransferSktTcp::newData()

{

QMutexLocker locker(&m\_door);

QTcpSocket\* s = qobject\_cast<QTcpSocket\*>(sender());

if (!s) return;

Conn\* conn = (Conn\*)s->property(PROP\_CONN).value<void\*>();

if (!conn) return;

QTcpSocket\* d = (s == conn->src) ? conn->dst : conn->src;

qint64 bufLen = s->bytesAvailable();

char\* buf = TK::createBuffer(bufLen, MAXBUFFER);

if (!buf) return;

qint64 readLen, writeLen, ioLen;

readLen = 0;

ioLen = s->read(buf, bufLen);

while (ioLen > 0)

{

readLen += ioLen;

ioLen = s->read(buf+readLen, bufLen-readLen);

}

if (ioLen >= 0)

{

recordRecv(readLen);

writeLen = 0;

ioLen = d->write(buf, readLen);

while (ioLen > 0)

{

writeLen += ioLen;

ioLen = d->write(buf+writeLen, readLen-writeLen);

}

if (ioLen >= 0)

{

recordSend(writeLen);

dump(buf, readLen, ((s==conn->src) ? TS2D:TD2S), conn->key);

}

}

TK::releaseBuffer(buf);

}

void TransferSktTcp::send(void\* cookie, bool s2d, const QByteArray& bin)

{

Conn\* conn = (Conn\*)cookie;

QTcpSocket\* d = s2d ? conn->dst : conn->src;

QHostAddress a = s2d ? dstAddr() : conn->src->peerAddress();

quint16 p = s2d ? dstPort() : conn->src->peerPort();

const char \* src = bin.constData();

qint64 srcLen = bin.length();

qint64 writeLen = 0;

qint64 ioLen = d->write(src, srcLen);

while (ioLen > 0)

{

writeLen += ioLen;

ioLen = d->write(src+writeLen, srcLen-writeLen);

}

if (writeLen != srcLen)

{

show(QString("failed to send data to %1:%2 [%3]")

.arg(a.toString()).arg(p).arg(writeLen));

return;

}

recordSend(writeLen);

dump(src, srcLen, (s2d?SS2D:SD2S), conn->key);

}

TransferSktUdp::TransferSktUdp(QObject \*parent)

:TransferSkt(parent)

{

}

TransferSktUdp::~TransferSktUdp()

{

m\_server.disconnect(this);

}

void TransferSktUdp::error()

{

QUdpSocket\* s = qobject\_cast<QUdpSocket\*>(sender());

show(QString("UDP socket error %1, %2").arg(s->error()).arg(s->errorString()));

if (s != &m\_server)

s->deleteLater();

}

bool TransferSktUdp::open()

{

if (m\_server.bind(srcAddr(), srcPort(), QUdpSocket::ShareAddress))

{

connect(&m\_server, SIGNAL(readyRead()), this, SLOT(newData()));

connect(&m\_server, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));

connect(&m\_timer, SIGNAL(timeout()), this, SLOT(check()));

m\_timer.start(2000);

return true;

}

else

{

setError(QString("%1, %2").arg(m\_server.error()).arg(m\_server.errorString()));

}

return false;

}

bool TransferSktUdp::close(void\* cookie)

{

Conn\* conn = (Conn\*)cookie;

if (conn->dst)

conn->dst->disconnect(this);

delete conn->dst;

delete conn;

return true;

}

void TransferSktUdp::close(QObject\* obj)

{

Conn\* conn = (Conn\*)obj->property(PROP\_CONN).value<void\*>();

if (!conn) return;

unsetCookie(conn->key);

delete conn;

}

void TransferSktUdp::close()

{

m\_timer.disconnect(this);

m\_timer.stop();

m\_server.close();

m\_server.disconnect(this);

}