90 TCP通信概述

90.1 TCP协议

TCP（Transmission Control Protocol，传输控制协议）是一种被大多数互联网协议，如HTTP、FTP等，广泛采用的，有连接、可靠、面向数据流的传输层协议。它主要用于对传输可靠性要求较高的连续数据传输。与UDP协议不同，基于TCP协议的通信双方，需要建立持久的套接字连接，通过该连接发送数据无需要指定目的地址和端口。

90.2 QTcpServer和QTcpSocket

Qt提供QTcpServer和QTcpSocket两个类，用于建立TCP连接和基于该连接通信。服务器借助QTcpServer对象监听端口，等待并接受客户机的连接请求。连接建立后，客户机和服务器通过QTcpSocket对象收发数据。

90.2.1 QTcpServer

90.2.1.1 公有函数

```
bool         QTcpServer::listen();                // 启动监听
bool         QTcpServer::isListening();           // 是否正在监听
QHostAddress QTcpServer::serverAddress();         // 获取正在监听的IP地址
quint16      QTcpServer::serverPort();            // 获取正在监听的端口
bool         QTcpServer::waitForNewConnection();  // 以阻塞方式等待连接
QTcpSocket*  QTcpServer::nextPendingConnection(); // 获取下一个待决连接
void         QTcpServer::close();                 // 终止监听
```

90.2.1.2 信号函数

```
void QTcpServer::newConnection();                                    // 新连接被接受时发射
void QTcpServer::acceptError(QAbstractSocket::SocketError socketError); // 接受连接出错时发射
```

90.2.1.3 保护函数

```
virtual void QTcpServer::incomingConnection(qintptr socketDescriptor); // 新连接进入时调用
// 该函数在QTcpServer类中的实现，是当新连接进入时，首先创建一个QTcpSocket对象，并将其排入待决连接队列，然后发射newConnection信号
// QTcpServer类的子类可以覆盖此函数，以自己的方式处理新连接，但必须通过addPendingConnection函数，将表示该连接的QTcpSocket对象，排入待决连接队列
void QTcpServer::addPendingConnection(QTcpSocket* socket); // 将连接对象socket排入待决连接队列
```

90.2.2 QTcpSocket

90.2.2.1 公有函数

```
void         QTcpSocket::connectToHost(QHostAddress& address, quint16 port);
// 连接服务器，连接成功发射connected信号
void         QTcpSocket::disconnectFromHost();
// 断开与服务器的连接，断开成功发射disconnected信号
```

```
bool          QTcpSocket::waitForConnected();
// 等待直到连接成功
bool          QTcpSocket::waitForDisconnected();
 // 等待直到连接断开
QHostAddress QTcpSocket::localAddress();
// 获取本机IP地址
quint16       QTcpSocket::localPort();
 // 获取本机端口
QHostAddress QTcpSocket::peerAddress();
 // 获取对方IP地址
QString       QTcpSocket::peerName();
// 获取对方主机名
quint16       QTcpSocket::peerPort();
// 获取对方端口
qint64        QTcpSocket::readBufferSize();
// 获取读缓冲区的大小
void          QTcpSocket::setReadBufferSize(qint64 size);
// 设置读缓冲区的大小
qint64        QTcpSocket::bytesAvailable();
// 获取读缓冲区中可被读取的字节数
bool          QTcpSocket::canReadLine();
 // 可否读取一行数据
SocketState   QTcpSocket::state();
 // 获取套接字状态
```

90.2.2.2 信号函数

```
void QTcpSocket::hostFound();                                                    //
connectToHost函数找到主机时发射
void QTcpSocket::connected();                                                    //
connectToHost函数连接服务器成功时发射
void QTcpSocket::disconnected();                                                 //
disconnectFromHost函数断开与服务器的连接时发射
void QTcpSocket::readyRead();                                                    // 读
缓冲区中有新的可读数据时发射
void QTcpSocket::stateChanged(QAbstractSocket::SocketState socketState); // 套
接字状态改变时发射
void QTcpSocket::error(QAbstractSocket::SocketError socketError);        // 套
接字发生错误时发射
```

90.2.3 服务器与客户机

服务器首先调用QTcpServer对象的listen方法，启动对指定IP地址和端口的监听。一旦客户机发起连接请求，该对象内部的incomingConnection方法即会创建一个用于与客户机通信的QTcpSocket对象，并发射newConnection信号。在与该信号相连接的槽函数中，调用QTcpServer对象的nextPendingConnection方法，获取QTcpSocket对象，与客户机通信。QTcpSocket类提供了基于TCP协议的通信接口。在此基础上，可以实现诸如POP3、SMTP、NNTP等标准网络协议，甚至可以实现自己定义的网络协议。

客户机首先通过QTcpSocket对象的connectToHost方法连接服务器，这里需要提供服务器的IP地址和端口。connectToHost方法不会阻塞程序运行，连接成功后会发射connected信号。如果需要以阻塞方式连接服务器，可以使用waitForConnected方法，该函数会阻塞程序运行，直到连接成功或失败时才返回。如果连接服务器成功，客户机、服务器双方就可以向写缓冲区写入数据或者从读缓冲区读取数据，实现数据通信。当读缓冲区有新数据进入时，会发射readyRead信号，在与该信号相连接的槽函数中读

取缓冲区中的数据。一个QTcpSocket对象可以同时接收和发送数据，两个过程是异步执行的，且各自拥有独立的缓冲区。