Programmation réseau avec Qt

Table des matières

Qt est une API orientée objet qui offre des composants d'interface graphique, d'accès aux données, de connexions réseaux,, d'analyse XML, etc. Qt est connu pour être la bibliothèque sur laquelle repose l'un des environnements de bureau les plus utilisés dans le monde GNU/Linux.

1. Introduction

Le module QtNetwork offre des classes qui vous permettent d'écrire vos propres clients et serveurs TCP/IP. Il offre des classes comme QFtp qui implémente le protocole spécifique au niveau de l'application, mais aussi des classes de plus bas niveau comme QTcpSocket, QTcpServer et QUdpSocket qui représentent les concepts réseau de bas niveau, et des classes plus haut niveau comme QNetworkRequest, QNetworkReply et QNetworkAccessManager qui permettent d'effectuer des opérations réseau avec des protocoles communs. Ce module offre aussi des classes comme QNetworkConfiguration, QNetworkConfigurationManager et QNetworkSession, qui implémentent la gestion de la connexion.

2. Classes Qt pour la programmation réseau

Les classes suivantes fournissent un support pour la programmation réseau avec Qt :

QAbstractSocket	La fonctionnalité de base commune à tous les types de sockets
QAuthenticator	Objet d'authentification
QFtp	Implémentation du côté client du protocole FTP
QHostAddress	Adresse IP
QHostInfo	Fonctions statiques pour les recherches de nom d'hôte
QNetworkAccessManager	Permet à l'application d'envoyer des requêtes réseau et de recevoir des réponses
QNetworkAddressEntry	Enregistre une adresse IP prise en charge par une interface réseau, avec son masque de réseau (netmask) associé et l'adresse de diffusion (broadcast)
QNetworkConfiguration	Abstraction d'une ou plusieurs configurations de points d'accès
QNetworkConfigurationManager	Gère la configuration réseau fournie par le système
QNetworkInterface	Liste des adresses IP et des interfaces réseau de l'hôte
QNetworkProxy	Proxy de couche réseau
QNetworkProxyFactory	Sélection très précise du proxy
QNetworkReply	Contient les données et en-têtes d'une requête envoyée avec QNetworkAccessManager
QNetworkRequest	Contrôle une requête à envoyer avec QNetworkAccessManager
QNetworkSession	Contrôle les points d'accès du système et permet la gestion de sessions pour les cas où plusieurs clients accèdent au même point d'accès
QSocketNotifier	Support au suivi de l'activité sur un descripteur de fichier
QSsl	Déclare des énumérations communes à toutes les classes SSL dans QtNetwork
QSslCertificate	API pratique pour un certificat X509
QSslCipher	Représente un procédé de chiffrement SSL
QSslConfiguration	Contrôle la configuration et l'état d'une connexion SSL
QSslError	Erreur SSL

QSslKey	Interface de clés privées et publiques
QSslSocket	Socket SSL cryptée à la fois pour les clients et les serveurs
QTcpServer	Serveur basé sur TCP
QTcpSocket	Socket TCP
QUdpSocket	Socket UDP
QUrl	Interface pratique pour travailler avec les URL
QUrlInfo	Stocke des informations sur les URL

3. Opérations de haut niveau pour les protocoles HTTP et FTP

L'API d'accès réseau est une collection de classes qui permettent de faire des opérations réseau communes. L'API fournit un niveau d'abstraction sur les opérations et protocoles spécifiques utilisés (par exemple pour obtenir et envoyer des données en HTTP), et n'a de classes, fonctions et signaux pour les concepts généraux et haut niveau.

Les requêtes réseaux sont représentées par la classe QNetworkRequest, qui sert aussi de conteneur pour les informations associées à la requête, comme les informations d'en-tête et l'encryptage utilisé. L'URL fourni pour construire un objet requête détermine le protocole qui sera utilisé. Pour l'instant, les protocoles HTTP, FTP et les URL locales sont supportés pour l'upload et le download.

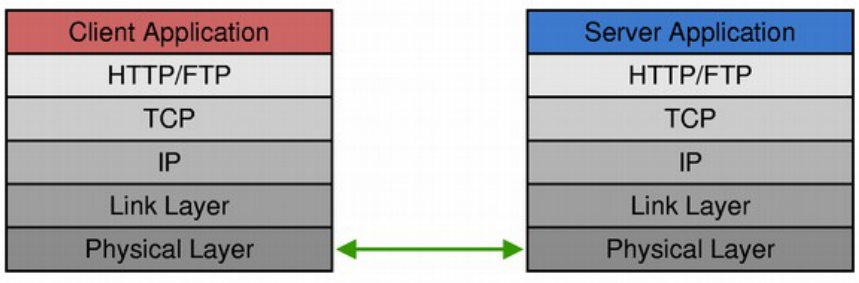
La coordination des opérations réseaux est gérée par la classe QNetworkAccessManager. Une fois qu'une requête a été créée, cette classe est utilisée pour la traiter et envoie des signaux pour informer sur l'avancement. Cette classe gère aussi l'utilisation des cookies pour enregistrer des données sur le client, les authentifications et les proxys.

Les réponses aux requêtes sont représentées par la classe QNetworkReply; elles sont créées par QNetworkAccessManager lorsqu'une requête est traitée. Le signal que fournit QNetworkReply peut être utilisé pour contrôler chaque réponse individuellement, ou on peut choisir d'utiliser celui fourni par le manager et ne pas se servir des références aux réponses. Puisque QNetworkReply est une classe fille de QIODevice, les requêtes peuvent être gérées de façon synchrone ou asynchrone, c'est-à-dire, des opérations bloquantes ou non bloquantes.

Chaque application ou bibliothèque peut créer une ou plusieurs instances de QNetworkAccessManager pour gérer les communications réseaux.

3. Créer des clients FTP avec QFtp

Le protocole FTP (File Transfer Protocol) est utilisé quasi-uniquement pour parcourir des dossiers distants et transférer des fichiers.



Le protocole FTP utilise deux connections réseaux, une pour envoyer les commandes et l'autre pour transférer les données. Ce protocole a un état et requiert que le client envoie plusieurs commandes avant de pouvoir transférer un fichier. Les clients FTP établissent une connexion et la gardent ouverte pendant la session. Dans chaque session, des transferts multiples peuvent avoir lieu.

La classe QFtp fournit un support du protocole FTP côté client avec les caractéristiques ci-dessous.

- `Comportement non bloquant`

QFtp est asynchrone. Vous pouvez mettre en place une série de commandes qui seront exécutées plus tard, lorsque le contrôle reviendra à la boucle d'événements de Qt.

- `Identifiants de commandes`

Chaque commande a un identifiant numérique unique qui peut être utilisé pour suivre l'exécution de la commande. Par exemple, QFtp émet les signaux commandStarted() et commandFinished() avec l'ID de la commande pour chaque commande exécutée.

- `Indicateurs de progression de transfert`

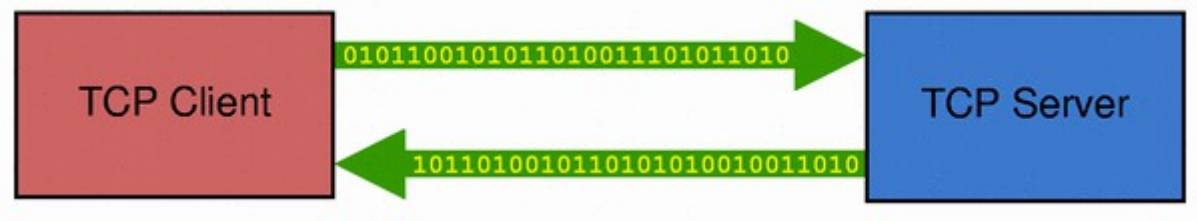
QFtp émet des signaux à chaque fois que des données sont transférées (QFtp::dataTransferProgress(), QNetworkReply::downloadProgress() et QNetworkReply::uploadProgress()). Vous pourriez connecter ces signaux à QProgressBar::setProgress() ou QProgressDialog::setProgress(), par exemple : `Support de QIODevice.` La classe supporte les uploads et downloads depuis et vers des QIODevices, en plus des QByteArrays.

Il y a deux façons différentes d'utiliser QFtp. La plus commune est de garder l'ID de la commande et de suivre l'évolution de chaque commande en connectant les signaux appropriés. L'autre façon est de prévoir toutes les commandes d'un coup et connecter uniquement le signal done(), qui est émis lorsque toutes les commandes prévues ont été effectuées. La première approche requiert plus de travail mais permet un meilleur contrôle sur l'exécution de chaque commande et permet de faire de nouvelles commandes en fonction des résultats obtenus. Elle permet aussi de fournir à l'utilisateur un suivi détaillé.

L'exemple FTP illustre comment créer un client FTP. Créer son propre serveur FTP (ou HTTP) est possible en utilisant les classes bas niveaux QTcpSocket et QTcpServer.

4. Utiliser le protocole TCP avec QTcpSocket et QTcpServer

Le protocole TCP (Transmission Control Protocol) est un protocole réseau bas niveau utilisé par la plupart des protocoles d'Internet, dont HTTP et FTP, pour le transfert de données. C'est un protocole de transport fiable, orienté flux, avec connexion. Il est particulièrement adéquat pour les transmissions continues de données.



La classe QTcpSocket fournit une interface pour le protocole TCP. Vous pouvez utiliser QTcpSocket pour implémenter des protocoles réseau standard comme POP, SMTP et NNTP, aussi bien que des protocoles personnalisés.

Une connexion TCP doit être établie vers un hôte distant et un port avant que le transfert de données puisse commencer. Une fois la connexion établie, l'adresse IP et le port du pair sont disponibles avec les fonctions QTcpSocket::peerAddress() et QTcpSocket::peerPort(). À n'importe quel moment, le pair peut fermer la connexion et le transfert de données s'arrêtera immédiatement.

La classe QTcpSocket est asynchrone et émet des signaux pour reporter des changements de statuts et des erreurs, comme QNetworkAccessManager et QFtp. Elle repose sur la boucle d'événements pour détecter des données arrivantes et automatiquement envoyer les données partantes. Vous pouvez écrire des données dans le socket avec QTcpSocket::write() et en lire avec QTcpSocket::read(). QTcpSocket représente deux flux indépendants de données : un pour écrire et l'autre pour lire.

Puisque QTcpSocket hérite de QIODevice, vous pouvez l'utiliser avec QTextStream et QDataStream. En lisant depuis un QTcpSocket, vous devez être sûr qu'assez de données sont disponibles en appelant QTcpSocket::bytesAvailable() avant.

Si vous avez besoin de gérer des connections TCP entrantes (dans une application serveur), utilisez la classe QTcpServer. Appelez la fonction QTcpServer::listen() pour mettre en place le serveur et connecter le signal QTcpServer::newConnection() à votre slot de réception. Dans votre slot, appelez la fonction QTcpServer::nextPendingConnection() pour accepter la connexion et utiliser le QTcpSocket retourné pour communiquer avec le client.

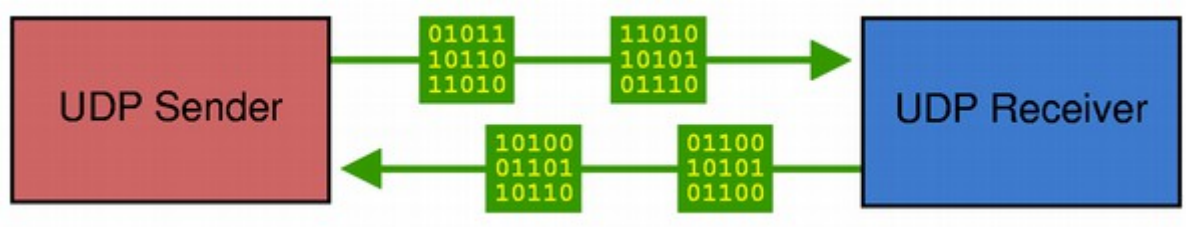
Bien que presque toutes ses fonctions soient asynchrones, il est possible d'utiliser QTcpSocket de façon synchrone (c'est-à-dire bloquante). Pour avoir un comportement bloquant, appeler les fonctions waitFor.....() de QTcpSocket; elles suspendent le processus jusqu'à ce qu'un signal soit émis. Par exemple, après avoir appelé la fonction non bloquante QTcpSocket::connectToHost(), appelez la fonction QTcpSocket::waitForConnected() pour bloquer le processus jusqu'à ce que le signal connected() soit émis.

Les sockets synchrones simplifient souvent le code. Mais le problème est que les événements ne seront pas gérés tant que la fonction waitFor....() bloquera le processus. Si elles sont utilisées dans le processus d'interface graphique, l'interface utilisateur peut se bloquer. Pour cette raison, nous recommandons l'utilisation de sockets synchrones uniquement dans les processus non relatifs à l'interface utilisateur. Lorsqu'elle est utilisée de façon synchrone, la classe QTcpSocket n'a pas besoin de boucle d'événements.

Les exemples client Fortune et serveur Fortune montrent comment utiliser les classes QTcpSocket et QTcpServer pour créer des applications TCP client-serveur. Voir aussi client Fortune bloquant pour un exemple sur comment utiliser un QTcpSocket dans un processus séparé (sans utiliser de boucle d'événements) et serveur Fortune multi-processus pour un exemple de serveur TCP multi-processus avec un processus par client actif.

5. Utiliser le protocole UDP avec QudpSocket

Le protocole UDP (User Datgram Protocol) est un protocole léger, non-fiable, sans connexion, orienté packet. Il peut être utilisé lorsque la fiabilité n'est pas importante. Par exemple, un serveur qui donne le temps pourrait choisir le protocole UDP. Si un packet avec le temps est perdu, le client peut simplement faire une autre requête.



La classe QUdpSocket permet d'envoyer et de recevoir des packets UDP. Elle hérite de QAbstractSocket et elle partage du coup la plupart de l'interface de QTcpSocket. La différence principale est que QUdpSocket transfère les données en tant que packets au lieu d'un flux continu. En cours, un packet est une partie de données de taille limitée (normalement plus petite que 512 octets), contenant l'adresse IP et le port du destinataire et de l'émetteur en plus des données transférées.

QUdpSocket supporte le broadcasing IPv4. Le broadcasting est souvent utilisé pour implémenter des protocoles réseaux de découvertes, comme chercher l'hôte sur le réseau qui a le plus d'espace disque disponible. Un hôte broadcast un packet sur réseau que tous les autres hôtes reçoivent. Chaque hôte recoit une requête et ensuite renvoie une réponse avec la quantité d'espace disque disponible. L'hôte de départ attend d'avoir reçu la réponse de chacun des hôtes et ensuite choisit le serveur avec le plus d'espace disque disponible pour stocker des données. Pour broadcaster un packet, il suffit de l'envoyer à l'adresse spéciale QHostAddress::Broadcast (255.255.255.255) ou à votre addresse de broadcast locale.

La fonction QUdpSocket::bind() prépare le socket pour les packets entrants, comme la fonction QTcpServer::listen() pour les serveurs TCP. À chaque fois qu'un ou plusieurs packets arrivent, QUdpSocket émet le signal readyRead(). Appelez la fonction QUdpSocket::readDatagram() pour lire le packet.

Les exemples émetteur de Broadcast et receveur de Broadcast illustrent comment créer un émetteur UDP et un receveur UDP avec Qt.

6. Résoudre les noms d'hôtes avec QHostInfo

Avant d'établir une connexion réseau, les classes QTcpSocket et QUdpSocket font une recherche de nom, pour traduire le nom d'hôte auquel vous vous connectez en une adresse IP. Cette opération est usuellement faite grâce au protocole DNS (Domain Name Service).

La classe QHostInfo fournit une fonction statique qui vous permet d'effectuer une telle recherche vous-même. En appelant QHostInfo::lookupHost() avec un nom d'hôte, un pointeur QObject et une signature de slot, QHostInfo fera une recherche du nom d'hôte et appellera le slot donné lorsque les résultats seront prêts. La recherche est effectuée dans un processus séparé, en utilisant les méthodes du système d'exploitation pour la faire.

QHostInfo fournit aussi une fonction statique appelée QHostInfo::fromName(), qui prend un nom d'hôte en argument et retourne les résultats. Dans ce cas, la recherche est effectuée dans le même processus. Cette fonction est utile pour faire des recherches dans une application sans interface graphique ou dans un processus indépendant (si le processus gère l'interface graphique, celle-ci peut être bloquée le temps de la recherche).

7. Support des proxys réseau

Les communications réseaux avec Qt peuvent être faites à travers de proxys, qui dirigent ou filtrent

le trafic.

Les proxys individuels sont représentés par la classe QNetworkProxy, qui est utilisée pour décrire et configurer la connexion à un proxy. Les types de proxys qui opèrent sur des niveaux de communications réseaux différents sont supportés, avec le support de SOCKS 5 qui permet le passage par proxy du trafic à bas niveau et le passage par proxy des connections HTTP et FTP au niveau du protocole.

Voir QNetworkProxy::ProxyType pour plus d'informations.

Le passage par proxy peut être établi pour un socket ou pour toutes les communications réseaux d'une application. Un nouveau socket peut utiliser un proxy en appelant sa fonction QAbstractSocket::setProxy() avant de le connecter. Le passage par proxy de toute l'application peut être activé pour toutes les connections ultérieures avec l'utilisation de la fonction QNetworkProxy::setApplicationProxy().

Les usines à proxys sont utilisées pour créer des politiques pour l'utilisation de proxys. QNetworkProxyFactory fournit des proxys basés sur les requêtes pour un certain type de proxys. Les requêtes elles-mêmes sont encodées dans des objets QNetworkProxyQuery qui permetten aux proxys d'être sélectionnés sur des critères clés comme l'utilisation (TCP, UDP, TCP serveur, requête URL), le port local, l'hôte et port distant et le protocole utilisé (HTTP, FTP, etc.).

QNetworkProxyFactory::proxyForQuery() est utilisé pour demander un proxy directement à l'usine. Une politique pour toute l'application sur l'utilisation d'un proxy peut être implémentée en passant une usine à la fonction QNetworkProxyFactory::setApplicationProxyFactory() et une politique personnalisée peut être créée en héritant QNetworkProxyFactory ; voir la documentation de la classe pour les détails.

8. Support de la gestion porteuse

La gestion porteuse contrôle l'état de connectivité d'un appareil pour que l'application puisse démarrer ou arrêter l'interface réseau et changer de façon transparente de point d'accès.

La classe QNetworkConfigurationManager contrôle la liste de configurations de réseaux connus à l'appareil. Une configuration réseau décrit un ensemble de paramètres utilisés pour lancer une interface réseau et est représentée par la classe QNetworkConfiguration.

Une interface réseau est lancée en ouvrant une QNetworkSession basée sur une configuration réseau donnée. Dans la plupart des cas, la création d'une session réseau basée sur la configuration de base spécifiée par le système d'exploitation est appropriée. La configuration réseau de base est retournée par la fonction QNetworkConfigurationManager::defaultConfiguration().

Sur certains systèmes d'exploitation, il est requis que l'application ouvre une session réseau avant d'effectuer des opérations réseaux. Cela peut être testé par la présence du flag QNetworkConfigurationManager::NetworkSessionRequired dans la valeur retournée par la fonction QNetworkConfigurationManager::capabilities().