Урок 8 \* Инструкция

1. Сетевые протоколы

Чтобы использовать классы, связанные с запросами в сеть, следует подключить сетевой модуль в pro файле:



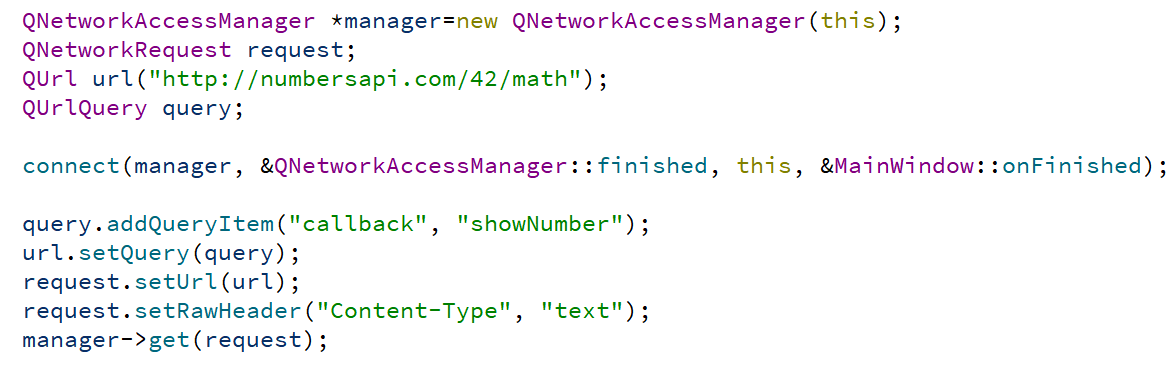
Начиная с QT4.4, для выполнения операций, связанных с HTTP и FTP протоколами в Qt, используются классы QNetWorkRequest, QNetWorkReply и QNetWorkAccessManager:

* QNetworkAccessManager – основной класс, который реализует механизм работы с сетью;
* QNetworkRequest – класс, хранящий конфигурацию запроса и позволяющий его отправить;
* QNetworkReply — класс, содержащий ответ от сервера. Хранит результат выполнения запроса – response.

Схема создания, отправки и обработки запроса производится в несколько шагов. В данном процессе используется объект класса QNetworkRequest для создания запроса. Затем, в зависимости от типа запроса, он передается в объект класса QNetworkAccessManager, вместе с дополнительными данными, если они есть. Ответ на запрос сохраняется в объекте класса QNetworkReply. Так как запросы выполняются асинхронно, то при получении сигнала finished от объекта reply вызывается лямбда-функция. Она проверяет наличие ошибок и вызывает соответствующий обратный вызов (callback). Также слот отвечает за освобождение ресурсов, связанных с запросом. Стоит обратить внимание, что без самостоятельной настройки SSL QNetWorkAccessManager позволяет работать со следующими протоколами:

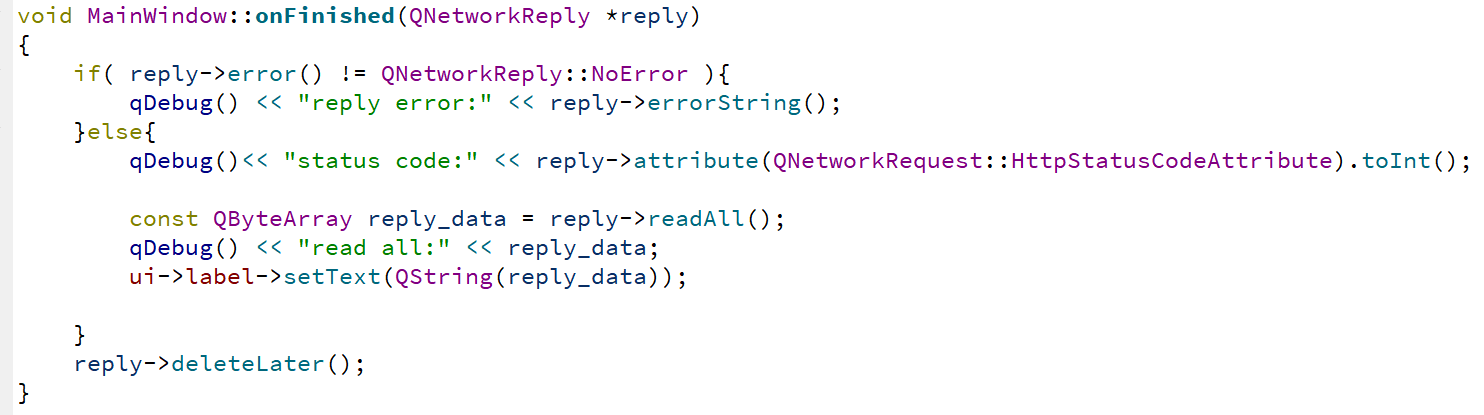
* ftp
* qrc
* file
* data
* http

Пример отправки GET запроса на сервис генерации интересных фактов о числах [NumbersAPI](http://numbersapi.com/):



Для дополнения GET запроса параметрами используется класс QUrlQuery и метод addQueryItem().

Пример слота обработки:



Для обработки запроса стоит проверить, не появились ли ошибки, и присвоить ответ с сервера экземпляру класса QByteArray. Метод deleteLater() является внутренним аналогом очистки памяти с помощью delete.

1. Работа с базами данных

Для возможности использовать базы данных в своем проекте следует подключить пакет sql в pro файле:



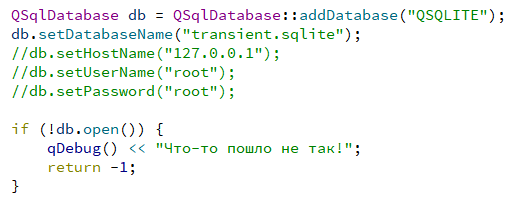
Работа с базами данных в Qt происходит на различных уровнях:

* Слой драйверов – низкоуровневый мост между определенными базами данных и слоем SQL API. Он включает в себя QSqlDriver, QSqlDriverCreator, QSqlDriverCreatorBase, QSqlDriverPlugin и QSqlResult;
* Слой SQL API – cоединения устанавливаются с помощью класса QSqlDatabase. Взаимодействие с базой данных осуществляется с помощью класса QSqlQuery.
* Слой пользовательского интерфейса – дата-ориентированные виджеты. Сюда входят такие классы, как QSqlQueryModel, QSqlTableModel и QSqlRelationalTableModel.

Чтобы получить доступ к базе данных с помощью QSqlQuery и QSqlQueryModel, необходимо создать и открыть соединение с базой данных. Qt может работать со следующими базами данных:

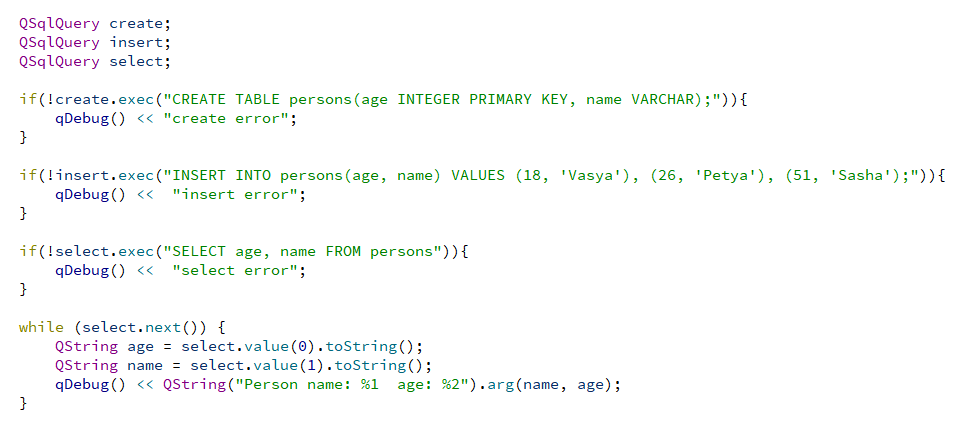
* QDB2 — IBM DB2 (версия 7.1 и выше
* QIBASE — Borland InterBase
* QMYSQL — MySQL
* QOCI — Драйвер Oracle Call Interface
* QODBC — Open Database Connectivity (ODBC) — Microsoft SQL Server и другие ODBC-совместимые базы данных
* QPSQL — PostgreSQL (версия 7.3 и выше)
* QSQLITE2 — SQLite версии 2
* QSQLITE — SQLite версии 3
* QTDS — Драйвер Sybase Adaptive Server

Важно отметить, что не все плагины поставляются с Qt Open Source Edition. Соединить с базой данных SQLite можно следующим образом:



Первая строка создает объект соединения, а метод open() открывает его. В промежутке инициализируется некоторая информация о соединении, включая имя соединения, имя базы данных, имя узла, имя пользователя, пароль. Для каждой базы данных количество информации может отличаться.

Класс QSqlQuery обеспечивает интерфейс для выполнения SQL запросов и навигации по результирующей выборке. Для выполнения SQL запросов следует просто создать в любой части программы объект QSqlQuery и вызвать QSqlQuery::exec():



Если QSqlQuery используется для выборки данных, то с помощью метода next() можно передвигаться по строкам таблицы ответа, а используя метод value(), получать значения из ячеек в строке.