Giao tiếp giữa C++ và QML

1. Đưa dữ liệu vào QML

* Binding dữ liệu (C++ object) sang QML:
  + C++ Object kế thừa QObject, khai báo các thuộc tính QPROPERTY muốn binding (binding nghĩa là gán 1 giá trị cho thuộc tính của QML sử dụng “:”)
  + Register object với QML
* Sử dụng hàm setProperty. Cần phải lấy được rootObject của QML (rootObject có thể hiểu là thẻ chứa thuộc tính cần thay đổi)
* Sử dụng signal phía C++ và slot phía QML để thay đổi thuộc tính cần thay đổi

Note: Nên sử dụng cách đầu tiên. Cách dễ maintain và sử dụng nhất. Khi thuộc tính trong dataobject thay đổi dữ liệu trong QML sẽ tự động thay đổi theo. Khi sử dụng phép gán bằng cho 1 thuộc tính trong QML, thuộc tính đó sẽ mất binding.

1. Lấy dữ liệu từ QML

* Binding dữ liệu (C++ object) sang QML:
  + C++ Object kế thừa QObject, khai báo các thuộc tính QPROPERTY muốn lấy trên giao diện QML
  + Sử dụng hàm property(). Cần phải lấy được rootObject của QML (rootObject có thể hiểu là thẻ chứa thuộc tính cần lấy dữ liệu)
  + Sử dung signal(phía QML) và SLOT(C++) để lấy dữ liệu

1. Liên kết giữa các file QML có thể sử dụng Connection
2. Xử lý business trên C++

* Viết lớp kế thừa từ QObject và đăng ký với QML sử dung qmlRegisterType. Sử dụng hàm setContextProperty để truy cập trong QML (khi sử dụng setContextProperty biến được set sẽ như biến global và có thể sử dụng bất cứ đâu trong QML)
* Viết lớp kế thừ từ QObject không đăng ký với QML và sử dụng signal (phía QML) và SLOT(phía C++) để xử lý nghiệp vụ
* Với những nghiệp vụ đơn giản như đọc ghi file, play media có thể viết thành các utility service (plugin (QML Extension Plugin)) để sử dụng như 1 Item trong QML.
* Có thể kế thừa QQuickItem hoặc QQuickPaintedItem để tạo component cho QML (đăng ký với QML)
* Tham khảo :
  + http://qmlbook.github.io/ch16/index.html#the-application-window
  + https://doc.qt.io/qt-5/qtqml-cppintegration-topic.html
  + https://doc.qt.io/qt-5/qtqml-tutorials-extending-qml-example.html

Note: với những ứng dụng đơn giản, giao diện ít form hiển thị, nghiệp vụ ít thì có thể sử dụng setContextProperty. Với ứng dụng phức tạp nhiều giao diện, nghiệp vụ nên sử dụng cách 2, 3.

“***Context properties*** *are easy to use for small applications. They do not require many effort you just expose your system API with kind of global objects. It is helpful to ensure there will be no naming conflicts (e.g by using a special character for this ($) for example $.currentTime). $ is a valid character for JS variables.*

***Registering QML types*** *allows the user to control the lifecycle of an c++ object from QML. This is not possible with the context properties. Also it does not pollute the global namespace. Still all types need to be registered first and by this all libraries need to be linked on application start, which in most cases is not really a problem.*

*The most flexible system is provided by the* ***QML extension plugins****. They allow you to register types in a plugin which is loaded when the first QML file calls the import identifier. Also by using a QML singleton there is no need to pollute the global namespace anymore. Plugins allow you to reuse modules across projects, which comes quit handy when you do more than one project with Qt..*”

1. Sử dụng javascript tạo Utility sử dụng chung giữa các file QML
2. Chú ý khi chuyển đổi dữ liệu giữa QML và C++
   1. <https://doc.qt.io/qt-5/qtqml-cppintegration-data.html>
3. use findChild

với root object của file qml là windown

Window {

...

Loader {

id: notifyUser

objectName: "notifyUser"

}

}

trên c++, để lấy được đối tượng Loader sử dụng findChild thì cần sử dụng findChild của QObject, không được sử dụng findChild của QQmlContext

example:

------

QQmlEngine engine;

QQmlComponent component(&engine, "qrc:/main.qml");

QObject \*object = nullptr;

object = component.create();

QObject \*itemNotifyUser = nullptr;

itemNotifyUser = object->findChild<QObject\*>("notifyUser");

=> result is address of object notifyUser

itemNotifyUser = QQuickLoader(0x19ac7460, name = "notifyUser")

------

QQmlEngine engine;

QQmlComponent component(&engine, "qrc:/main.qml");

QQmlContext \*context = engine.rootContext();

QObject \*itemNotifyUser = nullptr;

itemNotifyUser = object->findChild<QObject\*>("notifyUser");

=> itemNotifyUser = 0x00

refer: http://doc.qt.io/qt-5/qtqml-cppintegration-interactqmlfromcpp.html

1. data of signal-slot in c++ & qml

a. signal in qml file, slot in c++ file: argument can is QString, QVariant, int ...

ex:

QObject::connect(itemDltItemInListView, SIGNAL(userChooseDeleteItem(int)),

&display, SLOT(updateListItem(int)));

QObject::connect(itemDltItemInListView, SIGNAL(userChooseSearchName(QString)),

&display, SLOT(updateListAfterSearch(QString)));

b. signal in c++ file, slot in qml file: argument must is QVariant, if other then error

ex:

QObject::connect(&myPowerData, SIGNAL(currentPowerChanged(QVariant)),

item, SLOT(receiveSignalChangePower(QVariant)));

cpp file:

signals:

void currentPowerChanged(QVariant currentPower);

qml file:

function receiveSignalChangePower(currentPower) { ... }

Ghi chú:

* Trong 1 file QML có parent và children, thứ tự khởi tạo từ dưới lên trên (growing up). Parent luôn được khởi tạo trước (TrungNB).
* Gọi setContextProperty trước khi load source (TungDT)