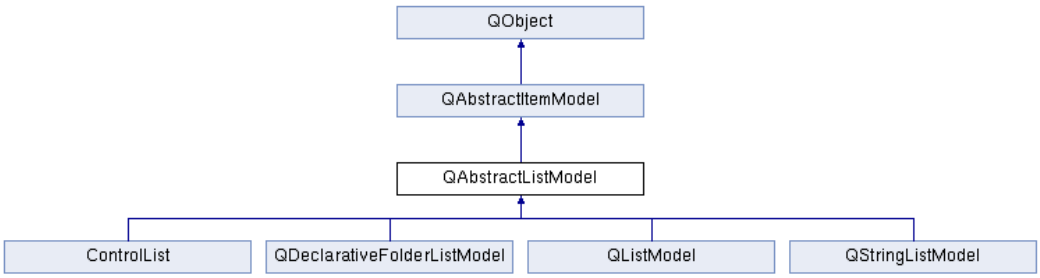
MVC pattern For QT

Model과 Vie,w Controller로 나눠서 UI를 개발할 때 자주 쓰이는 패턴 중 하나인 MVC 패턴은 현재까지도 웹과 모바일을 불문하고 많이 사용하고 있습니다. 위와 같은 MVC 아키텍쳐를 QT에서도 사용할 수 있습니다. 다만 QT에서는 Controller 부분이 View로 합쳐지면서 Model/View 아키텍쳐로 불립니다.

https://www.slideshare.net/ICSinc/indepth-modelview-with-qml

QT Model을 통해서 C++의 비즈니스 로직과 QML의 UI 로직으로 독립시킬 수 있습니다.
그에 따라 C++의 데이터 세트 조작에 영향을 주지 않고 QML UI를 구현할 수 있습니다. 이는 프로젝트에 있어서 유지보수, 테스팅, 개발에 큰 이점을 주고 있습니다.

이런 Model과 View를 나눔에 있어서 가장 많이 쓰이는 방법으로는 QAbstractItemModel이 있습니다. 또한 다량의 데이터를 리스트로 구현하려고 할 때 QAbstractItem Model 하위 클래스안 QAbstractlistmodel이 있습니다. 그렇기에 Qabstractlistmodel은 C++로 구현된 모델을 QML에서 참조 할 수 있는 인기 있는 클래스입니다.



To use AbstractListModel

먼저 두가지 메소드를 지정해줘야합니다
QModelIndex를 가지는 RowCount와 Qvariant data가 있습니다.

```
int rowCount(const QModelIndex &parent = QModelIndex()) const;
QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const;
```

int rowCount(const QModelIndex &parent = QModelIndex()) const;
QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const;

만약 여기서 데이터를 바꾸는 Set 동작을 한다면
QabstractListModel::SetData()를 사용합니다

```cpp
bool ToDoModel::setData(const QModelIndex &index, const QVariant &value, int role)
{
    if (!mList)
        return false;

    ToDoItem item = mList->items().at(index.row());
    switch (role) {
    case DoneRole:
        item.done = value.toBool();
        break;
    case DescriptionRole:
        item.description = value.toString();
        break;
    }

    if (mList->setItemAt(index.row(), item)) {
        emit dataChanged(index, index, QVector<int>() << role);
        return true;
    }
    return false;
}
```

그 다음은 모델 데이터에 접근할 선택자를 추가해줘야 합니다.

```cpp
QHash<int, QByteArray> MyModel::roleNames() const
{
    static QHash<int, QByteArray> mapping {
        {NameRole, "name"},
        {FlagRole, "flag"},
        {PopulationRole, "population"}
    };

    return mapping;
}
```

이 때 QHash<int, QBtyteArray> roleNames();를 사용하여 role들을 지정하고 mapping해줍니다. enum을 key로 문자열 값 매핑을하 rkh 매핑된 Qhash를 리턴합니다 .static Qhash를 사용하는 이유는 roleNames()메소드가 호출이 될 때 맵을 유지하기 위함입니다.

그리고 모델의 데이터를 CRUD와 같은 작업을 할 때 Signal과 Slot을 통해 함수 작업을 진행합니다. 이 때 CRUD시 시작과 끝 매소드를 잡아줘야하는데

```cpp
void ToDoList::appendItem()
{
    emit preItemAppended();

    ToDoItem item;
    item.done = false;
    mItems.append(item);

    emit postItemAppended();
}
```

위와 같은 방식으로 view단에 해당 작업이 수행되고 끝났음을 알립니다.