44 Qt的MV结构

44.1 什么是MV结构?

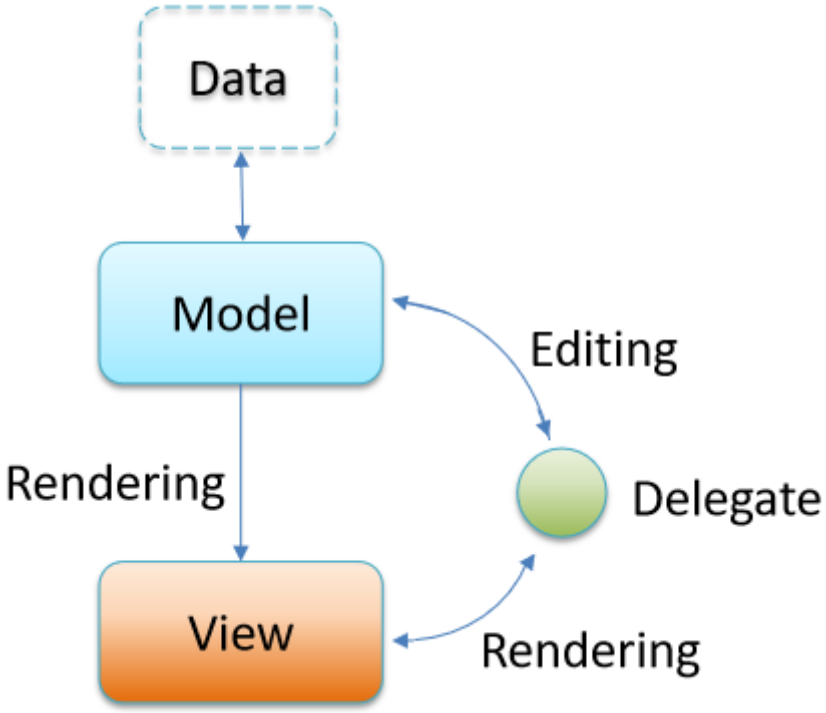
带有图形用户界面的应用程序通常都提供这样的功能，用户通过界面浏览和修改数据。用户所做的修改应直接反映到存储之中，存储中数据的变化也应同步反映到界面的显示。

MV结构，即模型（Model）视图（View）结构，是Qt提供的一种连接用户界面与数据存储的机制。其中，视图是显示、编辑数据的界面组件，而模型则是界面组件与数据存储间的接口。

将界面与存储分离，同时又以数据源的形式将二者连接起来，是解决界面——存储同步问题的绝佳方案。

44.2 MV结构中的角色

MV结构中的角色如下图所示：



- 数据（Data）：实际的数据，如数据库中的一张表、一条SQL语句的查询结果、内存中的一个字符串表、计算机磁盘上的文件系统，等等
- 模型（Model）：与数据建立通信，并为视图提供接口，从数据中提取需要的内容，在视图中显示和编辑。Qt通过不同的模型类管理不同形式的数据，如通过QStringListModel类管理内存中的字符串表，通过QSqlTableModel类管理数据库中的表，等等
- 视图（View）：屏幕上的界面组件，视图从模型中根据索引获取数据，并在可视化组件中显示。Qt提供了一系列用于创建视图组件的类，如QListView、QTableView、QTreeView等。同一个模型可以在不同的视图中显示，甚至可以在不修改模型的情况下，自己定义特殊的显示视图
- 代理（Delegate）：为视图赋予编辑数据的能力。当数据需要被编辑时，代理通过索引与模型通信，并为需要编辑的数据提供一个编辑器，比如一个QLineEdit组件

44.3 MV角色间的交互

视图、模型和代理之间借助信号和槽通信：

- 当数据发生变化时，模型发射信号，通知视图响应这些变化
- 当用户通过界面操作数据时，视图发射信号，表示这些操作
- 当数据正被编辑时，代理发射信号，告知模型和视图编辑器的当前状态