一去、二三里的博客

http://blog.sina.com.cn/liang19890820

个人资料 [管理]

一去、二三里

Qing 微博

博客等级：18
博客积分：78
博客访问：407,809
关注人气：378
获赠金笔：12
赠出金笔：0
荣誉徽章：

相关博文

- QtQuickControls模块 一去、二三里
- Qt之模型/视图（自定义风格） 一去、二三里
- 交换机(Q-in-Q)技术学习 hyccxc
- QTQTableView用法小结 雨儿
- Test1QTQTreeView左键和右键事件 又一只兔子
- Qt的树形控件 woshihaoren
- Qt之模型/视图（实时更新数据） 一去、二三里
- Qt之QTreeView（三） 一去、二三里
- Qt之模型/视图（委托） 一去、二三里
- Qt解析XML文件（QXmlStreamReade 一去、二三里

更多>>

正文 字体大小：大 中 小

Qt之模型/视图

(2014-01-08 15:48:01) [编辑][删除] 转载

标签： qt qt使用mvc qtableview qlistview qtreeview 分类： Qt

关于Qt中MVC的介绍与使用，助手中有一节模型/视图编程（Model/View Programming）讲解的很清晰。

Qt包含一组使用模型/视图结构的类，可以用来管理数据并呈现给用户。这种体系结构引入的分离使开发人员更灵活地定制项目，并且提供了一个标准模型的接口，以允许广泛范围的数据源被使用到到现有的视图中。

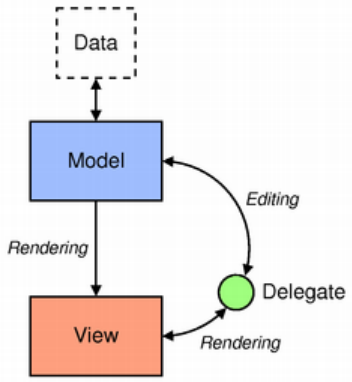
模型 - 视图 - 控制器（MVC）是一种设计模式，由三类对象组成：

- 模型：应用程序对象。
- 视图：屏幕演示。
- 控制器：定义了用户界面响应用户输入的方式。

在引入MVC之前，用户界面的设计往往是将这些对象组合在一起。MVC的解耦带来了灵活性和重用性。

如果视图和控制器对象相结合，其结果是模型/视图结构，仍然分离了数据与呈现给用户的方式，但提供了基于相同原理的简单框架。这种分离使得它可以在几个不同的视图中显示相同的数据，并且实现新类型的视图，而无需改变底层的数据结构。为了灵活地处理用户输入，则引入了委托的概念。在此框架引入委托的优点是：它允许项目数据显示和自定义编辑。

模型/视图结构



模型与数据源进行通信，在这个体系结构中为其它组件提供了一个接口。通信的性质依赖于数据源的类型以及模型的实现方式。

视图从模型中得到模型索引，这些都引用到数据项。通过为模型提供模型索引，视图可以从数据源中检索数据项。

在标准的视图里，委托呈现数据项目。当一个项目被编辑，委托与模型直接利用模型索引进行通信。

模型/视图/委托通信

模型、视图、委托使用信号和槽相互通信：

- 模型的信号：通知视图关于改变由数据源保持的数据。
- 视图的信号：提供了关于用户交互显示的项目信息。
- 委托的信号：当编辑时告诉模型和视图编辑器的状态。

模型

所有的模型都基于QAbstractItemModel类。这个类定义了一个使用视图和委托来访问数据的接口。数据本身不是必须要存储在模型中，可以在一个数据结构或一个单独的类、文件、数据库、或其它一些应用组件。

QAbstractItemModel为数据提供了一个接口，它足够的灵活性来处理表格、列表、树形式的数据视图。然而，实现新的列表和类似于表的数据结构模型时，QAbstractListModel和QAbstractTableModel类是更好的起点，因为它们提供了适当的常用的功能的默认实现。这些类可以派生子类，用来提供支持特定种类的列表和表格

的模型。

Qt提供了一些现成的模型，可以用来处理数据项：

- QStringListModel：用于存储简单的QString的列表项。
- QStandardItemModel：管理更复杂的树结构件，其中每一个项目可以包含任意数据。
- QFileSystemModel：提供有关本地文件系统的文件和目录信息。
- QSqlQueryModel、QSqlTableModel、QSqlRelationalTableModel：使用模型/视图约定来访问数据库。

如果这些标准模型不能满足要求，则可以继承化QAbstractItemModel、QAbstractListModel或QAbstractTableModel来创建自定义模型。

视图

提供了完整的实现为各种不同的视图：而QListView显示项目列表，QTableView中从一个表模型中显示数据，QTreeView则显示了层次列表的数据模型项目。每个类是基于QAbstractItemView抽象基类。虽然这些类是准备使用的实现，他们也可以被子类化，以提供自定义的视图。

委托

QAbstractItemDelegate在模型/视图框架中代表抽象的基类。默认的委托实现由QStyledItemDelegate提供，这被Qt的标准视图用作默认的委托。然而，QStyledItemDelegate和QItemDelegate独立替代绘画，且为视图项提供编辑器。它们之间的区别在于QStyledItemDelegate使用当前样式来绘制项目。因此，建议实现自定义委托或当与Qt样式表一起使用时，使用QStyledItemDelegate作为基类。

排序

在模型/视图结构中有两种接近的排序方式，选择哪种方式取决于你的基础模型。

如果你的模型是可排序的，也就是说，如果重新实现了QAbstractItemModel::sort()方法，QTableView和QTreeView都提供了一个API，允许以编程方式排序来排序模型数据。此外，可以启用交互式排序（即允许用户将数据通过单击视图的标题进行排序），由QHeaderView::sortIndicatorChanged()信号分别连接到QTableView::sortByColumn()槽或QTreeView::sortByColumn()槽。

另一种方法，如果模型没有所需的接口，或者如果想使用一个列表视图来显示数据，使用代理模型呈现数据视图之前应转换模型的结构。

方便的类

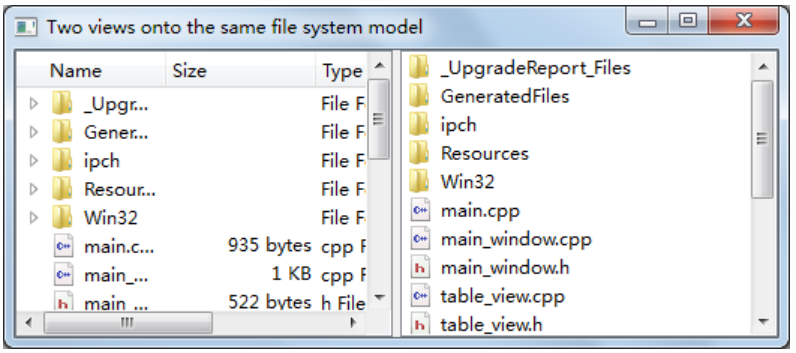
一些便利类都源于标准视图类的依赖Qt的项目为基础的项目视图和表类应用的好处。他们不打算被继承。这些类的实例包括QListWidget，QTreeWidget和QTableWidget。

这些类比视图类灵活性差，且不能与任意的模型使用。建议使用模型/视图的方法来处理在项目视图中的数据，除非强烈需要一个基于项目的类。

如果想利用模型/视图提供的特性方法，同时使用一个基于项目的接口，可以考虑使用视图类，例如：QListView、QTableView、QTreeView与QStandardItemModel。

使用视图与现有的模型

QListView和QTreeView类是最合适的视图来使用QFileSystemModel。下面介绍的示例在树视图中显示一个目录，旁边列表视图中显示相同的信息。该视图共享用户的选择，这样选择的项目在两个视图中高亮显示。



代码如下：

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QSplitter *splitter = new QSplitter;

    QFileSystemModel *model = new QFileSystemModel;
    model->setRootPath(QDir::currentPath());

    QTreeView *tree = new QTreeView(splitter);
    tree->setModel(model);
    tree->setRootIndex(model->index(QDir::currentPath()));

    QListView *list = new QListView(splitter);
```

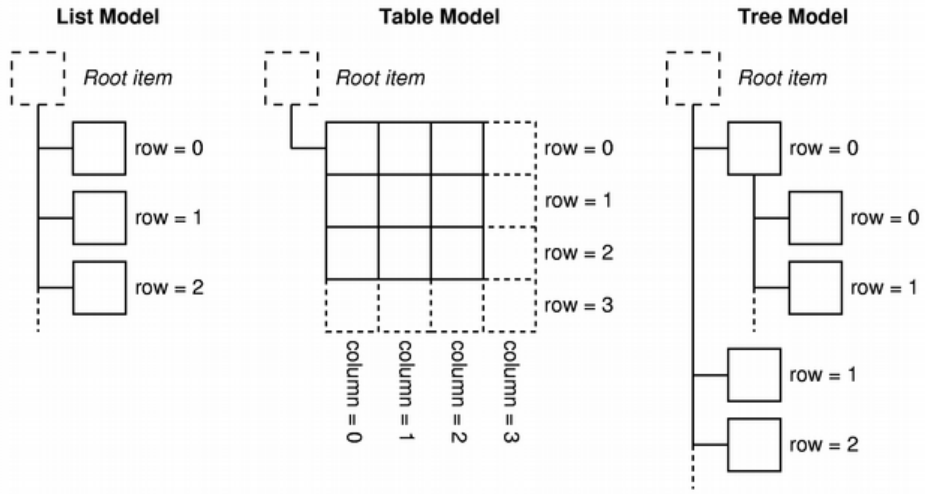
```
    list->setModel(model);
    list->setRootIndex(model->index(QDir::currentPath()));

    splitter->setWindowTitle("Two views onto the same file system model");
    splitter->show();
    return app.exec();
}
```

上面的例子中，我们忽略提及如何处理选择的项目。下面将详细讲述在视图中处理所选的项目。

基本概念

在模型/视图结构中，模型提供了视图与委托访问数据的标准接口。在Qt中，标准的接口由QAbstractItemModel类定义。无论多么数据项被存储在任何底层的数据结构中，QAbstractItemModel的所有子类所代表的数据作为包含视图项的分层结构。视图使用这个约定来访问模型中的数据项，但并不限制将该信息传达给用户的方式。



Model indexes

为确保数据被分开被访问，模型索引的概念被引入。可以通过模型索引来获得每条信息。视图与委托使用这些索引来请求显示的数据项。

因此，模型只需要知道如何获取数据，并通过模型管理的数据的类型可以被相当普遍定义。型号索引包含一个指向创建它们的模型的指针，在处理多个模型时可以防止混乱。

```
QAbstractItemModel *model = index.model();
```

模型索引提供临时参考信息，并且可以用于通过模型来检索或修改数据。由于模型可能重组其内部结构，模型的索引可能会变得无效，不宜存储。如果需要长期参考一条信息，必须创建一个持久性模型索引。这为模型保持最新信息提供了一个参考。临时模型索引由QModelIndex类提供，持久性模型索引由QPersistentModelIndex类提供。

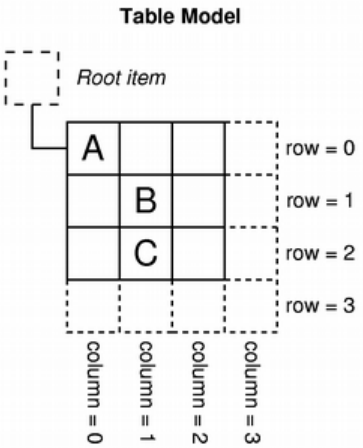
取得对应于数据项的模型索引，模型中必须制定三个属性：一个行号、一个列号，以及父项的模型索引。

行和列

在最基本的形式中，模型可以被一个简单的表访问，表项位于行号和列号，这并不意味着底层数据存储在数据结构中，使用行号和列号只是一个惯例，以允许组件相互通信。我们可以通过指定行号和列号的模型索引有关的任何特定信息，通过下面的方式得到项目的索引：

```
QModelIndex index = model->index(row, column, ...);
```

模型提供的接口简单，单级的数据结构如列表和表格不需要提供任何其他信息。但是，正如上面的代码所示，当获得一个模型索引时，我们需要提供更多的信息。



行和列

图中显示了一个基本的表模型，其中的每个项目的位置由一对行号和列号表示。我们通过模型索引（一个项目数据）行号和列号来获取。

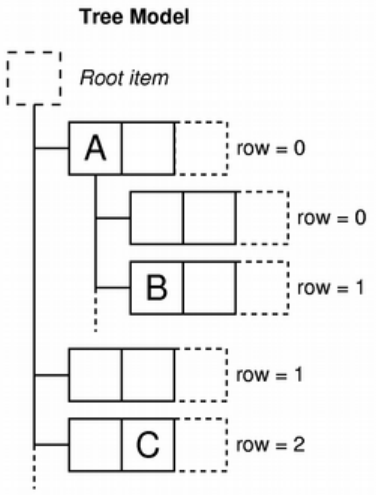
```
QModelIndex indexA = model->index(0, 0, QModelIndex());
QModelIndex indexB = model->index(1, 1, QModelIndex());
QModelIndex indexC = model->index(2, 1, QModelIndex());
```

父节点

由模型提供的类似于表的接口给数据项是理想的当在表格或列表视图中使用数据，行号和列号准确地映射到视图显示项目的方式。然而，结构，如树视图需要更模型为项目暴露一个更灵活的接口。因此，每个项目也可以是另一个表的父项，大致相同的方式，在一个树视图中的顶级项目可以包含另一个列表项。

当请求的一个模型项的索引时，必须提供有关该项目的父项目的一些信息。在模型外，指定一个项目的唯一途径是通过一个模型索引，所以父模型索引也必须如下给出：

```
QModelIndex index = model->index(row, column, parent);
```



父项，行和列

该图显示了一个树模型，其中每个项目都依赖于由一个父项，一个行号和一个列号。

项目的“A”和“C”表示模型顶层的兄弟姐妹：

项目“A”有很多孩子，可以通过如下方式由“A”索引得到“B”索引：

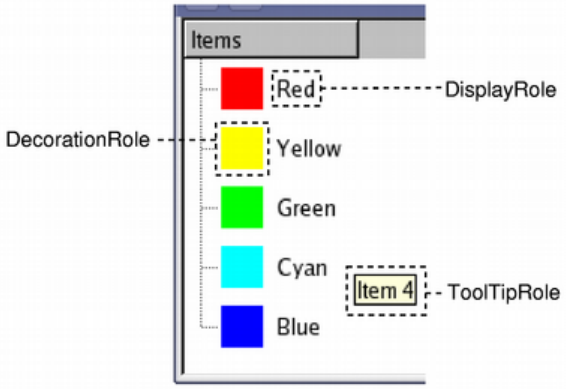
```
QModelIndex indexB = model->index(1, 0, indexA);
QModelIndex indexA = model->index(0, 0, QModelIndex());
QModelIndex indexC = model->index(2, 1, QModelIndex());
```

项目角色

模型中的项目可以为其他组件演绎不同的角色，允许为不同的情况提供不同类型的数据。例如，Qt::DisplayRole可以在用于访问视图中被显示为文本的字符串。通常情况下，包含数据的项目用于若干不同的角色，且标准角色被Qt::ItemDataRole定义。

我们可以通过模型索引传递给相应的项目向模型请求项目数据，并通过指定一个角色来获取想要的数据类型，如下：

```
QVariant value = model->data(index, role);
```



数据类型被称为模型的角色指示器。视图可以以不同的方式显示角色，因此，为每个角色提供相应的信息非常重要。

项目数据最常见的用途是覆盖在Qt::ItemDataRole中定义的标准角色。通过为每个角色提供相应的项目数据，模型可以为视图和委托提供有关项目应如何呈现给用户的指示，不同的视图可以根据需要来解释或忽略此信息。此外，也可以为应用程序的特定目的而定义附加的角色。

使用模型索引

为了演示如何将数据从一个模型中进行检索，使用模型索引，我们创建了一个QFileSystemModel，在窗体上

没有视图以及显示文件和目录的名称。虽然这并不是使用模型的正常方式，它表明模型在处理模型索引上的约定。

我们用以下述方式构建了一个文件系统模型：

```
QFileSystemModel *model = new QFileSystemModel;
QModelIndex parentIndex = model->index(QDir::currentPath());
int numRows = model->rowCount(parentIndex);
```

在这种情况下，我们设置了一个默认QFileSystemModel，由该模型使用index()的特定实现来获取父索引，使用rowCount()函数来计算行号。

为简单起见，我们只关心模型中的第一列中的项目。我们检查每一行，依次获取每一行中的第一个项目的模型索引，以及读出所存储在该模型项目中的数据。

```
for (int row = 0; row <<span style=" color:#c0c0c0;"> numRows; ++row) {
    QModelIndex index = model->index(row, 0, parentIndex);
```

为了获得一个模型索引，我们指定了行号、列号（第一列为零），以及所有我们想要的项目的父项目的模型索引。每个项目中的文本检索可以使用模型的data()函数来获取。我们指定了模型索引和Qt::DisplayRole来获取数据项中的字符串。

```
    QString text = model->data(index, Qt::DisplayRole).toString();
    // Display the text in a widget.
}
```

使用模型

我们创建一个字符串列表模型作为例子，设置一些数据，并构造一个视图来显示模型的内容。这都可以在一个单一的函数执行：

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QStringList numbers;
    numbers << "One" << "Two" << "Three" << "Four" << "Five";
```

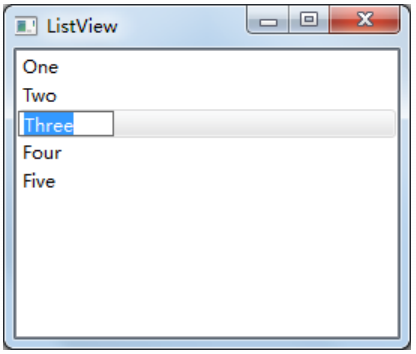
请注意，QStringListModel被声明为一个QAbstractItemModel。这使我们能够为模型使用抽象接口，并确保代码仍然有效，即使我们使用不同的模型替换了字符串列表模型。

由而QListView提供的列表视图足以展示字符串列表模型中的项目。我们构建视图，并设置模型可以使用下面代码：

```
    QAbstractItemModel *model = new QStringListModel(numbers);
    QListView *view = new QListView;
    view->setModel(model);

    视图正常显示方式
    view->show();
    return app.exec();
}
```

视图展现模型的内容，通过模型的接口访问数据。当用户试图编辑一个项目时，视图使用缺省代表提供一个编辑器部件。



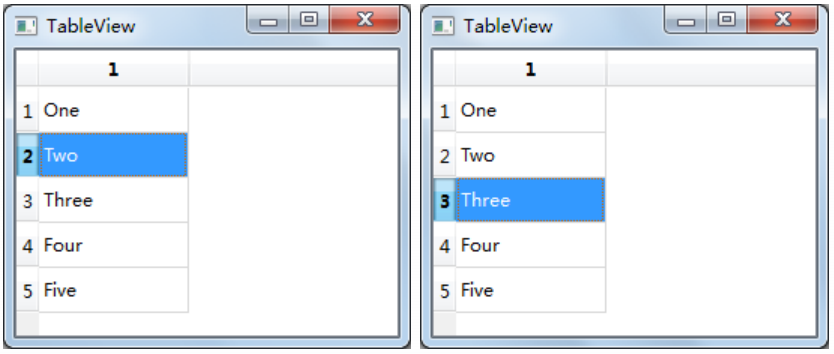
上面的图显示QListView如何使用字符串列表模型表示数据。由于模型可编辑，视图会自动允许列表中的每个项目使用默认的委托进行编辑。

一个模型的多个视图

一个模型可以为多个视图所使用。在下面的代码中，我们创建两个表视图，使用的均是创建好的同一个模型。

```
QTableView *firstTableView = new QTableView;
QTableView *secondTableView = new QTableView;
firstTableView->setModel(model);
secondTableView->setModel(model);
```

在模型/视图框架中使用信号和槽是指更改模型可以传递给所有相连的视图，以确保始终可以访问相同的数据，而不管所使用的视图。



上面的图显示了统一模型的两种不同的视图，每个都包含了一些选定的项目。尽管模型中的数据在视图显示一致，每个视图维护它自己的内部选择模型。这在某些情况下有用，但对于许多应用来说，则需要一个共享的选择模型。

视图共享选择

虽然视图类提供自己的默认选择模型很方便，但当我们使用多个视图到同一个模型时，通常需要所有的模型数据和用户的选择在所有视图显示一致。由于视图类允许其内部选择模型进行更换，那么可以使用如下方式实现视图之间的统一：

```
secondTableView->setSelectionModel(firstTableView->selectionModel());
```

第二个视图给出了第一个视图的选择模型。这两种视图现在在同一个选择模型进行操作，保持了数据和所选项目的同步。

注：

作者：`\☆奋斗ing❤孩子` 原文：http://blog.sina.com.cn/s/blog_a6fb6cc90101hh20.html。

前一篇：开始使用
后一篇：Qt之模型/视图（委托）

发评论

〈 前一篇
开始使用

后一篇 〉
Qt之模型/视图（委托）