

Таблицы, деревья и списки

Таблицы, деревья и списки

- Основанные на элементах:
 - QListWidget
 - QTableWidget
 - QTreeWidget
- Использующие архитектуру модель/представление:
(Model/View)
 - QListView
 - QTableView
 - QTreeView
 - QColumnView

QTableWidget – таблица

1. Устанавливаем число столбцов и строк:

```
tableWidget->setRowCount(10);  
tableWidget->setColumnCount(5);
```

2. Добавляем элементы:

```
QTableWidgetItem *newItem = new QTableWidgetItem(tr("текст"));  
tableWidget->setItem(row, column, newItem);  
// за удаление newItem теперь отвечает tableWidget
```

3. Устанавливаем заголовки:

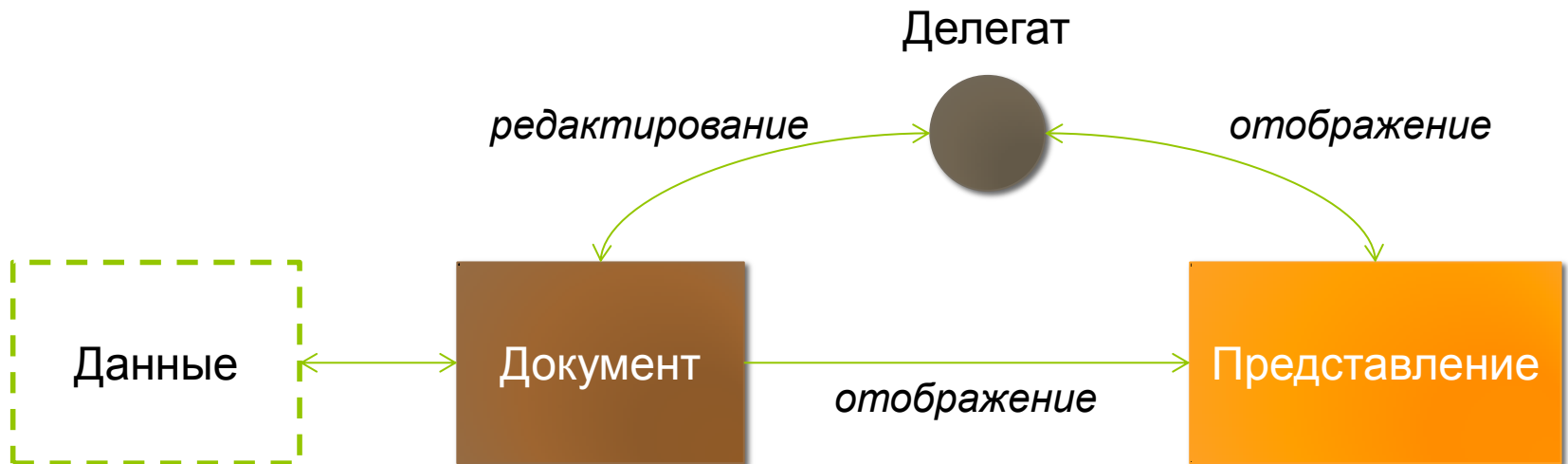
```
tableWidget->setHorizontalHeaderLabels(  
    QStringList() <<tr("Фамилия") <<tr("Имя") <<tr("Отчество"));
```

4. Для красоты:

```
tableWidget->resizeColumnsToContents();
```

Архитектура модель/представление

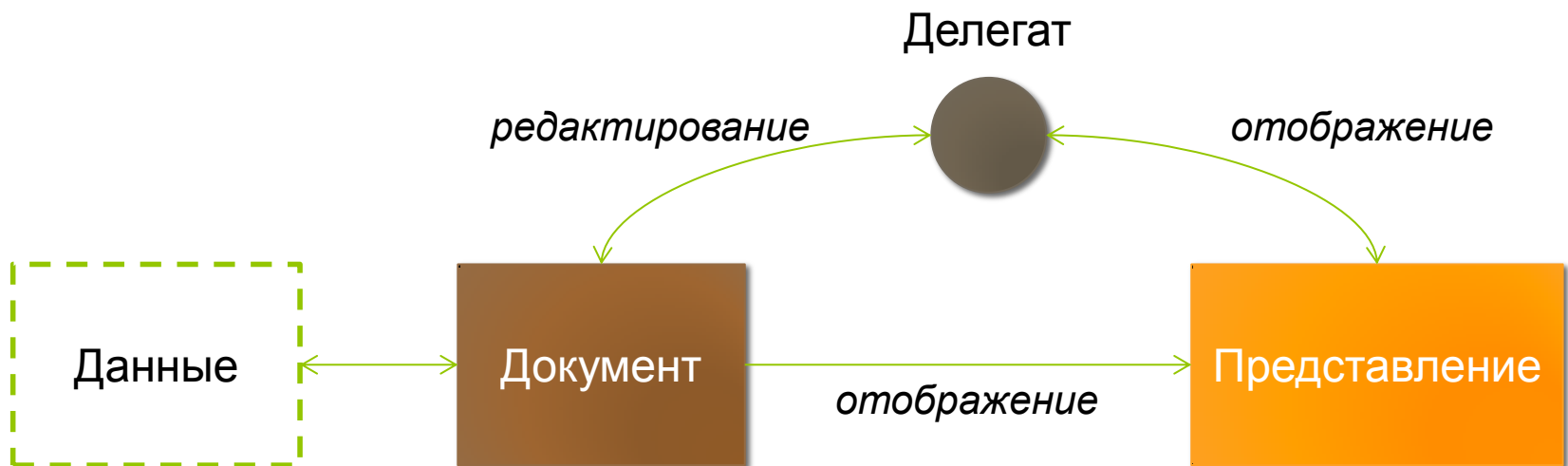
- **Модель** (model) – обеспечивает хранение данных или доступ к внешнему источнику.
- **Представление** (view) – отображает данные на экране и обеспечивает взаимодействие с пользователем. Для получения данных обращается к документу и запрашивает данные с использованием **индекса** (model index).
- **Делегат** (delegate) – обеспечивает отображение и редактирование элементов данных.



Архитектура модель/представление

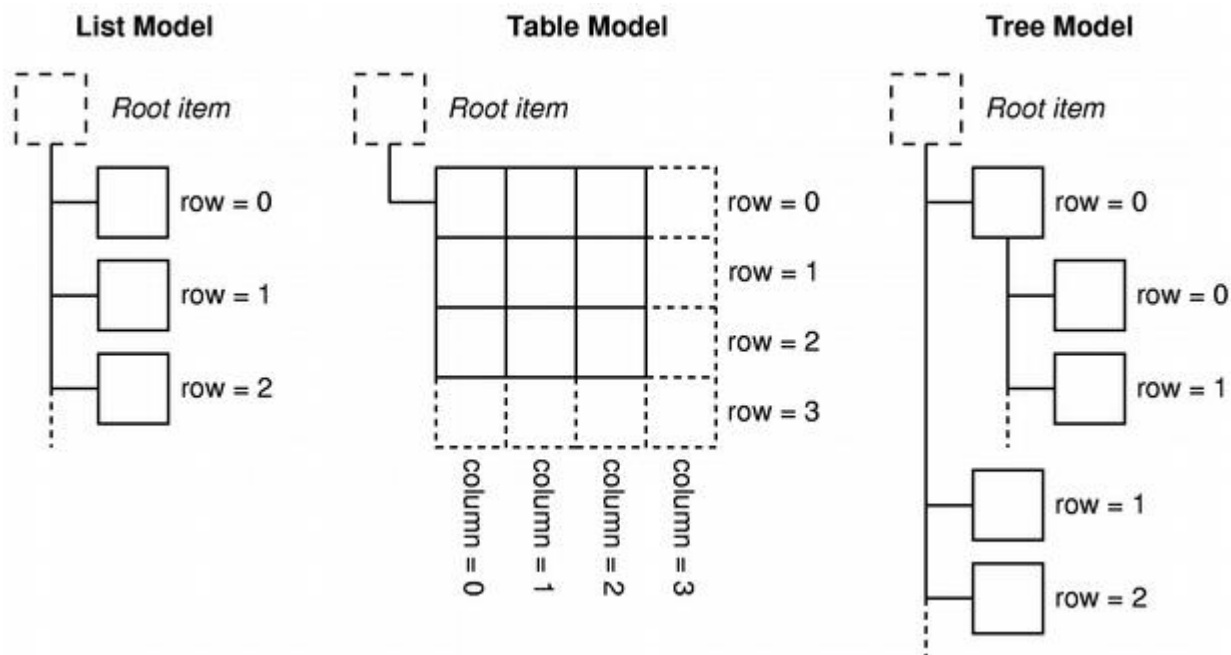
Модели, представления и делегаты взаимодействуют друг с другом, используя сигналы и слоты:

- Сигналы от модели информируют представление об изменениях данных в источнике данных.
- Сигналы от представления предоставляют информацию о пользовательском взаимодействии с отображаемыми элементами.
- Сигналы от делегата используются во время редактирования, сообщают модели и представлению о состоянии редактора.



Модель

- В Qt модель наследуется от `QAbstractItemModel` и представляет данные в виде иерархической структуры, хранящей таблицы из элементов:

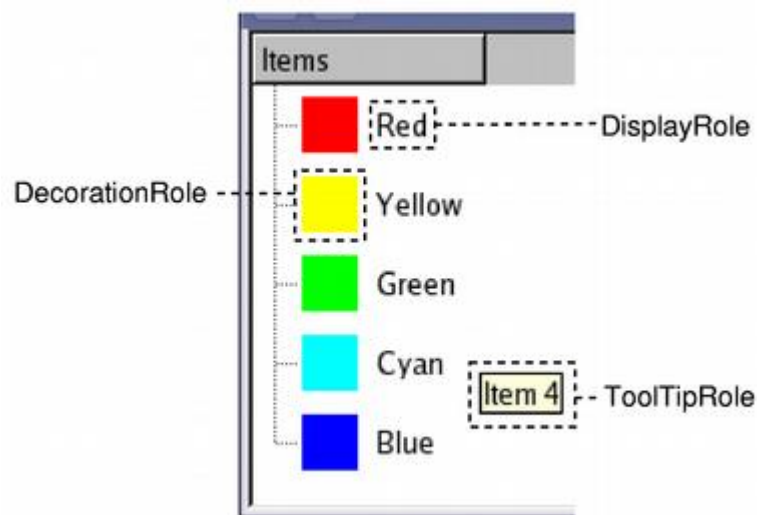


Индексы

- Элементы адресуются с помощью индекса, представленного классом `QModelIndex`.
- Расположение индекса можно узнать с помощью методов:
 - `int row();`
 - `int column();`
 - `QModelIndex parent();`
- У элементов верхнего уровня родителем является **недопустимый индекс**, создаваемый конструктором без параметров `QModelIndex()`.
- Создать допустимый индекс можно функцией `QModelIndex QAbstractItemModel::index (int row, int column, const QModelIndex & parent = QModelIndex())`

Роли

- Каждый элемент представляется набором данных, имеющих разные роли:
 - Qt::DisplayRole – текст элемента (QString)
 - Qt::DecorationRole – иконка (QIcon, QPixmap, QColor)
 - Qt::EditRole – данные, подходящие для редактора (QString)
 - Qt::ToolTipRole – текст всплывающей подсказки (QString)
 - ...



Как создать минимальную модель

Создайте класс, унаследованный от **QAbstractListModel** или **QAbstractTableModel**:

```
class MyModel : public QAbstractTableModel
{
    Q_OBJECT
public:
    MyModel(QObject *parent = 0);

    int rowCount(const QModelIndex &parent = QModelIndex()) const;
    int columnCount(const QModelIndex &parent = QModelIndex())
const;

    QVariant data(const QModelIndex &index, int role) const;

    QVariant headerData(int section, Qt::Orientation orientation,
                        int role = Qt::DisplayRole) const;

    ...
}
```

Как создать минимальную модель

```
QVariant MyModel::headerData(int section,
                             Qt::Orientation orientation, int role) const
{
    if(role!=Qt::DisplayRole)
        return QVariant();

    if(orientation==Qt::Horizontal)
    {
        switch(section)
        {
            case 0: return QVariant(tr("Артефакт"));
            case 1: return QVariant(tr("Цена"));
            default: return QVariant();
        }
    }
    else
        return QVariant(tr("%1").arg(section+1));
}
```

Как создать минимальную модель

```
QVariant MyModel::data(const QModelIndex &index, int role) const
{
    if(role!=Qt::DisplayRole)
        return QVariant();

    switch(index.column())
    {
        case 0: return artefactList[index.row()].name;
        case 1: return artefactList[index.row()].cost;
        default: return QVariant();
    }
}
```

Редактируемые модели

- `Qt::ItemFlags flags(const QModelIndex &index) const;`
 - Нужно вернуть `QAbstractItemModel::flags(index) | Qt::ItemsEditable`
- `bool setData(const QModelIndex &index, const QVariant &value, int role = Qt::EditRole);`
 - Изменяет данные
 - Активирует сигнал `dataChanged(const QModelIndex & topLeft, const QModelIndex & bottomRight)`
- Добавление/удаление строк или столбцов:
 - `bool insertRows(int position, int rows, const QModelIndex &index = QModelIndex());`
 - `bool removeRows(int position, int rows, const QModelIndex &index = QModelIndex());`
- Если вы хотите, чтобы при редактировании элементов использовались выпадающие списки, spinbox'ы и т.п., то необходимо реализовать своего делегата, смотрите `QAbstractItemDelegate` и его наследников.

Как подключить свою модель

Используйте

```
QAbstractItemView::setModel  
    ( QAbstractItemModel * model )
```