**QML中的Model View——一个ListView实例**

http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif(2011-11-14 15:14:09)

|  |  |
| --- | --- |
| 标签：  **[杂谈](http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag" \t "_blank)** |  |

**引言**

Qt 中的 Model View 大家已经比较熟悉了（详情请参阅[model view programming](http://doc.qt.nokia.com/4.7/model-view-programming.html)）。在QML中也有类似的模式。本文就向大家初步讲解一下 QML 中的 Model View 的用法，并提供一个 ListView 的例子程序供大家参考。

**Model**

Models 是用来提供数据的，它既可以以 QML 的形式出现也可以是 C++的类。QML中的Model有 [ListModel](http://doc.qt.nokia.com/4.7/qml-listmodel.html)、[XmlListModel](http://doc.qt.nokia.com/4.7/qml-xmllistmodel.html)、 [VisualItemModel](http://doc.qt.nokia.com/4.7/qml-visualitemmodel.html)；C++中的 Model 有 QAbstractItemModel、QStringList、 QList<QObject\*>等。

**View**

View 是用来显示 Model 中的数据的，如果有 Delegates，则 View 会通过 Delegates 的帮助来显示 Model 中的每一个 Item。

QML 中有[ListView](http://doc.qt.nokia.com/latest/qml-listview.html), [GridView](http://doc.qt.nokia.com/latest/qml-gridview.html), [PathView](http://doc.qt.nokia.com/latest/qml-pathview.html), [Repeater](http://doc.qt.nokia.com/qt-maemo-4.7/qml-repeater.html) 这几种，当然大家也可以在这几种View的基础上扩展写出来自己的View。这些View都自动实现了动力滚动和弹簧效果。

**Delegates**

Delegates 是用来为 Model 中的每一个 Item 创建其对应的实例，并被 View 拿来做显示用的。

这里再补充一点，在大家在看诸如 ListView、GridView、PathView 这些 QML Element 的时候会发现它们都有一个叫 **Highlight** 的属性，它定义的是怎样对 Item 进行高亮突出显示。

其实 **Highlight** 和 Delegates 都是一个[QML Component](http://doc.trolltech.com/4.7/qml-component.html)。我们通常会把一个 Component 写到一个 .qml 文件里（文件名第一个字母大写），从而利于程序其他地方以及日后的复用。 Highlight 和 Delegates 的作用本质都是相同的，都是为了描述如何显示 Model 中的每一块数据的，只是 Highlight 只会在 Item 在选中的情况下起作用。

**例程主要代码分析**

首先我们来到 **main.qml** 文件，这是 qml 入口文件（ ”m“是小写的，与其他大写字母开头的Component有区别）。

ListView {

id: listView

z:0

anchors.fill: parent

model: MyModel {}

delegate: myDelegate

}

”ListView“ ：我们这里直接使用了QML ListView Element。给它一个id，为了在其他地方可以引用到它。

”model: MyModel {}“：我们用了一个 model，叫做 MyModel，它是写在一个叫 MyModel.qml 文件中的，model 的名字是由文件名决定的，model名首字母要大写 。 我们这里直接写“MyModel {} “，只要有MyModel.qml这个文件，QML engine 就会认识的。其实我们也可以不这样写，如果这个 Model 有个 id，我们直接在这里写它的 id 也可以。

“delegate: myDelegate”：myDelegate是 我们自定义的一个 Delegate 的 id。

下面我们看下 myDelegate 是如何定义的：

Component {

id: myDelegate

//以下全部省略

然后我们继续看 myDelegate 里面还有什么：

states: [

State {

name: "Details"

PropertyChanges { target: listView; z:2}

PropertyChanges { target: background; color: "ivory" }

PropertyChanges { target: myImage; width: 180; height: 180 } // Make picture bigger

PropertyChanges { target: myContainer; detailsOpacity: 1; } // Make details visible

PropertyChanges { target: myContainer; height: listView.height } // Fill the entire list area with the detailed view

PropertyChanges { target: idTitle; color: "black" }

PropertyChanges { target: idTitle; font.pointSize: 11 }

PropertyChanges { target: topLayout; x: 10; }

PropertyChanges { target: myContainer.ListView.view; explicit: true; contentY: myContainer.y }

PropertyChanges { target: ListView.view; interactive: false }

},

State {

name: ""

PropertyChanges { target: listView; z:0}

PropertyChanges { target: background; color : "black"}

PropertyChanges { target: myImage; width: 47; height: 47 }

PropertyChanges { target: myContainer; detailsOpacity: 0; }

PropertyChanges { target: myContainer; height: 65 }

PropertyChanges { target: idTitle; color: "white" }

PropertyChanges { target: idTitle; font.pointSize: 9 }

PropertyChanges { target: topLayout; x: 50; }

PropertyChanges { target: myContainer.ListView.view; interactive: true }

}]

transitions: Transition {

ParallelAnimation {

ColorAnimation { property: "color"; duration: 400 }

NumberAnimation { duration: 400; properties: "detailsOpacity,x,contentY,height,width" }

}

以上这部分代码也出现在 myDelegate 中。

它定义的是两种状态，一个叫 "Details" （自己起的名字），另一个叫 ""也就是一个空的字符串，这是这个 Item 的默认状态。在这两种不同状态下，delegate 中不同对象所对应的属性值都是不同的。

当发生状态切换的时候，就会触发动画效果（由 transitions 定义），其中 contentY 的变化会使得原本列表中的一个小 Item 伸展充满整个屏幕。

最后我们来到 MyModel.qml 文件，看 model 是如何定义的：

ListModel {

ListElement {

title: "Qt"

picture: "content/pics/Qt.png"

detailstr: "<html>

<ol>

<li> Intuitive C++ class library

<li> cross-platform

<li> Integrated development tools

<li> High runtime performance

</ol>

</html>"

method: "<html>

<ul>

<li> Qt is a cross-platform application and UI framework. Using Qt, you can write web-enabled applications once and deploy them across desktop, mobile and embedded operating systems without rewriting the source code.

<li> Intuitive C++ class library

<li> Portability across desktop and embedded operating systems

<li> Integrated development tools with cross-platform IDE

<li> High runtime performance and small footprint on embedded

</ul>

</html>"

}

ListElement {

...

//以下省略

我们这里使用了 ListModel，里面包含着一个又一个的 ListElement 作为数据 Item。这里要注意的是，我们在 Model 里面定义的属性，在 Delegate 中是可以直接访问的到的。比如在 myDelegate 中有：

Text {

text: detailstr

//以下省略

于是这个文本就会显示 Model 中 detailstr 的具体内容。

此外，我们还可以使用 Javascript 对 Model 进行 append(), insert(), move(), remove() 等动态操作，从而实现对 Model 数据的增加删除和修改。

**使用数据库存储Model数据**

上面的Model数据是直接写到代码里的，但是在实际应用中我们通常会从某个地方读取数据，而不是直接写到代码里。比如我们可以把数据存到数据库里，程序启动的时候从数据库中读取数据，退出的时候把Model中的数据存放回数据库中。主要代码如下所示：

ListModel {

id: mymodel

Component.onCompleted: loadImageData()

Component.onDestruction: saveImageData()

**function** loadImageData() {

**var** db = openDatabaseSync("MyDB", "1.0", "My model SQL", 50000);

db.transaction(

**function**(tx) {

*// Create the database if it doesn't already exist*

tx.executeSql('CREATE TABLE IF NOT EXISTS Images(id INTEGER primary key, title TEXT, picture TEXT)');

**var** rs = tx.executeSql('SELECT \* FROM Images');

**var** index = 0;

**if** (rs.rows.length > 0) {

**var** index = 0;

**while** (index < rs.rows.length) {

**var** myItem = rs.rows.**item**(index);

mymodel.append( {

"id": myItem.id,

"title": myItem.title ,

"picture": myItem.picture });

index++;

}

} **else** {

mymodel.append( {

"id": 1,

"title": 'apple' ,

"picture": 'content/pics/apple.png' });

mymodel.append( {

"id": 2,

"title": 'Qt Quick!' ,

"picture": 'content/pics/Qt.png' });

}

}

)

}

**function** saveImageData() {

**var** db = openDatabaseSync("MyDB", "1.0", "My model SQL", 50000);

db.transaction(

**function**(tx) {

tx.executeSql('DROP TABLE Images');

tx.executeSql('CREATE TABLE IF NOT EXISTS Images(id INTEGER primary key, title TEXT, picture TEXT)');

**var** index = 0;

**while** (index < mymodel.count) {

**var** myItem = mymodel.get(index);

tx.executeSql('INSERT INTO Images VALUES(?,?,?)', [myItem.id, myItem.title, myItem.picture]);

index++;

}

}

)

}

}

动态添加数据是非常简单的，比如我们在 onClicked 事件中可以这样做：

onClicked: mymodel.append( { "title": 'Qt', "picture": 'content/pics/Qt.png' })

删除数据：

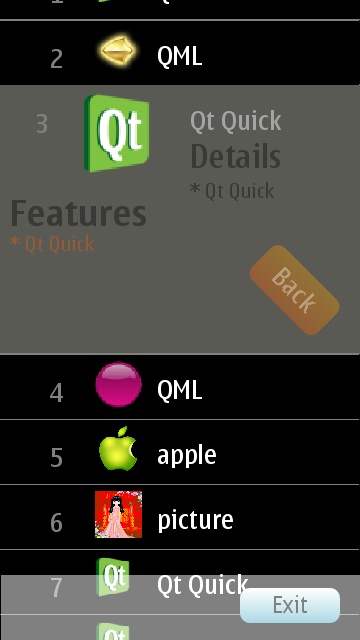
onClicked: mymodel.remove(listView.currentIndex)

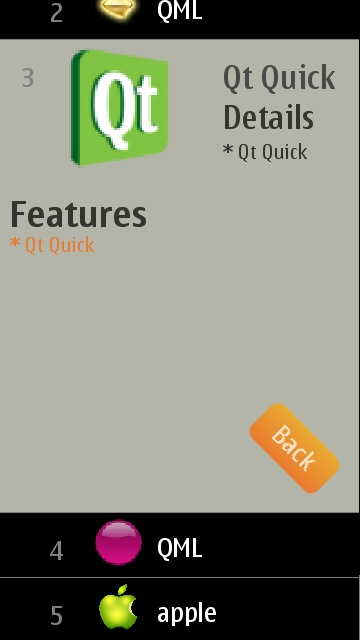
例程：[Media:ListModelViewWithDatabase.zip](http://www.developer.nokia.com/Community/Wiki/images/1/12/ListModelViewWithDatabase.zip?20110111074501)

**应用程序截图**

下面是程序在N8上面的运行效果：

[](http://www.developer.nokia.com/Community/Wiki/File:QMLListView1.JPG)

[](http://www.developer.nokia.com/Community/Wiki/File:QMLListView2.JPG)

[](http://www.developer.nokia.com/Community/Wiki/File:QMLListView3.JPG)