第 8 章 Qt Quick 元素布局

在 Qt Quick 中有两套与元素布局相关的类库，一套叫作 Item Positioner（定位器），一套叫作 Item Layout（布局）。其实我们前面还讲了一个锚布局，它通过 Item 的 anchors 属性实现，是 Qt Quick 中非常灵活的一种布局方式。

定位器包括 Row（行定位器）、Column（列定位器）、Grid（表格定位器）、Flow（流式定位器）。

布局管理器包括行布局（RowLayout）、列布局（ColumnLayout）、表格布局（GridLayout）。

我们先讲定位器，然后再讲布局管理器。

8.1 定位器

定位器是一种容器元素，专门用来管理界面中的其他元素，与传统的 Qt Widgets 中的布局管理器类似。使用定位器，你可以很方便地把众多的元素组织在一起，形成非常规则的界面效果。不过有一点要注意的是，定位器不会改变它管理的元素的大小，即便用户调整了界面尺寸，它也坚持不干涉孩子们的尺寸。这可能与你的期望不同，也与你使用 Qt Widgets 中的布局管理器的经验不同，不过如果你希望使用“自动根据界面尺寸变化调整孩子们的尺寸”这种特性，可以使用 Qt Quick 中的布局管理器，它们的行为与你的经验和期望完全一致。

常用的定位器元素有下列几种：

- Row
- Column
- Grid
- Flow

咱们一一来看。

8.1.1 Row

Row 沿着一行安置它的孩子们，在你需要水平放置一系列的 Item 时，它比锚布局更加方便。一旦你把一个 Item 交给 Row 来管理，那就不要再使用 Item 的 x、y、anchors 等属性

了，Row 会安排得妥妥的。

在一个 Row 内的 Item，可以使用 Positioner 附加属性来获知自己在 Row 中的详细位置信息。Positioner 有 index、isFirstItem、isLastItem 三个属性。

我们来看示例 row.qml：

```
import QtQuick 2.2

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    Row {
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        spacing: 4;

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }
    }
}
```

执行“qmlscene row.qml”命令，效果如图 8-1 所示。

因为 Row 本身是一个 Item，所以你可以使用锚布局来定位一个 Row，示例中这么做了，使用 anchors.left 和 anchors.bottom 属性把 Row 定位在界面的左下角。

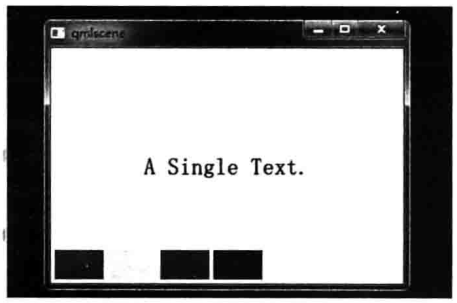

图 8-1　行定位器

Row 有一个 spacing 属性，用来指定它管理的 Item 之间的间隔。还有一个 layoutDirection 属性，可以指定布局方向，取值为 Qt.LeftToRight 时从左到右放置 Item，这是默认行为，取值为 Qt.RightToLeft 时从右向左放置 Item。

Row 还有 add、move、populate 三个 Transition 类型的属性，分别指定应用于 Item 添加、Item 移动、定位器初始化创建 Items 三种场景的过渡动画，等我们学习了动画相关的内容之后再来实验这些属性。

8.1.2　Colomun

Column 与 Row 类似，不过是在垂直方向上安排它的子 Item。在你需要垂直放置一系列的 Item 时，它比锚布局更加方便。

Column 本身也是一个 Item，可以使用 anchors 布局来决定它在父 Item 中的位置。Column 的 spacing 属性描述子 Item 之间的间隔。

看示例 column.qml：

```
import QtQuick 2.2

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    Column {
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        spacing: 4;
```

```
        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                        Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }
    }
}
```

代码与 row.qml 类似，不用解释了。图 8-2 是执行“qmlscene column.qml”命令后的效果。

图 8-2　列定位器

与 Row 类似，Column 内的子 Item 也可以使用 Positioner 附加属性。

Column 也有 add、move、populate 属性，指定添加、移动、定位器初始化 Item 时的动画效果。

8.1.3　Grid

Grid 在一个网格上安置它的子 Item，它会创建一个拥有很多单元格的网格，足够容纳它的所有子 Item。Grid 会从左到右、从上到下把它的子 Item 一个个塞到单元格里。Item 默认会被放在一个单元格左上角，即(0,0)位置。

你可以通过 rows 和 columns 属性设定表格的行、列数。如果不设置，默认只有 4 列，而行数则会根据实际的 Item 数量自动计算。rowSpacing 和 columnSpacing 指定行、列间距，单位是像素。

Grid 的 flow 属性描述表格的流模式，Grid.LeftToRight 是默认值，这种流模式从左到右一个挨一个放置 Item，一行放满再放下一行。flow 取值为 Grid.TopToBottom 时，从上到下一个挨一个放置 Item，一列放满再放下一列。

horizontalItemAlignment 和 verticalItemAlignment 指定单元格对齐方式。默认的单元格对齐方式和 layoutDirection 以及 flow 有关。

先看个简单的例子，grid.qml：

```
import QtQuick 2.2

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    Grid {
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        rows: 3;
        columns: 3;
        rowSpacing: 4;
        columnSpacing: 4;

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
```

```
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }
    }
}
```

为了看出 flow 属性取值不同时的效果，我特意将行、列数都设置为 3，创建了 7 个 ColorPicker 实例。在上面的示例代码中没有显式设置 flow 属性，它的默认值是 Grid.LeftToRight。图 8-3 是执行“qmlscene grid.qml”后的效果图。

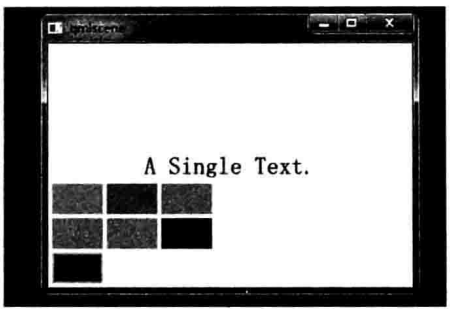

图 8-3　Grid 定位器，流模式从左到右

如果你在声明 Grid 对象时添加了“flow: Grid.TopToBottom;”语句，效果马上就会变成图 8-4 的样子。

图 8-4　Grid 定位器，流模式从上到下

示例还使用锚布局将 Grid 定位在界面左下角，老掉牙的老调了。

Grid 也有 add、move、populate 属性，指定添加、移动、定位器初始化 Item 时的动画效果。它还有其他的一些属性，请参考 Qt 帮助手册研究其细节。

8.1.4 Flow

Flow 其实和 Grid 类似，不同之处是它没有显式的行、列数，它会计算子 item 的尺寸，然后与自身尺寸比较，按需折行。Flow 的 flow 属性，默认取值 Flow.LeftToRight，从左到右安排 Item，直到 Flow 本身的宽度不能容纳新的子 Item 时折行；当 flow 取值 Flow.TopToBottom 时，从上到下安排 Item，直到 Flow 本身的高度不能容纳新的子 Item 时开始在下一列上安排 Item。

Flow 的 spacing 属性描述 Item 之间的间隔。与 Row 等定位器元素一样，Flow 也有 add、move、populate 三个与动画相关的属性。

看个示例，flow.qml：

```
import QtQuick 2.2

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.horizontalCenter: parent.horizontalCenter;
        anchors.top: parent.top;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    Flow {
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        width: 280;
        height: 130;
        spacing: 4;

        ColorPicker {
            width: 80;
            height: 20;
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            width: 100;
            height: 40;
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }
```

```
        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            width: 80;
            height: 25;
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            width: 35;
            height: 35;
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            width: 20;
            height: 80;
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(),
                           Math.random(), Math.random(), 1.0);
            onColorPicked: setTextColor(clr);
        }
    }
}
```

我改变了 ColorPicker 实例的大小，以便观察 Flow 布局的特点：根据自身的宽、高是否被 Item 超出而自动折行。

图 8-5 是 flow 为 LeftToRight（代码中未设置 flow 属性，默认值是 LeftToRight）时的效果图。

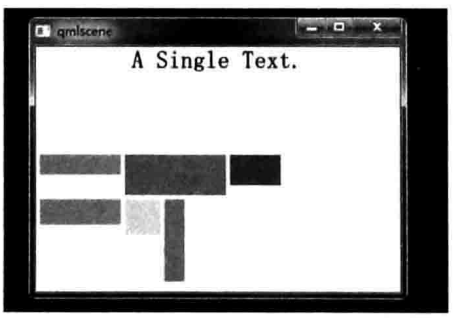

图 8-5　Flow 定位器，流模式从左到右

如果你在声明 Flow 对象时添加了“flow: Flow.TopToBottom;”语句，那么就可以看到图 8-6 所示的效果。

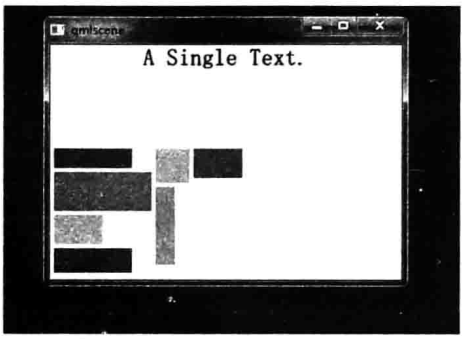

图 8-6　Flow 定位器，流模式从上到下

如你所见，效果大大不同。

其实可以把流布局想象成英文文字排版系统，一个 Item 对应一个单词，横版模式是从左到右，一行一行安排单词的位置，当接近一行的宽度时，如果下一个单词摆上去就会超出行宽，那就把这个单词放到下一行，继续排排排……；竖版模式也是类似的……也许你看过竖版书，很容易理解这件事情。

8.1.5　定位器嵌套

Qt Quick 中的定位器元素是可以嵌套的，比如通过 Row 和 Column 的嵌套就能够实现与 Grid 类似的效果。因为 Row、Column、Grid、Flow 等对象本身是从 Item 继承而来的，所以你在嵌套定位器时完全可以把它们当作一个普通的 Item 看待。

看下 nested_positioner.qml：

```
import QtQuick 2.2

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    Row {
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        spacing: 4;

        Column {
            spacing: 4;
```

```
            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }

            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }
        }

        Column {
            spacing: 4;
            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }

            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }
        }

        Column {
            spacing: 4;
            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }

            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }
        }
    }
}
```

我在一个 Row 内嵌套了 3 个 Column，实现了 2×3 的表格布局。执行“qmlscene nested_positioner. qml”命令，效果如图 8-7 所示。

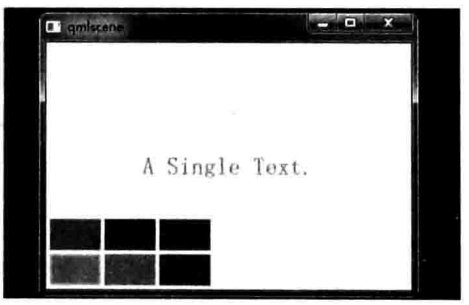

图 8-7　Row 嵌套 Column 实现 Grid

定位器嵌套时，比如放在 Row 内的 Column，其实对于 Row 来讲和其他非容器类的 Item 一样，没什么区别。

我们修改上面的代码，嵌套两个 Column，让其他两个 ColorPicker 实例与 Column 处在同一层级。改动的代码如下：

```
Rectangle {
    ...

    Row {
        ...

        //Column {
        //    spacing: 4;
            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }

            ColorPicker {
                color: Qt.rgba(Math.random(),
                               Math.random(), Math.random(), 1.0);
                onColorPicked: setTextColor(clr);
            }
        //}
    }
}
```

如你所见，我只是把最后一个 Column 对象声明给注释掉了，图 8-8 是运行效果图。

图 8-8　混合使用定位器与普通 Item

看起来效果与图 8-7 有所不同，什么原因？你懂的……

在介绍 Row 时（8.1.1 节），提到被定位器元素管理的 Item 不能使用 anchors 属性，嗯……定位器嵌套时也是一样的。

8.2　布局管理器

Qt Quick 中的布局管理器与 Qt Widgets 中的相似，它与定位器的不同之处在于：布局管理器会自动调整子 Item 的尺寸来适应界面大小的变化。

要使用布局管理器，需要引入 Layouts 模块，这样：

```
import QtQuick.Layouts 1.1
```

8.2.1　GridLayout

我们先说 Qt Quick 布局管理器中最复杂的 GridLayout 吧，因为 RowLayout 和 ColumnLayout 实际上可以看作是 GridLayout 的两个特例。

GridLayout 与 Qt C++中的 QGridLayout 功能类似，它在一个表格中安排它管理的 Item，如果用户调整界面尺寸，GridLayout 会自动重新调整 Item 的位置。

GridLayout 会根据 flow 属性来排列元素，这与 Grid 定位器类似，flow 属性的默认值是 GridLayout.LeftToRight，从左到右安排元素，一行结束再另起一行。而判定行结束的一个条件是 columns 属性，它指定一个 GridLayout 的列数。如果 flow 取值 GridLayout.TopToBottom，GridLayout 则从上到下安排元素，一列结束再另起一列。rows 属性指定 GridLayout 的行数，它将决定何时新开一列来排布剩余的元素。

看一个简单的示例 grid_layout.qml，它从 grid.qml 示例演变而来，仅仅做了三处改动：

```
import QtQuick 2.2
//[1]
import QtQuick.Layouts 1.1

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    GridLayout { //[2]
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        rows: 3;
        columns: 3;
        rowSpacing: 4;
        columnSpacing: 4;
        flow: GridLayout.TopToBottom; //[3]

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
```

```
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }
    }
}
```

如果你执行“qmlscene grid_layout.qml”命令，就会发现，出来的界面与 grid.qml 完全一样。其实我们略做修改，就可以看出 GridLayout 与 Grid 的不同之处。

GridLayout 所管理的 Item，可以使用下列附加属性（这正是布局管理器和定位器之关键不同点）：

- Layout.row
- Layout.column
- Layout.rowSpan
- Layout.columnSpan
- Layout.minimumWidth
- Layout.minimumHeight
- Layout.preferredWidth
- Layout.preferredHeight
- Layout.maximumWidth
- Layout.maximumHeight
- Layout.fillWidth
- Layout.fillHeight
- Layout.alignment

单单从名字上就可以看出这些附加属性的用途了。我们稍稍修改一下 grid_layout.qml，然后再来看效果。新的 grid_layout.qml 的内容如下：

```
import QtQuick 2.2
```

```
import QtQuick.Layouts 1.1

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    GridLayout {
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        width: 300; //[1]
        rows: 3;
        columns: 3;
        rowSpacing: 4;
        columnSpacing: 4;
        flow: GridLayout.TopToBottom;

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
            Layout.columnSpan: 3; //[2]
            Layout.rowSpan: 3;    //[2]
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
            Layout.fillWidth: true; //[3]
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }
```

```
        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }
    }
}
```

改动之处我已经通过注释标注出来了，有三处，第一处改动设置 GridLayout 的宽度为 300 像素，第二处改动设置第一个 ColorPicker 对象占用三行三列（单元格合并），第三处改动设置第二个 ColorPicker 对象填充 GridLayout 的所有可用宽度。执行“qmlscene grid_layout.qml”命令，效果如图 8-9 所示。

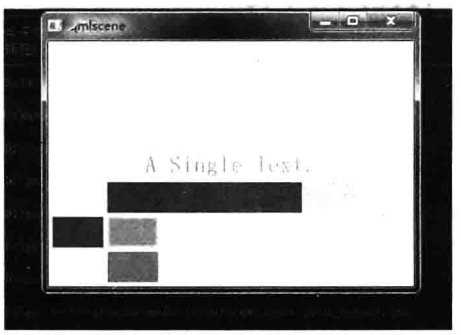

图 8-9　GridLayout 附加属性的效果

拿图 8-9 与 8-8 比较，立马就看出不同来了吧。

GridLayout 就简单说到这里，用到时如有不明之处，请查阅 Qt 帮助手册进一步学习。

8.2.2　RowLayout

RowLayout 可以看作是只有一行的 GridLayout，它的行为与 Row 类似，不同之处在于，它所管理的元素可以使用下列附加属性：

- Layout.minimumWidth
- Layout.minimumHeight
- Layout.preferredWidth
- Layout.preferredHeight
- Layout.maximumWidth
- Layout.maximumHeight
- Layout.fillWidth
- Layout.fillHeight
- Layout.alignment

我们以 row.qml 为基础来构造 row_layout.qml，代码如下：

```
import QtQuick 2.2
import QtQuick.Layouts 1.1  //[1]

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    RowLayout {  //[2]
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        anchors.right: parent.right;  //[3]
        anchors.rightMargin: 4;       //[3]
        spacing: 4;

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(), Math.random());
            onColorPicked: setTextColor(clr);
            Layout.fillWidth: true;  //[4]
        }
    }
}
```

执行“qmlscene row_layout.qml”命令，效果如图 8-10 所示。

如你所见，最后一个 ColorPicker 对象填充了 RowLayout 所有的剩余空间，这正是 Layout.fillWidth 附加属性的含义。

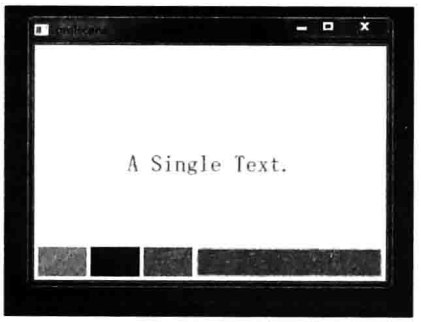

图 8-10　RowLayout

8.2.3　ColumnLayout

ColumnLayout 可以看作是只有一列的 GridLayout，它的行为与 Column 类似，不同之处在于，它所管理的元素可以使用下列附加属性：

- Layout.minimumWidth
- Layout.minimumHeight
- Layout.preferredWidth
- Layout.preferredHeight
- Layout.maximumWidth
- Layout.maximumHeight
- Layout.fillWidth
- Layout.fillHeight
- Layout.alignment

我们以 column.qml 为基础来构造 column_layout.qml，代码如下：

```
import QtQuick 2.2
import QtQuick.Layouts 1.1  //[1]

Rectangle {
    width: 360;
    height: 240;
    color: "#EEEEEE";
    id: rootItem;

    Text {
        id: centerText;
        text: "A Single Text.";
        anchors.centerIn: parent;
        font.pixelSize: 24;
        font.bold: true;
    }

    function setTextColor(clr){
        centerText.color = clr;
    }

    ColumnLayout { //[2]
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
```

```
        height: 180; //[3]
        spacing: 4;

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(),
                                                    Math.random());
            onColorPicked: setTextColor(clr);
            Layout.fillHeight: true; //[4]
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(),
                                                    Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(),
                                                     Math.random());
            onColorPicked: setTextColor(clr);
        }

        ColorPicker {
            color: Qt.rgba(Math.random(), Math.random(),
                                                    Math.random());
            onColorPicked: setTextColor(clr);
        }
    }
}
```

改动处已标出，执行“qmlscene column_layout.qml”命令，效果如图 8-11 所示。

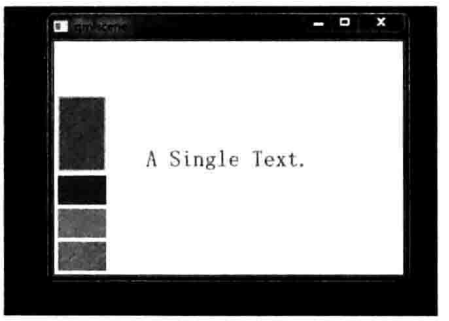

图 8-11　ColumnLayout

8.3　其他的布局方式

Qt Quick 中的 Item 都有 x、y、width、height 属性，其实你可以使用它们来定位 Item，这就是我们常用的绝对坐标布局。

Qt Quick 的 Model/View 框架其实也可以用来布局，比如 VisualItemModel 允许你把 QML Item 当作一个 Model，结合相应的 View，就可以达到特定的布局效果。你可以在学习 Model/View 后做做实验看看效果。