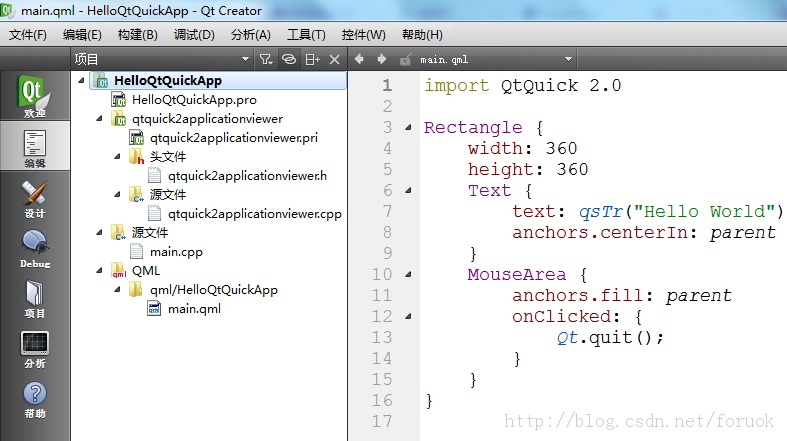
上一篇《[Qt Quick 之 Hello World 图文详解](http://blog.csdn.net/foruok/article/details/28850879" \t "_blank)》我们已经分别在电脑和 Android 手机上运行了第一个 Qt Quick 示例—— HelloQtQuickApp ，这篇呢，我们就来介绍 Qt Quick 编程的一些基本概念，为创建复杂的 Qt Quick 应用奠定基础。

    版权所有 foruok ，如需转载请注明来自博客 <http://blog.csdn.net/foruok> 。

    首先看一下《[Qt Quick 之 Hello World 图文详解](http://blog.csdn.net/foruok/article/details/28850879" \t "_blank)》中的 main.qml 文件：



    现在我们结合 main.qml 文件来讲解。

**import 语句**

    main.qml 文件第一行代码：import QtQuick 2.0 。这行代码引入了 QtQuick 模块， import 语句的作用与 C++ 中的 #include 类似，与 Java 中的 import 效果一样。不再啰嗦了。

**Qt Quick 基本元素**

    Qt Quick 作为 QML 的标准库，提供了很多基本元素和控件来帮助我们构建 Qt Quick 应用。如果拿 C++ 来比拟， QML 就相当于 C++ 语言本身，而 Qt Quick 相当于 STL 。好吧，你可能觉得有点驴头不对马嘴，没关系，有这么点儿意思就成。

**Rectangle**

    main.qml 的第三行代码，定义了一个 Rectangle 类型的对象作为 QML 文档的根对象。关于对象在 qml 文件中的描述，《[Qt on Android：QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543" \t "_blank)》一文中已经讲解，这里不再赘述。下面咱们看看 Rectangle 到底是什么。

    Rectangle 用来绘制一个填充矩形，可以带边框，也可以不带，可以使用纯色填充，也可以使用渐变色填充，甚至还可以不填充而只提供边框……

    Rectangle 有很多属性。

    width 用来指定宽， height 用来指定高，我们已经见识过了。

    color 属性可以指定填充颜色，而 gradient 属性则用来设置渐变色供填充使用，如果你同时指定了 color 和 gradient ，那么 gradient 生效；如果你设置 color 属性为 transparent ，那么就可以达到只绘制边框不填充的效果。

    border.width 指定边框的宽度， border.color 指定边框颜色。

    Rectangle 还可以绘制圆角矩形，你只要设置 radius 属性就行了。

    下面我们来看一个简单的示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. Rectangle {
2. width: 320;
3. height: 480;
4. color: "blue";
5. border.color: "#808080";
6. border.width: 2;
7. radius: 12;
8. }

    你可以修改 HelloQtQuickApp 的 main.qml 文件来查看效果，也可以建立一个新的工程。

    上面的 Rectangle 对象，我们

**颜色**

    关于颜色值， QML 中可以使用颜色名字，如 blue / red / green / transparent 等，也可以使用 "#RRGGBB" 或者 "#AARRGGBB" 来指定，还可以使用 Qt.rgba() / Qt.lighter() 等等方法来构造。详情请参考 Qt SDK 中 "QML Basic Type: color" 页面。

    color 类型有 r 、 g 、 b 、 a 四个属性，分别表示一个颜色值的 red 、 green 、 blue 、 alpha 四个成分。你可以这样使用它们：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. Text {
2. color: "red"
4. // prints "1 0 0 1"
5. Component.onCompleted: console.log(color.r, color.g, color.b, color.a)
6. }

**渐变色**

    QML 中渐变色的类型是 Gradient ，渐变色通过两个或多个颜色值来指定， QML 会自动在你指定的颜色之间插值，进行无缝填充。Gradient 使用 GradientStop 来指定一个颜色值和它的位置（取值在 0.0 与 1.0 之间）。

    好吧，无码不欢，快快看一个示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. Rectangle {
2. width: 100;
3. height: 100;
4. gradient: Gradient {
5. GradientStop { position: 0.0; color: "#202020"; }
6. GradientStop { position: 0.33; color: "blue"; }
7. GradientStop { position: 1.0; color: "#FFFFFF"; }
8. }
9. }

    Gradient 只能用来创建垂直方向的渐变，不过其它方向的，可以通过给 Rectangle 指定 rotation 属性来实现。下面是示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. Rectangle {
2. width: 100;
3. height: 100;
4. rotation: 90;
5. gradient: Gradient {
6. GradientStop { position: 0.0; color: "#202020"; }
7. GradientStop { position: 1.0; color: "#A0A0A0"; }
8. }
9. }

    刚刚我们使用了 rotation 属性，其实它来自 Rectangle 的父类 Item 。

**Item**

    Item 是 Qt Quick 中所有可视元素的基类，虽然它自己什么也不绘制，但是它定义了绘制图元所需要的大部分通用属性，比如 x 、 y 、 width 、 height 、 锚定（ anchoring ）和按键处理。

    Item 的属性特别多，除了前面提到的，还有 scale / smooth / anchors / antialiasing / enabled / visible / state / states / children 等等，详情参考 Qt 帮助文档。

    你可以使用 Item 来分组其它的可视图元。如：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
6. Item {
7. id: gradientGroup;
8. Rectangle {
9. x: 20;
10. y: 20;
11. width: 120;
12. height: 120;
13. gradient: Gradient {
14. GradientStop { position: 0.0; color: "#202020"; }
15. GradientStop { position: 1.0; color: "#A0A0A0"; }
16. }
17. }
19. Rectangle {
20. x: 160;
21. y: 20;
22. width: 120;
23. height: 120;
24. rotation: 90;
25. gradient: Gradient {
26. GradientStop { position: 0.0; color: "#202020"; }
27. GradientStop { position: 1.0; color: "#A0A0A0"; }
28. }
29. }
30. }
32. Component.onCompleted: {
33. console.log("visible children: " ,gradientGroup.visibleChildren.length);
34. console.log("visible children: " ,gradientGroup.children.length);
35. **for**(**var** i = 0; i < gradientGroup.children.length; i++){
36. console.log("child " , i, " x = ", gradientGroup.children[i].x);
37. }
38. }
39. }

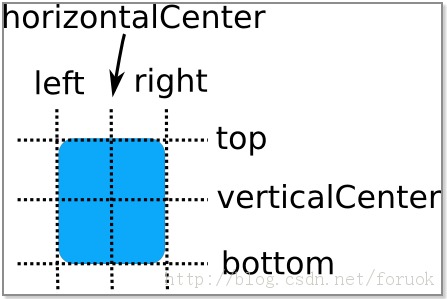
    分组后可以通过 Item 的 children 或 visibleChildren 属性来访问孩子元素，如上面的代码所示。

    另外你可能注意到了， x 、 y 、 width 、 height 四个属性结合起来，可以完成 Qt Quick 应用的界面布局，不过这种采用绝对坐标的方式来布局，不太容易适应多种多样的移动设备分辨率。而如果你看了《》，可能会注意到示例代码中多次出现的 anchors 属性，它 Item 的属性，是 Qt Quick 引入的一种新的布局方式。

**使用 anchors 进行界面布局**

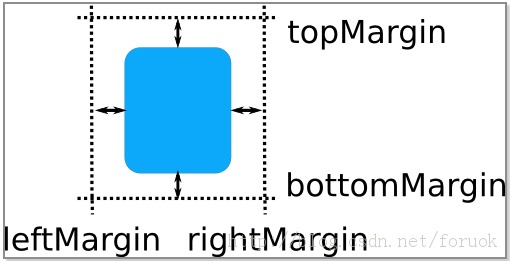
    anchors 提供了一种方式，让你可以通过指定一个元素与其它元素的关系来确定元素在界面中的位置。

    你可以想象一下，每个 item 都有 7 条不可见的锚线：左（left）、水平中心（horizontalCenter）、上（top）、下（bottom）、右（right）、垂直中心（verticalCenter）、基线（baseline）。看下图就明白了：



    在上图中，没有标注基线，基线是用于定位文本的，你可以想象一行文字端坐基线的情景。对于没有文本的图元，baseline 和 top 一致。

    使用 anchors 布局时，除了对齐锚线，还可以在指定上（topMargin）、下（bottomMargin）、左（leftMargin）、右（rightMargin）四个边的留白。看个图就明白了：



    好了，基础知识介绍完毕，可以看一些例子了。

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
7. Rectangle {
8. id: rect1;
9. anchors.left: parent.left;
10. anchors.leftMargin: 20;
11. anchors.top: parent.top;
12. anchors.topMargin: 20;
13. width: 120;
14. height: 120;
15. gradient: Gradient {
16. GradientStop { position: 0.0; color: "#202020"; }
17. GradientStop { position: 1.0; color: "#A0A0A0"; }
18. }
19. }
21. Rectangle {
22. anchors.left: rect1.right;
23. anchors.leftMargin: 20;
24. anchors.top: rect1.top;
25. width: 120;
26. height: 120;
27. rotation: 90;
28. gradient: Gradient {
29. GradientStop { position: 0.0; color: "#202020"; }
30. GradientStop { position: 1.0; color: "#A0A0A0"; }
31. }
32. }
33. }

    上面的代码运行后与之前使用 Item 分组的示例代码（绝对坐标布局）效果一样。这里的第二个矩形的左边从第一个矩形的右边开始、顶部向第一个矩形的顶部对齐。而对第一个矩形的引用，是通过的 id 属性来完成的，请参看《[Qt on Android：QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543" \t "_blank)》。

    Item 的 anchors 属性，除了上面介绍的，还有一些，如 centerIn 表示将一个 item 居中放置到另一个 item 内； fill 表示充满某个 item ……更多请参考 Item 类的文档。这里再举个使用 centerIn 和 fill 的示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
7. Rectangle {
8. color: "blue";
9. anchors.fill: parent;
10. border.width: 6;
11. border.color: "#888888";
13. Rectangle {
14. anchors.centerIn: parent;
15. width: 120;
16. height: 120;
17. radius: 8;
18. border.width: 2;
19. border.color: "black";
20. antialiasing: **true**;
21. color: "red";
22. }
23. }
24. }

**Z 序 与 透明度**

    Item 除了 x 、 y 属性，其实还有一个 z 属性，用来指定图元在场景中的 Z 序。 z 属性的类型是 real ，数值越小，图元就越垫底（远离我们），数值越大，图元就越靠近我们。

    Item 的属性 opacity 可以指定一个图元的透明度，取值在 0.0 到 1.0 之间。

    结合 Z 序和透明度，有时可以达到不错的效果。下面是一个简单的示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
7. Rectangle {
8. x: 20;
9. y: 20;
10. width: 150;
11. height: 100;
12. color: "#606080";
13. z: 0.5;
14. }
16. Rectangle {
17. width: 100;
18. height: 100;
19. anchors.centerIn: parent;
20. color: "#a0c080";
21. z: 1;
22. opacity: 0.6;
23. }
24. }

    除了视觉效果，有时我们也需要安排图元在场景中的 Z 序。比如一个图片浏览器，可能在加载图片时要显示一个 loading 图标，这个图标要显示在图片之上，此时就可以设置 loading 图元的 Z 序大于图片元素的 Z 序。

**按键处理**

    前面提到 Item 可以处理案件，所有从 Item 继承的元素都可以处理按键，比如 Rectangle ，比如 Button 。这点我们在《[Qt on Android：QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543" \t "_blank)》一文中介绍附加属性时已经提到。

    Item 通过附加属性 Keys 来处理按键。Keys 对象是 Qt Quick 提供的，专门供 Item 处理按键事件的对象。它定义了很多针对特定按键的信号，比如 onReturnPressed ，还定义了更为普通的 onPressed 和 onReleased 信号，一般地，你可以使用这两个信号来处理按键（请对照 Qt C++ 中的 keyPressEvent 和 keyReleaseEvent 来理解）。它们有一个名字是 event 的 KeyEvent 参数，包含了按键的详细信息。如果一个按键被处理， event.accepted 应该被设置为 true 以免它被继续传递。

    这里举一个简单的例子，检测到 Escape 和 Back 键时退出应用，检测到数字键，就通过 Text 来显示对应的数字。代码如下：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
6. color: "#c0c0c0";
7. focus: **true**;
8. Keys.enabled: **true**;
9. Keys.onEscapePressed: Qt.quit();
10. Keys.onBackPressed: Qt.quit();
11. Keys.onPressed: {
12. **switch**(event.key){
13. **case** Qt.Key\_0:
14. **case** Qt.Key\_1:
15. **case** Qt.Key\_2:
16. **case** Qt.Key\_3:
17. **case** Qt.Key\_4:
18. **case** Qt.Key\_5:
19. **case** Qt.Key\_6:
20. **case** Qt.Key\_7:
21. **case** Qt.Key\_8:
22. **case** Qt.Key\_9:
23. keyView.text = event.key - Qt.Key\_0;
24. **break**;
25. }
26. }
28. Text {
29. id: keyView;
30. font.bold: **true**;
31. font.pixelSize: 24;
32. text: qsTr("text");
33. anchors.centerIn: parent;
34. }
35. }

    示例中用到了 onPressed / onEscapePressed / onBackPressed 三个附加信号处理器，其中 onPressed 信号的参数是 event ，包含了按键信息，程序中使用 switch 语句与 Qt 对象的枚举值比较来过滤我们关注的按键。

    Item 还有很多的属性，不再一一演示用法，请移步 Qt 帮助进一步了解。

    你肯定注意到了，上面的示例使用了 Text 这个对象，接下来我们就介绍它。

**Text**

    Text 元素可以显示纯文本或者富文本（使用 HTML 标记修饰的文本）。它有 font / text / color / elide / textFormat / wrapMode / horizontalAlignment / verticalAlignment 等等属性，你可以通过这些属性来决定 Text 元素如何显示文本。

    当不指定 textFormat 属性时， Text 元素默认使用 Text.AutoText ，它会自动检测文本是纯文本还是富文本，如果你明确知道要显示的是富文本，可以显式指定 textFormat 属性。

    下面是一个简单示例，显示蓝色的问题，在单词分界处断行：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
6. Text {
7. width: 150;
8. height: 100;
9. wrapMode: Text.WordWrap;
10. font.bold: **true**;
11. font.pixelSize: 24;
12. font.underline: **true**;
13. text: "Hello Blue Text";
14. anchors.centerIn: parent;
15. color: "blue";
16. }
17. }

    下面的例子仅仅把 "Text" 字样以蓝色显示：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
6. Text {
7. width: 150;
8. height: 100;
9. wrapMode: Text.WordWrap;
10. font.bold: **true**;
11. font.pixelSize: 24;
12. font.underline: **true**;
13. text: "Hello Blue <font color=\"blue\">Text</font>";
14. anchors.centerIn: parent;
15. }
16. }

    Text 元素的 style 属性提供了几种文字风格，Text.Outline 、 Text.Raised 、 Text.Sunken ，可以使文字有点儿特别的效果。而 styleColor 属性可以和 style 配合使用（如果没有指定 style ，则 styleColor 不生效），比如 style 为 Text.Outline 时，styleColor 就是文字轮廓的颜色。看个简单的示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Rectangle {
4. width: 300;
5. height: 200;
6. Text {
7. id: normal;
8. anchors.left: parent.left;
9. anchors.leftMargin: 20;
10. anchors.top: parent.top;
11. anchors.topMargin: 20;
12. font.pointSize: 24;
13. text: "Normal Text";
14. }
15. Text {
16. id: raised;
17. anchors.left: normal.left;
18. anchors.top: normal.bottom;
19. anchors.topMargin: 4;
20. font.pointSize: 24;
21. text: "Raised Text";
22. style: Text.Raised;
23. styleColor: "#AAAAAA" ;
24. }
25. Text {
26. id: outline;
27. anchors.left: normal.left;
28. anchors.top: raised.bottom;
29. anchors.topMargin: 4;
30. font.pointSize: 24;
31. text: "Outline Text";
32. style: Text.Outline;
33. styleColor: "red";
34. }
35. Text {
36. anchors.left: normal.left;
37. anchors.top: outline.bottom;
38. anchors.topMargin: 4;
39. font.pointSize: 24;
40. text: "Sunken Text";
41. style: Text.Sunken;
42. styleColor: "#A00000";
43. }
44. }

    这个示例除了用到 Text 元素，还使用 anchors 来完成界面布局。

    Text 就介绍到这里，下面看 Button 。

**Button**

    按钮可能是 GUI 应用中最常用的控件了。 QML 中的 Button 和 QPushButton 类似，用户点击按钮会触发一个 clicked() 信号，在 QML 文档中可以为 clicked() 指定信号处理器，响应用户操作。

    要使用 Button ，需要引入 import QtQuick.Controls 1.1 。

    先看一个简单的示例，点击按钮，退出应用。代码如下：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 300;
6. height: 200;
7. Button {
8. anchors.centerIn: parent;
9. text: "Quit";
10. onClicked: Qt.quit();
11. }
12. }

    你可以运行它看看效果。

    现在我们再来看 Button 都有哪些属性。

    text 属性指定按钮文字，见过了。

    checkable 属性设置 Button 是否可选。如果 Button 可选 checked 属性则保存 Button 选中状态。其实我一直没用过这个属性……

    iconName 属性指定图标的名字，如果平台的图标主题中存在该名字对应的资源， Button 就可以加载并显示它。iconSource 则通过 URL 的方式来指定 icon 的位置。iconName 属性的优先级高于 iconSource 。

    isDefault 属性指定按钮是否为默认按钮，如果是默认的，用户按 Enter 键就会触发按钮的 clicked() 信号。

    pressed 属性保存了按钮的按下状态。

    menu 属性，允许你给按钮设置一个菜单（此时按钮可能会出现一个小小的下拉箭头），用户点击按钮时会弹出菜单。默认是 null 。

    action 属性，允许设定按钮的 action ，action 可以定义按钮的 checked ， text ，tooltip 等属性。默认是 null 。

    activeFocusOnPress ，指定当用户按下按钮时是否获取焦点，默认是 false 。

    style 属性用来定制按钮的风格，与它配套的有一个 ButtonStyle 类，允许你定制按钮的背景。

    其实 Button 比较简单好用，我不准备再啰嗦下去了，咱再看下 style 的使用就结束对 Button 的介绍。

**ButtonStyle**

    要使用 ButtonStyle 需要引入 QtQuick.Controls.Styles 1.1 。

    ButtonStyle 类有 background 、 control 、 label 三个属性。我们通过重写 background 来定制一个按钮。 control 属性指向应用 ButtonStyle 的按钮对象，你可以用它访问按钮的各种状态。 label 属性代表按钮的文本，如果你看它不顺眼，也可以替换它。

    background 实际是一个 Component 对象， Component（组件） 的概念我们回头讲。这里我们简单的使用 Rectangle 来定制按钮的背景。看下面的示例：

**[cpp]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. import QtQuick 2.0
2. import QtQuick.Controls 1.1
3. import QtQuick.Controls.Styles 1.1
5. Rectangle {
6. width: 300;
7. height: 200;
8. Button {
9. text: "Quit";
10. anchors.centerIn: parent;
11. style: ButtonStyle {
12. background: Rectangle {
13. implicitWidth: 70;
14. implicitHeight: 25;
15. border.width: control.pressed ? 2 : 1;
16. border.color: (control.hovered || control.pressed) ? "green" : "#888888";
17. }
18. }
19. }
20. }

    我通过给 style 对象指定一个 ButtonStyle 对象来定制 Button 的风格。这个就地实现的 ButtonStyle 对象，重写了 background 属性，通过 Rectangle 对象来定义按钮背景。我定义了背景的建议宽度和高度，根据按钮的 pressed 属性（ control 是实际按钮的引用 ）来设置背景矩形的边框粗细，而边框颜色则随着按钮的 hovered 和 pressed 属性来变化。

    最终的效果是这样的：当鼠标悬停在按钮上时，边框颜色为绿色；当鼠标按下时，边框变粗且颜色为绿色。

    对于 ButtonStyle ，如果有多个按钮同时用到，上面的方式就有点繁琐了，可以像下面这样使用：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
3. **import** QtQuick.Controls.Styles 1.1
5. Rectangle {
6. width: 300;
7. height: 200;
9. Component{
10. id: btnStyle;
11. ButtonStyle {
12. background: Rectangle {
13. implicitWidth: 70;
14. implicitHeight: 25;
15. color: "#DDDDDD";
16. border.width: control.pressed ? 2 : 1;
17. border.color: (control.hovered || control.pressed) ? "green" : "#888888";
18. }
19. }
20. }
22. Button {
23. id: openButton;
24. text: "Open";
25. anchors.left: parent.left;
26. anchors.leftMargin: 10;
27. anchors.bottom: parent.bottom;
28. anchors.bottomMargin: 10;
29. style: btnStyle;
30. }
32. Button {
33. text: "Quit";
34. anchors.left: openButton.right;
35. anchors.leftMargin: 6;
36. anchors.bottom: openButton.bottom;
37. style: btnStyle;
38. }
39. }

    这次我们定义了一个组件，设置其 id 属性的值为 btnStyle ，在 Button 中设定 style 属性时直接使用 btnStyle 。

    好啦， ButtonStyle 就介绍到这里。下面该介绍 Image 了。

**Image**

    Image 可以显示一个图片，只要是 Qt 支持的，比如 JPG 、 PNG 、 BMP 、 GIF 、 SVG 等都可以显示。它只能显示静态图片，对于 GIF 等格式，只会把第一帧显示出来。如果你要显示动画，可以使用 AnimateSprite 或者 AnimateImage 。

    Image 的 width 和 height 属性用来设定图元的大小，如果你没有设置它们，那么 Image 会使用图片本身的尺寸。如果你设置了 width 和 height ，那么图片就可能会拉伸来适应这个尺寸。此时 fillMode 属性可以设置图片的填充模式，它支持 Image.Stretch（拉伸） 、 Image.PreserveAspectFit（等比缩放） 、 Image.PreserveAspectCrop（等比缩放，最大化填充 Image ，必要时裁剪图片） 、 Image.Tile（在水平和垂直两个方向平铺，就像贴瓷砖那样） 、 Image.TileVertically（垂直平铺） 、 Image.TileHorizontally（水平平铺） 、 Image.Pad（保持图片原样不作变换） 等模式。

**Image 默认会阻塞式的加载图片，如果要显示的图片很小，没什么问题，如果分辨率很高，麻烦就来了。此时你可以设置 asynchronous 属性为 true 来开启异步加载模式，这种模式下 Image 使用一个线程来加载图片，而你可以在界面上显示一个等待图标之类的小玩意儿来告诉用户它需要等会儿。然后当 status（枚举值） 的值为 Image.Ready 时再隐藏加载等候界面。**

    比较强悍的是， Image 支持从网络加载图片。它的 source 属性类型是 url ，可以接受 Qt 支持的任意一种网络协议，比如 http 、 ftp 等。而当 Image 识别到你提供的 source 是网络资源时，会自动启用异步加载模式。此时呢，Image 的 progress（取值范围 0.0 至 1.0 ），status（枚举值）都会适时更新，你可以根据它们判断何时结束你的加载等候提示界面。

    算，先到这儿，看看怎么用吧。下面是最简的示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
3. Image {
4. source: "images/yourimage.png"
5. }

    source 替换为一个实际存在的图片路径就可以看到效果。

**显示网络图片**

    下面是一个稍微复杂点儿的示例，显示网络上的图片，在下载和加载前显示一个转圈圈的 Loading 图标，图片加载成功后隐藏 Loading 图标，如果加载出错，则显示一个简单的错误消息。看代码：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 480;
6. height: 320;
7. color: "#121212";
9. BusyIndicator {
10. id: busy;
11. running: **true**;
12. anchors.centerIn: parent;
13. z: 2;
14. }
16. Label {
17. id: stateLabel;
18. visible: **false**;
19. anchors.centerIn: parent;
20. z: 3;
21. }
23. Image {
24. id: imageViewer;
25. asynchronous: **true**;
26. cache: **false**;
27. anchors.fill: parent;
28. fillMode: Image.PreserveAspectFit;
29. onStatusChanged: {
30. **if** (imageViewer.status === Image.Loading) {
31. busy.running = **true**;
32. stateLabel.visible = **false**;
33. }
34. **else** **if**(imageViewer.status === Image.Ready){
35. busy.running = **false**;
36. }
37. **else** **if**(imageViewer.status === Image.Error){
38. busy.running = **false**;
39. stateLabel.visible = **true**;
40. stateLabel.text = "ERROR";
41. }
42. }
43. }
45. Component.onCompleted: {
46. imageViewer.source = "http://image.cuncunle.com/Images/EditorImages/2013/01/01/19/20130001194920468.JPG";
47. }
48. }

    图片资源是我从网络上搜索到的，仅仅用于演示程序，如有版权问题请提示我修改。

    Image 对象，设置了 asynchronous 属性为 true，不过对于网络资源 Image 默认异步加载，这个属性不起作用，只有你想异步加载本地资源时才需要设置它。 cache 属性设置为 false ，告诉 Image 不用缓存图片。 fillMode 属性我设置了等比缩放模式。

    onStatusChanged 是信号处理器，Image 的 status 属性变化时会发射 statusChanged() 信号。之前在《[QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543" \t "_blank)》中介绍信号处理器时我们知道，信号处理器遵循 on{Signal} 语法，所以我们这里的名字是 onStatusChanged 。在信号处理器的代码块中，我通过 Image 对象的 id 访问它的 status 属性，根据不同的状态来更新界面。

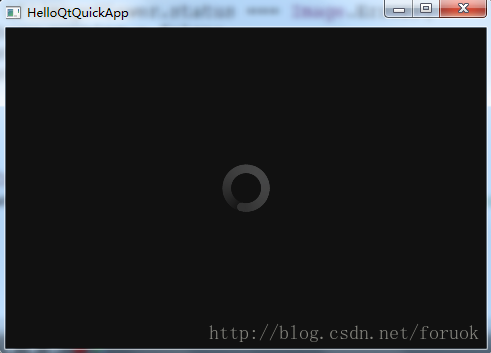
    可能你会奇怪，在 Qt 帮助中， Image 类的参考手册里没有明确提到 statusChanged 信号。其实呢，还有很多的信号， Qt 的文档里都没有提到，呜呜，怎么办呢？教你个诀窍，去 SDK 头文件中找，比如 Image 的头文件是 Qt5.2.0\5.2.0\mingw48\_32\include\QtQuick\5.2.0\QtQuick\private\qquickimage\_p.h ，阅读这个头文件你会看到 QML 中的 Image 对应的 Qt C++ 中的 QQuickImage 类，而 QQuickImage 的父类是 QQuickImageBase ，QQuickImageBase 的类声明在文件 Qt5.2.0\5.2.0\mingw48\_32\include\QtQuick\5.2.0\QtQuick\private\qquickimagebase\_p.h 中，找到这里就找到 status 属性的真身了，看下面的代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

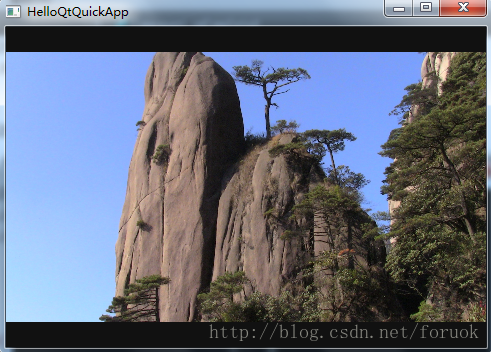
1. Q\_PROPERTY(Status status READ status NOTIFY statusChanged)

    Q\_PROPERTY 宏用来定义 QObject 及其派生类的属性，这样定义的属性可以在 QML 中访问。上面的语句定义了只读的 status 属性并且指明当属性变化时发送 statusChanged 信号。噢耶，K.O. ！

    现在来看运行效果图吧（我偷了个懒，都是直接修改[HelloQtQuickApp](http://blog.csdn.net/foruok/article/details/28850879" \t "_blank) 的 main.qml 文件来看各种示例的效果）。下图是加载过程：



    我设置了作为 QML 文档根元素的 Rectangle 对象的填充颜色为 "#121212"，所以背景是接近黑色的。下图是图片加载后的效果：



    怎么样，还不错吧，等比缩放模式生效了。

    Qt Quick 是如此方便，以至于我不得不爱它！你看嘛，就不到 50 行代码就可以实现一个基于网络的图片浏览器……

    说说这个示例中出现的新内容：BusyIndicator 。

**BusyIndicator**

    BusyIndicator 用来显示一个等待图元，在进行一些耗时操作时你可以使用它来缓解用户的焦躁情绪。

    BusyIndicator 的 running 属性是个布尔值， 为 true 时显示。 style 属性允许你定制 BusyIndicator 。默认的效果就是前面图示的那种，一个转圈圈的动画。

    至于 BusyIndicator 的使用，下面是显示网络图片示例的代码，再温习下：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/28859415)[在CODE上查看代码片](https://code.csdn.net/snippets/383274)

1. BusyIndicator {
2. id: busy;
3. running: **true**;
4. anchors.centerIn: parent;
5. z: 2;
6. }

    虽然 BusyIndicator 只有 running 和 style 两个属性，但它的祖先们有很多属性，上面用到的 anchors 和 z ，都是从 Item 继承来的属性，可以直接使用。

    好嘛，总算到一阶段，可以和简单教程说再见了。

    版权所有 foruok ，如需转载请注明来自博客 <http://blog.csdn.net/foruok> 。

    回顾一下前几篇：

* [Qt Quick 简介](http://blog.csdn.net/foruok/article/details/28620525)
* [QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543)
* [Qt Quick 之 Hello World 图文详解](http://blog.csdn.net/foruok/article/details/28850879)

    如果你有耐心看到这里，我想你肯定可以根据已经介绍的内容来完成一些比较复杂的 Qt Quick 应用了。恭喜你！