第3章 QML语言基础

要使用 Qt Quick，必须知道怎样使用 QML 语言来编写 QML 应用。你可以联想 C++与 STL 的关系来理解 QML 与 Qt Quick 的关系。

QML 实现并扩展了 ECMAScript，是一种说明性语言，用来描述基于 Qt 对象系统的用户界面。QML 提供了高可读性的、声明式的、类 CSS 的语法，支持结合动态属性绑定的 ECMAScript 表达式。

Qt Quick 模块是开发 QML 应用的标准库，提供了使用 QML 创建用户界面所需的一切东西，包括可视化类型、交互类型、动画、模型与视图、粒子特效与着色器等。

在介绍 Qt Quick 之前，我们先要介绍 QML 的语法，了解怎样撰写 QML 文档。

本章希望能让你快速了解 QML 语言的语法，使你能够飞速进入 Qt Quick 的世界中体验一番，所以笔者直接将 QML 及 Qt Quick 最基础的概念拉出来秀给你看。如果你愿意先了解 QML 最基本的 ECMAScript，则可以先到第 5 章（它的篇幅有点长嗳）看看再回来。

3.1 对象

QML 文件的后缀是.qml，其实就是文本文件。下面是一个简单的 QML 文件：

```
import QtQuick 2.2

Rectangle {
    width: 320;
    height: 480;

    Image {
        source: "images/IMG_001.jpg";
        anchors.centerIn: parent;
    }
}
```

这个简单的 QML 文件的开始是 import 语句，如 import QtQuick 2.2 这句，会引入 QtQuick 2.2 模块。哇，真是废话！接着废话吧。import 和 C++中的#include 类似，与 Java 中的 import 效果一样，与 JavaScript 中的……唐僧了，打住。

现在我们可以简单地将对象理解为类的实例，如果你有 C++或其他面向对象语言的基础，

那么对此应当了如指掌。

Rectangle{}语句，定义了一个类型为 Rectangle 的对象。对象要用一对花括号来描述，花括号前要写上对象的类型名字（类名）。就这么简单！

示例 QML 文档中有两个对象：一个是 Rectangle，一个是 Image。至于 Rectangle 和 Image，它们都是 Qt Quick 提供的内置类，先不必深究其含义。

在花括号之间，是对象的属性初始化语句（还可以有其他的，后面再说），形如“property: value”。如你所见，示例代码初始化了 Rectangle 对象的 width、height 等属性。

属性初始化语句可以分行书写，此时语句后可以不要“;”号，不过笔者建议 C++程序员都加上“;”，这会避免你患上精神分裂症。当然，也可以把多个属性初始化语句写在一行内，它们之间必须以“;”分隔，如下所示。

```
Rectangle {
    width: 320; height: 480; color: "#121212";
}
```

这种形式不利于阅读，笔者强烈建议你不要这么干！除非有代码以外的原因，比如排版需要，比如老板觉得你的代码行数太多……

3.2　表达式

QML 支持 ECMAScript 表达式。你可以这样初始化 Rectangle 对象的宽、高属性：

```
Rectangle {
    width: 23*10;
    height: 6*80;
    color: "#121212";
}
```

这里的 23*10 之类毫无意义的表达式只是示意啊，你在实际项目中可别这么写，这种行为往不好听里说，有点儿脑残……当然我也可以举一个有意义的示例：

```
Button {
    text: "Quit";
    style: ButtonStyle {
        background: Rectangle {
            implicitWidth: 70;
            implicitHeight: 25;
            border.width: control.activeFocus ? 2 : 1;
        }
    }
}
```

在这个示例代码片段中我定制了按钮风格，指定一个 Rectangle 对象来作为按钮的背景，在按钮有焦点时边框宽度为 2，没有焦点时宽度为 1。语句“border.width: control.activeFocus ? 2 : 1”使用了 ECMAScript 的“?:”三元运算符（C++中貌似也有……）。

另外，慧眼如你，可能已经注意到，上面的表达式中使用了“control.activeFocus”，没错，在表达式中可以引用其他对象及其属性。当你这么做的时候，待赋值的属性就和你所引用的对象的那个属性建立了关联，当被引用属性发生变化时，表达式的值会重新计算，而待赋值的属性也会变化。

也许你心中已经有了疑问：如何引用一个对象呢？答案是：通过对象的 id 值来引用一个对象。看这里：

```
Rectangle {
    width: 320;
    height: 480;

    Button {
        id: openFile;
        text: "打开";
        anchors.left:  parent.left;
        anchors.leftMargin: 6;
        anchors.top: parent.top;
        anchors.topMargin: 6;
    }

    Button {
        id: quit;
        text: "退出";
        anchors.left: openFile.right;
        anchors.leftMargin: 4;
        anchors.bottom: openFile.bottom;
    }
}
```

上面的示例代码片段，退出按钮使用 id（openFile）引用了打开按钮。在一个 QML 文档中，每个对象都可以指定一个唯一的 id，在代码中可以通过这个 id 来引用某个对象，访问其属性、方法，这个 id，就像是 C++中的一个具有文件作用域的全局变量一样。

有人要问了，anchors 是什么东西……先别管它，后面会讲到。

3.3 注释

在 QML 中，注释与 C++中一样，单行以“//”开始，多行以“/*”开始、以“*/”结束。

注释是不被执行的，可添加注释对代码进行解释或者提高其可读性。注释同样还可用于防止代码执行，这对跟踪问题是非常有用的。

使用注释的示例 QML：

```
/*
 * the root element of QML
 */
Rectangle {
    width: 320;
    height: 480;

    Button {
        id: quit;
        text: "退出";
        //use anchors to layout
        anchors.left: openFile.right;
        anchors.leftMargin: 4;
        anchors.bottom: openFile.bottom;
        //set z-order
        z: 1;
    }
}
```

3.4　属性

其实，QML 中的属性，对应于我们非常熟悉的 C++、Java 中类的成员变量……

这节是本章最复杂的，因为 QML 的世界围绕着对象属性展开，我们不得不做相对全面的介绍。

3.4.1　属性命名

属性名的首字母一般以小写开始，如我们看烦了的 width 属性。

如果属性名以多个单词表示，那么第二个及以后的单词，首字母大写。这也是驼峰命名法。比如 Rectangle 从 Item 继承而来的属性 implicitWidth。

3.4.2　属性的类型

可以在 QML 文档中使用的类型大概有三类：

- 由 QML 语言本身提供的类型。
- 由 QML 模块（比如 Qt Quick）提供的类型。
- 导出到 QML 环境中的 C++类型。

我们先看 QML 语言提供的基本类型。

（1）基本类型

QML 支持的基本类型包括 int、real、double、bool、string、color、list、font 等。这些基本类型有些是和 ECMAScript 语言的基本类型对应的，比如 string。

还是之前的示例，修改了一下，通过注释标注了属性类型：

```
Rectangle {
    width: 320; //int
    height: 480;

    Button {
        id: quit;
        text: "退出"; //string
        anchors.left: openFile.right;
        anchors.leftMargin: 4;
        anchors.bottom: openFile.bottom;
        z: 1.5; //real
        visible: false; //bool
    }
}
```

注意，QML 中对象的属性是有类型安全检测的，也就是说，你只能指定与属性类型匹配的值，否则会报错。

请使用 Qt 帮助的索引模式，以“qml basic types”为关键字检索，找到 QML Basic Types 页面来查看完整的类型列表和每种类型的详情。

Qt 的 QML 模块还为 QML 引入了很多 Qt 相关的类型，如 Qt、QtObject、Component、Connections、Binding 等，请使用 Qt 帮助检索“qt qml qml types”来了解。

（2）id 属性

前面在介绍表达式时提到了 id 属性，这里展开描述一下。

一个对象的 id 属性是唯一的，在同一个 QML 文件中不同对象的 id 属性的值不能重复。当给一个对象指定了 id 后，就可以在其他对象或脚本中通过 id 来引用该对象了。在“表达式”一节中我们已经演示了如何通过 id 来引用一个对象。

请注意，id 属性的值，首字符必须是小写字母或下画线，并且不能包含字母、数字、下画线以外的字符。

（3）列表属性

QML 对象的列表属性（类型是 list）类似于下面这样：

```
Item {
    children:[
        Image{},
        Text{}
    ]
}
```

列表是包含在方括号内，以逗号分隔的多个元素的集合。

其实列表和 ECMAScript 的数组（Array）是类似的，其访问方式也一样：

- 可以用[value1，value2，...，valueN]这种形式给 list 对象赋值。
- length 属性提供了列表内元素的个数。
- 列表内的元素通过数组下标来访问（[index]）。

值得注意的是，列表内只能包含 QML 对象，不能包含任何基本类型的字面量（如 8、true，如果非要包含，需要使用 var 变量）。这点与 JSON 是不一样的。下面是访问列表的示例。

```
Item {
    children:[
        Text{
            text: "textOne";
        },
        Text{
            text: "textTwo";
        }
    ]
    Component.onCompleted:{
        for (var i = 0; i < children.length; i++)
            console.log("text of label ", i, " : ", children[i].text)
    }
}
```

如果一个列表内只有一个元素，也可以省略方括号，如下所示：

```
Item {
    children:Image{}
}
```

不过，笔者还是建议你始终使用方括号，即使其中只有一个元素。

有没有什么问题？有就要说啊，闷在心里会憋坏自己的。好吧，你不说我就说了。在我们访问列表的示例中出现了一个新的内容，Component.onCompleted :{}，这是什么东西呢？接下来我们就来讲它。

（4）信号处理器

信号处理器，其实等价于 Qt 中的槽。但是我们没有看到类似于 C++中的明确定义的函数……没错，就是这样，你的的确确只看到了一对花括号！对啦，这是 ECMAScript 中的代

码块。代码块就是一系列语句的组合，它的作用就是使语句序列一起执行。

让我们回头再看信号处理器，它的名字还有点儿特别，一般是 on<Signal>这种形式。比如 Qt Quick 中的 Button 元素有一个信号 clicked()，那么你可能会写出这样的代码：

```
Rectangle {
    width: 320;
    height: 480;

    Button {
        id: quit;
        text: "退出";
        anchors.left: parent.left;
        anchors.leftMargin: 4;
        anchors.bottom: parent.bottom;
        anchors.bottomMargin: 4;
        onClicked: {
            Qt.quit();
        }
    }
}
```

上面的 QML 代码其实已经是一个简单的 QML 应用了，这个应用在窗口的左下角放了个“退出”按钮，当用户点击它时会触发按钮的 clicked()信号，而我们定义了 onClicked 信号处理器来响应 clicked()信号——调用 Qt.quit()退出应用。

你看到了，当信号是 clicked()时，信号处理器就命名为 onClicked。就这么简单，以 on 起始后跟信号名字（第一个字母大写）。

我们在前面用到的 Component.onCompleted:{}是附加信号处理器，马上就会介绍到它，而后面的第 6 章则会详细讲述它。

（5）分组属性

在某些情况下使用一个“.”符号或分组符号将相关的属性形成一个逻辑组。有时我们给分组属性赋值是一个个来的，类似于这样：

```
Text {
    font.pixelSize: 18;
    font.bold: true;
}
```

其实下面这样的写法在形式上更贴合分组的含义：

```
Text {
    font { pixelSize: 12; bold: true; }
}
```

其实可以这么理解，font 属性的类型本身是一个对象，这个对象又有 pixelSize、bold、italic、underline 等属性。对于类型为对象的属性值，可以使用“.”操作符展开对象的每一个成员对其赋值，也可以通过分组符号（一对花括号）把要赋值的成员放在一起给它们赋值。对于后者，其形式就和对象的定义一样了，起码看起来没有区别。所以，又可以这么理解上面的示例：Text 对象内聚合了 font 对象。OK，就是聚合。

（6）附加属性

属性真难搞！到现在还没讲完，不但你烦了，我也快坐不住了。我保证，这是最后一个要点了，不过也是最复杂、最难以理解的属性了。对于这种玩意儿，我一向的做法是，不能理解的话就接受，你就当它生来如此，存在即合理，只要学会怎么用它就 OK 了。

在 QML 语言的语法中，有一个附加属性（attached properties）的概念，这是附加到一个

对象上的额外的属性。

举个例子，下面的 Item 对象使用了附加属性：

```
import QtQuick 2.2

Item {
    width: 100;
    height: 100;

    focus: true;
    Keys.enabled: false;
}
```

你看，Item 对象设置 Keys.enabled 为 false，Keys 就是 Qt Quick 提供的供 Item 处理按键事件的附加属性。与附加属性相似的概念还有附加信号处理器，我们后面再讲。Keys 也留待后文细讲吧。

好啦，关于 QML 语言的基础性介绍就到这里，相信现在你已经可以看懂简单的 QML 文档了。或有疑问：这节还有一些东西只见用不见讲啊，比如 Rectangle、Text、Image、Item、Button、Component、Qt 等，抱歉，且听风吟——哦不，且听下回分解吧。