QML官方系列教程——Scalability

时间 2014-06-12 13:28:49 [CSDN博客](http://www.tuicool.com/sites/vQZRB3) [相似文章](http://www.tuicool.com/articles/dup?id=iaiuqi) (0) 原文  <http://blog.csdn.net/cloud_castle/article/details/30057169> [添加到推刊](http://www.tuicool.com/articles/iaiuqi#add-article-to-kan)

收藏到推刊 [创建推刊](http://www.tuicool.com/articles/iaiuqi#new-kan)

收 藏取消

已收藏到推刊！

请填写推刊名   
描述不能大于100个字符!   
权限设置： 公开 仅自己可见

创建取消

附网址：http://qt-project.org/doc/qt-5/scalability.html

Scalability —— (多屏)扩展性

Qt Quick使你开发的应用程序能够在 类似桌面，手持设备等 多个不同的平台上运行。特别地，它们还能够应付不同的的屏幕配置：尺寸，密度，方向，分辨率以及纵横比。

在这些情况下你需要考虑程序的扩展性：

· 你希望将应用程序部署在多个平台，比如Android，BlackBerry，iOS，或者不同的屏幕配置以及物理尺寸的设备上。

· 有些新的设备出现在你的应用程序第一次发布之后，你希望为它们提前做好准备。

我们这样使用 [***Qt Quick***](http://blog.csdn.net/cloud_castle/article/details/28649137) 来实现可扩展的应用程序：

· 使用提供了一组UI控件的 *Qt Quick Controls* 来设计UI界面。

· 使用 *Qt Quick Layouts* 来进行布局，它可以合理地对项目进行缩放。

· 使用 *property binding* 来实现布局无法处理的情况。例如，我们可以使用属性绑定来针对不同的屏幕显示不同dpi的图像，或是针对当前屏幕的方向自动旋转视图内容。

· 使用 *file selectors* 加载平台相关的资源。

· 使用 *Loader* 在需要时加载组件。

考虑一个应用程序可能需要被部署到不同屏幕配置，物理尺寸，或者UI风格的多个设备上，那么在设计你的应用程序时以下几种设计模式可以参考：

· 对于所有屏幕尺寸而言，视图的内容可以大致相似，但它需要具有可扩展的内容区域。当你使用 ApplicationWindow 这个来自Qt Quick Controls的QML类型时，它将自动根据它所包含的组件内容计算其窗口尺寸。而如果你使用Qt Quick Layouts来对其内部组件进行布局，它会改变这些组件的大小。

· 小设备上一个完整页面的内容可以作为大设备上布局中的一个元素。也就是说，考虑将UI分离成单独的组件（即是，在一个单独的QML文件中定义），这样，小设备上可以简单地只显示这一个单独的组件。这样在更大的设备上，当屏幕上有足够的位置，我们再使用 *loaders* 来加载另外的组件。例如，一个email浏览器，如果这个设备足够大，我们可以使用它展示一个email列表，这样email的读者就可以一条一条进行阅读。（补充来说，如果将一条email作为一个组件，那么小设备上可以只显示少量几条。译者注。）

· 两个设备之间的屏幕的尺寸和外形可能很相近，但是使用更贴近本地习惯的UI风格。

Resizing Screens Dynamically —— 动态调整屏幕

Qt Quick Controls 提供了一组UI控件以帮助创建用户界面。例如我们可以声明一个 ApplicationWindow 控件作为应用程序的根项目。 ApplicationWindow 以一个平台独立的方式提供了对其他控件布局的便利性，类似 MenuBar ， ToolBar ，以及 StatusBar 等等。当计算实际窗口的尺寸限制时 ApplicationWindow 使用内部组件的尺寸限制作为输入。

除了定义应用程序窗口所需的各种控件，controls还被提供用来创建views和menus。并且可以响应和接收用户的操作。你可以使用 Qt Quick Controls Styles 来设置这些预定义的控件的风格。

Qt Quick控件，类似 ToolBar 之类，都没有提供自己的layout，因此需要你对它们的内容进行布局。你可以使用Qt Quick Layouts来做这件事情。

Laying out Screens Dynamically —— 屏幕的动态编排

Qt Quick Layouts提供了多种方式来编排屏幕上的控件——行、列、网格等等，这需要使用 RowLayout 、 ColumnLayout 和 GridLayout 类型。设置它们的属性来决定其布局方向和各单元间的间隔。

你可以使用 Layout 类型为 布局中的元素 附加额外的属性。例如，你可以为对象的高度、宽度和尺寸指明其最小值、最大值和首选值。

这些布局类型保证了当窗口和屏幕的尺寸发生变化时你的UI组件能够进行适当的缩放，并总是保证空间的最大利用率。

不断地调整和重新计算显示尺寸会带来性能上的开销。比如移动设备和嵌入式设备可能没有这个能力在每一帧都计算动画对象的尺寸和位置。所以如果在使用layouts时遇到了性能瓶颈，你应该考虑使用其他方法，例如绑定，来进行替换。

在这些地方不要使用layouts：

· 不要绑定一个布局中元素的x，y，width或者height属性，因为这有可能与Layout产生冲突，并引起循环绑定。

· 不要定义会重复计算的复杂JavaScript表达式。这将带来很差的表现性能，尤其是在动画过渡中。

· 不要对包含者(container)或子项目的尺寸做假设。试试使用灵活的layout，它可以来适应可用空间的改变。

· 如果你的设计需要在像素级层面上表现完美，就不要使用layout。它会依赖于可用空间对内部组件的大小和位置自动调整。

Handling Pixel Density —— 处理像素密度

如果Qt Quick Layouts并不能满足你的需要，你可以转而使用 property binding 。绑定使得对象在其他对象属性值改变或者外部事件发生时能够自动调整他们的属性。

要为对象的属性分配值，一种方式是分配一个静态的数值，另一种是绑定到一个JavaScript表达式。第一种情况下，除非一个新的值被分配给该属性，否则它不会被改变；第二种情况下，一个属性绑定被创建后，只要当这个计算表达式的值发生变化，该属性值就会由QML引擎动态更新。

这种定位方式是高度动态的，然而，不断地计算JavaScript表达式会带来性能的损耗。

例如，绑定是一种处理不同像素很好的方式。下面的代码段使用了Screen.logicalPixelDesity附加属性来指明需要在不同分辨率屏幕下显示的不同图像：

Image {

source: {

if (Screen.logicalPixelDensity < 40)

"image\_low\_dpi.png"

else if (Screen.logicalPixelDensity > 300)

"image\_high\_dpi.png"

else

"image.png"

}

}

·

Loading Files Depending on Platform —— 根据不同平台加载相应文件

你可以使用 QQmlFileSelector 来应用一个 QFileSelector 以加载QML文件。这使你在应用程序运行时可以根据不同平台加载不同的资源。例如，当程序运行在Android和BlackBerry设备上时，你可以使用+android和+blackberry的文件选择器(file selectors)来加载不同的图像文件。

你可以针对一个特殊的平台使用多个文件选择器选择一个单例对象。

文件选择器是静态的，并且强制要求平台相关的文件存放在与平台名相同的子目录下。如果你需要一个更动态的方式来根据需求加载你的UI组件，可以使用Loader。

Loading Components on Demand —— 根据需求加载组件

Loader能够加载一个QML文件（使用source属性）或者一个组件对象（使用sourceComponent属性）。它对于需要时才创建的组件特别有用。例如由于需要或者性能原因可以延迟创建组件的时候。

有时由于某些平台并不支持的功能，我们的部分UI并不需要在这些平台上显示，这时我们同样可以使用Loaders。对于程序运行时不需要显示在设备上的视图，我们可以将它隐藏，并使用loaders在它原本的地方显示一些别的东西。

Switching Orientation —— 转向切换

Screen.orientation附加属性包含了屏幕的当前方向（横纵），我们从加速度计取到这个值（如果有的话）。在桌面电脑上，这个值通常不会发生改变。

如果primaryOrientation跟随orientation，那么意味着当屏幕转向时内部所有组件都会随之旋转，而这取决于你如何拿着这个设备。如果orientation发生了改变但primaryOrientation没有变化，那么该设备可能不能对自己的显示进行旋转。在这种情况下，你需要使用 Item.rotation 或者 Item.transform 来旋转你的内容。

应用程序的顶级页面以及可复用的组件应该在一个QML layout定义中进行声明。它应该被设计成能够适应各个设备的方向以及纵横比。旋转屏幕时的表现性是十分关键的，因此这是保证朝向改变时，所有组件都可以被正确加载的一个好方法。

相反的是，如果在屏幕旋转时选择使用 Loader 来加载额外需要的QML组件，你需要进行彻底的测试。因为这样十分影响性能。

为了在转向时正常显示，锚定义必须设置在相同包含层面的组件上（有点类似父子关系中的兄弟姐妹，只不过这里是从显示的角度。译者注）。因此页面或是组件的结构应该是包含一组共同的子项目，一个共同的锚定义，一个状态的集合（在 StateGroup 中定义），以此对不同的纵横比进行支持。

如果包含一个特定组件的一个页面需要建立在多种不同物理尺寸的平台上，那么这个组件布局的状态就应该依赖于这个页面的纵横比（它的父包含者。同样类似父对象，这里是视图上的。译者注）。类似的，一个组件的不同实例可能位于多种不同的容器中，并且它的布局应该取决于它父包含者的宽高比。结论就是，布局状态应该始终跟随其直接容器的宽高比（并不是当前设备屏幕的“orientation”）。

在每一个布局 State ，你都应该使用本地QML layout定义对象之间的关系。在两个状态的转变期间（由顶级orientation改变引起），在锚布局的用例中，AnchorAnimation元素可以被用来控制过渡。某些情况下，你也可以使用NumberAnimation 在例如组件的宽度上控制过渡。要记得避免在动画的帧切换中使用复杂的JavaScript计算式。使用简单的锚定义和锚动画可以解决大部分问题。

这里举了一些实际的应用例子：

· 如果你有一个单一的页面，它在展示风景和肖像时完全不同（比如，所有的子项目都不一样）该怎么做？那么对于每个单一的页面，我们都有两套子组件，并定义不同的布局，然后使其中一个在某种状态下透明化。你可以将 NumberAnimation 应用于透明度以达到淡入淡出的动画效果。

· 那如果这个页面在显示风景和肖像时有30%或以上的布局内容相同呢？考虑创建一套组件应用于风景和肖像两个状态，以及一组各自独立的子项目，其透明度（或是位置）取决于状态的改变。你可以 对那些多个状态通用的组件中使用布局，而对 其他项目使用淡入/淡出或是动画的开始/结束。

· 如果你有两个页面需要同时在手持设备的屏幕上显示，例如更大规格的设备(脑补surface...)？注意在这种情况下，你的可视化组件不再能够占领整个屏幕。因此记住这一点十分重要：所有的组件（特别是，列表的委托项目）应该依赖于包含该组件的宽度尺寸，而不是屏幕的宽度。这种情况下在Component.onCompleted()中设置对象的宽度是必要的，它可以保证列表委托项目在被赋值前已经被构造。

如果同时保持两个方向上的内容占用了太多内存？必要的话可以使用Loader，但是你需要当心布局切换时动态过渡的表现性能。一个解决方式是可以有两个"splash screen（载入屏幕）"对象作为这个页面的子对象，然后在旋转的时候添加淡入该虚拟屏幕的效果。然后你可以使用Loader来实际加载组件，它加载了另一套子组件实际的数据，并在Loader完成后结束这个虚拟屏幕的淡入效果