第3章

布局

WPF 布局是通过面板(Panel)对页面元素进行全面规划和安排。简单地说,就是把一些控件有条理地摆放在界面上合适的位置。在应用程序界面设计中,合理的元素布局至关重要,它可以方便用户操作,并用清晰的页面逻辑呈现用户信息。如果内置布局控件不能满足需要,用户还可以创建自定义的布局元素。

3.1 布局原则

WinForm 的布局是采用基于坐标的方式,当窗口内容发生变化时,里面的控件不会随之动态调整,用户体验不够好。而 WPF 采用了基于流的布局方式,像 Web 开发模式。流式布局特点是:所有的元素总是默认地自动向左上角靠近,在设计时,通过控制元素相对位置的方式使其达到预计的效果,即元素的位置依赖于相邻元素的位置和尺寸。

3.1.1 合成布局模型

WPF 的合成布局模型是用来满足广泛的应用场景布局,允许某种布局控件被嵌套在其他布局控件中。合成布局模型通过布局契约来实现子控件和父布局控件间的通信问题。布局契约包括两种设计思想,即根据内容调整尺寸和两段布局。

1. 根据内容调整尺寸

根据内容调整尺寸,即每个控件都根据内容来确定控件大小,这个设计思想应用于 UI 中的所有控件。例如,窗口能够调整大小来适应它们内部的控件,文本框控件能调整尺寸来适应它内部的文本。当然每个元素会被询问其期望的尺寸大小,以确保根据内容调整尺寸的设计思想能够实施。

2. 两段布局

两段布局是指在两个完全不同的阶段来确定控件的最佳尺寸。在这两个阶段布局模型让父布局控件和子控件达成元素最后尺寸的约定。两个阶段分别是测量(Measure)和排列(Arrange)。测量阶段需要做的主要工作是:对整个 UI 页面的检测,并询问每个元素的期望尺寸(Desired Size),元素返回一个可用的尺寸(Available Size),当所有的元素都被询问并测量好以后,就进入到排列阶段。在排列阶段,父元素通知每个子元素的实际尺寸(Actual Size)和位置。

在两段布局中,父元素和子元素需要协商出需要的尺寸大小,涉及可用尺寸、期望尺寸、实际尺寸,在此辨析 3 个尺寸。其中,可用尺寸是测量阶段的初始约束值,即父元素愿意给子元素的最大空间值;期望尺寸是子元素想要的尺寸;实际尺寸是父元素分配给子元素的

最终尺寸。这 3 个尺寸要符合下面的不等式条件。

Desired Size≤Actual Size≤Available Size

了解 WPF 合成布局模型，学习 WPF 布局机制，才能理解合成布局模型的来龙去脉，在页面布局时做到得心应手。

3.1.2 布局机制

WPF 界面上的每个元素的边界框尺寸和排列是 WPF 自动计算出来的。通过 WPF 合成布局模型的学习，了解 WPF 渲染布局的过程中，执行测量(Measure)和排列(Arrange)两个步骤。在布局机制中，详细分解在 WPF 布局的不同阶段，后台类的调用过程。在测量阶段，布局容器遍历所有子元素，并询问子元素所期望的尺寸；在排列阶段，布局控件在合适的位置放置子元素，并设置元素的最终尺寸；这是一个递归的过程，界面中任何一个容器元素都会被遍历到。

因为面板可以嵌套，所以处理过程是递归的，布局(Layout)处理的过程如图 3.1 所示。

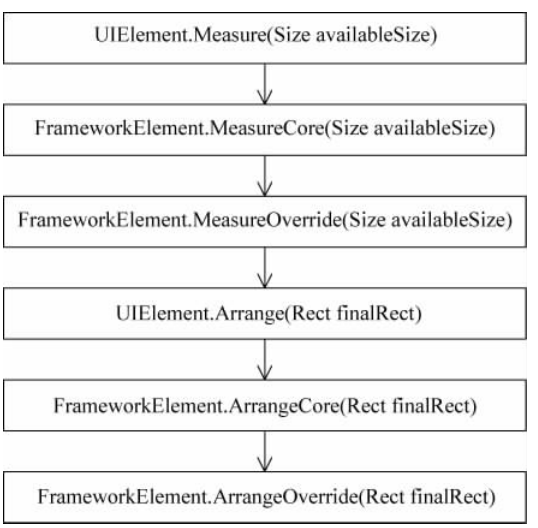

图 3.1 WPF 布局原理

在此，简要说明 WPF 布局处理过程。由 WPF 框架可知，所有 UI 元素的根元素是 UIElement 类型，在 UIElement 中定义了一些基本的关于 UI 显示的属性(如 Clip 和 Visibility)。在 UIElement. Measure(Size availableSize)方法执行阶段，就是对这些基本属性做评估，获得适合的 Size。同样，FrameworkElement. MeasureCore(Size availableSize)方法评估时，在 FrameworkElement 中定义且有可能影响 UI 布局的属性，得出更适合的 Size。这个 Size 将被传递给 FrameworkElement. MeasureOverride(Size availableSize)方法。WPF 提供的 Panel 类型(如 Grid)中就会重写该方法来处理，处理完后将得到一个系统期望的 Size(称为 DesiredSize)。布局系统将按照这个 Size 来显示该 Element，测量(Measure)阶段结束。Size 确定后，把 Size 包装为 Rect 实例，传递给 UIElement. Arrange(Rect finalRect)，进行排列(Arrange)处理。根据 Size 值，Arrange 方法为元素创建边界框，边框打包到 Rect 实例，传给 FrameworkElement. ArrangeCore(Rect finalRect)方法。ArrangeCore

将继续评估 DesiredSize，计算边界留白（Margin，Padding）等信息，获得 ArrangeSize，并传给 FrameworkElement. ArrangeOverride（Size finalSize）。这个方法也是可重写的，WPF 提供的 Panel 类型会重写该方法来处理，最终获得 finalSize。当 finalSize 确定后，ArrangeOverride 执行完毕，控制权回到 ArrangeCore 方法，ArrangeCore 把该 Element 放到它的边界框中。到此，该 Element 的 Layout 处理完成。

3.1.3 布局通用属性

所有的 WPF 布局面板都由 System. Windows. Controls. Panel 抽象类派生。Panel 就是所有布局元素的基类，用于放置和排列 WPF 元素，这个抽象类包含 3 个公共属性：Background、Children 和 IsItemHost（IsItemHost 标志着控件是不是类似 TreeView 和 ListView 这样的控件）。布局容器内的子元素对自身的对齐方式（HorizontalAlignment、VerticalAlignment）、宽度（Width）、高度（Height）、四周间隙（Margin）等有一定的决定权。子元素可以设置自身的布局属性来调整自己的位置和大小。表 3.1 给出了 WPF 布局面板中的通用属性。

表 3.1 WPF 布局面板中的通用属性

属性名称	表征意义
Background	布局面板背景着色，在响应鼠标事件时，该值非空（含透明）
Children	布局面板中存储的条目集合，条目还可以含更多的条目
IsItemHost	布尔值，是否为由 ItemsControl 生成的 UI 项的容器
HorizontalAlignment	水平对齐方式，有 Center、Left、Right、Stretch 属性值可选
VerticalAlignment	垂直对齐方式，有 Center、Left、Right、Stretch 属性值可选
MinWidth/MinHeight	最小宽度尺寸/最小高度尺寸，默认单位是像素
MaxWidth/MaxHeight	最大宽度尺寸/最大高度尺寸，默认单位是像素
Width/Height	宽度尺寸/高度尺寸，默认单位是像素
Margin	在元素周围空白尺寸，默认单位是像素

3.2 布局面板

WPF 的布局面板实现基本的布局，布局面板类型有 Canvas、DockPanel、StackPanel、WrapPanel、Grid 和 UniformGrid。在这一节中，不涉及 Grid，因为 Grid 灵活，适用场合颇多，需独立讲解。表 3.2 列出了布局面板的类型及其使用规则。

表 3.2 布局面板的类型及其使用规则

布局面板类型	使用规则
Canvas	画布。通过 Top、Left、Right 和 Bottom 4 个属性将子元素定位
DockPanel	停靠面板。让子元素停靠在整个面板的某一条边上，然后拉伸元素以填满全部宽度或高度。类似于 Windows Form 编程中控件的 Dock 属性
StackPanel	堆式面板。子元素按照声明的先后顺序，自上往下或从左往右摆放

续表

布局面板类型	使 用 规 则
WrapPanel	自动折行面板。子元素按照声明的先后顺序，从左往右摆放，摆满一行后，自动折行。与 HTML 中的流式布局相似
Grid	网格。通过定义行高和列宽来调整子元素。类似于 HTML 中的 Table
UniformGrid	简化网格布局。每个单元格具有相同的大小，自动创建相同的行列数

3.2.1 Canvas

Canvas(画布)是 WPF 中最简单的布局控件，是用于存储控件的容器，不会自动调整内部元素的排列及大小，它仅支持用显式坐标定位控件。可以使用 Left、Top、Right 和 Bottom 附加属性在 Canvas 中定位控件。实质上，在选择每个控件停靠角时，附加属性的值是作为外边距使用的。如果一个控件没有使用任何附加属性，它会被放在 Canvas 的左上方(等同于设置 Left 和 Top 为 0)。

Canvas 实例 1，设计要求：将 5 个 Button 按钮分别放在画布的左上角、右上角、左下角、右下角和中心位置。使用 XAML 代码如下。

```
<Canvas Width = "200" Height = "100" Background = "Beige">
    <Button Content = "Left, Top" Canvas.Left = "4" Canvas.Top = "4"/>
    <Button Content = "Right, Top" Canvas.Right = "4" Canvas.Top = "4"/>
    <Button Content = "Left, Bottom" Canvas.Left = "4" Canvas.Bottom = "4"/>
    <Button Content = "Right, Bottom" Canvas.Right = "4" Canvas.Bottom = "4"/>
    <Button Content = "Center" Canvas.Left = "75" Canvas.Top = "38"/>
</Canvas>
```

运行上述代码后，显示页面效果如图 3.2 所示。“画布”上的元素附加属性 Canvas.Left、Canvas.Top、Canvas.Right 和 Canvas.Bottom 来完成内部子元素(Button)的定位。这里的 Canvas.Left 和 Canvas.Top 属性与 Windows Form 窗体控件的 Left 和 Top 属性类似，Canvas.Left 属性表示距离“画布”左边的距离，而 Canvas.Top 属性表示距离“画布”顶部的距离。

Canvas 实例 2，设计要求：分别将黄色、粉红色和蓝紫色 3 个 Rectangle 放到画布上，使用 XAML 代码如下。

```
<Canvas>
      <Rectangle Canvas.ZIndex = "3" Width = "60" Height = "60" Canvas.Top = "30"
                 Canvas.Left = "30" Fill = "BlueViolet"/>
      <Rectangle Canvas.ZIndex = "1" Width = "60" Height = "60" Canvas.Top = "50"
                 Canvas.Left = "50" Fill = "Yellow"/>
      <Rectangle Canvas.ZIndex = "2" Width = "60" Height = "60" Canvas.Top = "70"
                 Canvas.Left = "70" Fill = "Pink"/>
</Canvas>
```

运行上述代码后，显示页面效果如图 3.3 所示。“画布”上的元素附加属性 Canvas.ZIndex 设置不同颜色的 Rectangle 声明的先后顺序，从视觉效果上看，后声明的蓝紫色矩形将先声明的矩形框覆盖，用专业术语则是 Canvas.ZIndex 设置重叠深度。

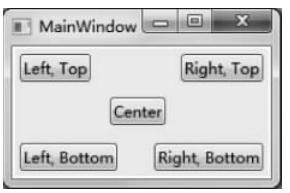

图 3.2　Canvas 附加属性定位

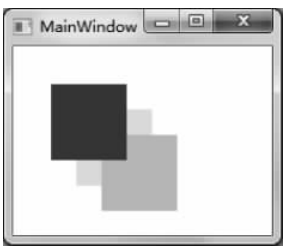

图 3.3　Canvas.ZIndex 设置重叠深度

3.2.2　DockPanel

DockPanel(停靠面板)让子元素停靠在整个面板的某一条边上。当多个子元素停靠在相同的边时,根据声明的先后顺序依次停靠,系统默认最后一个声明子元素来添满剩余空间。打开 Windows 计算机窗口,如图 3.4 所示,该窗口是利用 DockPanel 的常见布局方式。

图 3.4　Windows 计算机窗口

现将 Windows 计算机窗口分解后,它是由菜单栏、工具条、文件夹列表框、详细信息列表框和状态栏组成的,如图 3.5 所示。

现在使用停靠面板来创建 Windows 计算机窗口的基本结构,在此用 Button 来表示所有的子元素。XAML 代码如下。

```
<DockPanel>
    <Button DockPanel.Dock = "Top">菜单栏区域</Button>
    <Button DockPanel.Dock = "Top">工具条区域</Button>
    <Button DockPanel.Dock = "Bottom">状态栏区域</Button>
    <Button DockPanel.Dock = "Left">文件夹区域</Button>
    <Button DockPanel.Dock = "Right" >详细信息列表区域</Button>
</DockPanel>
```

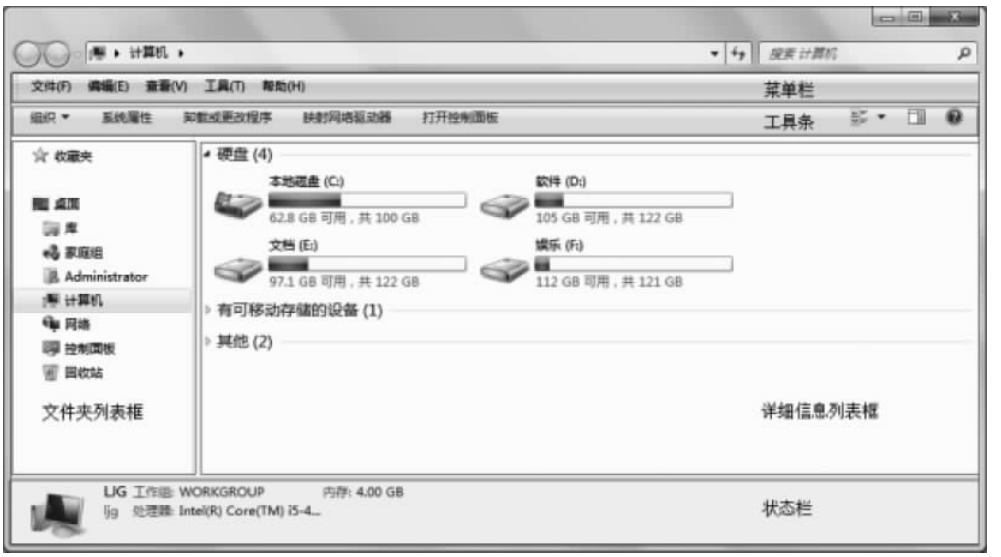

图 3.5　Windows 计算机窗口分解

运行上述代码后，页面显示效果如图 3.6 所示。DockPanel.Dock 作为 Button 的附加属性，用来设置上(Top)、下(Bottom)、左(Left)、右(Right)的停靠位置。

图 3.6　DockPanel 构建窗口常用布局

下面通过调整 Button 声明的顺序来改变页面布局结构。XAML 代码如下。

```
<DockPanel>
    <Button DockPanel.Dock = "Top">菜单栏区域</Button>
    <Button DockPanel.Dock = "Left">文件夹区域</Button>
    <Button DockPanel.Dock = "Top">工具条区域</Button>
    <Button DockPanel.Dock = "Bottom">状态栏区域</Button>
    <Button DockPanel.Dock = "Right" >详细信息列表区域</Button>
</DockPanel>
```

运行上述代码后，页面显示效果如图 3.7 所示。读者留意会发现，在文件夹区域与详细

信息列表区域之间没有分隔条。具有分隔条的 WPF 控件是 GridSplitter，它依赖于 Grid 控件，后续章节将对其进行讨论。

图 3.7 调整 Button 声明顺序改变布局结构

3.2.3 StackPanel

StackPanel(堆式面板)将子元素按照声明的先后顺序堆在一起。设置 Orientation 属性值确定以水平或垂直方向堆放。

在 StackPanel 中，根据子元素的最大尺寸来确定最佳尺寸。设计 StackPanel 实例，页面显示效果如图 3.8 所示。

设计要求：在最外层是边框环绕，内部一个 StackPanel 中再嵌套两个 StackPanel，子元素用 Button，XAML 代码如下。

```
<Border BorderBrush = "Black" BorderThickness = "1"
        HorizontalAlignment = "Center" VerticalAlignment = "Center">
    <StackPanel Height = "154" Width = "221">
        <StackPanel Margin = "2" Background = "Yellow" Orientation = "Vertical">
            <Button Content = "One" Margin = "1"/>
            <Button Content = "Two" Margin = "1"/>
            <Button Content = "Three" Margin = "1"/>
            <Button Content = "Four" Margin = "1"/>
            <Button Content = "Five" Margin = "1"/>
        </StackPanel>
        <StackPanel Margin = "2" Background = "pink" Orientation = "Horizontal">
            <Button Content = "One" Margin = "3"/>
            <Button Content = "Two" Margin = "3"/>
            <Button Content = "Three" Margin = "3"/>
            <Button Content = "Four" Margin = "3"/>
            <Button Content = "Five" Margin = "3"/>
        </StackPanel>
    </StackPanel>
</Border>
```

StackPanel 会根据方向使用无限宽度和高度来测量子控件，当有其他布局同时出现时，会影响布局的整体风格。本例中，通过最外层的 Border 对 StackPanel 进行约束。

StackPanel 适合局部页面布局，接下来用它做一个简单的用户搜索页面。页面效果如图 3.9 所示。XAML 代码如下。

```
<StackPanel Background = "AliceBlue">
        <TextBlock Margin = "4"> Look for:</TextBlock>
        <ComboBox Margin = "4"></ComboBox>
        <TextBlock Margin = "4"> Filtered by:</TextBlock>
        <ComboBox Margin = "4"></ComboBox>
        <Button Margin = "4,5"> Search </Button>
        <CheckBox Margin = "4"> Search in titles only </CheckBox>
        <CheckBox Margin = "4"> Search in Keyword </CheckBox>
</StackPanel>
```

图 3.9 是由两个 TextBlock、两个 ComboBox、两个 CheckBox 和一个 Button 构成的页面，设置 Margin 属性让页面保持清爽。

图 3.8　3 个 StackPanel 嵌套

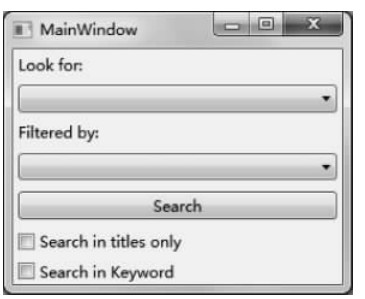

图 3.9　用户搜索页面

3.2.4　WrapPanel

WrapPanel（自动折行面板）允许任意多的子元素按照声明的先后顺序，从左往右摆放，摆满一行后，自动折行。折行面板的经典的例子就是工具条布局。

默认情况下，WrapPanel 根据子元素的内容，自动调整控件的大小。也可以设置 ItemWidth 和 ItemHeight 属性来约束子元素的宽度和高度。WrapPanel 布局 10 个 Button，如图 3.10 与图 3.11 所示。

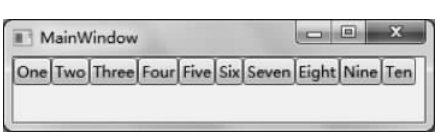

图 3.10　WrapPanel 布局 10 个 Button

图 3.11　折行的 Button

由图 3.10 可知，XAML 代码如下。

```
<WrapPanel Background = "AliceBlue">
        <Button> One </Button>
```

```
        <Button>Two</Button>
        <Button>Three</Button>
        <Button>Four</Button>
        <Button>Five</Button>
        <Button>Six</Button>
        <Button>Seven</Button>
        <Button>Eight</Button>
        <Button>Nine</Button>
        <Button>Ten</Button>
</WrapPanel>
```

把图 3.10 中的窗口宽度变窄，调整到一行放 5 个 Button，图 3.10 会变成图 3.11 的效果。

3.2.5 UniformGrid

UniformGrid(简化网格布局)隐含在 System.Windows.Controls.Primitives 名称空间中。每个单元格具有相同的大小。在使用 UniformGrid 布局时，需要设定行值与列值。下面使用 UniformGrid 对 6 个 Button 布局，XAML 代码如下。

```
<Border BorderBrush="Black" BorderThickness="1"
        HorizontalAlignment="Center" VerticalAlignment="Center">
    <UniformGrid Rows="3" Columns="2" Background="Green" >
        <Button Content="One" Margin="1"/>
        <Button Content="Two" Margin="1"/>
        <Button Content="Three" Margin="1"/>
        <Button Content="Four" Margin="1"/>
        <Button Content="Five" Margin="1"/>
        <Button Content="Six" Margin="1"/>
    </UniformGrid>
</Border>
```

运行上述代码，页面显示效果如图 3.12 所示。UniformGrid 布局是绿色背景，6 个 Button，按照 3 行 2 列生成大小相同的单元格。

在使用 UniformGrid 布局时，如果只设定列值，那么行值等于子元素的数目除以列值；如果只设定行值，那么列值等于子元素的数目除以行值。接下来，使用 UniformGrid 布局 6 个 Button 为 2 行 3 列，XAML 代码如下。

```
<Border BorderBrush="Black" BorderThickness="1"
        HorizontalAlignment="Center" VerticalAlignment="Center">
    <UniformGrid Columns="3" Background="Green" >
        <Button Content="One" Margin="1"/>
        <Button Content="Two" Margin="1"/>
        <Button Content="Three" Margin="1"/>
        <Button Content="Four" Margin="1"/>
        <Button Content="Five" Margin="1"/>
        <Button Content="Six" Margin="1"/>
    </UniformGrid>
</Border>
```

运行上述代码，页面显示效果如图 3.13 所示。代码中只设置了“Columns="3"”，

UniformGrid 布局 6 个 Button，系统行值等于 6 除于 2，自动生成 2 行 3 列的 UniformGrid 网格布局。

图 3.12　网格 3×2 的 UniformGrid

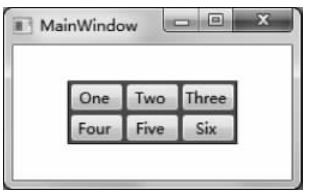

图 3.13　网格 2×3 的 UniformGrid

使用 UniformGrid 布局的网格宽度是所有子元素中的最大宽度值；高度是所有子元素中的最大高度值。用 UniformGrid 布局 3×3 的网格，XAML 代码如下。

```
<UniformGrid Columns = "3" Rows = "3" Background = "AliceBlue" >
    <Button Content = "One" Margin = "1"/>
    <Button Content = "Two" Margin = "1"/>
    <Button Content = "Three" Margin = "1"/>
    <Button Content = "Four" Margin = "1"/>
    <Button Content = "Five" Margin = "1"/>
    <Button Content = "Six" Margin = "1"/>
    <Button Content = "Seven" Margin = "1"/>
</UniformGrid>
```

运行上述代码，页面显示效果如图 3.14 所示。由图 3.14 可知，当代码中给出的单元格数大于子元素的数目，依次摆放后，余下的单元格为空。

修改上述 XAML 代码，将“Rows＝"3"”改成“Rows＝"2"”，运行代码，页面显示效果如图 3.15 所示。可知，当系统提供的单元格数小于子元素的数目，UniformGrid 把最后一个子元素放到了边界外。

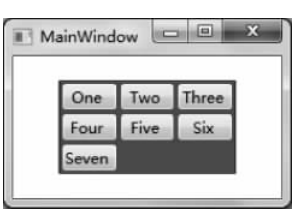

图 3.14　网格 3×3 的 UniformGrid

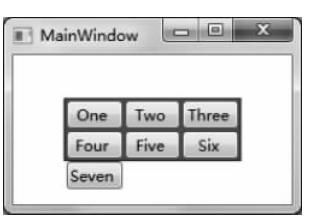

图 3.15　子元素大于网格单元数的 UniformGrid

3.3　Grid

尽管 UniformGrid 能够布局统一单元格，但是很多布局中需要构建单元格大小不等，具有跨越式单元格(Span Cells)和空白列等功能。而 Grid 能实现上述这些功能，它是一个使用灵活、能构建复杂 UI 的布局控件。

Grid 最简单的用法是通过设置 RowDefinitions 和 ColumnDefinitions 属性定义单元格的总数。其中，RowDefinitions 指行数，ColumnDefinitions 指列数。添加子元素时，使用

Grid.Row 和 Grid.Column 附加属性设定子元素在单元格中的位置。接下来，用 Grid 布局 6 个 Button，单元格排列成 3 行 2 列。XAML 代码如下。

```
<Grid Background = "green">
        <Grid.RowDefinitions>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Button Grid.Row = "0" Grid.Column = "0" Margin = "2"> One </Button>
        <Button Grid.Row = "0" Grid.Column = "1" Margin = "2" > Two </Button>
        <Button Grid.Row = "1" Grid.Column = "0" Margin = "2"> Three </Button>
        <Button Grid.Row = "1" Grid.Column = "1" Margin = "2"> Four </Button>
        <Button Grid.Row = "2" Grid.Column = "0" Margin = "2" > Five </Button>
        <Button Grid.Row = "2" Grid.Column = "1" Margin = "2"> Six </Button>
</Grid>
```

运行上述代码，页面显示效果如图 3.16 所示。当调整窗口大小时，Button 大小随着窗口的变化而变化。

在很多应用程序中都会有登录页面的设计，在此，使用 Grid 布局用户登录页面，XAML 代码如下。

```
<Grid Background = "AliceBlue">
   <Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
  </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
     <RowDefinition></RowDefinition>
     <RowDefinition></RowDefinition>
     <RowDefinition></RowDefinition>
    </Grid.RowDefinitions>
     <TextBlock Text = "Username: " VerticalAlignment = "Center" FontSize = "20"
     HorizontalAlignment = " Center" FontFamily = "Times New Roman" >
     </TextBlock>
     <TextBlock Text = "Password: " Grid.Row = "1" VerticalAlignment = "Center"
     HorizontalAlignment = "Center" FontSize = "20" FontFamily = "Times New Roman">
     </TextBlock>
     <TextBox Grid.Column = "1" Margin = "25" FontSize = "20"></TextBox>
     <TextBox Grid.Column = "1" Grid.Row = "1" Margin = "25" FontSize = "20"></TextBox>
     <Button Grid.Row = "2" Margin = "25" Content = "Login" FontSize = "20" FontFamily = "Times
     New Roman" Background = "LightBlue">
     </Button>
        <Button Grid.Row = "2" Margin = "25" Content = "Cancel" Grid.Column = "1" FontSize =
     "20" FontFamily = "Times New Roman" />
     </Grid>
```

运行上述代码，页面显示效果如图 3.17 所示。该登录页面含有两个 TextBlock、两个 TextBox 和两个 Button，共 6 个子元素。当窗口大小变化时，页面子元素会随着窗口的大小动态调整。

图 3.16　3×2 的 Grid

图 3.17　Grid 布局登录界面

上面的两个例子，显示了基本的 Grid 布局网格的用法，但并没有充分实现 Grid 的重要特征。下面从结构中分离布局、尺寸模型、共享尺寸组、跨越行和列与 GridSplitter 等方面来演示 Grid 的多种用法。

3.3.1　从结构中分离布局

前面讲到的 WPF 布局面板，都是通过改变元素的声明顺序来改变布局结构，因为声明顺序不同，所以布局结构也会不同。WPF 除 Grid 以外的布局面板，首先，将布局面板融入可视化树中，便于调用布局算法。然后，根据声明子控件声明的顺序，设定布局结构。

Grid 能从结构中分离布局，指的是不依赖子元素的声明顺序来改变布局结构。因为子元素可以通过两个附加属性进行定位。下面对 6 个 Button 通过设置 Grid.Row 和 Grid.Column 两个附加属性值，让页面显示与 Button 的定义顺序相反。XAML 代码如下。

```
<!-- 略与上例代码相同部分 -->
    <Button Grid.Row="2" Grid.Column="1" Margin="2">One</Button>
    <Button Grid.Row="2" Grid.Column="0" Margin="2">Two</Button>
    <Button Grid.Row="1" Grid.Column="1" Margin="2">Three</Button>
    <Button Grid.Row="1" Grid.Column="0" Margin="2">Four</Button>
    <Button Grid.Row="0" Grid.Column="1" Margin="2">Five</Button>
    <Button Grid.Row="0" Grid.Column="0" Margin="2">Six</Button>
</Grid>
```

运行上述代码后，页面显示效果如图 3.18 所示。6 个 Button 布局效果与 Button 定义顺序相反，是通过设置附加属性 Grid.Row 和 Grid.Column 的值来改变 Button 的输出顺序。

接下来选用 Uniform Grid 布局 6 个 Button，XAML 代码如下。

```
<Border BorderBrush="Black" BorderThickness="1"
        HorizontalAlignment="Center" VerticalAlignment="Center">
        <UniformGrid Columns="2" Background="Green">
```

```
            <Button Content = "Six" Margin = "1"/>
            <Button Content = "Five" Margin = "1"/>
            <Button Content = "Four" Margin = "1"/>
            <Button Content = "Three" Margin = "1"/>
            <Button Content = "Two" Margin = "1"/>
            <Button Content = "One" Margin = "1"/>
        </UniformGrid>
</Border>
```

运行上述代码，页面显示效果如图 3.19 所示，在使用 UniformGrid 布局时，只能通过改变元素定义顺序改变布局结构。

图 3.18 设置 Grid 附加属性调整布局结构

图 3.19 Uniform Grid 改变元素顺序改变布局结构

Grid 布局页面时，在不影响代码的情况下，通过设置 Grid.Row 和 Grid.Column 两个附加属性值，调整布局结构，这种从结构中分离布局使 Grid 能够创建控件结构。

3.3.2 尺寸模型

在 Canvas 布局中，要使用元素绝对值来分割空间；而 Grid 引入了百分比尺寸，也就是列或行的高度能设置为星花(*)单位。星花允许行和列在按照内容尺寸或绝对尺寸分配空间后，占用网格空间的一个百分比值。

接下来理解 * 的用法，XAML 代码如下。

```
<Grid Background = "green">
        <Grid.RowDefinitions>
            <RowDefinition Height = "50"></RowDefinition>
            <RowDefinition Height = "1 * "></RowDefinition>
            <RowDefinition Height = "2 * "></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width = "80"></ColumnDefinition>
            <ColumnDefinition ></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Button Grid.Row = "0" Grid.Column = "0" Margin = "2"> One </Button>
        <Button Grid.Row = "0" Grid.Column = "1" Margin = "2" > Two </Button>
        <Button Grid.Row = "1" Grid.Column = "0" Margin = "2"> Three </Button>
        <Button Grid.Row = "1" Grid.Column = "1" Margin = "2"> Four </Button>
        <Button Grid.Row = "2" Grid.Column = "0" Margin = "2"> Five </Button>
        <Button Grid.Row = "2" Grid.Column = "1" Margin = "2"> Six </Button>
</Grid>
```

运行上述代码后，页面显示效果如图 3.20 所示。第 0 行第 0 列的 Button One 的高度(Height＝"50")是 50 像素，宽度(Width＝"80")是 80 像素。Button Three 所在行占余下空间的 1/3，Button Five 所在行占余下空间的 2/3。还使用了 Grid.Row、Grid.Column 附加属性把子元素定位到单元格。在此，需要声明的是，Grid 的行和列都是从 0 开始计数的，如果没有指定子元素的行列值，则子元素默认位于第 0 行第 0 列。

将上述代码中的 Height 与 Width 全改成星花(＊)单位的 XAML 代码如下。

```
<Grid Background = "green">
        <Grid.RowDefinitions>
            <RowDefinition Height = "1 * "></RowDefinition>
            <RowDefinition Height = "2 * "></RowDefinition>
            <RowDefinition Height = "3 * "></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width = "1 * "></ColumnDefinition>
            <ColumnDefinition Width = "1 * " ></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Button   Grid.Row = "0" Grid.Column = "0"   Margin = "2"> One </Button>
        <Button   Grid.Row = "0" Grid.Column = "1"   Margin = "2" > Two </Button>
        <Button   Grid.Row = "1" Grid.Column = "0"   Margin = "2"> Three </Button>
        <Button   Grid.Row = "1" Grid.Column = "1"   Margin = "2"> Four </Button>
        <Button   Grid.Row = "2" Grid.Column = "0"   Margin = "2"> Five </Button>
        <Button   Grid.Row = "2" Grid.Column = "1"   Margin = "2"> Six </Button>
</Grid>
```

运行上述代码后，页面显示效果如图 3.21 所示。第 0 行第 0 列的高度(Height＝"1＊")是加权百分比，表示占总数(1＋2＋3＝6)中的 1 份，即六分之一；第 1 行第 1 列的高度占六分之二，第 2 行第 2 列的高度占六分之三。同理，可求 Width 宽度所占的百分比，各为二分之一。

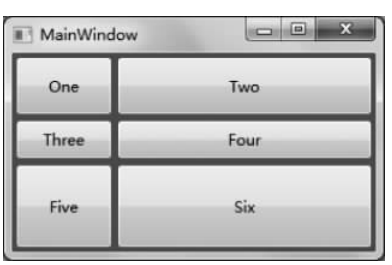

图 3.20　设置 Grid 附加属性调整布局结构

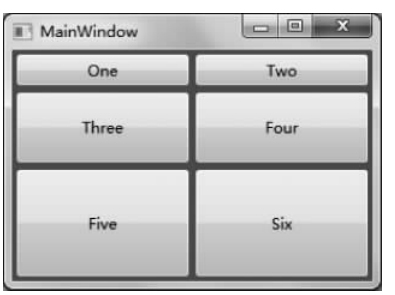

图 3.21　Uniform Grid 改变元素顺序改变布局结构

Height 属性和 Width 属性可以被设置成绝对值、百分比和 Auto。当采用绝对值时，像素是默认单位，可省略。

设置行高或者列宽时，除了可以使用像素作为单位外，还能使用厘米(Centimeter)、英寸(Inch)和点(Point)。它们与像素之间的换算关系如下。

- 1cm＝(96/2.54)pixel；
- 1in＝96pixel；

- 1pt=(96/72)pixel。

3.3.3 共享尺寸组

共享尺寸组是使用 Grid 布局时,位于同一列的子元素具有相同的尺寸大小。若在一个窗体中,允许一组控件具有相同的尺寸,尺寸大小由最宽的那个控件来决定。

设置共享尺寸,需要两个步骤。首先,在某一列或行上设置 SharedSizeGroup 属性;在控件上设置“IsSharedSizeScope="True"”。

为了演示这个效果,对位于同一窗口的 4 个 Button 实现共享尺寸。XAML 代码如下。

```
<StackPanel>
    <Grid IsSharedSizeScope = "True">
            <Grid.RowDefinitions>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width = "Auto" SharedSizeGroup = "a"/>
                <ColumnDefinition Width = "Auto" SharedSizeGroup = "a"/>
            </Grid.ColumnDefinitions>
            <Button Grid.Row = "0" Grid.Column = "0">One</Button>
            <Button Grid.Row = "0" Grid.Column = "1">Two</Button>
            <Button Grid.Row = "1" Grid.Column = "0">Three(which is longer)</Button>
            <Button Grid.Row = "1" Grid.Column = "1">Four</Button>
    </Grid>
</StackPanel>
```

运行上述代码后,页面显示效果如图 3.22 所示。尽管列中设置了 Width="Auto",但是 4 个 Button 全与最宽的那个 Button 保持一致,因为列上设置了共享尺寸分组(SharedSizeGroup="a"),并且在 Grid 中设置“IsSharedSizeScope="True"”。

将代码中的“IsSharedSizeScope="True"”和“SharedSizeGroup="a"”去掉,运行后,页面显示效果如图 3.23 所示。Button(Two)与 Button(Four)不再与最宽的 Button 保持一致。

图 3.22 设置共享尺寸的 Grid

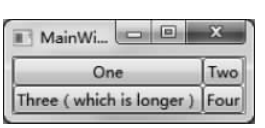

图 3.23 未设置共享尺寸的 Grid

3.3.4 跨越行和列

前面的例子,用 Grid.Row 和 Grid.Column 附加属性把子元素定位到单元格。还可以使用另外两个附加属性 Grid.RowSpan 和 Grid.ColumnSpan,让子元素跨越多个单元格。下面使用 Grid 设计计算器的页面,XAML 代码如下。

```
<Grid Background = "Green" >
```

```
        <Grid.RowDefinitions>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <TextBox Grid.Row="0" Grid.ColumnSpan="4" Margin="2,4,2,2"/>
        <Button  Grid.Row="1" Grid.Column="0"  Margin="2">1</Button>
        <Button  Grid.Row="1" Grid.Column="1"  Margin="2">2</Button>
        <Button  Grid.Row="1" Grid.Column="2"  Margin="2">3</Button>
        <Button  Grid.Row="1" Grid.Column="3"  Margin="2">+</Button>
        <Button  Grid.Row="2" Grid.Column="0"  Margin="2">4</Button>
        <Button  Grid.Row="2" Grid.Column="1"  Margin="2">5</Button>
        <Button  Grid.Row="2" Grid.Column="2"  Margin="2">6</Button>
        <Button  Grid.Row="2" Grid.Column="3"  Margin="2">-</Button>
        <Button  Grid.Row="3" Grid.Column="0"  Margin="2">7</Button>
        <Button  Grid.Row="3" Grid.Column="1"  Margin="2">8</Button>
        <Button  Grid.Row="3" Grid.Column="2"  Margin="2">9</Button>
        <Button  Grid.Row="3" Grid.Column="3" Margin="2">*</Button>
        <Button  Grid.Row="4" Grid.ColumnSpan="2" Margin="2">0</Button>
        <Button  Grid.Row="4" Grid.Column="2" Margin="2">.</Button>
        <Button  Grid.Row="4" Grid.Column="3" Margin="2">/</Button>
        <Button  Grid.Row="5" Grid.ColumnSpan="2" Margin="2">Del</Button>
        <Button  Grid.Row="5" Grid.Column="2" Grid.ColumnSpan="2" Margin="2">=</Button>
</Grid>
```

运行上述代码，页面显示效果如图 3.24 所示。Grid 布局 6 行 4 列网格。页面是由一个 TextBox、17 个 Button 构成。

其中，< TextBox Grid.Row="0" Grid.ColumnSpan="4" Margin="2,4,2,2"/>这条代码中的“Grid.Row="0"”用来设置 TextBox 位于第 0 行；“Grid.ColumnSpan="4"”设置 TextBox 占据 4 列，即从第 0 列～第 3 列；“Margin="2,4,2,2"”设置 TextBox 与四周边界的左、上、右、下相距四周的间隙分别是 2 像素、4 像素、2 像素、2 像素。

修改上述代码中的两条语句：

```
<Button  Grid.Row="4" Grid.ColumnSpan="2" Margin="2">0</Button>
<Button  Grid.Row="5" Grid.ColumnSpan="2" Margin="2">Del</Button>
```

修改后的语句：

```
<Button  Grid.Row="4" Grid.RowSpan="2" Margin="2">0</Button>
<Button  Grid.Row="4" Grid.RowSpan="2" Grid.Column="1" Margin="2">Del</Button>
```

运行修改后的代码，页面显示效果如图 3.25 所示。其中“Grid.RowSpan＝"2"”实现跨越 2 行。

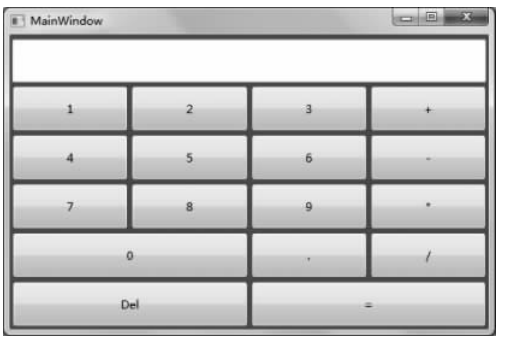

图 3.24　Grid 布局计算器页面

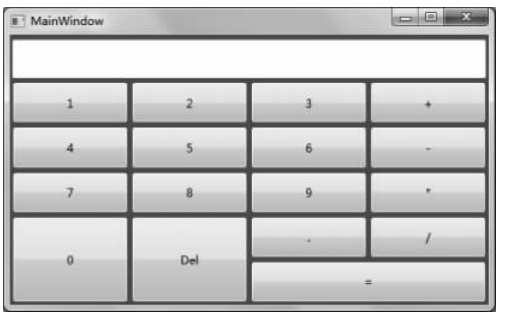

图 3.25　设置附加属性的计算器页面

3.3.5　GridSplitter

GridSplitter 是网格分割线。它支持用户在运行时编辑行或列，当用户移动分割线时，还可以改变行高列宽。在很多聊天软件的页面布局都含有分割线。接下来，设计常用的“一上二下式”布局。该布局的上面是一个菜单栏，下面是两列网格，两列网格间是一条分割线。在布局中的子元素全部用 Button，XAML 代码如下。

```
<DockPanel>
        <Button Height = "20" DockPanel.Dock = "Top"> Menu </Button>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width = "3 * "/>
            <ColumnDefinition Width = "7 * "/>
        </Grid.ColumnDefinitions>
            <Button VerticalContentAlignment = "Center"
                    HorizontalContentAlignment = "Center"> Column1 </Button>
            <GridSplitter Width = "3"></GridSplitter>
            <Button VerticalContentAlignment = "Center" Grid.Column = "1"
                    HorizontalContentAlignment = "Center" > Column2 </Button>
    </Grid>
</DockPanel>
```

运行上述代码，页面显示效果如图 3.26 所示，当程序处在运行状态下，用户可以根据需要，移动 Column1 与 Column2 之间的 GridSplitter。

将图 3.26 页面中的 Column2 中加上一条垂直分割线，XAML 代码如下。

```
<DockPanel>
        <Button Height = "20" DockPanel.Dock = "Top"> Menu </Button>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width = "3 * "/>
                <ColumnDefinition Width = "7 * "/>
            </Grid.ColumnDefinitions>
```

```
            <Button VerticalContentAlignment = "Center"
                    HorizontalContentAlignment = "Center"> Column1 </Button>
            <GridSplitter Width = "3"></GridSplitter>
            <Button VerticalContentAlignment = "Center" Grid.Column = "1"
                    HorizontalContentAlignment = "Center" > Column2 </Button>
            <Grid Name = "GridRightColumn" Grid.Row = "0" Grid.Column = "1">
                <Grid.RowDefinitions>
                    <RowDefinition></RowDefinition>
                    <RowDefinition></RowDefinition>
                </Grid.RowDefinitions>
                <Button> Row1 </Button>
                <GridSplitter Height = "3"  HorizontalAlignment = "Stretch"
                    VerticalAlignment = "Bottom"></GridSplitter>
                <Button Grid.Row = "1"> Row2 </Button>
            </Grid>
        </Grid>
    </DockPanel>
</Window>
```

运行上述代码，页面显示效果如图 3.27 所示。分析布局结构发现，在 DockPanel 下是 Grid，Grid 下还有一个 Grid。其中，语句“< Grid Name＝"GridRightColumn" Grid.Row＝"0" Grid.Column＝"1">”指的是为第 2 个 Grid 命名，并使用了第一个 Grid 的附加属性定位；“< GridSplitter Height＝"3" HorizontalAlignment＝"Stretch" VerticalAlignment＝"Bottom">”语句定义垂直分割线水平拉伸。

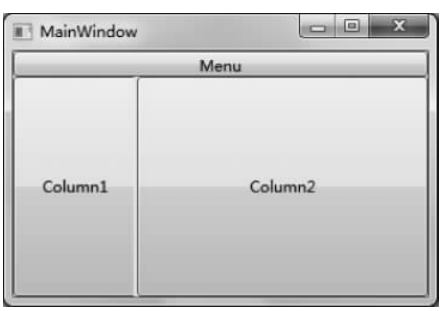

图 3.26　GridSplitter 实现水平分割

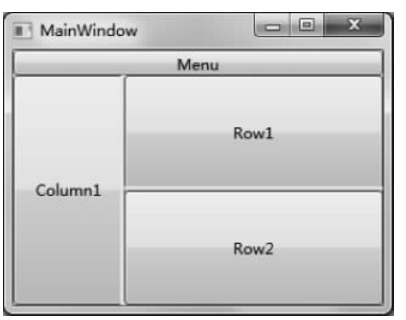

图 3.27　GridSplitter 实现水平＋垂直分割

3.4　小　　结

本章重点介绍了 WPF 布局原则以及布局面板中的 Canvas、DockPanel、StackPanel、WrapPanel 和 UniformGrid 的适用场合；详细介绍了 Grid 从结构中分离布局、尺寸模型、共享尺寸组和跨越行列等特征，并演示了 Grid 的多种用法。本章中的案例涉及 Windows 窗口页面、用户搜索页面、用户登录页面、计算器页面及常用布局。但是布局内容远不止这些，当了解更多的控件以后，可以做出个性化的布局。一个好的布局，能让 UI 根据用户的需求调整屏幕大小，窗口的大小，并根据内容来改变尺寸。在以后的章节中还会陆续补充布局的相关知识。

习题与实验 3

1. 简述流式布局的特点。

2. 分析如图 3.28 所示的 QQ 聊天页面，用 Grid 布局页面，页面中出现的元素用 Button 表示，其效果如图 3.29 所示。

图 3.28　QQ 聊天页面

图 3.29　Grid 布局 QQ 聊天页面

操作提示：QQ 聊天页面的结构：先分为左右结构，右半部分又分成上、中、下结构，用 Grid 嵌套实现。在内部页面共用 3 根 GridSplitter 分割线。最下面的分割线，需要指定这个属性“Grid.Row＝"1"”。