在代码中创建一个Hi-Lo-Open-Close图表

以编程方式创建一个HiLowOpenClose 图表，通过以下代码：

C#

```
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries() {
XValueBinding = new System.Windows.Data.Binding("NumberOfDay"),
HighValueBinding = new System.Windows.Data.Binding("High"),
LowValueBinding = new System.Windows.Data.Binding("Low"),
OpenValueBinding = new System.Windows.Data.Binding("Open"),
CloseValueBinding = new System.Windows.Data.Binding("Close"),
SymbolStrokeThickness = 1, SymbolSize = new Size(5, 5)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="119093ab-9226-4c28-8d46-74df4a211561"><ac:plain-text-body><![CDATA[} ds.PlotElementLoaded += (s, e) => {
PlotElement pe = (PlotElement)s; double open = (double)pe.DataPoint["OpenValues"]; double close = (double)pe.DataPoint["Cl
oseValues"]; if (open > close)
]]></ac:plain-text-body></ac:structured-macro>
{ pe.Fill = green; pe.Stroke = green;
```

} else { pe.Fill = red; pe.Stroke = red; }
};
例如，如果值由应用程序以集合的方式提供，则您可以使用以下代码创建数据系列：

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries(); ds.XValuesSource = dates; // 数值沿着X轴方向 ds.OpenValuesSource
= open; ds.CloseValuesSource = close; ds.HighValuesSource = hi; ds.LowValuesSource = lo;
// 向图表添加数据系列
chart.Data.Children.Add(ds);
// 设置图表类型
chart.ChartType = isCandle ? ChartType.Candle
: ChartType.HighLowOpenClose;
```

另一种选择是使用数据绑定。例如，如果可用的数据为StockQuote对象的集合，比如说：

C#

```
public class Quote
{ public DateTime Date { get; set; } public double Open { get; set; } public double Close { get; set; } public double High
{ get; set; } public double Low { get; set; }
}
```

Then the code that creates the data series would be as follows:

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries();
// 绑定全部的五个值
ds.XValueBinding = new Binding("Date"); // 数值沿着X轴方向 ds.OpenValueBinding = new Binding("Open"); ds.CloseValueBinding
= new Binding("Close"); ds.HighValueBinding = new Binding("High");
```

```
ds.LowValueBinding = new Binding("Low");
// 向图表添加数据系列
chart.Data.Children.Add(ds);
// 设置图表类型
chart.ChartType = isCandle ? ChartType.Candle
: ChartType.HighLowOpenClose;
```

在代码中创建一个Hi-Lo-Open-Close图表 以编程方式创建一个HiLowOpenClose 图表，通过以下代码：

C#

```
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries() {
XValueBinding = new System.Windows.Data.Binding("NumberOfDay"),
HighValueBinding = new System.Windows.Data.Binding("High"),
LowValueBinding = new System.Windows.Data.Binding("Low"),
OpenValueBinding = new System.Windows.Data.Binding("Open"),
CloseValueBinding = new System.Windows.Data.Binding("Close"),
SymbolStrokeThickness = 1, SymbolSize = new Size(5, 5)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="3e783df7-9b72-40cd-b70d-99a0af509237"><ac:plain-text-body><![CDATA[} ds.PlotElementLoaded += (s, e) => {
PlotElement pe = (PlotElement)s; double open = (double)pe.DataPoint["OpenValues"]; double close = (double)pe.DataPoint["Cl
oseValues"]; if (open > close)
]]></ac:plain-text-body></ac:structured-macro>
{ pe.Fill = green; pe.Stroke = green; } else { pe.Fill = red; pe.Stroke = red; }
};
```

例如，如果值由应用程序以集合的方式提供，则您可以使用以下代码创建数据系列：

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries(); ds.XValuesSource = dates; // 数值沿着X轴方向
```

```
ds.OpenValuesSource = open; ds.CloseValuesSource = close; ds.HighValuesSource = hi; ds.LowValuesSource = lo;

// 向图表添加数据系列
chart.Data.Children.Add(ds);
// 设置图表类型
chart.ChartType = isCandle ? ChartType.Candle
: ChartType.HighLowOpenClose;
```

另一种选择是使用数据绑定。例如，如果可用的数据为StockQuote对象的集合，比如说：

C#

```
public class Quote
{ public DateTime Date { get; set; } public double Open { get; set; } public double Close { get; set; } public double High
{ get; set; } public double Low { get; set; } }
```

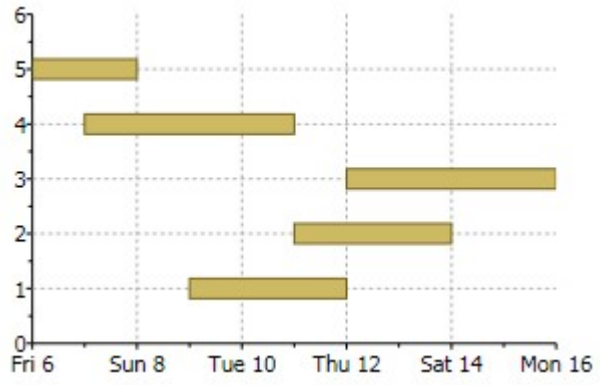
Then the code that creates the data series would be as follows:

C#

```
// 创建数据系列
HighLowOpenCloseSeries ds = new HighLowOpenCloseSeries();
// 绑定全部的五个值
ds.XValueBinding = new Binding("Date"); // 数值沿着X轴方向 ds.OpenValueBinding = new Binding("Open"); ds.CloseValueBinding
= new Binding("Close"); ds.HighValueBinding = new Binding("High"); ds.LowValueBinding = new Binding("Low");
// 向图表添加数据系列
chart.Data.Children.Add(ds);
// 设置图表类型
chart.ChartType = isCandle ? ChartType.Candle
: ChartType.HighLowOpenClose;
```

甘特图

甘特图使用类型为HighLowSeries的对象作为数据系列（在线文档 'HighLowSeries
类'）。每一个数据系列表示一个单独的任务，每一个任务都有一个起始值和结束值的集合。简单的任务有一个开始值和一个结束值。由多个顺序的子任务组成的任务有多个开始和结束值对。
下图表示一个甘特图：



为了演示甘特图，让我们从定义一个Task对象开始：

C#

```
class Task
{ public string Name { get; set; } public DateTime Start { get; set; } public DateTime End { get; set; } public bool IsGro
up { get; set; } public Task(string name, DateTime start, DateTime end, bool isGroup)
{
Name = name;
Start = start;
End = end;
IsGroup = isGroup;
}
}
```

接下来，我们定义一个方法，创建一组将显示为甘特图的任务对象：

C#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="f7e49328-0cba-4dd0-bfb4-eff6069c7c9d"><ac:plain-text-body><![CDATA[
```

```
Task[] GetTasks() { return new Task[]
]]></ac:plain-text-body></ac:structured-ma
{ new Task("Alpha", new DateTime(2008,1,1)
ew DateTime(2008,2,15), true), new Task("S
", new DateTime(2008,1,1), new DateTime(20
1,15), false), new Task("Prototype", new D
Time(2008,1,15), new DateTime(2008,1,31),
se), new Task("Document", new DateTime(200
,1), new DateTime(2008,2,10), false), new
k("Test", new DateTime(2008,2,1), new Date
e(2008,2,12), false), new Task("Setup", ne
ateTime(2008,2,12), new DateTime(2008,2,15
false),
```

```
new Task("Beta", new DateTime(2008,2,15), new DateTime(2008,3,15), true), new Task("WebPage", new DateTime(2008,2,15), new D
ateTime(2008,2,28), false), new Task("Save bugs", new DateTime(2008,2,28), new DateTime(2008,3,10), false), new Task("Fix
bugs", new DateTime(2008,3,1), new DateTime(2008,3,15), false), new Task("Ship", new DateTime(2008,3,14), new DateTime(2008,
3,15), false),
};
}
```

现在这个任务已经建立，我们已经准备好创建甘特图：

C#

```
private void CreateGanttChart()
{
// 清除当前图表
c1Chart.Reset(true);
// 设置图表类型
c1Chart.ChartType = ChartType.Gantt;
// 生成图表
var tasks = GetTasks(); foreach (var task in tasks)
{
// 为每一个任务创建一个数据系列
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="d7cc6824-da9f-40d5-88d2-2aa297a32ace"><ac:plain-text-body><![CDATA[ var ds = new HighLowSeries(); ds.Label =
task.Name; ds.LowValuesSource = new DateTime[] { task.Start }; ds.HighValuesSource = new DateTime[] { task.End };
ds.SymbolSize = new Size(0, task.IsGroup ? 30 : 10);
]]></ac:plain-text-body></ac:structured-macro>
// 添加数据系列至图表
c1Chart.Data.Children.Add(ds); }
// 沿着Y轴显示任务名称
c1Chart.Data.ItemNames =
(from task in tasks select task.Name).ToArray();
// 自定义Y轴
var ax = c1Chart.View.AxisY; ax.Reversed = true; ax.MajorGridStroke = null;
// 自定义X轴
ax = c1Chart.View.AxisX; ax.MajorGridStrokeDashes = null; ax.MajorGridFill = new SolidColorBrush(Color.FromArgb(20, 120,
120, 120)); ax.Min = new DateTime(2008, 1, 1).ToOADate();
}
```

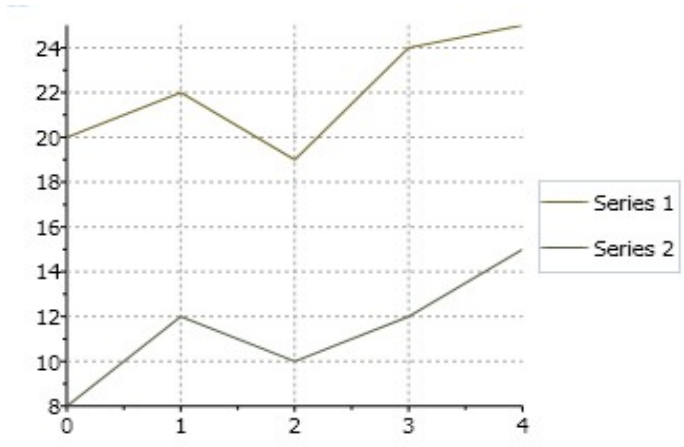
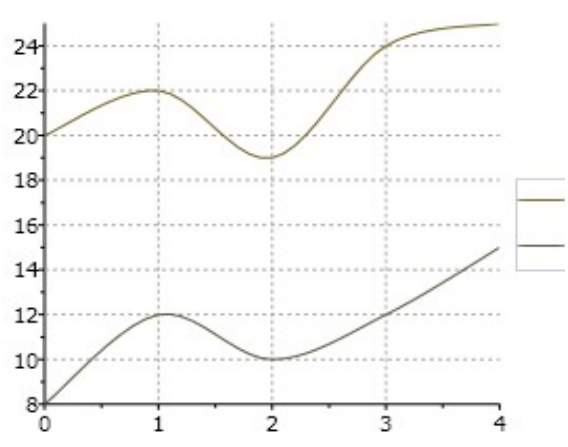
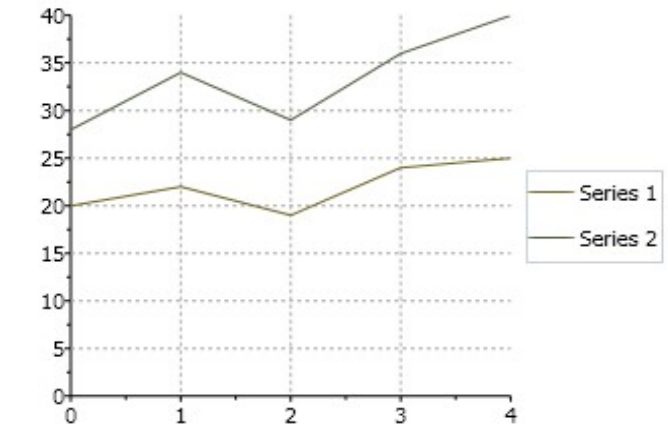
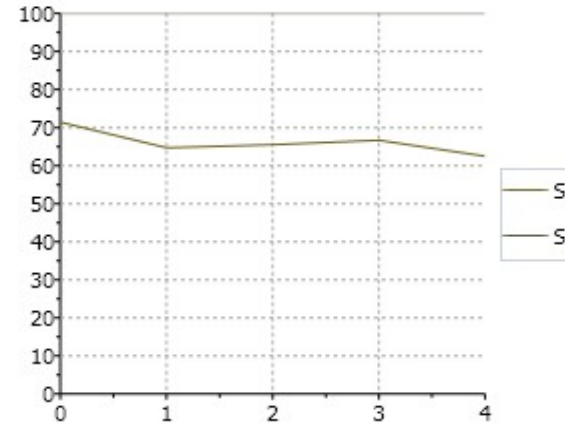
在清除完C1Chart并设置了图表类型之后，该代码枚举每一个任务并为每一个任务创建了一个HighLowSeries对象。除了设置系列标签，LowValuesSource以及HighValuesSource
属性之外，该代码通过SymbolSize属性以设置每一个任务条形的高度。在这个示例中，我们将一些任务定义为"组"任务，并使它们比一般任务更高。
接下来，我们使用LINQ语句来提取任务名称并将其设置给ItemNames属性。这将使得C1Chart沿Y轴显示任务名称。最后，该代码还对坐标轴进行自定义。Y轴被设置为反向显示，因此第一个任务显示在图表的顶部。坐标轴同时被设置显示垂直的网格线和交替色的条带。
在标记语言中创建甘特图为创建甘特图，使用下面的XAML标记：

XAML

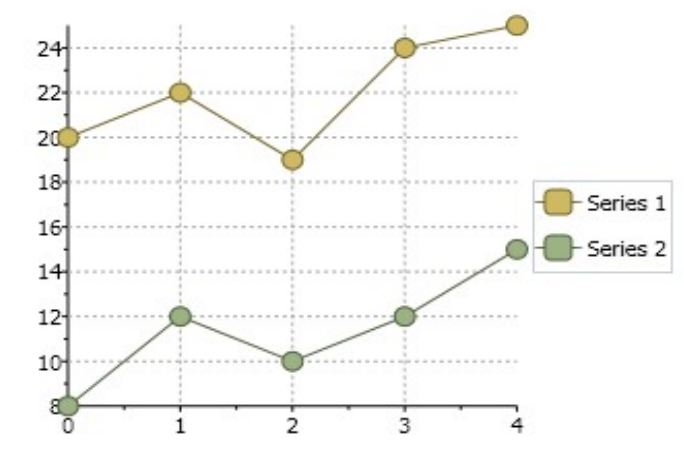
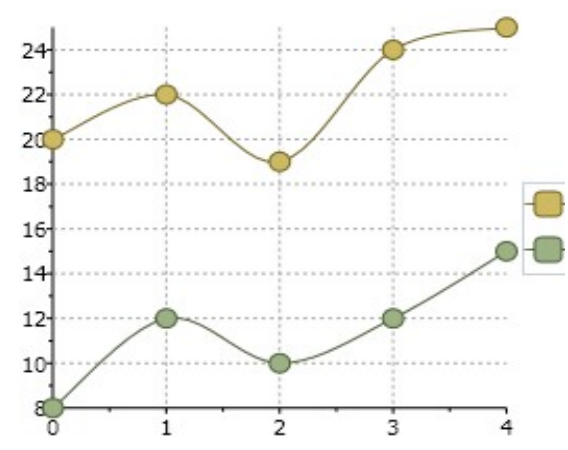
```
<c1chart:C1Chart Margin="0" Name="c1Chart1" xmlns:sys="clr-namespace:System;assembly=mscorlib">

<c1chart:C1Chart.Resources>
<x:Array x:Key="start" Type="sys:DateTime" >
<sys:DateTime>2008-6-1</sys:DateTime>
<sys:DateTime>2008-6-4</sys:DateTime>
<sys:DateTime>2008-6-2</sys:DateTime>
</x:Array>
<x:Array x:Key="end" Type="sys:DateTime">
<sys:DateTime>2008-6-10</sys:DateTime>
<sys:DateTime>2008-6-12</sys:DateTime>
<sys:DateTime>2008-6-15</sys:DateTime>
</x:Array>
</c1chart:C1Chart.Resources>
<c1chart:C1Chart.Data>
<c1chart:ChartData>
<c1chart:ChartData.Renderer>
<c1chart:Renderer2D Inverted="True" ColorScheme="Point"/>
</c1chart:ChartData.Renderer>
<c1chart:ChartData.ItemNames>Task1 Task2 Task3</c1chart:ChartData.ItemNames>
<c1chart:HighLowSeries HighValuesSource="{StaticResource end}"
LowValuesSource="{StaticResource start}"/>
</c1chart:ChartData>
</c1chart:C1Chart.Data>
<c1chart:C1Chart.View>
<c1chart:ChartView>
<c1chart:ChartView.AxisX>
<c1chart:Axis IsTime="True" AnnoFormat="d"/>
</c1chart:ChartView.AxisX>
</c1chart:ChartView>
</c1chart:C1Chart.View>
</c1chart:C1Chart>
```

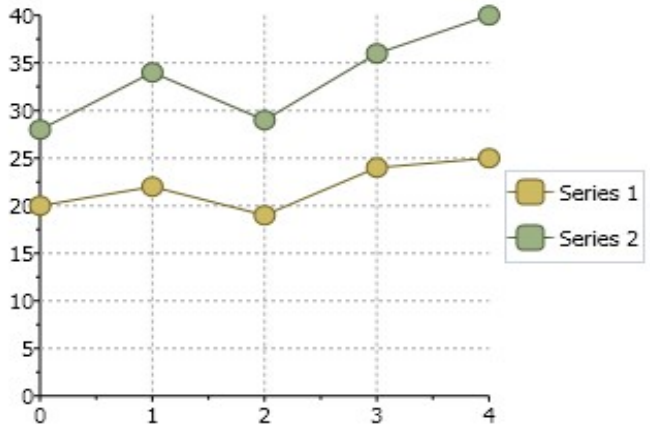
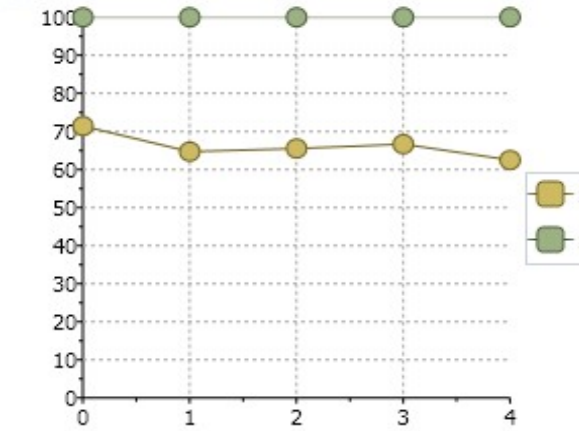
折线图

WPF及Silverlight版Chart支持以下类型的折线图：
Line
LineSmoothed
LineStacked
LineStacked100pc
LineSymbols
LineSymbolsSmoothed
LineSymbolsStacked
LineSymbolsStacked100pc 常规折线图常规折线图

Line	LineSmoothed
	
LineStacked	LineStacked100pc
	

LineSymbol Charts

LineSymbols	LineSymbolsSmoothed
	

LineSymbolsStacked	LineSymbolsStacked100pc
	

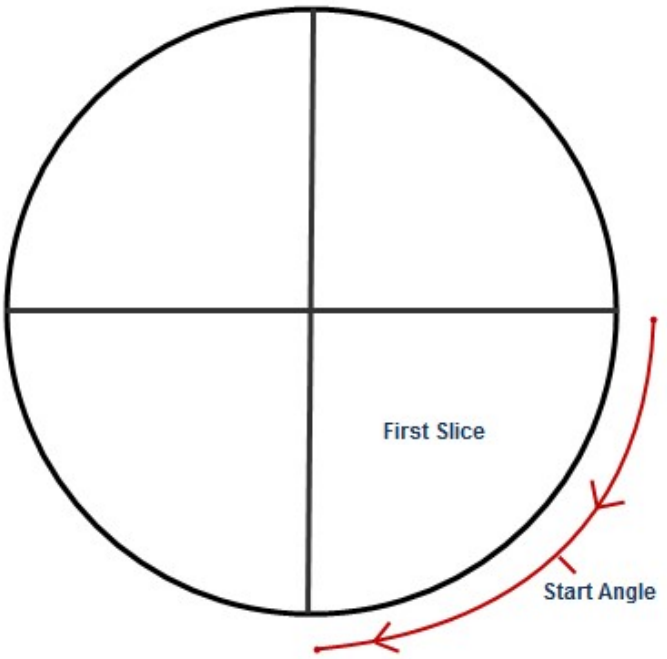
饼图

饼图通常用于显示简单的值。它们具有视觉上的吸引力，而且经常会显示三维效果，如阴影和旋转。
和C1Chart中其他的图表类型相比，饼图所具有的一个显著不同，每一个系列代表饼图中的一块。因此，你永远不会有一个单一系列的饼图（他们将是一个整圆）。在大多数情况下，饼图有多个系列（每片一个），在每个系列中仅具有一个数据点。C1Chart将具有多个数据点的系列在图表中展示为多个饼的形状。
如果您想通过XAML标记创建一个饼图，则标记语言应当类似以下的代码：

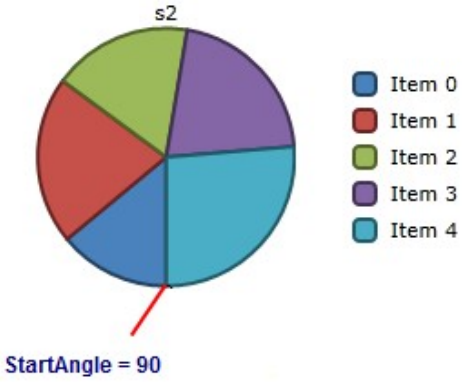
XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="Pie">
<c1chart:C1Chart.Data>
<c1chart:ChartData>
<c1chart:ChartData.ItemNames>P1 P2 P3 P4 P5</c1chart:ChartData.ItemNames>

<c1chart:DataSeries Values="20 22 19 24 25" />
</c1chart:ChartData>
</c1chart:C1Chart.Data>
<c1chart:C1ChartLegend DockPanel.Dock="Right" />
</c1chart:C1Chart>
```

有一些特殊的属性，可以帮助您自定义您的饼图控件。通过使用SetStartingAngle或Direction属性，您可以改变饼图显示的起始角度或者改变切片显示的方向。您同时也可以通过BasePieRenderer 属性将一个饼的切片从主图表分离显示，以高亮显示该值。
起始角起始角
PieOptions.SetStartingAngle属性定义饼图中的一个切片的位置。饼图第一片的位置始终从90度位置开始显示。启示角从90度角位置顺时针方向计算。

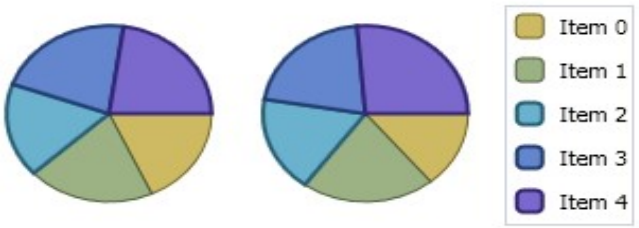


使用PieOptions.SetStartingAngle属性指定第一个系列的切片显示的位置。您可以在下图中看到当设置起始角为90度时会发生什么：

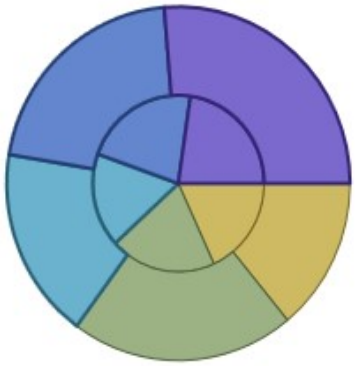


Direction
使用 PieOptions.Direction 附加属性指定图表绘制椭圆弧的方向。选项包括：
逆时针：指定在逆时针方向（负角）方向绘制圆弧。
顺时针：指定以顺时针方向（正角度）绘制圆弧。
SeriesLabelTemplate
使用PieOptions.SeriesLabelTemplate属性指定一个饼图的系列标签模板。创建一个DataTemplate
以设置系列标签的内容。分离显示饼图分离显示饼图
一个饼图的切片可以通过爆炸分离进行强调，从余下的饼图切片中突出显示。使用系列的Offset属性以设置分离显示的切片距离饼图中心的偏移量。该偏移量以饼图半径的百分比表示。
WPF及Silverlight版Chart支持以下类型的饼图：
Pie
Pie Stacked
3D Pie
3D Doughnut Pie
3D Exploded Pie
3D Exploded Doughnut Pie
Doughnut Pie
Exploded Pie
Exploded Doughnut Pie
常规饼图常规饼图 （（WPF 及及 Silverlight））

Pie	PieStacked



DoughnutPie

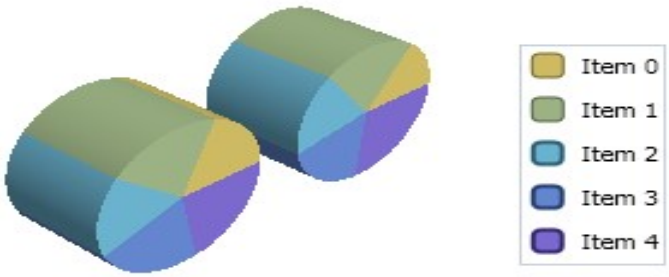


ExplodedPie

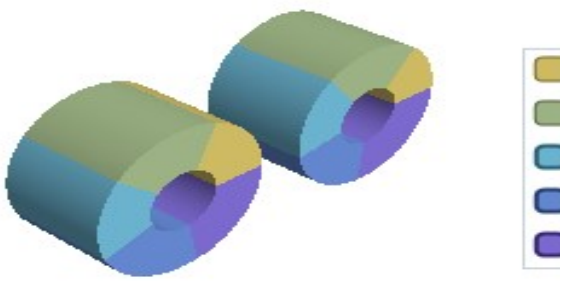
ExplodedDoughnutPie

3D Pie Charts (仅 WPF)

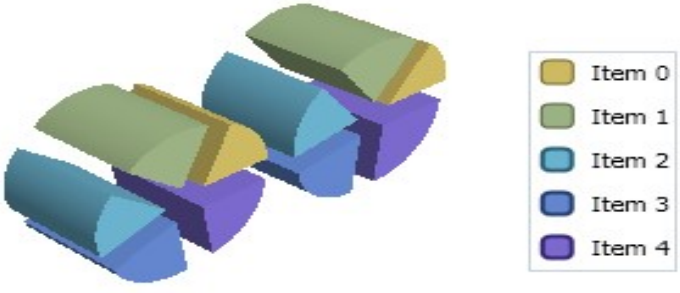
3DPie



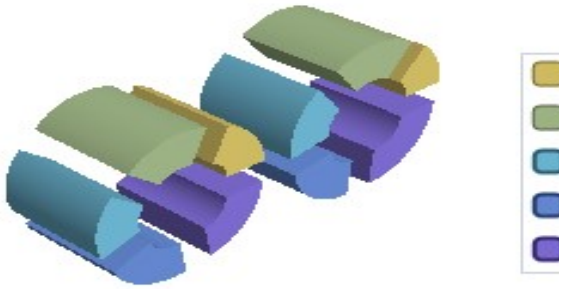
3DDoughnutPie



3DExplodedPie



3DExplodedDoughnutPie



增加连接线，以防止饼重叠您可以通过PlotElement.LabelLine 附加属性添加连接线，如以下XAML代码所示：

XAML

```
<c1:DataSeries.PointLabelTemplate>
<DataTemplate>
<Border BorderBrush="DarkGray" BorderThickness="1" Background="LightGray" c1:PlotElement.LabelAlignment="Auto" c1:PlotElem
ent.LabelOffset="30,0">
<TextBlock Text="{Binding Value, StringFormat=0}" />
<c1:PlotElement.LabelLine>
<Line Stroke="LightGray" StrokeThickness="2" />
</c1:PlotElement.LabelLine>
</Border>
</DataTemplate>
</c1:DataSeries.PointLabelTemplate>
```

添加标签到饼图
要将多个值添加到饼图标签中，可以创建如下标签模板：
XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="Pie">
<c1chart:C1Chart.Resources>
<DataTemplate x:Key="lbl">
<StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="{Binding Path=Name}" />
<TextBlock Text="=" />
<TextBlock Text="{Binding Path=Value}" />
</StackPanel>
<TextBlock Text="{Binding Path=PercentageSeries,Converter={x:Static c1chart:Converters.Format}, ConverterParameter=#.#%}"/>
</StackPanel>
</DataTemplate>
</c1chart:C1Chart.Resources>
<c1chart:C1Chart.Data>
<c1chart:ChartData>
<c1chart:ChartData.ItemNames>P1 P2 P3 P4 P5</c1chart:ChartData.ItemNames>
<c1chart:DataSeries Values="20 22 19 24 25" PointLabelTemplate="
{StaticResource lbl}" />
</c1chart:ChartData>
</c1chart:C1Chart.Data>
<c1chart:C1ChartLegend DockPanel.Dock="Right" />
</c1chart:C1Chart>
```

改变所有切片的偏移量
要更改所有饼图切片的偏移量，使用以下代码

C#

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="27b53df5-67be-440d-a228-5284eabdb248"><ac:plain-text-body><![CDATA[
```

```
chart.DataContext = new double[] { 1, 2, 3
}; chart.ChartType = ChartType.Pie;
chart.Loaded += (s, e) =>
((BasePieRenderer)chart.Data.Renderer).Off
= 0.1;
```

设置三维饼图的默认视角
要设置"三维饼图"的默认视角，使用以下代码：

C#

```
chart.View.Camera.Transform = new RotateTransform3D(new AxisAngleRotation3D(new

Vector3D(0,0,1),45));
```

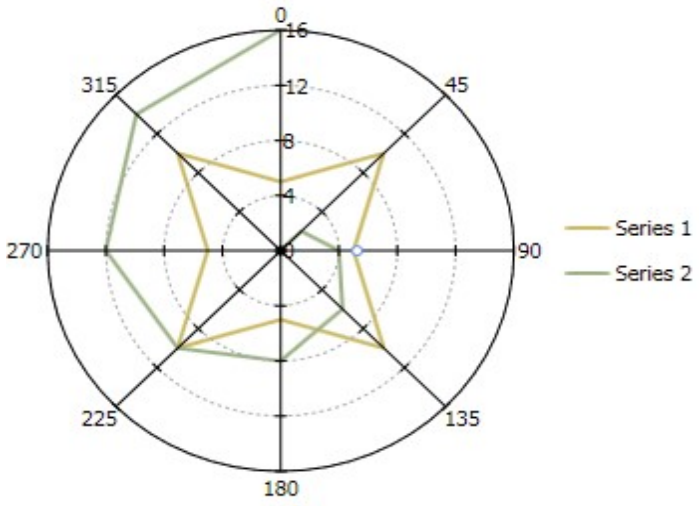
极坐标图表

极坐标图表将每一个系列作为（θ，r）的值在X和Y轴上进行绘制。
θ-从图表起始位置的旋转量。θ可以以角度指定（默认值），或者以弧度表示。r-表示到图表原点的距离
因为X轴是一个圆，X轴的最大值和最小值是固定的。
极坐标图表无法和其他图表类型组合在同一张图表区域。
下面的XAML标记为XYDataSeries指定数据值，并用此数据系列创建图像：

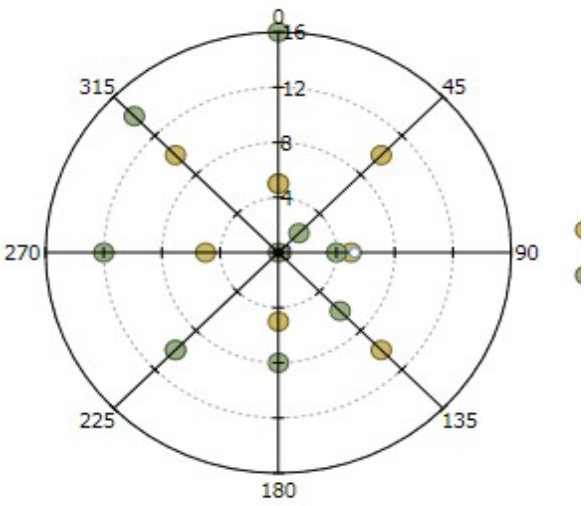
XAML

```
<c1chart:C1Chart Name="c1Chart1" ChartType="PolarLinesSymbols">
<c1chart:C1Chart.Data>
<c1chart:ChartData>
<c1chart:XYDataSeries Label="Series 1" Values="5 10 5 10 5 10 5 10 5" XValues="0 45 90 135 180 225 270 315 0"/>
<c1chart:XYDataSeries Label="Series 2" Values="0 2 4 6 8 10 12 14 16"
XValues="0 45 90 135 180 225 270 315 0"/>
</c1chart:ChartData>
</c1chart:C1Chart.Data>
<c1chart:C1ChartLegend DockPanel.Dock="Right" /> </c1chart:C1Chart>
```

PolarLines



PolarSymbols



PolarLinesSymbols