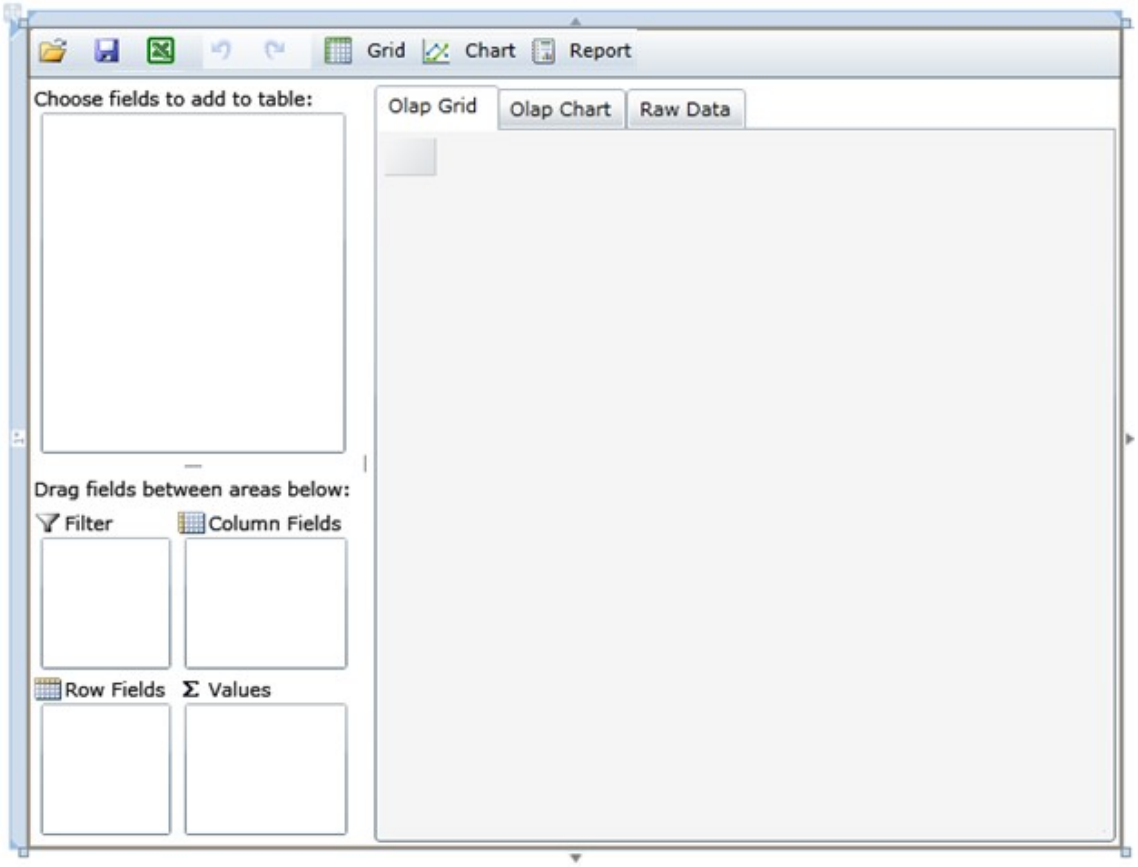
WPF_C1Olap快速入门

本节介绍WPF及Silverlight应用程序的代码演示，并介绍其常用功能。

一个简单的OLAP应用程序

创建一个最简单的C1OLAP应用程序，需要先创建一个新的WPF或Silverlight的应用程序，然后再拖拽一个C1OlapPage控件到页面中，允许通过移除所有的边距和对齐方式设置填将C1OlapPage控件充整个页面。



现在让我们为应用程序设置一个数据源。
在本示例中，我们从XML数据文件中载入Northwind产品数据，我们使用的ComponentOne Data 可以为我们提供熟悉的DataSet和DataTable对象，我们也使用ComponentOne Zip 来解压客户端上的XML压缩文件。

Visual Basic

```
' 从压缩文件中载入数据
Dim ds = New DataSet()
Dim asm = Assembly.GetExecutingAssembly()
Using s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip")
Dim zip = New C1ZipFile(s)
```

```
Using zr = zip.Entries(0).OpenReader()
' 载入数据
ds.ReadXml(zr) End Using
End Using
```

C#

```
// 从压缩文件中载入数据
var ds = new DataSet(); var asm = Assembly.GetExecutingAssembly(); using (var s = asm.GetManifestResourceStream("OlapQuick
Start.nwind.zip"))
{ var zip = new C1ZipFile(s);
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="3b23c8de-64a3-4d00-b20a-1f8bc1fcb43e"><ac:plain-text-body><![CDATA[ using (var zr =
zip.Entries[0].OpenReader())
]]></ac:plain-text-body></ac:structured-macro>
{
// 载入数据 ds.ReadXml(zr); }
}
```

此时我们可以对C1OlapPage控件的DataSource属性进行简单的设置，我们可以使用此控件的数据绑定方法。

```
Visual Basic
```

```
' 为olap页面绑定数据
_c1OlapPage.DataSource = ds.Tables(0).DefaultView
```

```
C#
```

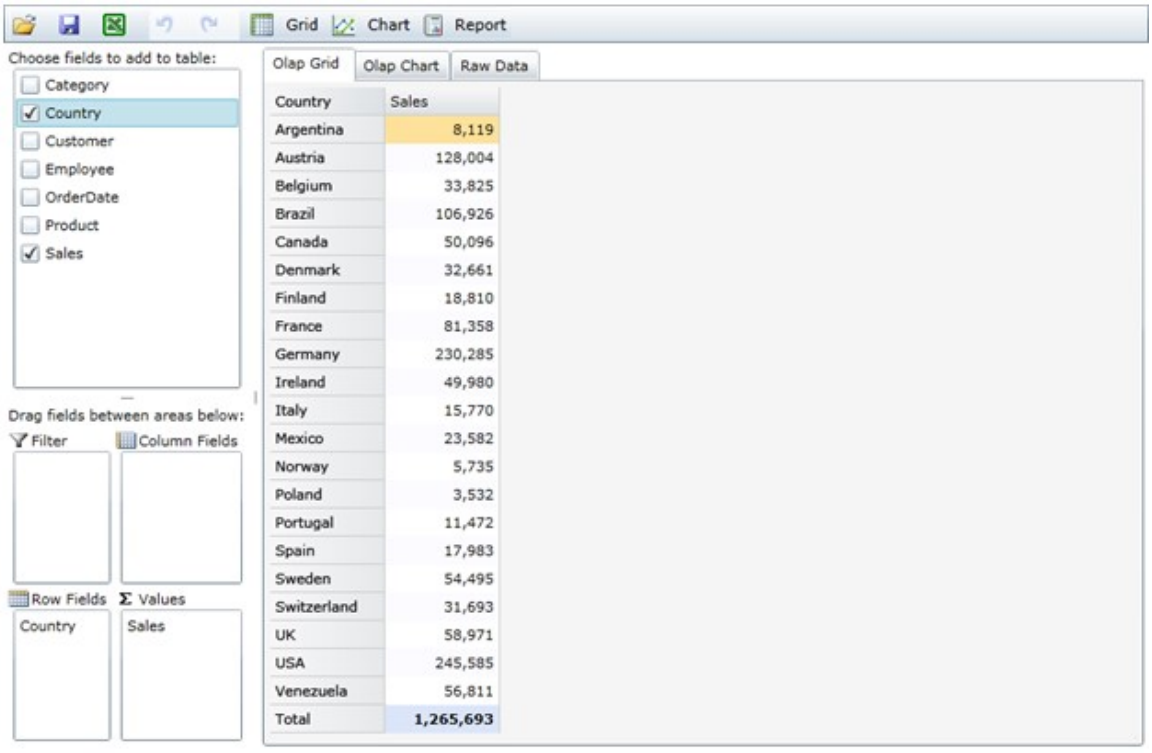
```
// 为olap页面绑定数据
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="d62bb42f-8c75-4f1d-9af0-4ee49bc0594e"><ac:plain-text-body><![CDATA[_c1OlapPage.DataSource
= ds.Tables[0].DefaultView;
```

```
]]></ac:plain-text-b
```

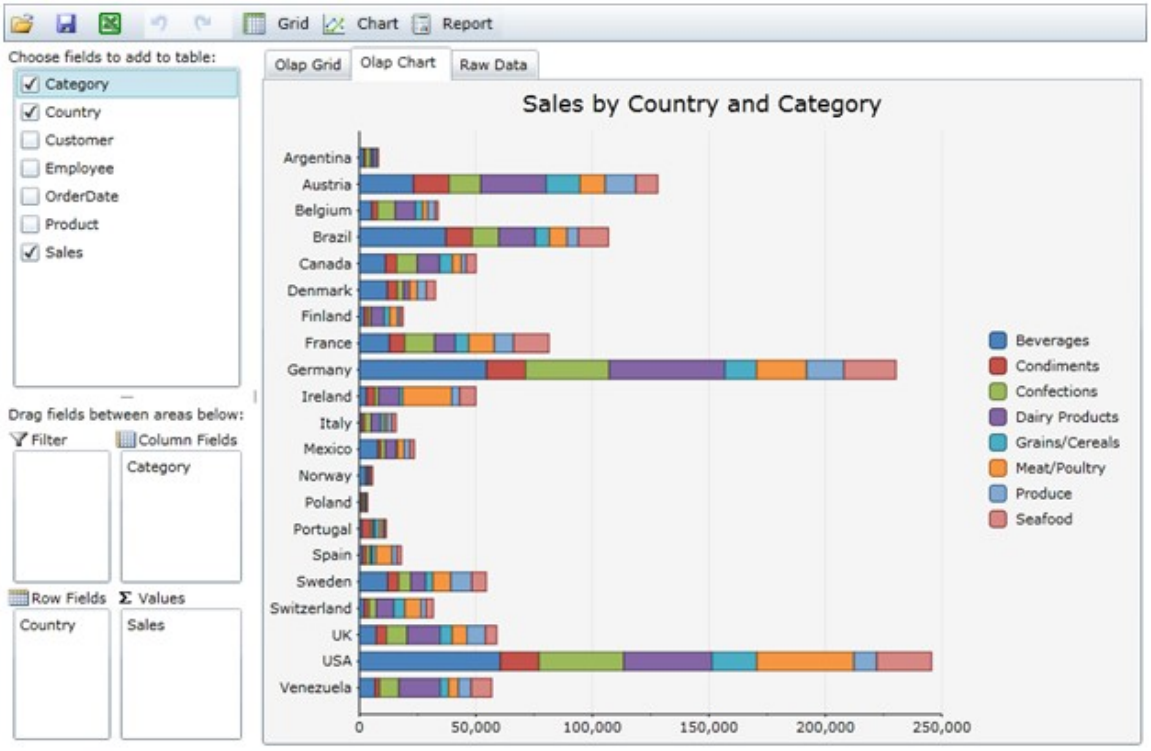
应用现已准备就绪了，以下各节将介绍默认提供的基本功能，并不用做任何编码就可以配置我们的数据源。

创建OLAP视图

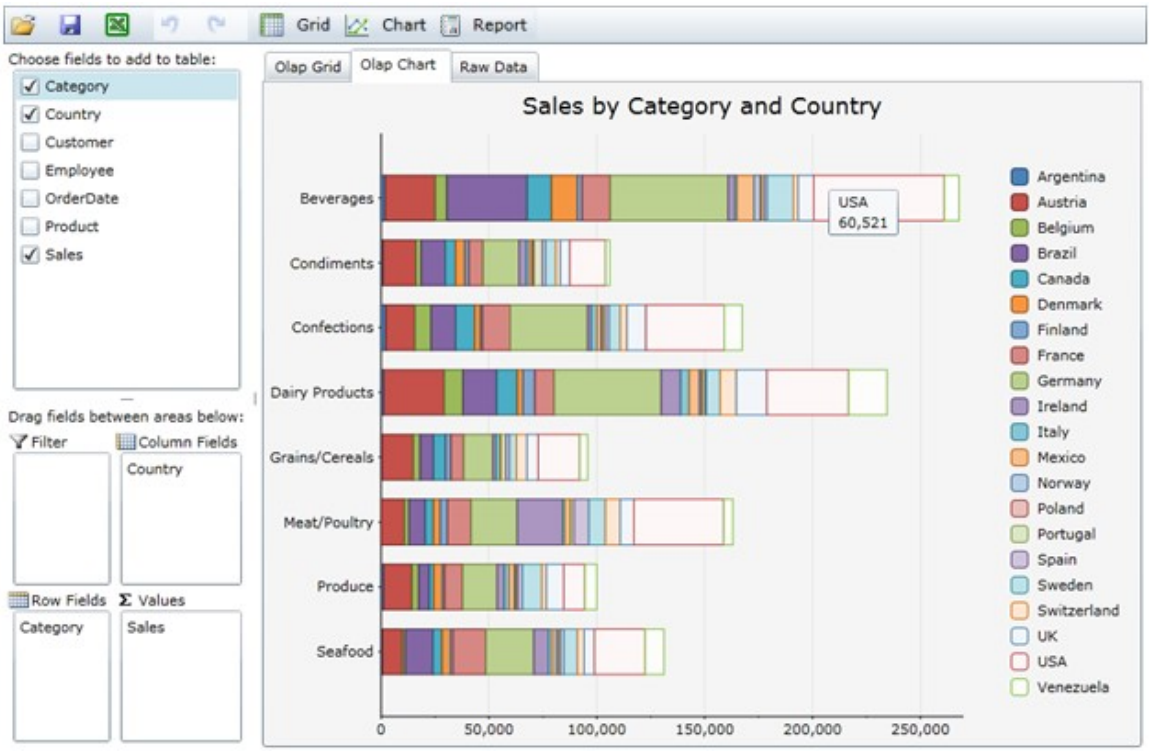
运行应用程序，此时您会看到类似于Microsoft Excel的用户界面，拖动"Country"字段到"Row Fields"列表，"Sales"到"Values"列表，您将会看到按照国家分类的一个价格总和，其显示如下：



点击"Olap Chart"选项卡，此时您将看到相同数据的图表格式，可以看出该产品的主要客户为US，Germany，和Austria。
现在拖动"Category"字段到"Column Fields"列表可以看到一个新的汇总，显示每个国家每个产品这段时间的销量，如果您仍然选择图表选项卡，此时显示与前者类似，其柱状图被切分割开以显示每个销售人员的销量：



将鼠标移到图表上，你会看到工具提示会显示产品的名字及销售的数量。
现在通过拖动"Category"与"Country"字段来交换列表的位置来创建一个新的视图，此时新的图表将强调产品，而不是国家：

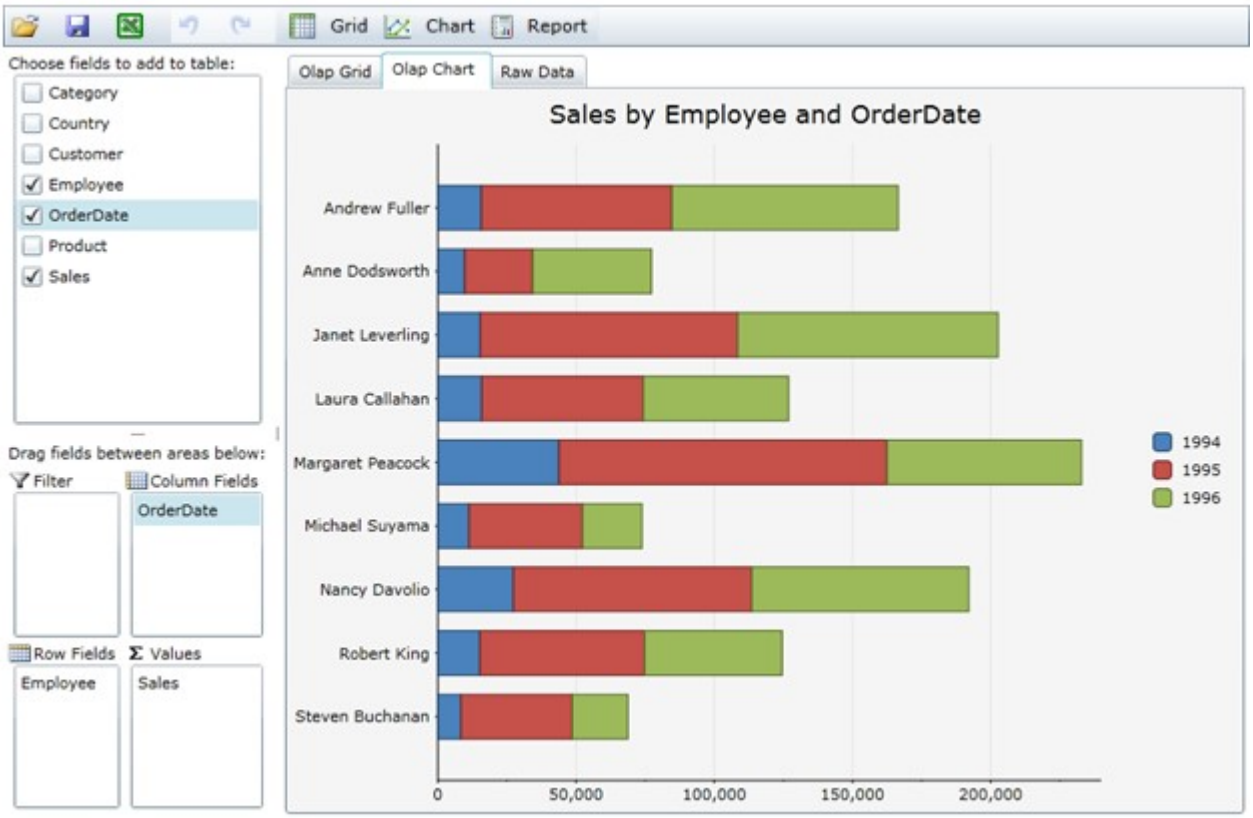


图表显示，这一阶段的饮料销售量分析得到的结果是最高的，其次是乳制品。
随着我们对视图进行更改，C1OlapPanel控件将保存比较高的记录，我们可以单击C1OlapPanel菜单中的撤消按钮以返回之前创建的视图。

汇总数据

让我们用不同的方法创建一个新的视图以演示如何轻松的对数据汇总。
这次，我们将"Employee"字段拖动到"Row Field"列表，"OrderDate"字段拖动到"Column Field"列表中，视图中则包含一列订单，这一列数据代表每一天接到的订单，然而这些信息不是非常有用，因为有太多的列来显示趋势，因此我们希望能想按月或年汇总数据。
一种方法是对数据源进行修改，通过创建新的SQL查询或使用LINQ，这两种技术将在后面几节中叙述。另一种方法是简单地修改"OrderDate"字段，要做到这一点，请先右键单击"OrderDate"字段并选择"Field

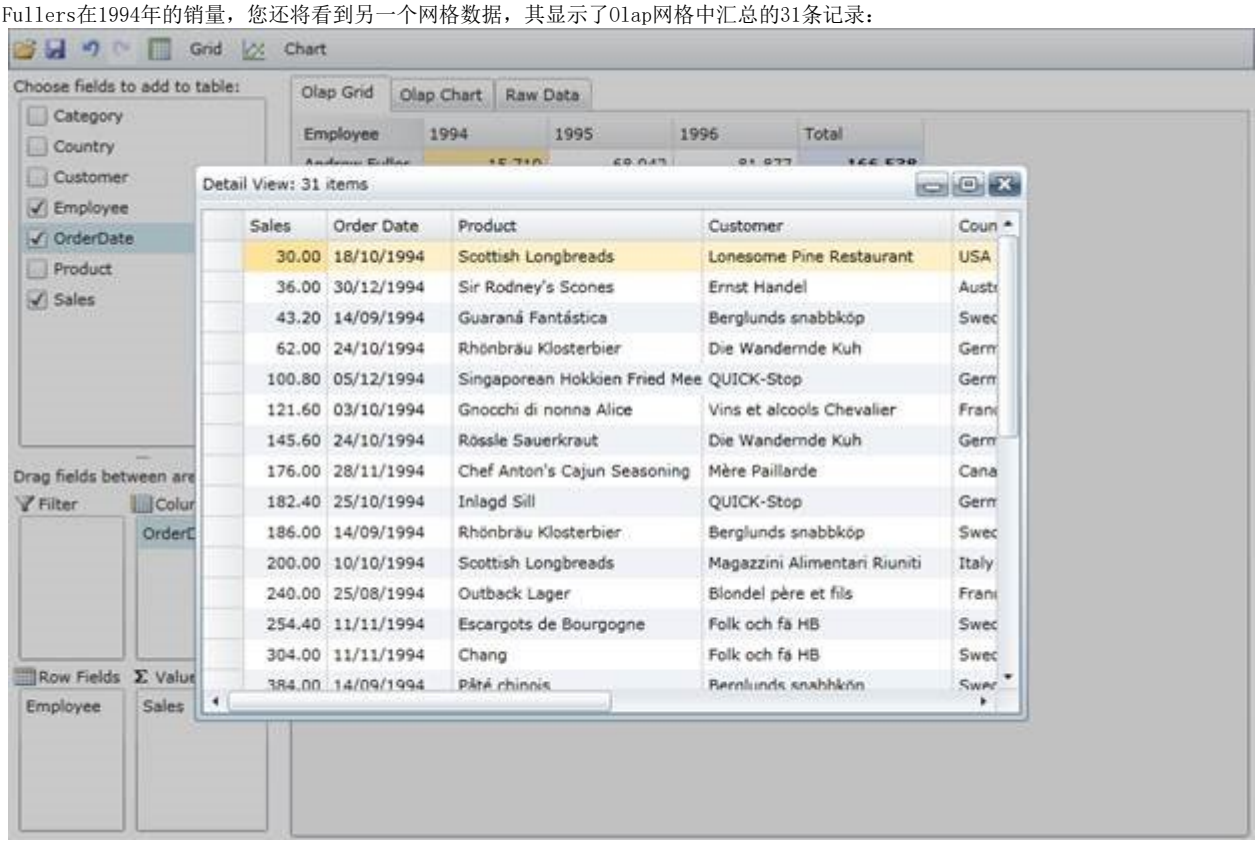
Setting"菜单，然后在对话框中选择"Format"选项卡，选择"Custom"格式并输入"yyyy"，最后单击确定。
此时数据将按照年汇总的格式显示，其OLAP图表显示如下：



如果您想要按照月份或者星期来查看销量，您仅需要改变格式为"MMMM"或"dddd"。

数据回溯

正如我们之前提到的，OLAP网格的每一个单元格代表了数据源几组记录的一个汇总，通过双击单元格您可以看到网格中单元格的基础数据。
想要查看此数据，需要点击"Olap Grid"选项卡并双击网格的第一个单元，其代表了Andrew
Fullers在1994年的销量，您还将看到另一个网格数据，其显示了Olap网格中汇总的31条记录：



自定义C1OlapPage

前面的示例展示如何仅使用C1OlapPage控件和极少的代码创建一个完整的OLAP应用程序，这对于用户非常方便，但在大多数情况下你可能希望在一定程度上自定义应用程序以及用户界面。

在代码中配置字段

OLAP应用程序的一个主要优势是交互性，用户可以轻松并快速的创建和修改视图，并可以立即看到结果。与Excel相类似，WPF及及Silverlight版版OLAP的界面也十分友好且对话简单。
但在某些情况下，你可能想要使用代码来配置视图，WPF及及Silverlight版版OLAP也提供简单且功能强大的对象模型，尤其是Field与Filter类。
下面的示例将演示如何在WPF及及Silverlight版版OLAP中配置一个视图。

Visual Basic

```
' 按照用户和产品显示销量
Dim olap = _c1OlapPage.OlapPanel.OlapEngine
olap.DataSource = ds.Tables(0).DefaultView olap.BeginUpdate() olap.RowFields.Add("Country")
olap.ColumnFields.Add("Category")
```

```
olap.ValueFields.Add("Sales") olap.Fields("Sales").Format = "n0" olap.EndUpdate()
```

C#

```
// 按照用户和产品显示销量
var olap = _c1OlapPage.OlapPanel.OlapEngine;
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="e58e0bbf-50b2-4d4c-9964-df1156a81aaf"><ac:plain-text-body><![CDATA[olap.DataSource
= ds.Tables[0].DefaultView; olap.BeginUpdate(); olap.RowFields.Add("Country");
olap.ColumnFields.Add("Category"); olap.ValueFields.Add("Sales"); olap.Fields["Sales"].Format
= "n0"; olap.EndUpdate();
```

```
]]></ac:plain-text-body></a
```

首先需要调用BeginUpdate方法，可以实现对输出表格的自动更新，它可以对行，列及字段集自动的添加字段，而用户不需要手动执行此操作，因此我们可以隐藏应用程序的C1OlapPanel部分，这段代码也适用数字格式的"Sales" 字段，并最终调用Endupdate方法。
如果您运行示例，您将看到一个类似于第一个示例中的OLAP视图。
接下来，我们将使用WPF及及Silverlight版版OLAP对象模型改变订单日期和扩展价格的格式：

Visual Basic

```
' 格式化订单日期
Dim field = olap.Fields("OrderDate") field.Format = "yyyy"
' 格式化扩展价格并改变分类汇总类型
' 用以显示扩展价格的平均值（而不是总价）
field = olap.Fields("Sales") field.Format = "c" field.Subtotal =
C1.Olap.Subtotal.Average
```

C#

```
// 格式化订单日期
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="504873ed-820f-42f1-b30e-ddf6375aa21c"><ac:plain-text-body><![CDATA[var
field = olap.Fields["OrderDate"]; field.Format = "yyyy";
]]></ac:plain-text-body></ac:structured-macro>
// 格式化扩展价格并改变分类汇总类型
// 用以显示扩展价格的平均值（而不是总价）
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="5a3b3663-274b-4ea3-8621-cf1954c8502f"><ac:plain-text-body><![CDATA[field
= olap.Fields["Sales"]; field.Format = "c"; field.Subtotal =
C1.Olap.Subtotal.Average;
```

```
]]></ac:plain-text-body></ac:structur
```

Field集包含数据源中指定的所有字段，代码会检索Field集中的每个字段，此时它将分配所需的值
到Format和Subtotal属性中，Format采用常规的.NET 格式字符串，Subtotal决定在OLAP视图中如何聚合数值。在默认情况下聚合为合计，但还有许多其他聚合方式包括平均值、 最大值、 最小值以及标准偏差和方差。
现在假设你只感兴趣数据的某个子集，比如说某几个产品和某一年，用户可以用鼠标右键单击字段，并对它们进行筛选，你在代码中实现，如下所示：

Visual Basic

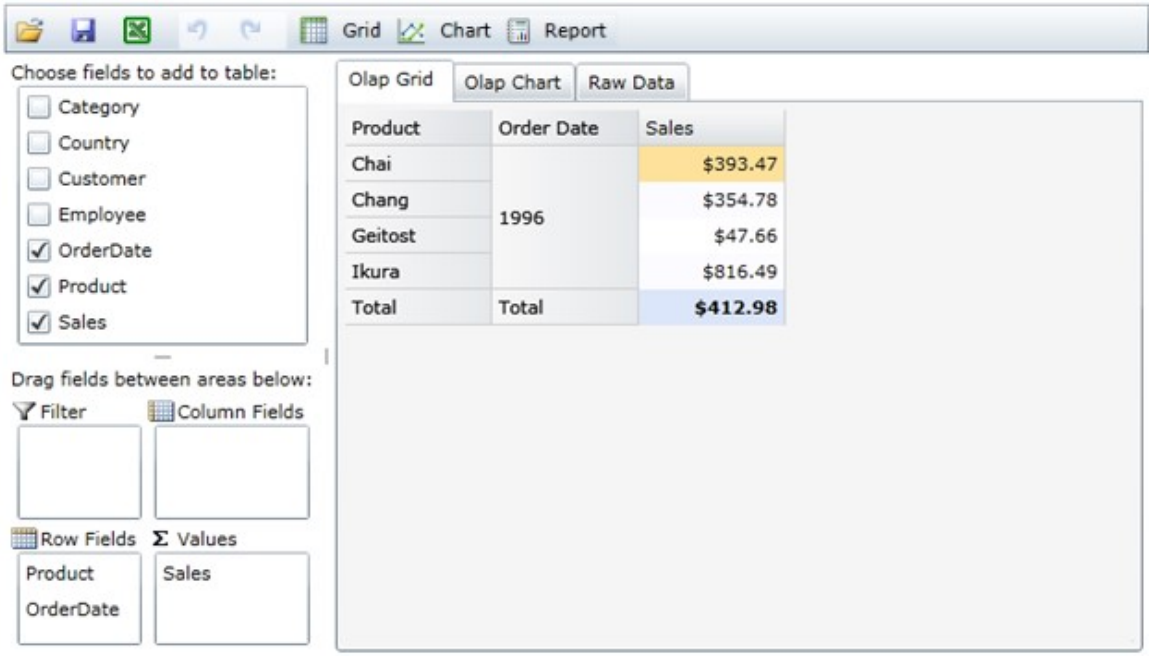
```
' 格式化订单日期和扩展价格
' 没有变化...
' 应用值筛选来只显示几个产品
Dim filter As C1.Olap.C1OlapFilter = olap.Fields("Product").Filter filter.Clear()
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(",",C)
' 应用条件筛选来显示几个日期
filter = olap.Fields("OrderDate").Filter filter.Clear()
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="2a2ea103-8d4f-4f16-b7b5-9d4b2c67ddaa"><ac:plain-text-body><![CDATA[filter.Condition1.[Operator]
= C1.Olap.ConditionOperator.GreaterThanOrEqualTo filter.Condition1.Parameter = New DateTime(1996, 1, 1)
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="40dca520-5c89-4884-92ae-27b53be35e94"><ac:plain-text-body><![CDATA[filter.Condition2.[Operator]
= C1.Olap.ConditionOperator.LessThanOrEqualTo filter.Condition2.Parameter = New DateTime(1996, 12, 31)
filter.AndConditions = True
```

]]></ac:plain-

C#

```
// 格式化订单日期和扩展价格
// 没有变化...
// 应用值筛选来只显示几个产品
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="15eee2ab-d059-471a-a485-81fed4bb1f0d"><ac:plain-text-body><![CDATA[C1.Olap.C1OlapFilter filter
= olap.Fields["Product"].Filter; filter.Clear();
]]></ac:plain-text-body></ac:structured-macro>
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(',');
// 应用条件筛选来显示几个日期
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="fc175bff-9490-454d-ba0b-e5daffe61b2e"><ac:plain-text-body><![CDATA[filter = olap.Fields["Order
Date"].Filter; filter.Clear();
]]></ac:plain-text-body></ac:structured-macro>
filter.Condition1.Operator = C1.Olap.ConditionOperator.GreaterThanOrEqualTo; filter.Condition1.Parameter =
new DateTime(1996, 1, 1); filter.Condition2.Operator = C1.Olap.ConditionOperator.LessThanOrEqualTo;
filter.Condition2.Parameter = new DateTime(1996, 12, 31); filter.AndConditions = true;
```

代码可以通过检索相关联的"Product"字段的C1OlapFilter对象，然后清空筛选器并将设置它的ShowValues，此属性会被筛选并显示数组值，在WPF及及Silverlight版版OLAP中我们称之为"value filter"。
接下来，代码将检索关联"OrderDate"字段的筛选器。这一次，我们需要显示的值为指定的某一年，但我们不需要枚举出指定某年的所有天。取而代之，我们将使用定义两个条件的"condition filter"。
第一个条件指定"OrderDate" 应该大于或等于1996年1月1日，第二个条件指定"OrderDate" 应该小于或等于1996年12 月31日，AndConditions属性指定应该如何应用的第一个与第二个条件(和或运算符)。在这种情况下，我们希望的日期的两个条件都为真，所以AndConditions应该设置为true。
如果您再次运行该项目，您应看到下图显示:



本地存储中持久化OLAP视图

虽然加载的默认视图已经可以满足用户的需求，但用户可能会厌倦并改变它，在Silverlight中我们可以将一些简单的

数据存储在独立的存储空间并保存会话之间的视图，我们首先会创建一个默认视图，并在独立存储空间中保持会话，IsolatedStorageSettings.ApplicationSettings类可以您轻松的保存和加载应用的设置，在默认情况下，独立的存储限制为1MB，但OLAP视图的大小不受此影响。
在此示例中，我们将保存当前应用Exit事件的视图，因此当用户关闭应用程序时，任何用户的自定义都会被自动的保存。

Visual Basic

```
' 当关闭应用时保存当前的视图到存储空间
Private Sub Current_Exit(sender As Object, e As EventArgs)
Dim userSettings = IsolatedStorageSettings.ApplicationSettings userSettings(VIEWDEF_KEY) = _c1OlapPage.ViewDefinition
userSettings.Save()
End Sub
```

C#

```
// 当关闭应用时保存当前的视图到存储空间
void Current_Exit(object sender, EventArgs e)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="b3938b56-3dfe-4542-9ec9-a8280e1dc36f"><ac:plain-text-body><![CDATA[{ var userSettings =
IsolatedStorageSettings.ApplicationSettings; userSettings[VIEWDEF_KEY] = _c1OlapPage.ViewDefinition; userSettings.Save();
]]></ac:plain-text-body></ac:structured-macro>
}
```

注意这里我们使用唯一的索引访问应用程序设置，我们在ViewDefinition属性中保存数据，其内容是一个XML格式的字符串以定义此数据集的视图，应用程序中的任何点我们可以通过转换第二行代码来还原OLAP视图，下一步我们将从独立存储空间中加载视图。

Visual Basic

```
Const VIEWDEF_KEY As String = "C1OlapViewDefinition"
```

C#

```
const string VIEWDEF_KEY = "C1OlapViewDefinition";
```

增加一行代码声明VIEWDEF_KEY常量，方便我们能使用相同的唯一值来访问我们存储的数据视图。
Application.Current.Exit += Current_Exit;
以上代码附加到我们的exit事件中，在关闭应用之前它将会被触发，接下来我们将通过转换需要保存的代码来载入独立存储中的视图。

Visual Basic

```
' 初始化olap视图
Dim userSettings = IsolatedStorageSettings.ApplicationSettings
If userSettings.Contains(VIEWDEF_KEY) Then
' 载入独立存储空间中最后一次被使用的olap视图
_c1OlapPage.ViewDefinition = TryCast(userSettings(VIEWDEF_KEY), String)
End If
```

C#

```
// 初始化olap视图
var userSettings = IsolatedStorageSettings.ApplicationSettings; if (userSetting
s.Contains(VIEWDEF_KEY))
{
// 载入独立存储空间中最后一次被使用的olap视图
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="0866b181-6ab8-49a8-89df-41b73d4e9cc8"><ac:plain-text-body><![CDATA[
_c1OlapPage.ViewDefinition = userSettings[VIEWDEF_KEY] as string; }
```

```
]]></ac:plain-text-body></ac:structured-ma
```

如果您现在运行项目，您将会看到通过代码创建的默认视图，如果您对视图做出改变，请先关闭应用程序并重启，此时您将会看到您的更改已经被保存了。

创建一个预定义视图

除了利用XML字符串表示的ViewDefinition属性来获取或设置当前视图，C1OlapPage控件还提供ReadXml与WriteXml方法来帮助您以文件或流的形式保存视图，当您单击内置菜单中"加载"和"保存"按钮时，C1OlapPage会自动调用这些方法。这些方法可以帮助您轻松的实现视图的预定义，您首先需创建一些视图并通过点击"保存"按钮来保存它们，对于本示例，我们将创建五个视图以显示销售额:

1. 产品与国家
2. 员工与国家
3. 员工与月份

4. 员工与星期
5. 员工与年

一旦您创建并保存了所有的视图，即创建了一个带有单个"OlapViews"节点且名为"DefaultViews.xml"的新XML文件，此时复制并粘贴所有您的默认视图到文档中，接下来添加一个"id"选项到每一个视图中并为每一个视图分配唯一的名字，这个名字将会显示在用户界面中（这在C1OlapGrid中并不需要），您的XML文件可能与下列相似：

XAML

```
<OlapViews>
<C1OlapPage id="Product vs Country">
<!-- 视图定义省略... -->
<C1OlapPage id="Employee vs Country">
<!-- 视图定义省略... -->
<C1OlapPage id="Employee vs Month">
<!-- 视图定义省略... -->
<C1OlapPage id="Employee vs Weekday">>

<!-- 视图定义省略... -->
<C1OlapPage id="Employee vs Year">
<!-- 视图定义省略... -->
</OlapViews>
```

现在添加此文件到项目中作为一个资源，首先需要添加一个新的文件叫到您的项目中，并命名为"Resource"，此时在解决方案视图中右键点击资源文件夹，并选择"Add Existing File..."选项，选择XML文件并点击OK。
现在视图已经被定义，我们需要在我们的菜单中显示它们以便用户能够选择，为实现此操作，请将下列代码添加到项目中：

Visual Basic

```
Public Sub New()
InitializeComponent()
'这里没有变化
'…
' 从XML资源中得到预定义视图
Dim views = New Dictionary(Of String, String)() Using s =
asm.GetManifestResourceStream("OlapQuickStart.Resources.OlapViews.xml") Using reader = XmlReader.Create(s)

' 读取预定义视图中的定义
While reader.Read()
If reader.NodeType = XmlNodeType.Element AndAlso reader.Name = "C1OlapPage" Then
```

Dim id = reader.GetAttribute("id") Dim def = reader.ReadOuterXml() views(id) = def End If End While End Using End Using ' 对预定义视图建立新的菜单 Dim menuViews = New C1MenuItem() menuViews.Header = "View" menuViews.Icon = GetImage("Resources/views.png") menuViews.VerticalAlignment = VerticalAlignment.Center ToolTipService.SetToolTip(menuViews, "Select a predefined Olap view.") For Each id As var In views.Keys Dim mi = New C1MenuItem() mi.Header = id mi.Tag = views(id) mi.Click += mi_Click menuViews.Items.Add(mi) Next ' 在页面主菜单中添加一个新的菜单 _c1OlapPage.MainMenu.Items.Insert(6, menuViews)End SubC#public MainPage(){ InitializeComponent(); //这里没有变化//… // 从XML资源中得到预定义视图 var views = new Dictionary<string, string>(); using (var s = asm.GetManifestResourceStream("OlapQuickStart.Resources.OlapViews.xml")) using (var reader = XmlReader.Create(s)) { // 读取预定义视图中的定义 while (reader.Read()) { if (reader.NodeType == XmlNodeType.Element && reader.Name == "C1OlapPage") {

```
<ac:structured-macro ac:name="unmigrated-wiki-markup"
ac:schema-version="1"
ac:macro-id="d8305216-a06f-4e54-a750-3bbecc7c04a5"><ac:plain
-text-body><![CDATA[
```

```
var id = reader.GetAttribute("id"); var def =
reader.ReadOuterXml(); views[id] = def;
]]></ac:plain-text-body></ac:structured-macro>
}
}
}
// 对预定义视图建立新的菜单
var menuViews = new C1MenuItem(); menuViews.Header = "View";
menuViews.Icon = GetImage("Resources/views.png");
menuViews.VerticalAlignment = VerticalAlignment.Center;
ToolTipService.SetToolTip(menuViews, "Select a predefined
Olap view."); foreach (var id in views.Keys)
<ac:structured-macro ac:name="unmigrated-wiki-markup"
ac:schema-version="1"
ac:macro-id="0f355ec0-3714-4ce3-8e80-1774496463f0"><ac:plain
-text-body><![CDATA[ { var mi = new C1MenuItem(); mi.Header
= id; mi.Tag = views[id]; mi.Click += mi_Click;
menuViews.Items.Add(mi);
]]></ac:plain-text-body></ac:structured-macro>
}
// 在页面主菜单中添加一个新的菜单
_c1OlapPage.MainMenu.Items.Insert(6, menuViews);
}
```

代码会载入OLAP预定义的XML文档，使用C1Menu创建一个新的下拉菜单，并为其添加视图的下拉选项，每一个菜单项Header属性中写入视图的名字，Tag属性中包含实际的XML节点，这个节点将在之后应用视图中被用户选择。但下拉菜单完成后，可以使用MainMenu属性在代码中将它添加到C1OlapPage，新的按钮会被添加到之前的几个按钮之后。
有一个简单的方法可以调用以上内容并载入新的菜单按钮图像，即GetImage方法。载入单张图像不需要大量的工作，然而如果您载入多张图像，您可能希望有一个可以被多次使用的常规方法。

```
Visual Basic
' 从URI从加载一个图像
Private Shared Function GetImage(name As String) As Image
Dim uri = New Uri(name, UriKind.Relative) Dim img = New Image() img.Source = New BitmapImage(uri) img.Stretch = Stretch.None img.VerticalAlignment = VerticalAlignment.Center img.HorizontalAlignment = HorizontalAlignment.Center Return img
End Function
C#
// 从URI中加载一个图像
static Image GetImage(string name)
{
var uri = new Uri(name, UriKind.Relative); var img = new Image(); img.Source = new BitmapImage(uri); img.Stretch = Stretch.None; img.VerticalAlignment = VerticalAlignment.Center; img.HorizontalAlignment = HorizontalAlignment.Center; return img;
}
```

当用户通过点击菜单项在C1OlapPage应用视图，这部分可以通过以下代码完成：

```
Visual Basic
' 应用一个预定义视图
Private Sub mi_Click(sender As Object, e As SourcedEventArgs)
Dim mi = TryCast(sender, C1MenuItem)
Dim viewDef = TryCast(mi.Tag, String)
_c1OlapPage.ViewDefinition = viewDef
End Sub
C#
// 应用一个预定义视图
void mi_Click(object sender, SourcedEventArgs e)
{ var mi = sender as C1MenuItem; var viewDef = mi.Tag as string; _c1OlapPage.ViewDefinition = viewDef;

}
```

代码会通过读取菜单的Tag属性检索OLAP在XML中的预定义，此时会将它分配给C1OlapPage.ViewDefinition属性。
如果您需要进一步的自定义，您也可以选择不使用C1OlapPage，并使用低级别的C1OlapPanel，C1OlapGrid及C1OlapChar控件来构建您的用户界面，C1OlapPage控件的源码被包含在包中并被用作起始点，在"创建一个自定义用户界面"节中的示例显示了它如何执行。

更新OLAP视图

您可能想要强制更新C1OlapPage或C1O来重新生成分析，您可以在C1OlapEngine中调用Update方法，为了能在您的UI中添加该功能，需要添加一个按钮及一个响应时间，通过以下代码来实现：

```
Visual Basic
' 重新生成olap视图
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)

_c1OlapPage.OlapPanel.OlapEngine.Update()
End Sub
C#
// 重新生成olap视图
void Button_Click(object sender, RoutedEventArgs e)
{
_c1OlapPage.OlapPanel.OlapEngine.Update();
}
```

条件格式

C1OlapGrid控件源于C1FlexGrid控件，因此您可以使用网格内单元格的自定义特性对基于内容的单元格应用样式，此示例展示了绿色背景下的100多个值的网格。
C1OlapGrid控件有一个CellFactory类，其主要功能是为网格中的每一个单元格创建显示，为了创建自定义单元格，您可以创建一个实现ICellFactory接口的类，并将该类分配给网格的CellFactory属性，像自定义列，ICellFactory自定义类可以实现高专业化和定制化应用，也可以实现通用的，可复用及可配置的类，通常情况下，ICellFactory自定义类会比自定义列更简便一些，因为ICellFactory类可以直接处理单元格。
以下代码实现了一个ConditionFactory类，并应用了一个自定义绿色背景且超过100个值的网格。

Visual Basic

```
Public Class ConditionalCellFactory
Inherits C1.Silverlight.FlexGrid.CellFactory
Public Overrides Function CreateCell(grid As C1FlexGrid, cellType__1 As
CellType, range As CellRange) As FrameworkElement
' 让基类执行大多数工作
Dim cell = MyBase.CreateCell(grid, cellType__1, range)
' 如果需要则应用绿色的背景
If cellType__1 = CellType.Cell Then
Dim cellValue = grid(range.Row, range.Column)
If TypeOf cellValue Is Double AndAlso CDbl(cellValue) > 100 Then
Dim border = TryCast(cell, Border)
border.Background = _greenBrush End If
End If
' 执行
Return cell
End Function
Shared _greenBrush As Brush = New SolidColorBrush(Color.FromArgb(&Hff, 88, 183,
112))
End Class
```

C#

```
public class ConditionalCellFactory : C1.Silverlight.FlexGrid.CellFactory
```

```
{
public override FrameworkElement CreateCell(C1FlexGrid grid, CellType cellType,
CellRange range)
{
// 让基类执行大多数工作
var cell = base.CreateCell(grid, cellType, range);
// 如果需要则应用绿色的背景
if (cellType == CellType.Cell)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="509943c4-b157-4480-aaa7-f2a56b1198b8"><ac:plain-text-body><![CDATA[ { var cellValue = grid[range.Row,
range.Column]; if (cellValue is double && (double)cellValue > 100)
]]></ac:plain-text-body></ac:structured-macro>
{ var border = cell as Border; border.Background = _greenBrush; }
} // 执行 return cell;
} static Brush _greenBrush = new SolidColorBrush(Color.FromArgb(0xff, 88, 183,
112));
}
```

以下代码需要应用于C1OlapGrid中：

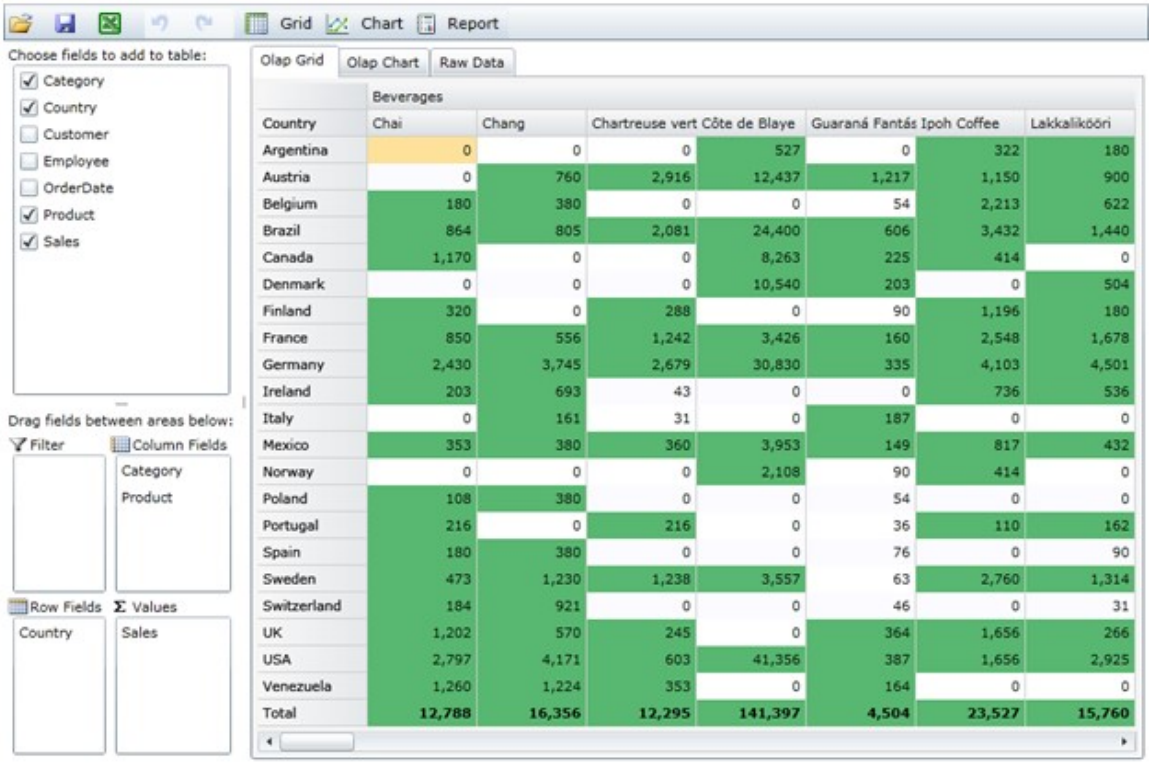
Visual Basic

```
' 对网格单元格应用条件格式
_c1OlapPage.OlapGrid.CellFactory = New ConditionalCellFactory()
```

C#

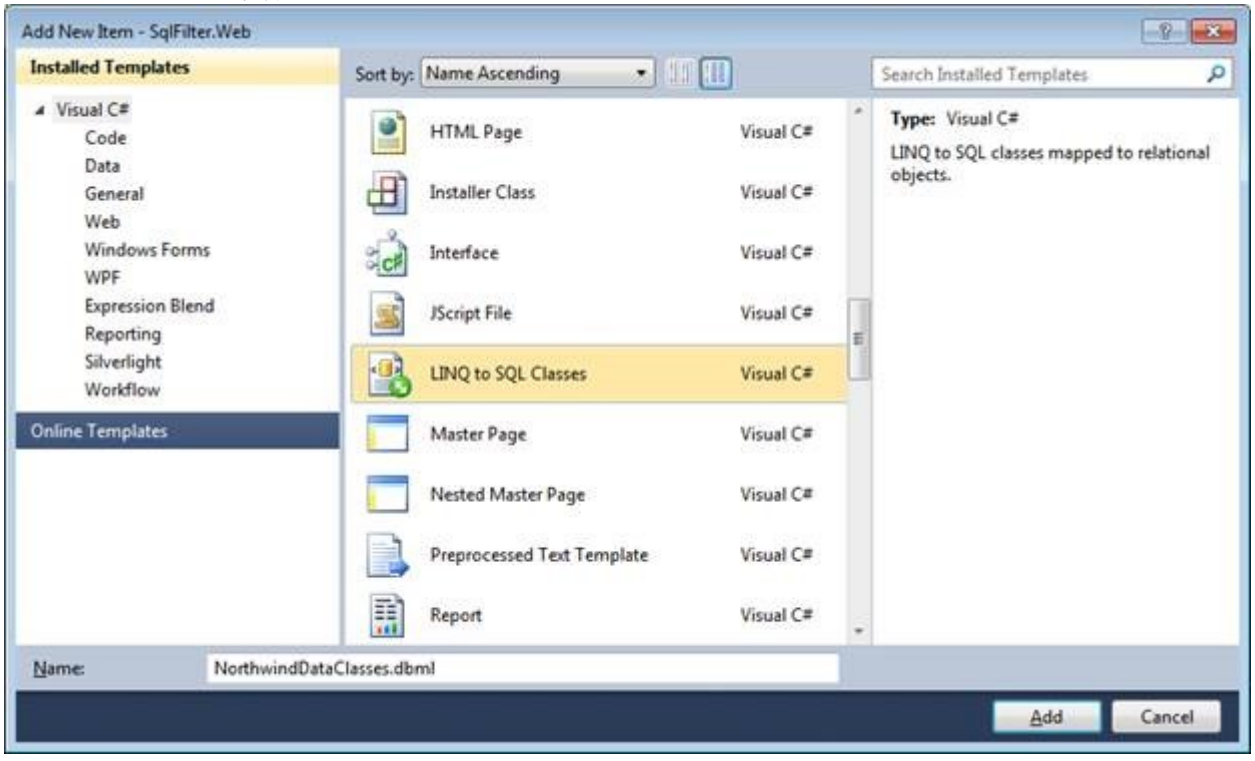
```
// 对网格单元应用条件格式
_c1OlapPage.OlapGrid.CellFactory = new ConditionalCellFactory();
```

如果您在之前的示例中添加此代码，您将看到如下运行结果：

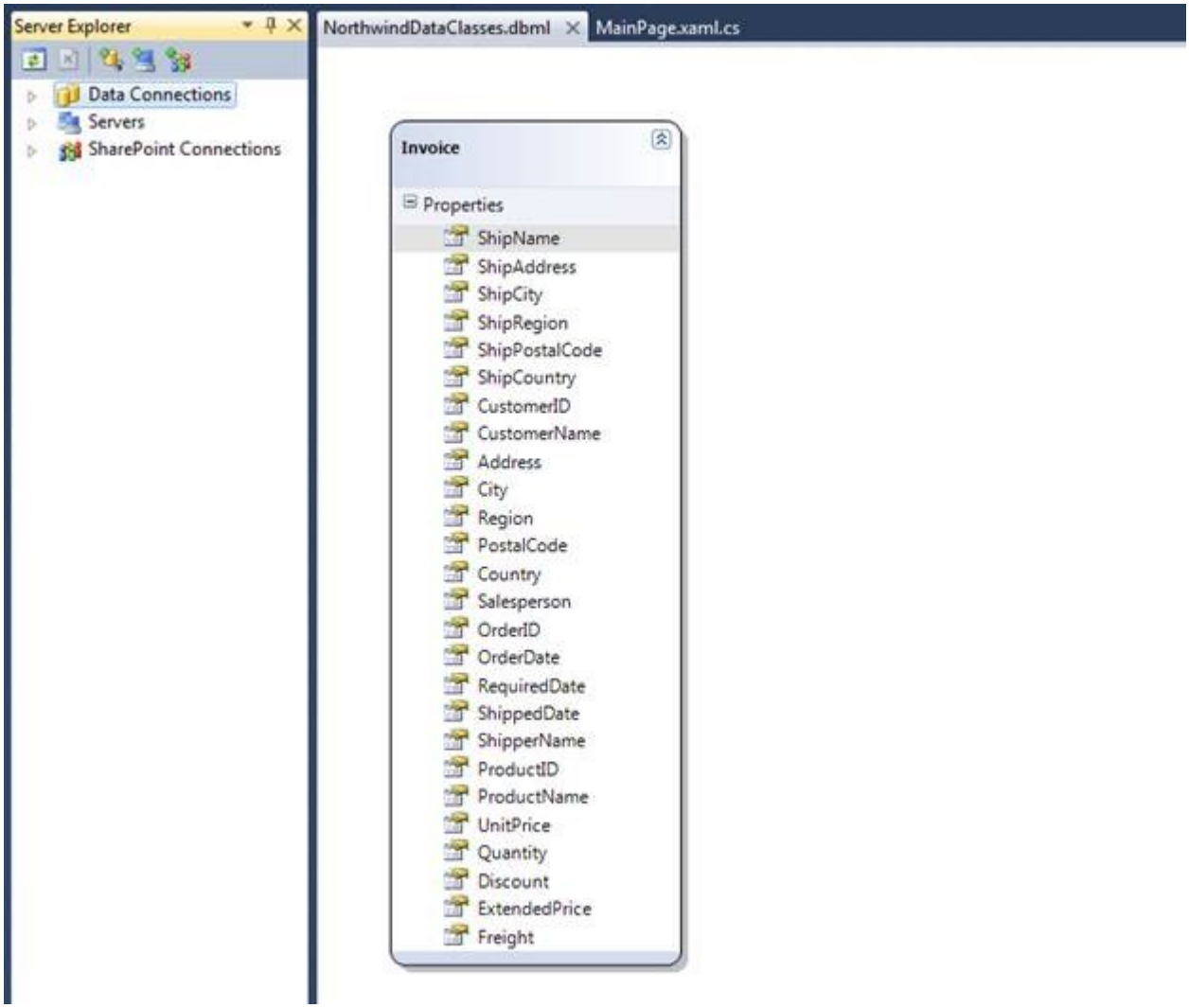


大数据资源

在许多情况下，我们需要一次性将大量的数据加载到内存中，例如一个表中包含一百万或更多的行，即使您在内存中载入了所有的数据，这个过程可能会消耗大量的时间。
有许多的方法可以处理这些场景，您可以在服务器上创建对汇总或者缓存数据的查询，或者使用web服务给您的Silverlight客户端传递数据，然而您也可以使用ClOlapPage处理这些表格。
在这个示例中，我们将使用一个WPF服务访问SQL数据库中的Northwind数据，该示例中最有趣的地方在于不是所有的数据都被一次性的加载到内存中的，ClOlapPage只能响应用户在筛选器中的当前数据。
对于本示例，我们将在ASP.Net web站点中创建一个Silverlight项目，我们也将使用LINQ to SQL类查询Northwind数据库样本中查询数据，LINQ to SQL是Visual Studio（2008或更高）中的一个ORM（相关对象映射）实现，它允许您在使用.NET类时实现一个相关数据模型，这个类可以方便您不使用LINQ就可以实现查询。
首先我们创建一个LINQ to SQL代表Northwind数据库，右键关联您Silverlight项目的web站点项目，并点击"Add New Item..."选择LINQ to SQL类并命名为NorthwindDataClasses.dbml。



接下来，我们将通过在Server Explorer中拖动选项将"Invoices"视图中的所有字段导入。



此时我们采用刚创建的LINQ及LINQ to SQL Classes(NorthwindDataClasses)创建WCF服务来查询数据，右键点击web 站点项目，并点击"Add New Item..."选择WCF Service并命名为NorthwindDataSerivce.svc。
使用以下代码替换NorthwindDataService.svc中代码：

Visual Basic

```
Imports System.Linq
Imports System.Runtime.Serialization
Imports System.ServiceModel
Imports System.ServiceModel.Activation
Imports System.Collections.Generic
Imports System.Text
Namespace SqlFilter.Web
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="1b2e30f1-97aa-4a9a-be80-1540b468016b"><ac:plain-text-body><![CDATA[ <ServiceContract([Namespace] := "")> _
]]></ac:plain-text-body></ac:structured-macro>
<AspNetCompatibilityRequirements(RequirementsMode :=
AspNetCompatibilityRequirementsMode.Allowed)> _
Public Class NorthwindDataService

''' <summary>/// 获取所有的发票. /// </summary> [OperationContract]


Public Function GetInvoices() As List(Of Invoice)
Dim ctx = New NorthwindDataClassesDataContext()
Dim invoices = From inv In ctx.Invoicesinv
```

Return invoices.ToList() End Function ''' <summary>/// 获取所有的客户<ac:structured-macro ac:name="unmigrated-wiki-markup"

ac:schema-version="1" ac:macro-id="e36cdb30-4787-49c8-93bc-7ca791fdf8fa"><ac:plain-text-body><![CDATA[. /// </summary>
[OperationContract]

Public Function GetCustomers() As List(Of String) Dim ctx = New NorthwindDataClassesDataContext() Dim customers = (From inv In ctx.Invoicesinv.CustomerName).Distinct() Return customers.ToList() End Function ''' <summary>/// 获取指定客户的所有发票. /// </summary> [OperationContract] Public Function GetCustomerInvoices(ParamArray customers As String()) As List(Of Invoice) ' 创建一个HashSet var hash = new HashSet<string>(); For Each c As String In customers hash.Add(c) Next Dim customerList As String() = hash.ToArray() ' 获取列表中客户的发票 var ctx = new NorthwindDataClassesDataContext(); Dim invoices = From inv In ctx.Invoices Where customerList.Contains(inv.CustomerName)inv Return invoices.ToList() End Function End ClassEnd NamespaceC# using System;using System.Linq;using System.Runtime.Serialization;using System.ServiceModel;using System.ServiceModel.Activation;using System.Collections.Generic;using System.Text;namespace SqlFilter.Web\{ \[ServiceContract(Namespace = "")\] \[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)\] public class NorthwindDataService \{ /// <summary>/// 获取所有发票. /// </summary>

[OperationContract]

public List GetInvoices() \{ var ctx = new NorthwindDataClassesDataContext(); var invoices = from inv in ctx.Invoices select inv; return invoices.ToList(); \} /// <summary>///获取所有客户. /// </summary>

[OperationContract]

]]>

```
public List<string> GetCustomers()
{ var ctx = new NorthwindDataClassesDataContext(); var customers =
(from inv in ctx.Invoices select inv.CustomerName).Distinct(); return customers.ToList();
}
/// <summary>/// 获取指定客户的所有发票. /// </summary>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="c5c17265-d50e-45a6-8757-3d292d12d9f3"><ac:plain-text-body><![CDATA[

[OperationContract]

public List<Invoice> GetCustomerInvoices(params string[] customers) {
]]></ac:plain-text-body></ac:structured-macro>
// 创建HashSet var hash = new HashSet<string>(); foreach (string c in customers)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="202d2c98-741b-4f07-8802-64860a905975"><ac:plain-text-body><![CDATA[ { hash.Add(c); } string[] customerList =
hash.ToArray();
]]></ac:plain-text-body></ac:structured-macro>
// 获取列表中客户的发票 var ctx = new
NorthwindDataClassesDataContext(); var invoices = from inv in ctx.Invoices where customerList.Contains(inv.CustomerName) s
elect inv; return invoices.ToList();
}
}
}
```

这里我们在web服务中定义了3个方法，头两个方法是简单的Get方法，用以使用LINQ和之前创建的LINQ to SQL类返回列表中的项目，GetCustomerInvoices方法是特殊的返回方法，它接收客户的数组数据作为参数，这就是我们在Silverlight C1OlapGrid项目客户端中定义的筛选器。
在移动到Silverlight项目之前我们必须创建web site项目，并在我们的web service中添加一个引用。想要添加一个引用，需要右键点击在解决方案视图中Silverlight项目节点，并点击"Add Service Reference"。这是再点击"Discover"并选择NorthwindDataService.svc，对它重命名为"NorthwindDataServiceReference"并点击OK。
现在数据源已经准备好了，我们现在需要将它连接到C1OlapPage以确保：

1. 用户可以看到筛选器中所有的客户（不仅仅是当前载入的）
2. 当用户变更筛选器，新的数据将被载入以显示新客户的需求

在我们完成任务之前，我们需要创建我们的UI，在MainPage.XAML中添加一个C1OlapPage控件和几个TextBlocks作为状态条：

XAML

```
<Grid x:Name="LayoutRoot">
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<olap:C1OlapPage x:Name="_c1OlapPage"/>
<TextBlock x:Name="_lblLoading"FontSize="24"Opacity=".5"Text="Loading
```

```
data..."HorizontalAlignment="Center"VerticalAlignment="Center"/>
<TextBlock x:Name="_lblStatus"Text="Ready"HorizontalAlignment="Right"Grid.Row="1"/>
</Grid>
```

此时添加下列代码到窗体中：

Visula Basic

```
Dim _allCustomers As ObservableCollection(Of String)
Dim _invoices As ObservableCollection(Of NorthwindDataServiceReference.Invoice)

Dim _customerFilter As C1OlapFilter
```

C#

```
ObservableCollection<string> _allCustomers;
ObservableCollection<NorthwindDataServiceReference.Invoice> _invoices;
C1OlapFilter _customerFilter;
```

这些字段将客户的列表完整的保存在数据库中，且当前的客户列表可以被用户选择，在任何时刻都可以选择客户的最大值。
我们需要分配完整的客户列表到C1OlapField.Values属性，该属性包含筛选器中显示的值，在默认情况下，
C1OlapPage中填充在原始数据中的值，原始数据仅包含部分列表，因此我们需要提供列表的完整的版
本，allCustomers ObservableCollection 将包含用户的整个集合供用户从中选择，C1OlapPage 实际可以使用
_invoice集合，即客户筛选得到的数据集。替换MainPage()中下列代码:

Visual Basic

```
Public Sub New()
InitializeComponent()
' 初始化OlapPage数据源 _invoices = new
ObservableCollection<SqlFilter.NorthwindDataServiceReference.Invoice>();
_c1OlapPage.DataSource = _invoices
' 初始化OlapPage视图 var olap = _c1OlapPage.OlapEngine; olap.BeginUpdate() olap.ColumnFields.Add("OrderDate")
olap.RowFields.Add("CustomerName") olap.ValueFields.Add("ExtendedPrice") olap.RowFields(0).Width = 200 olap.Fields("OrderD
ate").Format = "yyyy" olap.Fields("CustomerName").Filter.ShowValues = selectedCustomers.ToArray() olap.EndUpdate()
' 获取在数据库中所有客户的列表 var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient(); sc.GetCustomersCompleted += sc_GetCustomersCompleted

' 显示状态 _lblStatus.Text = "Retrieving customer list..."; sc.GetCustomersAsync()
End Sub
```

C#

```
public MainPage()
{
```

```
InitializeComponent();
// 初始化OlapPage数据源 _invoices = new
ObservableCollection<SqlFilter.NorthwindDataServiceReference.Invoice>();
_c1OlapPage.DataSource = _invoices;
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="f06ea99a-2170-42b9-8b2b-446b0be55b78"><ac:plain-text-body><![CDATA[ // 初始化OlapPage视图 var olap =
_c1OlapPage.OlapEngine; olap.BeginUpdate(); olap.ColumnFields.Add("OrderDate"); olap.RowFields.Add("CustomerName");
olap.ValueFields.Add("ExtendedPrice"); olap.RowFields[0].Width = 200; olap.Fields["OrderDate"].Format = "yyyy";
olap.Fields["CustomerName"].Filter.ShowValues = selectedCustomers.ToArray(); olap.EndUpdate();
]]></ac:plain-text-body></ac:structured-macro>
// 获取在数据库中所有客户的列表 var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient(); sc.GetCustomersCompleted +=
sc_GetCustomersCompleted; sc.GetCustomersAsync();
// 显示状态 _lblStatus.Text = "Retrieving customer list..."; }
```

这里我们将初始化C1OlapPage 数据源，首先创建一个默认的视图并得到数据库中所有客户的列表，我们需要的完成数据库所有客户的列表，因此用户可以选择他们希望看到的客户，注意这是一个长而紧凑的列表，它仅包含客户的姓名，而没有与之关联的详细信息，例如订单，订单明细等。
由于我们的数据来自于web service，它就可以异步的被检索，并且当数据完成载入时sc_GetCustomersCompleted事件会被激发。

Visula Basic

```
Private Sub sc_GetCustomersCompleted(sender As Object, e As
SqlFilter.NorthwindDataServiceReference.GetCustomersCompletedEventArgs)
' 隐藏'载入' 信息 _lblLoading.Visibility = Visibility.Collapsed;
' 监控CustomerName筛选 _customerFilter =

_c1OlapPage.OlapEngine.Fields["CustomerName"].Filter;


_customerFilter.PropertyChanged += filter_PropertyChanged
' 监控视图定义以确保CustomerName字段可用 _c1OlapPage.ViewDefinitionChanged +=
_c1OlapPage_ViewDefinitionChanged;
' 在"CustomerName"字段筛选器中显示有效的客户 _allCustomers = e.Result;
_customerFilter.Values = _allCustomers
' 获取数据 GetData();
End Sub
```

C#

```
void sc_GetCustomersCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomersCompletedEventArgs e)
{
// 隐藏'载入' 信息 _lblLoading.Visibility = Visibility.Collapsed;
// 监控CustomerName筛选 _customerFilter =

_c1OlapPage.OlapEngine.Fields["CustomerName"].Filter;


_customerFilter.PropertyChanged += filter_PropertyChanged;
// 监控视图定义以确保CustomerName字段可用 _c1OlapPage.ViewDefinitionChanged +=

_c1OlapPage_ViewDefinitionChanged;


// 在"CustomerName"字段筛选器中显示有效的客户 _allCustomers = e.Result;
_customerFilter.Values = _allCustomers;
// 获取数据 GetData();
}
```

该事件获取数据库中完整的列表，将它们保存并且在筛选器中显示，我们对C1OlapField.PropertyChanged事件进行监听，当用户修改包括筛选器中任何字段属性时事件会被激发，此时我们将检索被用户所选择的客户列表，并且将列表连接数据源。以下是当筛选项改变时更行数据的事件处理：

Visual Basic

```
' CustomerName字段筛选改变：得到一个新数据 void filter_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
If True Then
GetData()
End If
```

C#

```
// CustomerName字段筛选改变：得到一个新数据 void filter_PropertyChanged(object sender,

System.ComponentModel.PropertyChangedEventArgs e)
{
GetData();
}
```

如果"active"字段在视图中，字段的Filter 属性仅会出现在C1OlapEngine中，"Active"是RowFields, ColumnFields, ValueFields或FilterFields 集合中的成员，在这种情况下，"CustomerName"字段有一个特殊的筛选器且总是活动的，为了确保这点，我们必须处理ViewDefinitionChanged事件以确保"Customer"字段是Active的。下列的代买可以确保"CustomerName" 字段是活动的：

Visual Basic

```
' 确保Customer字段总是在视图中的 void _c1OlapPage_ViewDefinitionChanged(object sender,
EventArgs e)
If True Then
Dim olap = _c1OlapPage.OlapEngine
Dim field = olap.Fields("CustomerName") If Not field.IsActive Then olap.FilterFields.Add(field) End If
End If
```

C#

```
// 确保Customer字段总是在视图中的 void _c1OlapPage_ViewDefinitionChanged(object sender,
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="8a6b5c54-9c7f-449e-803e-d0cd782f69c9"><ac:plain-text-body><![CDATA[EventArgs e) { var olap =
_c1OlapPage.OlapEngine; var field = olap.Fields["CustomerName"]; if (!field.IsActive)
]]></ac:plain-text-body></ac:structured-macro>
{ olap.FilterFields.Add(field);
}
```

}
通过在筛选器中选择调用GetData方法获取用户的发票数据。

Visual Basic

```
' 获取选择客户的发票数据 void GetData()
If True Then
' 基于当前的筛选器设置重新创建活动客户列表 var selectedCustomers = new
ObservableCollection<string>();
For Each customer As String In _allCustomers If _customerFilter.Apply(customer) Then selectedCustomers.Add(customer) End
If
Next
_customerFilter.ShowValues = selectedCustomers.ToArray()
' 获取已选择的客户发票 var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient(); sc.GetCustomerInvoicesCompleted +=
sc_GetCustomerInvoicesCompleted sc.GetCustomerInvoicesAsync(selectedCustomers)
' 显示状态 _lblStatus.Text = string.Format("Retrieving invoices for {0} customers...", selectedCustomers.Count);
End If
```

C#

```
// 获取选择客户的发票数据 void GetData()
{
// 基于当前的筛选器设置重新创建活动客户列表 var selectedCustomers = new
ObservableCollection<string>(); foreach (string customer in _allCustomers)
{ if (_customerFilter.Apply(customer))
{ selectedCustomers.Add(customer); }
}
_customerFilter.ShowValues = selectedCustomers.ToArray();
// 获取已选择的客户发票 var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient(); sc.GetCustomerInvoicesCompleted +=
sc_GetCustomerInvoicesCompleted; sc.GetCustomerInvoicesAsync(selectedCustomers);
// 显示状态 _lblStatus.Text = string.Format("Retrieving invoices for {0} customers...", selectedCustomers.Count);
}
```

这里我们使用 C1OlapFilter (_customFilter) 并调用Apply方法创建用户所选择客户的列表，我们可以采用其他异步调用我们的web服务，可以采用以下事件返回筛选的发票：

Visual Basic

```

```

```
' 获取新的数据：在C1OlapPage上显示 void sc_GetCustomerInvoicesCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomerInvoicesCompletedEventArgs e)
If True Then
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="e49f6d28-837b-4d2f-9513-588ef3c32457"><ac:plain-text-body><![CDATA[ If
e.Cancelled OrElse e.[Error] IsNot Nothing Then
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="b53330c1-683d-47cc-a4d0-efe17da9e2ca"><ac:plain-text-body><![CDATA[
_lblStatus.Text = String.Format("** Error: {0}", If(e.[Error] IsNot
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="c6cc196f-ed3b-4c7b-9951-af50a14caacd"><ac:plain-text-body><![CDATA[Nothing,
e.[Error].Message, "Canceled"))
]]></ac:plain-text-body></ac:structured-macro>
Else
_lblStatus.Text = String.Format("Received {0} invoices ({1} customers).",
e.Result.Count, _customerFilter.ShowValues.Length)
' 开始更新 var olap = _c1OlapPage.OlapEngine; olap.BeginUpdate()
' 更新数据源 _invoices.Clear();
For Each invoice As var In e.Result
_invoices.Add(invoice)
' 完成更新 olap.EndUpdate();
Next
End If
End If
```

C#

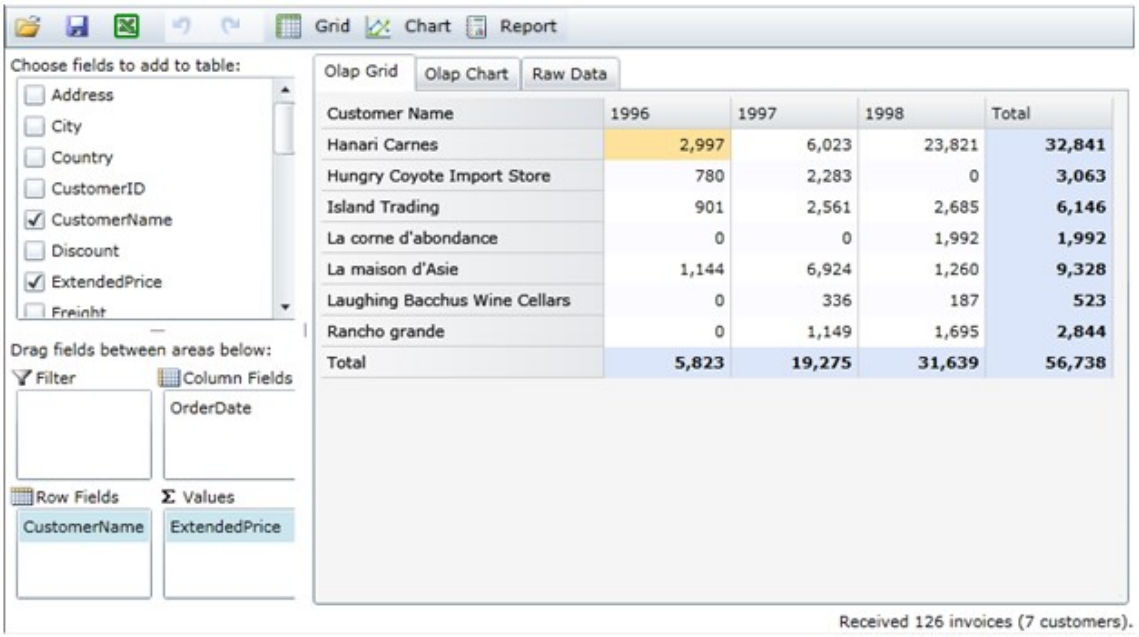
```
// 获取新的数据：在C1OlapPage上显示 void sc_GetCustomerInvoicesCompleted(object sender,

SqlFilter.NorthwindDataServiceReference.GetCustomerInvoicesCompletedEventArgs e) { if (
e.Cancelled
```

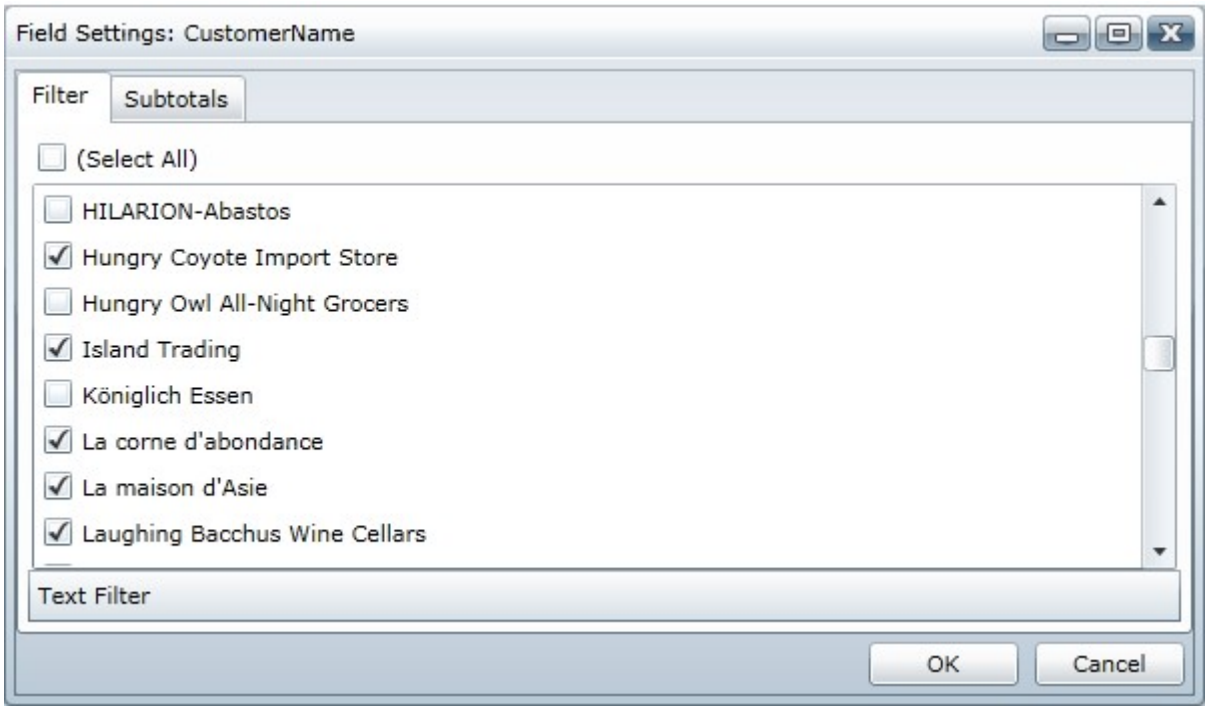
```
e.Error != null)
{
_lblStatus.Text = string.Format("*
* Error: {0}", e.Error != null ?
e.Error.Message : "Canceled"); } e
lse
{
_lblStatus.Text = string.Format("R
eceived {0} invoices ({1}
customers).", e.Result.Count,
_customerFilter.ShowValues.Length)

// 开始更新 var olap =
_c1OlapPage.OlapEngine; olap.Begin
Update();
// 更新数据源 _invoices.Clear(); f
oreach (var invoice in e.Result)
{
_invoices.Add(invoice);
}
// 完成更新 olap.EndUpdate();
}
}
```

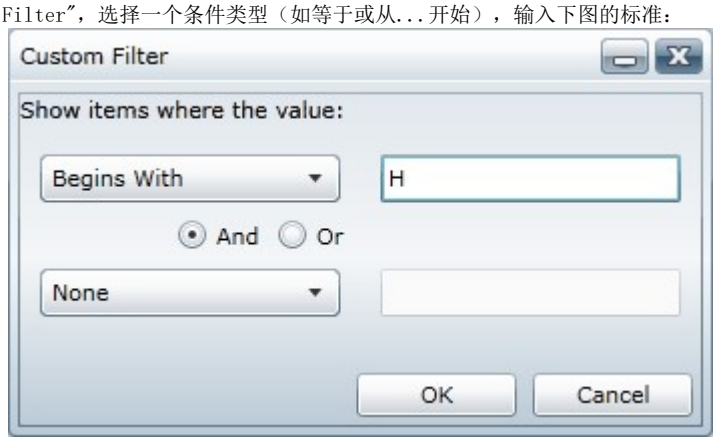
如果您现在运行应用程序，您将注意到只有"CustomerName"设置中的客户在视图中显示：



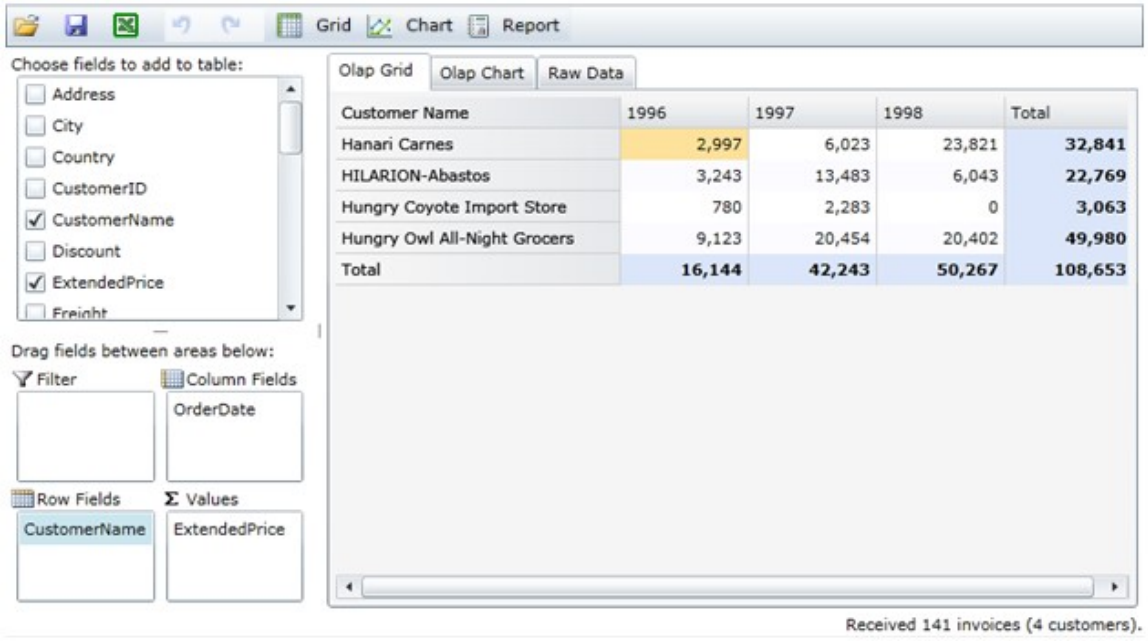
为了查看其它的客户，请双击"CustomerName" 字段并选择"Field Settings" 打开筛选器设置。



此时通过选择指定客户或者定义条件编辑筛选器，为了定义筛选条件，需要点击字段设置下方的"Text Filter"，选择一个条件类型（如等于或从...开始），输入下图的标准：



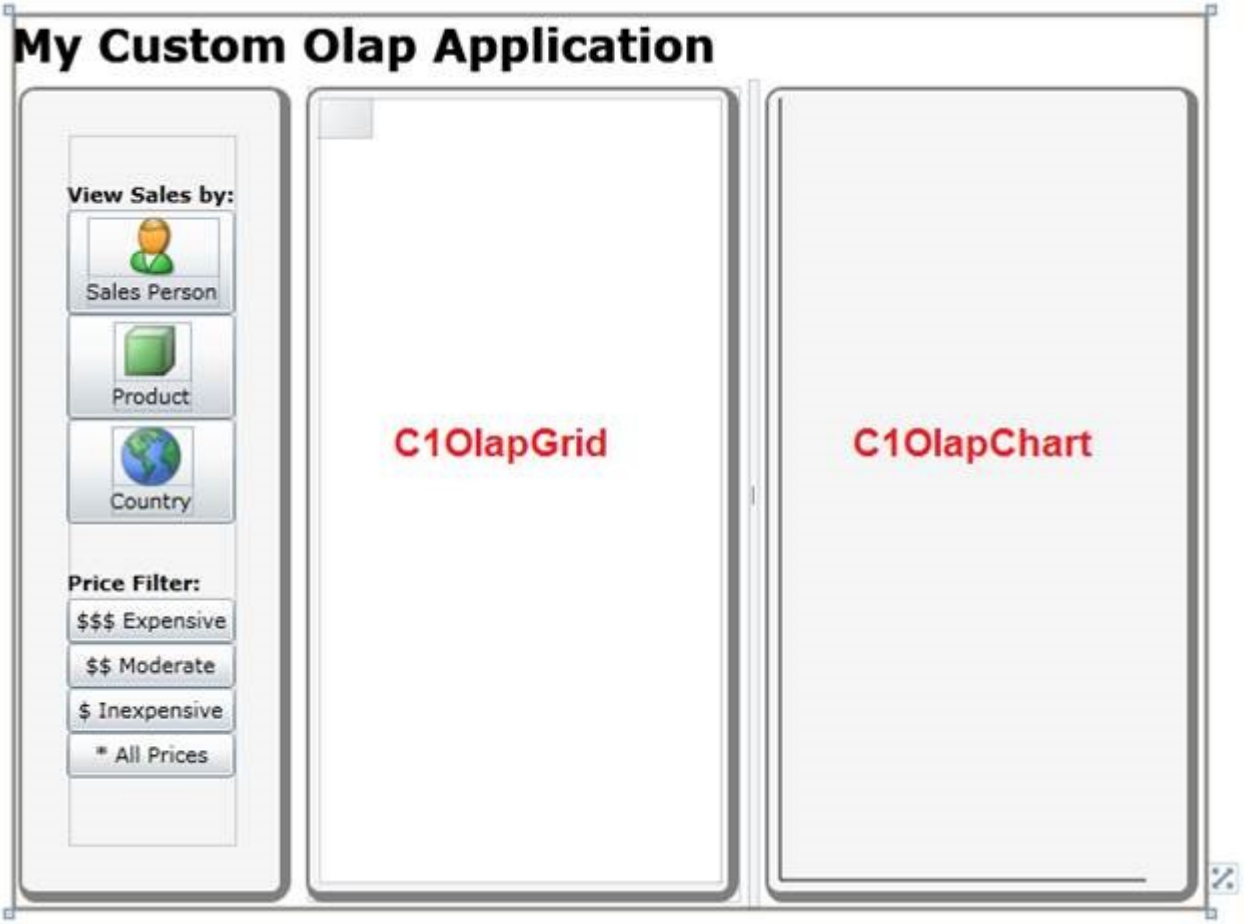
当您点击OK，应用程序会检测变化并从GetData方法中请求数据，一旦新的数据被加载，ClOlapPage将会自动的检测OLAP表的变化和更新：



查阅上一节"SqlFilter" 示例的完整实现，我们可以拓展该示例在本地存储的筛选器中保存OLAP视图。请参阅本地存本地存储中持久储中持久OLAP视图视图 (Section 7.2.2)。

绑定一个自定义用户界面

前一节中的所有示例均采用C1OlapPage控件，包含了一个完整的UI并且只需要极少的代码，在本节中，我们将不使用C1OlapPage创建一个OLAP应用程序，它将使用C1OlapGrid，C1OlapChart及一些标准的Silverlight控件创建一个完整的自定义UI。
完整的应用程序资源代码包含在Silverlight及及WPF版版OLAP的"CustomUI"示例中。
下图显示设计视图中的应用：



我们的网格布局包含2行4列，在最顶行有一个TextBlock以显示应用标题，在包含两组按钮的最左列有一个垂直StackPanel控件，上面的组允许用户从三个预定义视图中选择一个：按照销售人员、产品和国家的销售数据。下面的组允许用户基于产品的价

格（昂贵，中等或者廉价）应用一个筛选器。
其余列有一个空的C1OlapGrid，GridSplitter和一个空的C1OlapChart，这些控件都用来显示当前已选择的视图。
一旦所有的控件都已就绪，添加代码将它们连接起来，最终应用可以顺利运行。
在代码中，我们声明一个C1OlapPanel，在之前的例子中C1OlapPanel部分对最终用户是可见的，但是在本示例中我们在场景后面使用它，因此用户将无法看到它，这个不可见的控件作为网格和图表的数据源，并负责筛选和汇总数据，网格和图表都有DataSource属性并设置为C1OlapPanel。
C1OlapPanel _olapPanel = new C1OlapPanel();
以下代码首先从XML数据文件中载入Northwind数据，我们使用Silverlight版版Data为我们提供熟悉的数据集和数据表
格对象来读取数据，我们也可以使用Silverlight版版Zip从客户端中解压XML文件，我们将DataTable的结果设置为
C1OlapPanel.DataSource的属性，我们也可以将我们C1OlapPanel控件分配到我们的C1OlapGrid和C1OlapChart控件的DataSource属性中，最后我们通过点击两个按钮来对当前的视图与筛选器进行初始化。

```
Visual Basic
```

```
Public MainPage()
InitializeComponent()
Dim ds = New DataSet()
Dim asm = Assembly.GetExecutingAssembly()
Using s = asm.GetManifestResourceStream("CustomUI.Resources.nwind.zip")

Dim zip = New C1ZipFile(s)
```

' 载入数据 ds.ReadXml(zr); Using zr = zip.Entries(0).OpenReader() End Using End Using ' 绑定olap网格/图表到面板上 _olapChart.DataSource = _olapPanel; _olapGrid.DataSource = _olapPanel ' 为olap面板绑定数据 <ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="25a38177-1934-40da-9d0f-8fd429eb9602"><ac:plain-text-body><![CDATA[_olapPanel.DataSource = ds.Tables[0].DefaultView;

' 开始运行销售人员与所有产品视图 _btnSalesperson_Click(this, null); _btnAllPrices_Click(Me, Nothing)End Sub'选择当前视图的事件处理如下：Private Sub _btnSalesperson_Click(sender As Object, e As RoutedEventArgs) BuildView("SalesPerson")End SubPrivate Sub _btnProduct_Click(sender As Object, e As RoutedEventArgs) BuildView("ProductName")End SubPrivate Sub _btnCountry_Click(sender As Object, e As RoutedEventArgs) BuildView("Country")End Sub'所有的处理使用如下BuildView帮助方法：' 在按钮被点击后重新构建一个视图 BuildView(string fieldName)If True Then ' 获取olap机 var olap = _olapPanel.OlapEngine; ' 在执行前停止更新 olap.BeginUpdate(); ' 清除所有字段 olap.RowFields.Clear(); olap.ColumnFields.Clear() olap.ValueFields.Clear() ' 按照年分组格式化订单日期

var f = olap.Fields["OrderDate"];

f.Format = "yyyy" ' 生成视图 olap.ColumnFields.Add("OrderDate"); olap.RowFields.Add(fieldName) olap.ValueFields.Add("ExtendedPrice") ' 保存更新 olap.EndUpdate();End IfC#public MainPage()\{ InitializeComponent(); var ds = new DataSet(); var asm = Assembly.GetExecutingAssembly(); using (var s = asm.GetManifestResourceStream("CustomUI.Resources.nwind.zip")) \{ var zip = new C1ZipFile(s); using (var zr = zip.Entries\[0\].OpenReader()) \{ // 载入数据 ds.ReadXml(zr); \} \}]]>

```
// 绑定olap网格/图表到面板上 _olapChart.DataSource = _olapPanel;
_olapGrid.DataSource = _olapPanel;

// 为olap面板绑定数据 _olapPanel.DataSource = ds.Tables[0].DefaultView;


// 开始运行销售人员与所有产品视图 _btnSalesperson_Click(this, null);
_btnAllPrices_Click(this, null);
}
//选择当前视图的事件处理如下：
void _btnSalesperson_Click(object sender, RoutedEventArgs e) {
BuildView("SalesPerson");
} void _btnProduct_Click(object sender, RoutedEventArgs e) {
BuildView("ProductName");
} void _btnCountry_Click(object sender, RoutedEventArgs e) {
BuildView("Country");
}
//所有的处理使用如下BuildView帮助方法：
// 在按钮被点击后重新构建一个视图 BuildView(string fieldName)
{
// 获取olap机 var olap = _olapPanel.OlapEngine;
// 在执行前停止更新 olap.BeginUpdate(); // 清除所有字段 olap.RowFields.Clear(); olap.ColumnFields.Clear();
olap.ValueFields.Clear();

// 按照年分组格式化订单日期 var f = olap.Fields["OrderDate"];

f.Format = "yyyy";
// 生成视图 olap.ColumnFields.Add("OrderDate"); olap.RowFields.Add(fieldName); olap.ValueFields.Add("ExtendedPrice");
// 保存更新 olap.EndUpdate();
}
```

ClOlapPanel的BuildView方法可以获取ClOlapEngine对象的一个引用，可以调用BeginUpdate方法停止更新直到一个新的视图被完整的定义，这样可以提高性能。
代码可以设置"OrderDate"字段格式为"yyyy"，以便销量可以按照年来分组，并且可以通过清除机制中的RowFields, ColumnFields, 及ValueFields 集来重新构建视图，此时添加的字段会被显示，由调用方法传递的"fieldName"参数包含字段名可以改变示例中的视图。
当以上都完成时，代码会调用EndUpdate使ClOlapPanel更新输出表格。
在运行应用程序之前，可以查看代码实现筛选，事件处理如下：

Visual Basic

```
Private Sub _btnExpensive_Click(sender As Object, e As RoutedEventArgs)
SetPriceFilter("Expensive Products (price > $50)", 50, Double.MaxValue) End Sub

Private Sub _btnModerate_Click(sender As Object, e As RoutedEventArgs)
SetPriceFilter("Moderately Priced Products ($20 < price < $50)", 20, 50)
```

```
End Sub
Private Sub _btnInexpensive_Click(sender As Object, e As RoutedEventArgs)
SetPriceFilter("Inexpensive Products (price < $20)", 0, 20) End Sub
Private Sub _btnAllPrices_Click(sender As Object, e As RoutedEventArgs)
SetPriceFilter("All Products", 0, Double.MaxValue)
End Sub
```

C#

```
void _btnExpensive_Click(object sender, RoutedEventArgs e) {
SetPriceFilter("Expensive Products (price > $50)", 50, double.MaxValue);
} void _btnModerate_Click(object sender, RoutedEventArgs e) {
SetPriceFilter("Moderately Priced Products ($20 < price < $50)", 20, 50);
} void _btnInexpensive_Click(object sender, RoutedEventArgs e) {
SetPriceFilter("Inexpensive Products (price < $20)", 0, 20);
} void _btnAllPrices_Click(object sender, RoutedEventArgs e) {
SetPriceFilter("All Products", 0, double.MaxValue); }
```

所有的事件处理使用一个SetPriceFilter 帮助方法：

Visual Basic

```
' 为产品价格应用一个筛选器 oid SetPriceFilter(string footerText, double min, double max) If True Then
' 获取olap机 var olap = _olapPanel.OlapEngine;
' 在执行前停止更新 lap.BeginUpdate();

' 确保视图中的单元价格字段是活动的 var field = olap.Fields["UnitPrice"];

olap.FilterFields.Add(field)
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="ddca5243-42dc-428c-8ed5-46887bbb59c6"><ac:plain-text-body><![CDATA[ ' 自定义筛选器 var filter = field.Filter;
filter.Clear() filter.Condition1.[Operator] = C1.Olap.ConditionOperator.GreaterThanOrEqualTo filter.Condition1.Parameter =
min filter.Condition2.[Operator] = C1.Olap.ConditionOperator.LessThanOrEqualTo filter.Condition2.Parameter = max
]]></ac:plain-text-body></ac:structured-macro>
' 保存更新 olap.EndUpdate();
End If
```

C#

```
// 为产品价格应用一个筛选器 etPriceFilter(string footerText, double min, double max)
{
// 获取olap机 ar olap = _olapPanel.OlapEngine;
// 在执行前停止更新 olap.BeginUpdate();

// 确保视图中的单元价格字段是活动的 ar field = olap.Fields["UnitPrice"];
```

```
olap.FilterFields.Add(field);
// 自定义筛选器 var filter = field.Filter; filter.Clear(); filter.Condition1.Operator =
C1.Olap.ConditionOperator.GreaterThanOrEqualTo; filter.Condition1.Parameter = min; filter.Condition2.Operator =
C1.Olap.ConditionOperator.LessThanOrEqualTo; filter.Condition2.Parameter = max;
// 保存更新 olap.EndUpdate();
}
```

在这之前，代码会获取C1OlapEngine的一个引用，并立即调用BeginUpdate。
它可以获取到"UnitPrice"字段的一个引用以筛选数据，"UnitPrice" 字段添加到机制的FilterFields 集合中，因而筛选可以应用到当前的视图中。
这是一个重要的细节，如果一个字段没有被包含在任意的视图集中(RowFields, ColumnFields, ValueFields, FilterFields)，此时它将不会包含在任何的视图中，它的Filter 属性不会影响视图。
代码可以配置"UnitPrice"字段的Filter属性来设置指定应包含在视图中值范围的两个条件，通过确定"min"和"max"参数来定义范围，您可以提供应包含的值的列表而不使用条件，在处理数字值时条件通常更方便，而处理字符串值和枚举时列表更适合。最后，代码调用EndUpdate。最后我们需要随时更新C1OlapGrid中的C1OlapChart对列进行排列，数据值会按照下列排序显示：

Visual Basic

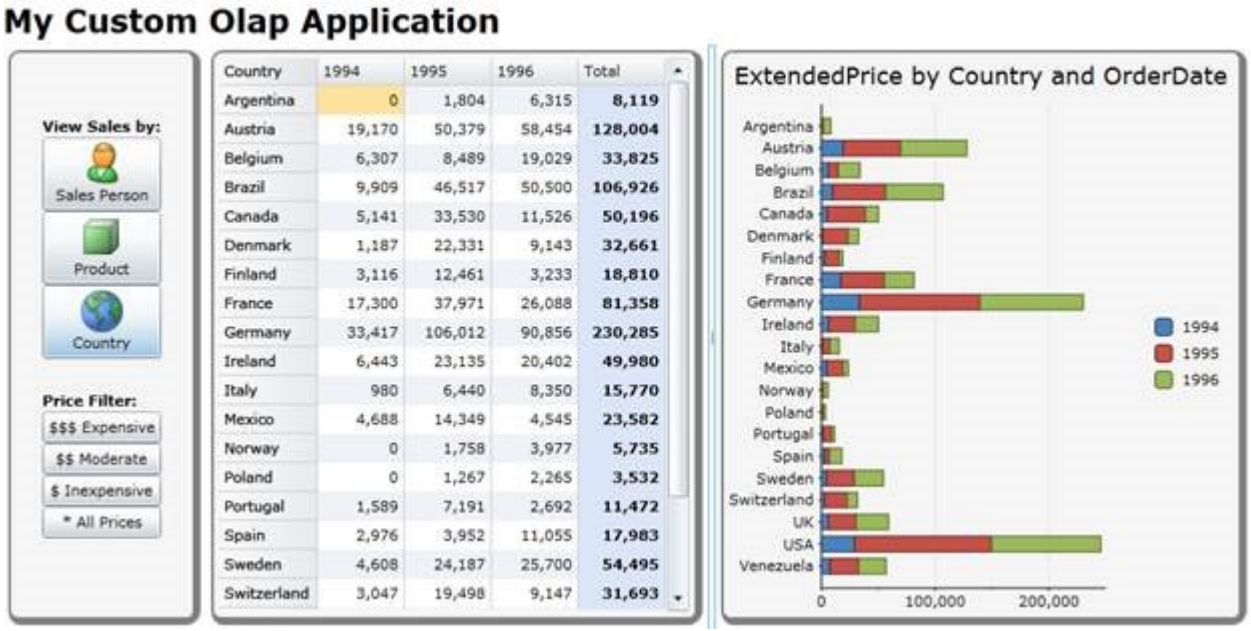
```
Private Sub _olapGrid_SortedColumn(sender As Object, e As
C1.Silverlight.FlexGrid.CellRangeEventArgs)
_olapChart.UpdateChart()
End Sub
```

C#

```
void _olapGrid_SortedColumn(object sender, C1.Silverlight.FlexGrid.CellRangeEventArgs e)

{
_olapChart.UpdateChart();
}
```

应用程序已经准备好了，您可以运行它并测试不同的视图，以及它筛选视图的能力，如下图所示：



该视图显示了按年和国家分组所有产品的销量，请注意图表是如何显示值接近$300,000。
如果您点击了"$$$ Expensive" 按钮，筛选器会立即应用视图的变化，请注意图表是如何显示值接近$80,000的，
Expensive 值表示三分之一的销量：