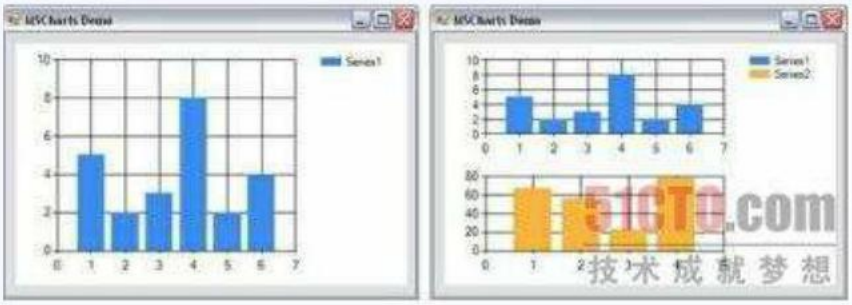
一、mschart 控件使用详解

ASP.NET 3.5 **ChartAreas 控件**中的 ChartAreas 属性是 ChartArea 对象的集合，ChartArea 负责显示容器的属性或图表的背景，由于不止一个，这就意味着 MSChart 控件可以包含多个图表。



在使用多个 ChartAreas 时理解下面几点内容非常重要：

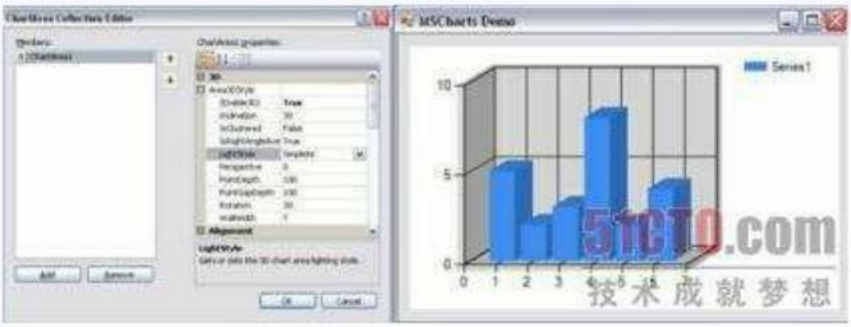
在技术上可以控制 ChartArea 的位置，因此多个 ChartArea 可以叠加，但不推荐这么做，建议在 MSChart 控件内的独立区域内绘制它们，为了合并或覆盖数据点，推荐在一个 ChartArea 内使用多个序列，后面将会有介绍。默认情况下，控件会为你自动调整大小和位置。

单个 ChartArea 将会独立调整以适应数据，正如上图所显示的，第二个 ChartArea 中的 Y 值更大，数据点也更少。

多个 ChartAreas 控件允许你使用多个不相容的 ChartTypes（序列对象属性，控制图表的显示类型，如条形、柱状和饼状）显示图表，图表任然显示在相同的 MSChart 控件内。

对于单个 ChartArea，有许多独立的属性可以设置和调整，这样你就可以自行调整图表区域以满足不同的需要，它的大部分属性和面板控件的属性都差不多，因此这里我们就不多说了，只说一下 ChartArea 唯一的属性，下面是这些唯一属性的清单：

3D 样式：使用 ChartArea 的 Area3DStyle 属性和子属性，我们可以创建漂亮的、十分抢眼的 3D 图表，无论是在设计器中还是在代码中都必需将 Enable3D 属性设置为 TRUE，其余的参数可以通过调整旋转、视角、照明方式和其它 3D 元素，让一个图像看起来具有 3D 效果。



坐标轴控制和样式：坐标轴集合包括 x 轴和 y 轴，以及第二个 x 轴和 y 轴，这四个项目的属性允许你设置样式、设置标签、定义间隔、设置工具提示、设置缩放等，如果你的图标要求精确的间隔、标签或其它特殊的显示需要，你可以使用这些属性。例如，你可以颠倒坐标轴的值，或控制如何在 x 轴上显示标签。如果你使用图表显示实时信息，可以使用 IntervalType 属性来配置基于日期和时间显示数据点。

选择光标：如果你对用户使用鼠标选择数据点或点击和拖拉范围非常感兴趣，这个时候就要用到 CursorX 和 CursorY 属性了，你可以启用选择，并设置最初的光标位置或范围。

Series

和 ChartAreas 属性一样，Series 属性是一个集合。

单个 ChartAreas 实例包括 3 个重要的属性：**ChartArea 属性、ChartType 属性和 Points 集合属性。**

ChartArea：识别使用哪个 ChartArea。

ChartType：识别表示数据时使用的图标类型，基本的类型有条形、柱状、饼状和线状，还有一些高级选项，如 K 线图、曲线图、追星图等。

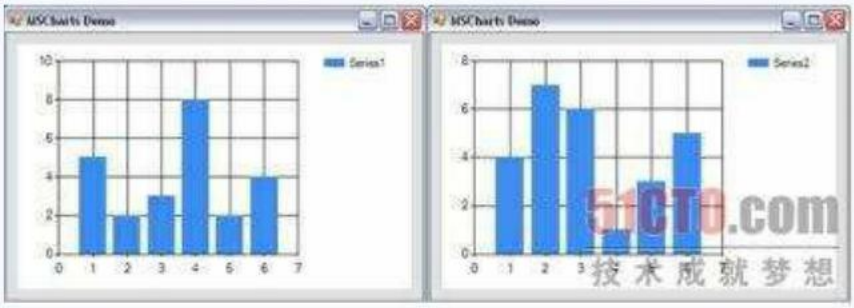
Points：它是 DataPoint 对象的集合，包括 x 值和 y 值，它们是绘在图表上的序列的一部分，数据绑定时最常用的增加数据点的方法，本文后面会做介绍。

Series 实例上的其他常用属性和人们广泛了解的还包括：

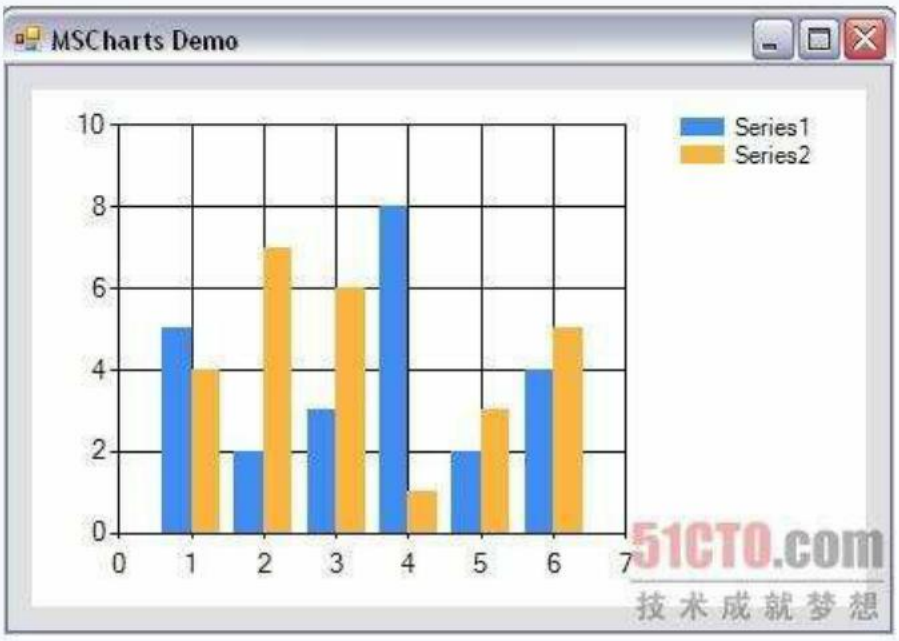
Color：这个属性用于单独设置每个数据点序列的颜色，默认情况下，这个属性是空白的，控件会自动改变颜色，以保证将多个序列区分开来。

IsValueShownAsLabel：将这个属性的值设为 TRUE 后（默认是 FLASE），图表将显示每个数据点的 Y 值。

在讲多个序列实例合并到一个 ChartArea 中时，例如下面两个独立的图表，每个图表都包括 6 个数据点。



假设你想比较这两个图表中的数据点，你可以将这两个 MSChart 控件放在一起，相互挨着，也可以在一个图表中使用两个 ChartAreas 控件，这两种方式都没问题，但都不能给你很好的视觉比较效果，这就是为什么 MSChart 要合并数据点，让你可以肩并肩地对比数据。将第二个图表中的数据作为第二个序列实例添加到第一个图表中，一下子就从视觉上改善了对比的效果。



使用多个序列实例时，记住每个序列使用的 ChartType 非常重要，不是所有 ChartType 选项放在一起都是兼容的。

总的说来，图表控件的层次如下：MSChart 控件有零到多个 ChartAreas，一个 ChartAreas 有零到多个序列（Series），一个序列有零到多个数据点（DataPoints）。

数据绑定

数据可以在设计时或运行时绑定，在设计时绑定要使用到数据源配置向导，在 MSChart 控件数据源属性下拉按钮中可以找到它，如果你已经配置过数据源，你可以在下拉列表中进行选择。

图表函数

DataBind()：绑定数据源的基础函数。

DataBindTable()：绑定图表到特定的数据表，但不允许绑定多个 Y 值，每个序列不同的数据源或 x 值、y 值有不同的数据源。

DataBindCrossTab()：将图表绑定到一个数据源，并允许基于一个数据列进行分组，在具体指定的列上每个唯一的值将自动创建一个单独的序列。

数据点函数

DataBind()：绑定一个序列到单一的数据源，并允许其它属性绑定到同一个数据源（如标签、工具提示、图例文本等）。

DataBindXY()：允许将 x 值和 y 值绑定到独立的数据源，它也用于为每个序列绑定单独的数据源。

DataBindY()：仅绑定序列中数据点的 Y 值。

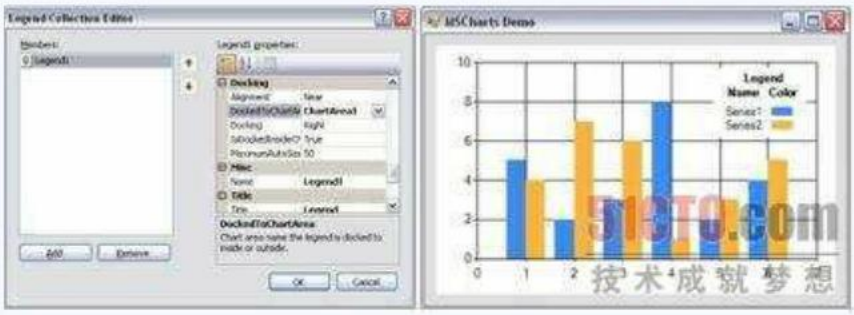
数据源配置好后，MSChart 控件可以绑定所有的实现了 IEnumerable 接口的对象，包括但不限于 DataReader、DataSet、Array 和 List。也允许绑定 SqlCommand、OleDbCommand、SqlDataAdapters 和 OleDbDataAdapter 对象。

图例

图例属性也是一个集合（叫做图例对象），为了快速理解使用图例可以做什么，可以把它想象成一个简单的表，假设你有一个表格，默认有两列，你可以从前面的例子看出，使用序列实例名的默认设置和序列的颜色，图例可以通过添加额外的单元列到 CellColumn 集合上进行扩展，还可以添加列标题，便于更好地理解图例。在下面的例子，标题“Name”和“Color”已经添加到默认图例实例上。

每个序列的图例文本由序列自身控制，在每个序列实例上使用 LegendText 属性以改变图表上图例的文本，你也可以给图例一个标题。在下面的例子，图例被标题为“图例”，除此之外，你还可以输入文本，你也可以设置属性来处理标题的对齐、颜色和字体，你会发现大部分 MSChart 属性这类操作和自定义，只要你愿意。

最后，图表的图例有两种放置方法，默认是在 ChartArea 外部，位于右侧，紧挨 ChartArea。另一种是通过坐标进行精确控制，停靠在 ChartArea 内，通过设置 ChartArea 对象的 DockedToChartArea 属性实现。



标题

标题和前面讨论到的其它属性类似，为每个标题创建独立的实例时，图表控件会在标题集合中保留这些标题实例，理解标题的最好方法是将其认为是一个标签控件，这意味着标题可以顶端居中、左端居中、顶端居左和底部居右。

在下面的例子中，图表拥有一个字体大小，顶端居中的标题，叫做“My First Chart”。

提示和技巧

3D 透明性：可以在序列上使用一些透明属性让 3D 图表看起来更漂亮，在设计时，可以将 Alpha 值添加到要使用的颜色的 RGB 代码中，在属性窗口中，选择序列集，选择一个序列，在该序列的属性窗口上，在现有的 3 个 RGB 值前添加一个 Alpha 值，你可以使用下面的代码完成同样的任务：

```
1.  chart1.Series["Series1"].
2.  Color = Color.FromArgb(220, 123, 123, 123);
```

利用设计器：在大多数时候，在代码运行时才配置图表控件的属性和配置选项是很愚蠢的，当你的应用程序以静态的方式使用图表时（如一直都是两个序列的条形图），在设计时，设计器允许你配置和查看图表。另一方面，如果你倾向于动态使用图表（如一会儿是饼图，一会儿是线状图，用户在运行时可以自行修改），这种情况需要代码在运行时修改设计，不用为每种图表类型都手动创建代码，使用设计器创建后，从设计文件中去除多余的代码，这样可以节约你编码和决定使用哪个属性的时间。

逻辑名：这是一个通用的优秀编程习惯，在创建 ChartAreas、Series 和 Titles 等时，请确定都给它们取了名字，并且是容易记住的，例如，chart1.Series["Series1"]…, chart1.Series["Series2"]…就

比 chart1.Series["salesrepearningquarter"]…, chart1.Series["salesrepcomdatabyquarter"]…更容易引入错误。

二、详解

chart 控件主要有 Titles 标题集合 Chart Area 图形显示区域 Series 图表集合 Legends 图列的集合

（1） 常用事件：

1. Series1.Points.DataBind()绑定数据点集合，如果要在一个 MSChart 控件的一个绘图区(ChartArea)内添加多个不同数据源的图表，就用这个主动绑定数据集合的方法。可以将表中指定字段的值绑定到指定的坐标轴上。

2. MSChart1.DataBind()给整个 MSChart 绑定一个数据源，该 MSChart 中的图表全部可以使用该数据源作为统计来源

（2）MSChart 的元素组成，最常用的属性包括：

ChartAreas：增加多个绘图区域，每个绘图区域包含独立的图表组、数据源，用于多个图表类型在一个绘图区不兼容时。

AlignmentOrientation：图表区对齐方向，定义两个绘图区域间的对齐方式。

AlignmentStyle：图表区对齐类型，定义图表间用以对其的元素。

AlignWithChartArea：参照对齐的绘图区名称。

InnerPlotPosition：图表在绘图区内的位置属性。

Auto：是否自动对齐。

Height：图表在绘图区内的高度（百分比，取值在 0-100）

Width：图表在绘图区内的宽度（百分比，取值在 0-100）

X,Y：图表在绘图区内左上角坐标

Position：绘图区位置属性，同 InnerPlotPosition。

Name：绘图区名称。

Axis：坐标轴集合

Title：坐标轴标题

TitleAlignment：坐标轴标题对齐方式

Interval：轴刻度间隔大小

IntervalOffset：轴刻度偏移量大小

MinorGrid：次要辅助线

MinorTickMark：次要刻度线

MajorGrid：主要辅助线

MajorTickMark：主要刻度线

DataSourceID：MSChart 的数据源。

Legends：图例说明。

Palette：图表外观定义。

（3）Series：最重要的属性，图表集合，就是最终看到的饼图、柱状图、线图、点图等构成的集合；可以将多种相互兼容的类型放在一个绘图区域内，形成复合图。

IsValueShownAsLabel：是否显示数据点标签，如果为 true，在图表中显示每一个数据值

Label：数据点标签文本

LabelFormat：数据点标签文本格式

LabelAngle：标签字体角度

Name：图表名称

Points：数据点集合

XValueType：横坐标轴类型

YValueType：纵坐标轴类型

XValueMember：横坐标绑定的数据源（如果数据源为 Table，则填写横坐标要显示的字段名称）

YValueMembers：纵坐标绑定的数据源（如果数据源为 Table，则填写纵坐标要显示的字段名称，纵坐标可以有两个）

ChartArea：图表所属的绘图区域名称

ChartType：图表类型（柱形、饼形、线形、点形等）

Legend：图表使用的图例名称

Titles：标题集合。

width：MSChart 的宽度。

height：MSChart 的高度。

（4）示例：

```
private void BindGrid()
        {
            chart2.Width = 800;
            chart2.Height = 600;
            //作图区的显示属性设置
            //chart2.ChartAreas["ChartArea1"].AxisX.IsMarginVisible = false;
            //chart2.ChartAreas["ChartArea1"].Area3DStyle.Enable3D = false;
            //背景色设置
            chart2.ChartAreas["ChartArea1"].ShadowColor = Color.Transparent;
            chart2.ChartAreas["ChartArea1"].BackColor = Color.FromArgb(209, 237,
```

```
254);          //该处设置为了由天蓝到白色的逐渐变化
            chart2.ChartAreas["ChartArea1"].BackGradientStyle = GradientStyle.TopBottom;
            chart2.ChartAreas["ChartArea1"].BackSecondaryColor = Color.White;
            //X,Y 坐标线颜色和大小
            chart2.ChartAreas["ChartArea1"].AxisX.LineColor = Color.FromArgb(64, 64, 64, 64);
            chart2.ChartAreas["ChartArea1"].AxisY.LineColor = Color.FromArgb(64, 64, 64, 64);
            chart2.ChartAreas["ChartArea1"].AxisX.LineWidth = 2;
            chart2.ChartAreas["ChartArea1"].AxisY.LineWidth = 2;
            chart2.ChartAreas["ChartArea1"].AxisX.Title = "时间";
            chart2.ChartAreas["ChartArea1"].AxisY.Title = "灰量";
            //中间 X,Y 线条的颜色设置
            chart2.ChartAreas["ChartArea1"].AxisX.MajorGrid.LineColor = Color.FromArgb(64,
64, 64, 64);
            chart2.ChartAreas["ChartArea1"].AxisY.MajorGrid.LineColor = Color.FromArgb(64,
64, 64, 64);
            //X.Y 轴数据显示间隔
            chart2.ChartAreas["ChartArea1"].AxisX.Interval = 1.0; //X 轴数据显示间隔
            chart2.ChartAreas["ChartArea1"].AxisX.IntervalType = DateTimeIntervalType.Days;
            chart2.ChartAreas["ChartArea1"].AxisX.IntervalOffset = 0.0;
            chart2.ChartAreas["ChartArea1"].AxisX.IntervalOffsetType =
DateTimeIntervalType.Days;
            chart2.ChartAreas["ChartArea1"].AxisX.LabelStyle.Format = "M-d";
```

```
            chart2.ChartAreas["ChartArea1"].AxisY.Interval = 200;// y 轴数据显示间隔
            //X 轴线条显示间隔
            //chart2.ChartAreas["ChartArea1"].AxisX.MajorGrid.IntervalType =
DateTimeIntervalType.Hours;
            chart2.Palette = ChartColorPalette.Pastel;
            string sql = "select sum(zhl) zhl, input_date, ash_type_name from (" +
            " select sum(t.second_load - t.first_load) as zhl," +
                  " to_date(to_char(t.input_date, 'dd/mm/yyyy')||'00:00','dd/mm/yyyy
hh24:mi:ss') as input_date," +
                  " u.ash_type_name" +
             " from transportation_bill t, ash_type u" +
             " where t.ash_type_id = u.ash_type_id" +
              " and (t.input_date between to_date('2009-6-11','yyyy-mm-dd') and
to_date('2009-6-20','yyyy-mm-dd'))" +//此处加按日期查询的条件
               " and t.sale_organization_id = 1" +//此处加用户所在机构查询条件
             " group by u.ash_type_name, t.input_date" +
             " order by t.input_date" +
             ")" +
             " group by ash_type_name,input_date" +
            " order by input_date";
            DataTable dt2 = OracleHelper.ExecuteDataTable(OracleHelper.ConnectionString,
CommandType.Text, sql, "T", null);
```

```
            chart2.DataBindCrossTable(dt2.DefaultView, "ash_type_name", "input_date", "zhl",
"", PointSortOrder.Ascending);
            foreach (Series sr in chart2.Series)
            {
                sr.ChartType = SeriesChartType.Spline;
                sr.XValueType = ChartValueType.Date;
                sr.MarkerStyle = MarkerStyle.Circle;//散点样式
                sr.MarkerSize = 5;//默认是5，散点大小
                sr.MarkerStep = 1;//默认是1，散点频率
                sr.MarkerColor = 1;//默认是透明，散点颜色
                sr.BorderWidth = 2;
            }
            //chart2.Legends["Default"].Docking = Docking.Left;
        }
```

1.什么是 MSChart

Chart: Microsoft Chart Controls for Microsoft .NET Framework 3.5

- 适用于.NET Framework 3.5 SP1 的 ASP.NET 和 Windows Form 图标控制项
- 开发商：2007 年 Dundas 开发出收费的 Chart 控件　　2008/9/8 微软发布免费的 MSChart 控件
- Support Service：　　system :win2003 sp2 ,win2008,windows vista,winxp sp3　　.net Framework: .NET Framework 3.5 SP1(VS2008)
- 处理器：400 MHz Pentium(最低)；1GHz Pentium（建议）　　RAM：96 MB（最小）256MB（建议）　　Hard：　500MB 可用磁盘空间
- 显卡：　800*600、256 色（小）　1024*768、256 色

2.MSchart 安装

- 安装完成 VS2008（.NET Framework 3.5 SP1）下载地址
-

1) dotnetfx35setup.exe
2) MSChart.exe
3) MSChart_VisualStudioAddOn.exe
4) MSChartLP_chs.exe

- 安装顺序：

（1）dotnetfx35setup.exe

（2）MSChart_VisualStudioAddOn.exe

（3）MSChartLP_chs.exe

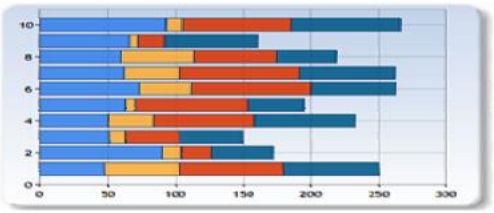
（4）MSChart.exe

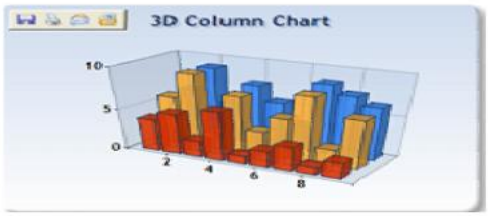
打开 **VS2008** 在工具中新增的 **Data Chart** 控件：

MSChart 图表类型

1.Bar and Column Charts 3D or 2D Bar or Column Charts like this page:

-

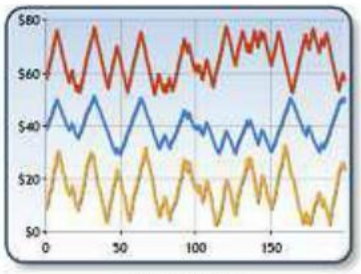


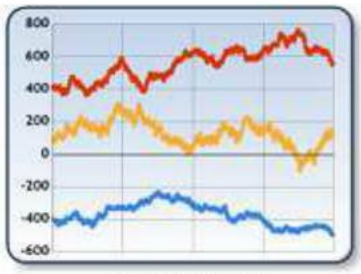


Stacked charts Bar and Column

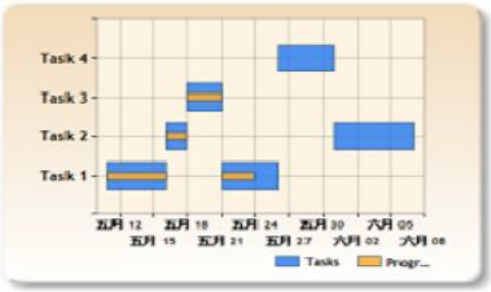
3D Bar and Column

2.Line Charts

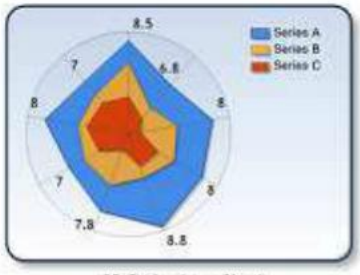


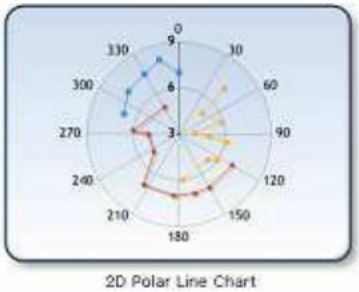


3. Range Charts



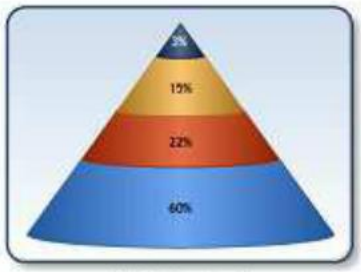
4. Circular charts





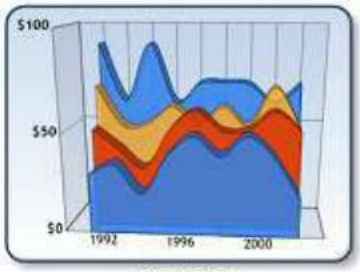
5. Accumulation charts

3D Funnel Chart with Point Gaps

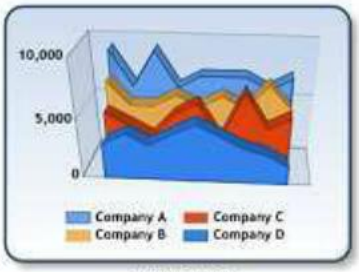


3D Pyramid Chart

6. Area Chart



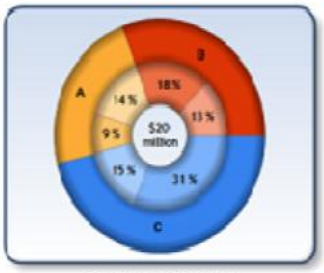
3D Spline Area



3D Area Chart

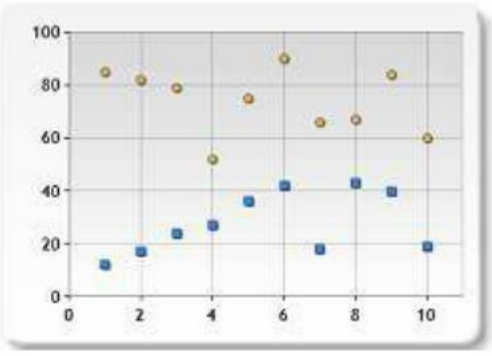
7. Pie and Doughnut charts

2D Doughrut Chart

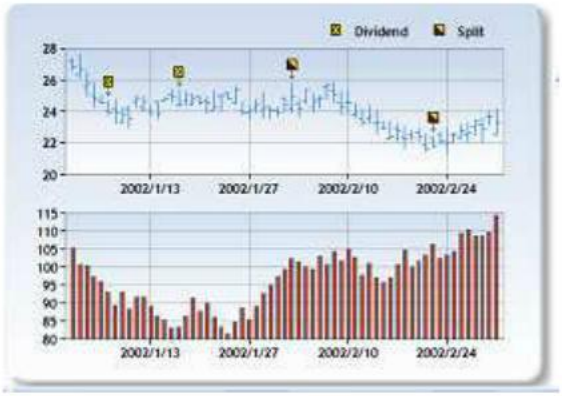


2D Grouped Pie Chart

8. Point charts

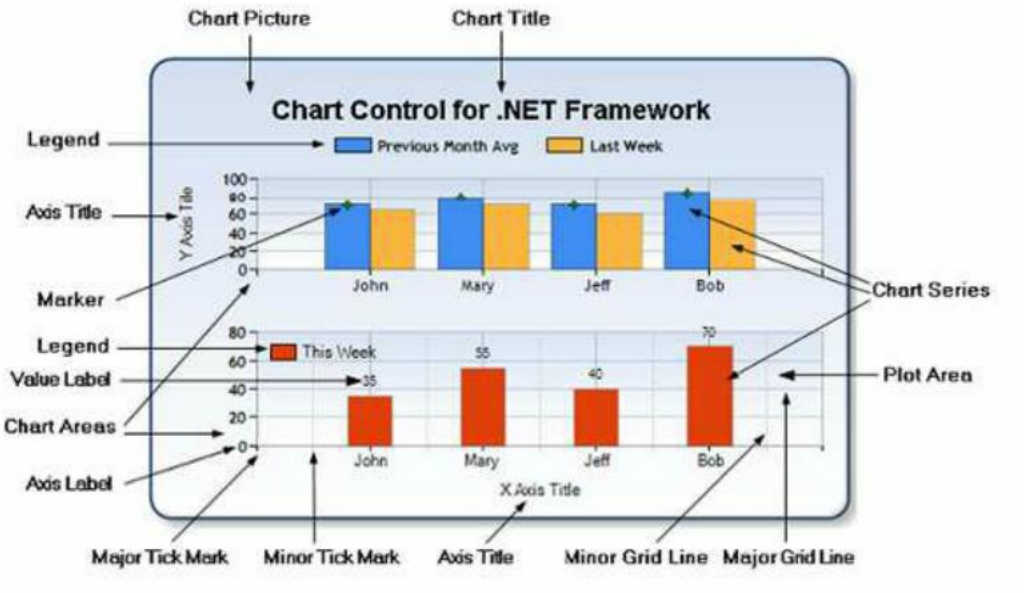


9. Advanced Financial charts



10. Combinational Charts

3.MSChart 的性介绍

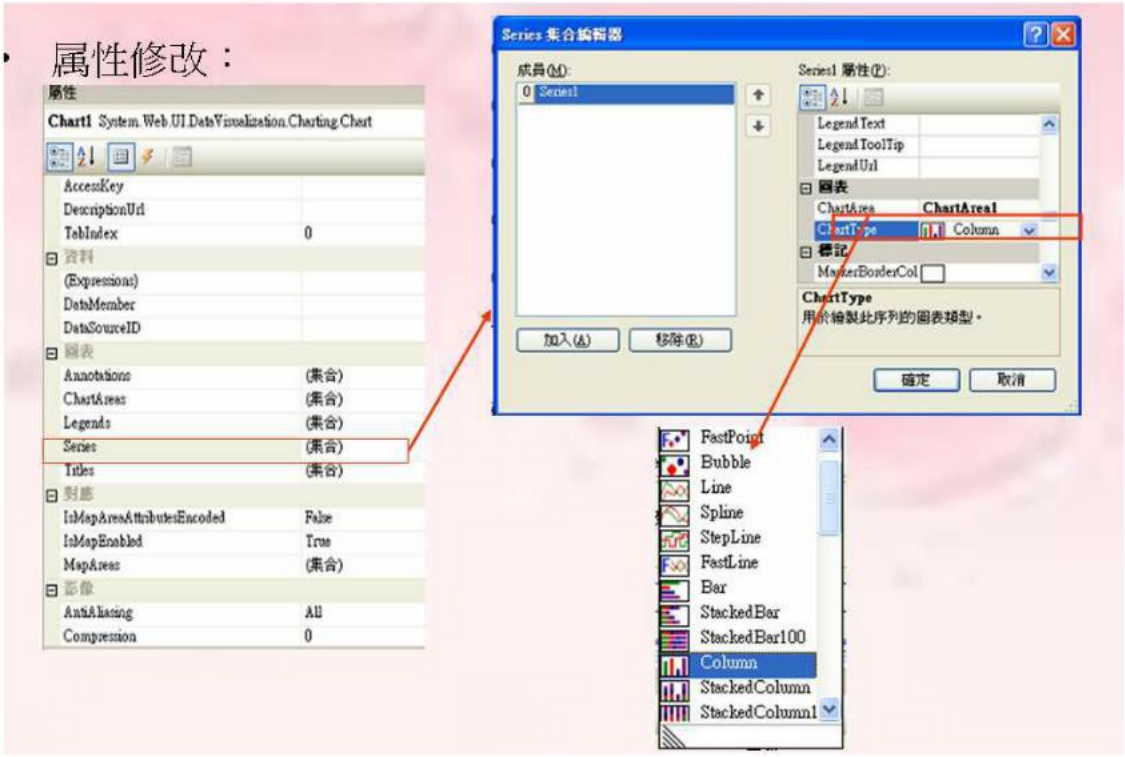


这些属性在添加 chart 的时候会有。

开启 VS2008 ，新建网站 ChartTest 打开主界面:

点击 Chart 在页面中添加 Chart 图形，在属性栏中选择 Chart 类型

- 属性修改：



==========

昨天在网上看到了微软发布了.NET 3.5 框架下的图表控件，第一时间抓下来看了一下，发觉功能很强劲，基本上能想到的图表都可以使用它绘制出来，给图形统计和报表图形显示提供了很好的解决办法，同时支持 Web 和 WinForm 两种方式，不过缺点也比较明显，只能在最新的开发环境中使用，需要.Net 3.5 Sp1 以及 VS 2008 的开发环境。

下面是下载地址：

控件：Microsoft .NET Framework 3.5 的 Microsoft 图表控件(Microsoft Chart Controls for Microsoft .NET Framework 3.5)–

- 包含英文版，中文版。上面提供的链接是中文版的，可以更改为英文版。

- 语言包：Microsoft Chart Controls for Microsoft .NET Framework 3.5 Language Pack
- Microsoft .NET Framework 3.5 的 Microsoft 图表控件 的语言包，包含 23 中语言。
- Microsoft Chart Controls Add-on for Microsoft Visual Studio 2008 –

这个只有英文的，没找到中文的。

- 文档 （Microsoft Chart Controls for .NET Framework Documentation）–

这个只有英文的，没找到中文的。

- WinForm 和 Asp.net 的例子（Samples Environment for Microsoft Chart Controls）–

这个只有英文的，没找到英文的。

- Demo 下载：http://code.msdn.microsoft.com/mschart

下了它的示例程序后，运行了一下，非常的强大，可以支持各种各样的图形显示，常见的：点状图、饼图、柱状图、曲线图、面积图、排列图等等，同时也支持 3D 样式的图表显示，不过我觉得最有用的功能还是**支持图形上各个点的属性操作，它可以定义图形上各个点、标签、图形的提示信息（Tooltip）以及超级链接、Javascript 动作等**，而不是像其它图形类库仅生成一幅图片而已，通过这些，加上微软自己的 Ajax 框架，可以建立一个可以互动的图形统计报表了。

一。安装

控件的安装相对比较简单，下载完后，先执行"MSChart.exe"程序，它会自动检测你的环境，安装到系统目录中去，如果要在 VS 2008 环境中直接使用，那么需要安装 For Vs2008 的插件，MSChart_VisualStudioAddOn.exe，还有一个中文语言包 MSChartLP_chs.exe。安装完后，打开 Vs2008，在建立项目的时候，你就能在工具栏中看到有一个 Chart 的控件了，如下图：

二。使用

安装好后，建立一个.NET3.5 的 Web 项目，像使用普通控件一样拖放到要使用的 Web 界面即可。初步研究了一下，整个图形控件主要由以下几个部份组成：

1.Annotations --图形注解集合

2.ChartAreas --图表区域集合

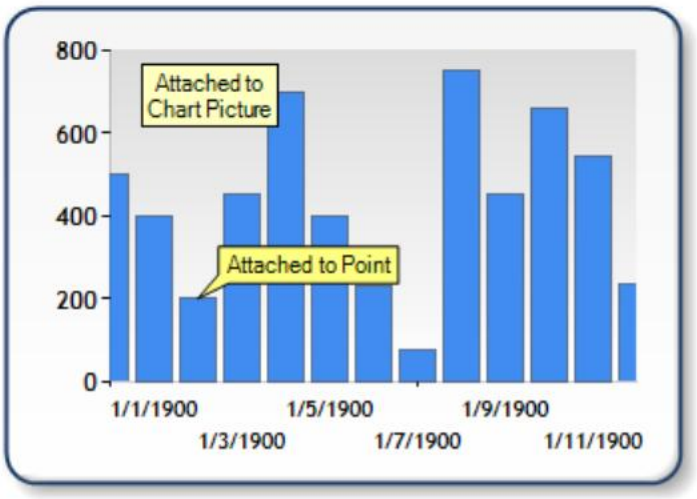
3.Legends --图例集合

4.Series --图表序列集合（即图表数据对象集合）

5.Titles --图标的标题集合

Annotations 注解集合

Annotations 是一个对图形的一些注解对象的集合，所谓注解对象，类似于对某个点的详细或者批注的说明，比如，在图片上实现各个节点的关键信息，如下图方框和黄色的小方框：



一个图形上可以拥有多个注解对象，可以添加十多种图形样式的注解对象，包括常见的箭头、云朵、矩行、图片等等注解符号，通过各个注解对象的属性，可以方便的设置注解对象的放置位置、呈现的颜色、大小、文字内容样式等常见的属性。

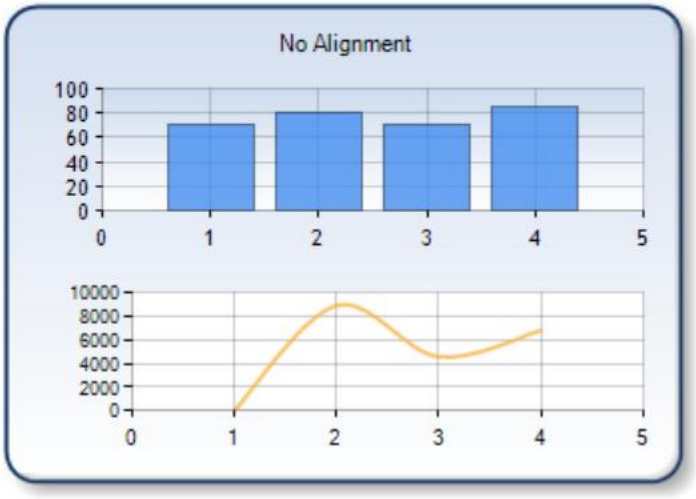
ChartAreas 图表区域集合

ChartAreas 可以理解为是一个图表的绘图区，例如，你想在一幅图上呈现两个不同属性的内容，一个是用户流量，另一个则是系统资源占用情况，那么你要在一个图形上绘制这两种情况，明显是不合理的，对于这种情况，可以建立两个 ChartArea，一个用于呈现用户流量，另一个则用于呈现系统资源的占用情况。

当然了，图表控件并不限制你添加多少个绘图区域，你可以根据你的需要进行添加。对于每一个绘图区域，你可以设置各自的属性，如：X,Y 轴属性、背景等。

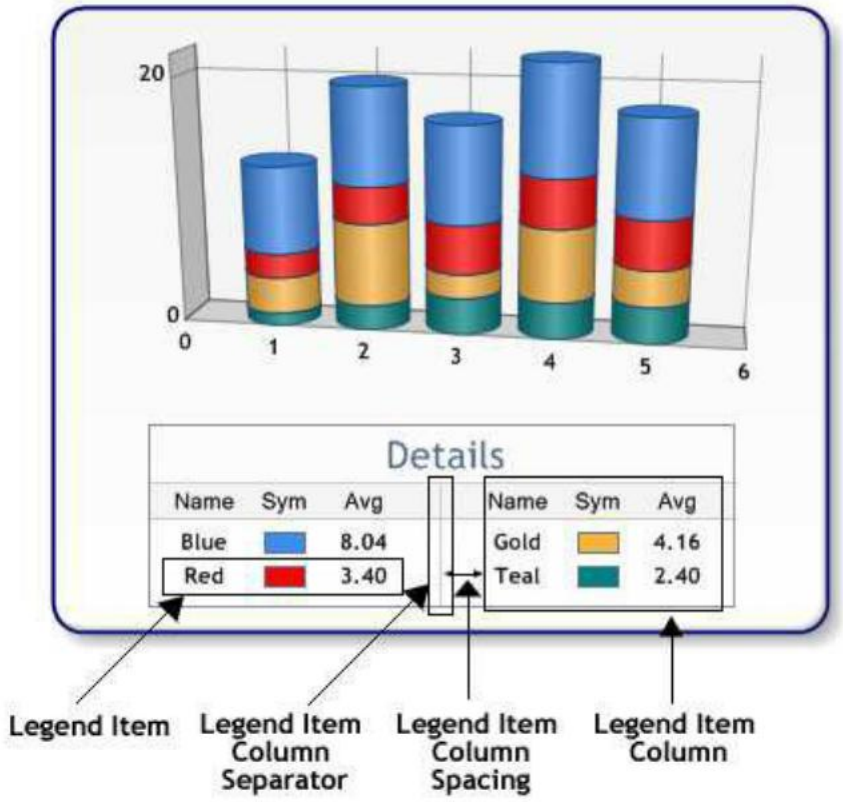
需要注意的是，绘图区域只是一个可以作图的区域范围，它本身并不包含要作图形的各种属性数据。

多绘图区效果图如下，分为上下两个绘图区域，分别表示不同的绘图数据：



Legends 图例集合

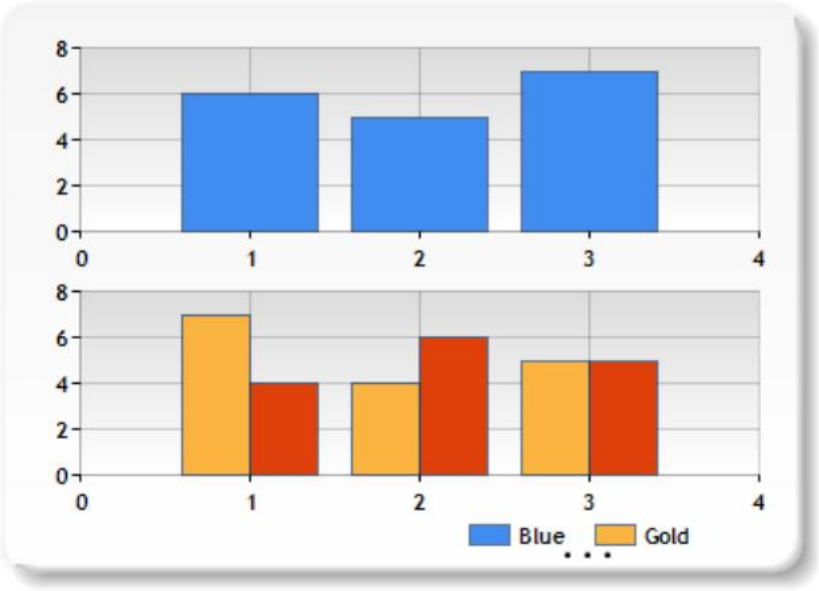
Legends 是一个图例的集合，即标注图形中各个线条或颜色的含义，同样，一个图片也可以包含多个图例说明，比如像上面说的多个图表区域的方式，则可以建立多个图例，每别说明各个绘图区域的信息，具体的图例配置说明此处就不详细说明了，可以参考一下官网的例子，写得丰富的详细了：）也上一张图例的效果图吧~

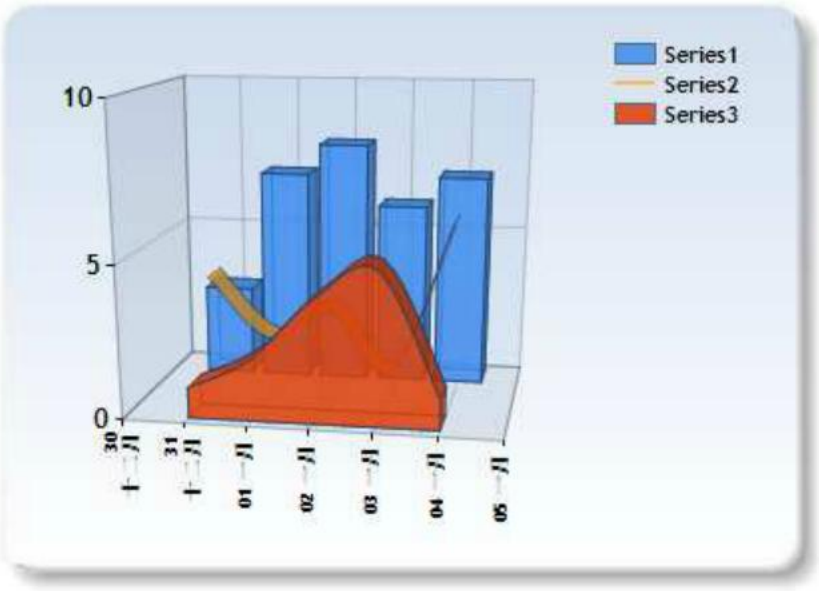


Series 图表序列

图表序列，应该是整个绘图中最关键的内容了，通俗点说，即是实际的绘图数据区域，实际呈现的图形形状，就是由此集合中的每一个图表来构成的，可以往集合里面添加多个图表，每一个图表可以有自己的绘制形状、样式、独立的数据等。

需要注意的是，每一个图表，你可以指定它的绘制区域（见 ChartAreas 的说明），让此图表呈现在某个绘图区域，也可以让几个图表在同一个绘图区域叠加，如下图：





上面两幅图，分别表示了把图表放在不同的绘制区域和放在同一个绘制区域的情况。

继续回到 ChartAreas 章节举的例子，同时要显示用户的流量还要显示系统的占用情况，对于这种时候，应该建立两个 Series，一个用于呈现用户的流量，另一个则用于呈现系统的占用情况。它们分别属于各自的绘图区域。

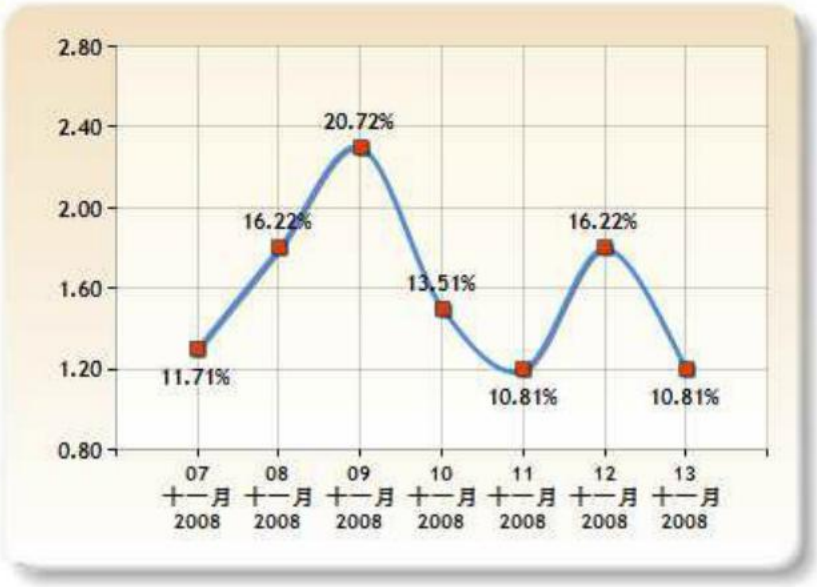
Titles 标题合集

根据字面含义即可以理解，是图表的标题配置，同样可以添加多个标题，以及设置标题的样式及文字、位置等属性。多看一下它的属性即能明白各自的含义。

三。其它属性

相对来说，我觉得比较有用的属性有三个，分别是：Label、Tooltip 以及 Url 链接。

Label 即标签的含义，可以在图片的关键位置进行一些关键数字或文字的描述，如下图：



像上图：X 轴和 Y 轴的文字便是标签，以及图表曲线中的红点上的文字，也是标签，添加了标签，可以让人更容易的对内容进行理解。

Tooltip 即提示的含义，用于在各个关键点，如：标签、图形关键点、标题等当鼠标移动上去的时候，提示用户一些相关的详细或说明信息，例如上图，可以给曲线中的每一个点增加 Tooltip 的属性，写上需要详细说明的内容，比如：详细的销售明细，那么，在鼠标移动到这个点的时候，会自动弹出提示信息。

Tooltip 可以支持简单方式以及自定义的方式，简单方式即像平时 Html 页面设置的 title 之类的属性效果，而自定义的方式，则可以实现图形、文本等各种复杂的提示信息显示。详细的方式请参考官方例子的：Interactivity and AJAX/Tooltips 以及 Interactivity and AJAX/Client Side Scripts 下面的相关例子。

Url 链接，图表控件中，有一大半的控件都有 Url 及 Tooltip 的属性，你可以设置此属性，在鼠标点击的时候，代到其它相应的页面去。

建议大家看看官方例子中的 **Interactivity and AJAX** 部份，很精彩：）

例子：建立一个 **Cpu** 信息和内存使用的实时统计表

下面写一个小例子，建立一个系统的内存实时统计图表，使用到了 Ajax 的方法，以及 Windows Api 取得系统内存的方法。

首先，建立一个 Aspx 页面，拖动一个图表控件到页面，设置图表控件的属性如下：

其中，MEMORY_INFO,ComputerInfo 是一个定义的结构体及调用 Win32 API 接口的一个访问类。程序分别取得每一个图表对象，每次加载的时候，都重新取得当前的内存和 Cpu 信息，再在图表上添加一个点，需要注意的是，一定要设置图表控件的 EnableViewState 属性为 True，否则无法记录状态。

```
<asp:Chart ID="ChartMemory" runat="server" BackColor="LightSteelBlue"
BackGradientStyle="TopBottom" BackSecondaryColor="White" EnableTheming="False"
EnableViewState="True" Height="363px" Width="415px">
<Legends>
<asp:Legend Alignment="Center" Docking="Bottom" Name="Legend1" Title="图例">
</asp:Legend>
</Legends>
<Titles>
<asp:Title Font="微软雅黑, 16pt" Name="Title1" Text="系统内存监控图表">
</asp:Title>
</Titles>
<Series>
<asp:Series BorderColor="White" BorderWidth="3" ChartArea="ChartArea1"
ChartType="Spline" Legend="Legend1" Name="已使用物理内存" XValueType="Double"
YValueType="Double">
</asp:Series>
<asp:Series BorderWidth="3" ChartArea="ChartArea1" ChartType="Spline"
Legend="Legend1" Name="全部占用内存">
</asp:Series>
<asp:Series ChartArea="ChartArea2" ChartType="StackedArea" Legend="Legend1"
Name="CPU">
</asp:Series>
</Series>
<ChartAreas>
<asp:ChartArea BackColor="224, 224, 224" BackGradientStyle="LeftRight"
Name="ChartArea1">
</asp:ChartArea>
<asp:ChartArea Name="ChartArea2">
</asp:ChartArea>
</ChartAreas>
</asp:Chart>
```

一共建立了两个绘图区，一个用于呈现内存使用情况的在 ChartArea1 区域，另一个则是呈现 Cpu 使用情况的，放置在 ChartArea2 区域了。一共有三个图表，分别表示已使用的物理内存、全部占用的物理内存，以及 Cpu 使用显示的情况。

添加一个 Ajax 的计时器以及 Ajax 的 ScriptManager，UpdatePanel，把计时器和图表控件都拖进 UpdatePanel 里面。设置计时器的间隔时间为一秒钟（1000），双击计时器，写如下代码：

```
static PerformanceCounter pc = new PerformanceCounter("Processor", "% Processor
Time", "_Total");
protected void Timer1_Tick(object sender, EventArgs e)
{
MEMORY_INFO MemInfo = new MEMORY_INFO();
ComputerInfo.GlobalMemoryStatus(ref MemInfo);
//UseMemory
Series series = ChartMemory.Series[0];
int xCount = series.Points.Count == 0 ? 0 : series.Points.Count - 1;
double lastXValue = series.Points.Count == 0 ? 1 : series.Points[xCount].XValue
+ 1;
double lastYValue = (double)(MemInfo.dwTotalPhys-MemInfo.dwAvailPhys)/1024/1024;
series.Points.AddXY(lastXValue, lastYValue);
//Total Memory
series = ChartMemory.Series[1];
lastYValue = (double)(MemInfo.dwTotalVirtual+MemInfo.dwTotalPhys-MemInfo.dwAvai
lPhys - MemInfo.dwAvailVirtual)/1024/1024;
series.Points.AddXY(lastXValue, lastYValue);

//CPU
series = ChartMemory.Series[2];
lastYValue = (double)pc.NextValue();
series.Points.AddXY(lastXValue, lastYValue);

// Remove points from the left chart side if number of points exceeds 100.
while (this.ChartMemory.Series[0].Points.Count > 80)
{
// Remove series points
foreach (Series s in this.ChartMemory.Series)
{
s.Points.RemoveAt(0);
}
}
// Adjust categorical scale
double axisMinimum = this.ChartMemory.Series[0].Points[0].XValue;
this.ChartMemory.ChartAreas[0].AxisX.Minimum = axisMinimum;
this.ChartMemory.ChartAreas[0].AxisX.Maximum = axisMinimum + 99;
}
```

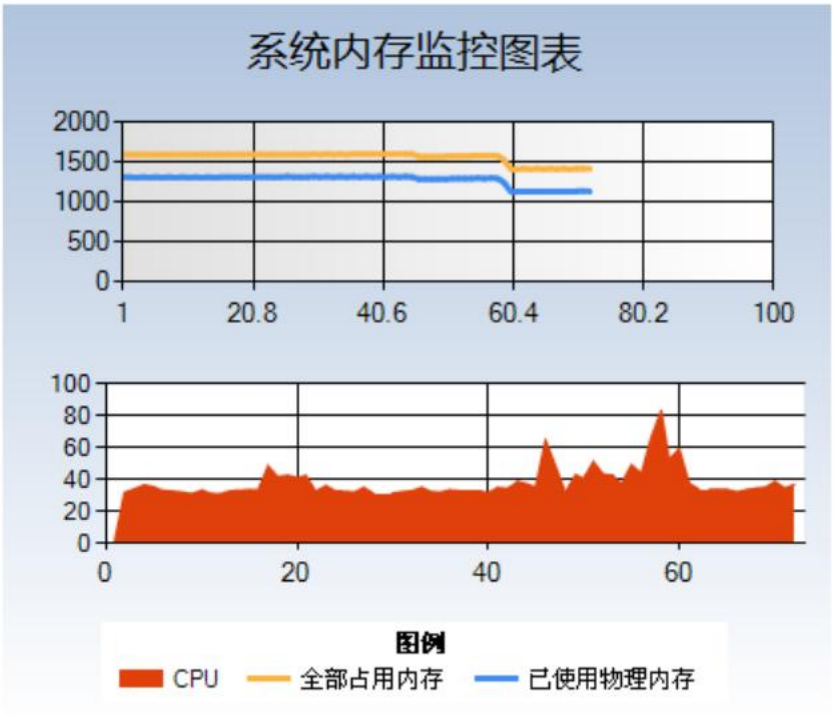
附上取得内存信息的类代码：

```
/// <summary>
///取得计算机的系统信息
/// </summary>
public class ComputerInfo
{
/// <summary>
/// 取得 Windows 的目录
/// </summary>
/// <param name="WinDir"></param>
/// <param name="count"></param>
[DllImport("kernel32")]
public static extern void GetWindowsDirectory(StringBuilder WinDir, int count);
/// <summary>
/// 获取系统路径
/// </summary>
/// <param name="SysDir"></param>
/// <param name="count"></param>
[DllImport("kernel32")]
public static extern void GetSystemDirectory(StringBuilder SysDir, int count);
/// <summary>
/// 取得 CPU 信息
/// </summary>
/// <param name="cpuinfo"></param>
[DllImport("kernel32")]
public static extern void GetSystemInfo(ref CPU_INFO cpuinfo);
/// <summary>
/// 取得内存状态
/// </summary>
/// <param name="meminfo"></param>
[DllImport("kernel32")]
public static extern void GlobalMemoryStatus(ref MEMORY_INFO meminfo);
/// <summary>
/// 取得系统时间
/// </summary>
/// <param name="stinfo"></param>
[DllImport("kernel32")]
public static extern void GetSystemTime(ref SYSTEMTIME_INFO stinfo);

public ComputerInfo()
{
}
}
```

```
//定义 CPU 的信息结构
[StructLayout(LayoutKind.Sequential)]
public struct CPU_INFO
{
public uint dwOemId;
public uint dwPageSize;
public uint lpMinimumApplicationAddress;
public uint lpMaximumApplicationAddress;
public uint dwActiveProcessorMask;
public uint dwNumberOfProcessors;
public uint dwProcessorType;
public uint dwAllocationGranularity;
public uint dwProcessorLevel;
public uint dwProcessorRevision;
}
//定义内存的信息结构
[StructLayout(LayoutKind.Sequential)]
public struct MEMORY_INFO
{
public uint dwLength;
public uint dwMemoryLoad;
public uint dwTotalPhys;
public uint dwAvailPhys;
public uint dwTotalPageFile;
public uint dwAvailPageFile;
public uint dwTotalVirtual;
public uint dwAvailVirtual;
}
//定义系统时间的信息结构
[StructLayout(LayoutKind.Sequential)]
public struct SYSTEMTIME_INFO
{
public ushort wYear;
public ushort wMonth;
public ushort wDayOfWeek;
public ushort wDay;
public ushort wHour;
public ushort wMinute;
public ushort wSecond;
public ushort wMilliseconds;
}
```

运行的效果图如下：



先写到这里吧~呵呵，第一次写教程，经验不足，希望各位提点意见哈~到时再看看有没有必要再继续写一篇