- 视频转换（见图 1）
  - 配置：

<appSettings>

<!--工具文件夹-->

<add key="ffmpeg" value="ffmpeg/ffmpeg.exe"/>

<add key="mencoder" value="mencoder/mencoder.exe"/>

<add key="mplayer" value="mencoder/mplayer.exe"/>

<!--上传文件的路径-->

<add key="upfile" value="UpFiles"/>

<!--上专文件图片路径-->

<add key="imgfile" value="ImgFile"/>

<!--上传文件图片大小-->

<add key="CatchFlvImgSize" value="240x180"/>

<add key="widthSize" value="400"/>

<add key="heightSize" value="350"/>

<!--转换后文件路径-->

<add key="playfile" value="PlayFiles"/>

</appSettings>

  - 前台：

标题：<asp:TextBox ID="txtTitle" runat="server" Width="358px"></asp:TextBox>

<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
ControlToValidate="txtTitle"

ErrorMessage="标题不为空"></asp:RequiredFieldValidator>

<asp:FileUpload ID="FileUpload1" runat="server" Width="339px" />

<asp:Button ID="btnUpload" runat="server" OnClick="btnUpload_Click" Text="上传视频
" Width="70px" />

文件类型(.asf|.flv|.avi|.mpg|.3gp|.mov|.wmv|.rm|.rmvb)

<asp:RegularExpressionValidator ID="imagePathValidator" runat="server"
ErrorMessage="文件类型不正确"

ValidationGroup="vgValidation" Display="Dynamic" ValidationExpression="^[a-zA-
Z]:(\\.+)(.asf|.flv|.avi|.mpg|.3gp|.mov|.wmv|.rm|.rmvb){1}quot;

ControlToValidate="FileUpload1">

</asp:RegularExpressionValidator>

<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
ControlToValidate="FileUpload1"

ErrorMessage="文件不为空"></asp:RequiredFieldValidator>

上传列表.

■ 后台：

using System;

using System.Data;

using System.Configuration;

```
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
public partial class _Default : System.Web.UI.Page
{
// 扩展名定义
string[] strArrFfmpeg = new string[] {"asf","avi","mpg","3gp","mov" };
string[] strArrMencoder = new string[] {"wmv","rm","rmvb" };
protected void Page_Load(object sender, EventArgs e)
{
}
//
protected void btnUpload_Click(object sender, EventArgs e)
{
string upFileName = "";
if (this.FileUpload1.HasFile)
{
```

```
string fileName = PublicMethod.GetFileName(this.FileUpload1.FileName);//
GetFileName();

if ((string)Session["file"] == fileName)

{

return;

}

upFileName = Server.MapPath(PublicMethod.upFile + fileName);

this.FileUpload1.SaveAs(upFileName);

string saveName = DateTime.Now.ToString("yyyyMMddHHmmssffff") ;

string playFile = Server.MapPath(PublicMethod.playFile + saveName);

string imgFile = Server.MapPath(PublicMethod.imgFile + saveName);

//System.IO.File.Copy(Server.MapPath(PublicMethod.playFile + "00000002.jpg"),
Server.MapPath(PublicMethod.imgFile+"aa.jpg"));

PublicMethod pm = new PublicMethod();

string m_strExtension =
PublicMethod.GetExtension(this.FileUpload1.PostedFile.FileName).ToLower();

if (m_strExtension == "flv")

{//直接拷贝到播放文件夹下

System.IO.File.Copy(upFileName, playFile+".flv");

pm.CatchImg(upFileName, imgFile);

}

string Extension = CheckExtension(m_strExtension);
```

```
if (Extension == "ffmpeg")
{
      pm.ChangeFilePhy(upFileName, playFile, imgFile);
}
else if (Extension == "mencoder")
{
pm.MChangeFilePhy(upFileName, playFile, imgFile);
}
InsertData(this.txtTitle.Text, fileName,saveName);
Session["file"] = fileName;
}
}
//
private string CheckExtension(string extension)
{
string m_strReturn = "";
foreach (string var in this.strArrFfmpeg)
{
if (var == extension)
{
m_strReturn = "ffmpeg"; break;
```

```
}
}
if (m_strReturn == "")
{
foreach (string var in strArrMencoder)
{
if (var == extension)
{
m_strReturn = "mencoder"; break;
}
}
}
return m_strReturn;
}
#region 插入数据到数据库中
private void InsertData(string MediaName,string fileName,string saveName)
{
//string name=fileName.Substring(0, fileName.LastIndexOf('.'));
string imgName = saveName + ".jpg";//图片文件名;
string playName = saveName + ".flv";
```

```
string sqlstr = "insert into
Media(FMediaName,FMediaUpPath,FMediaPlayPath,FMediaImgPath)
values(@MName,@MUppath,@MPlaypath,@MImgpath)";

//string constr = ConfigurationManager.ConnectionStrings["sqlcon"].ToString();

SqlDataSource1.InsertCommand = sqlstr;

SqlDataSource1.InsertCommandType = SqlDataSourceCommandType.Text;//
CommandType.Text;

SqlDataSource1.InsertParameters.Add("MName",MediaName);

SqlDataSource1.InsertParameters.Add("MUppath",PublicMethod.upFile+fileName);

SqlDataSource1.InsertParameters.Add("MPlaypath",PublicMethod.playFile+playName);

SqlDataSource1.InsertParameters.Add("MImgpath",PublicMethod.imgFile+imgName);

SqlDataSource1.Insert();

}

#endregion

}
```

PublicMethod 类如下：

在这个类里面主要是做文件转换和保存，在转换文件的时候 CPU 的利用率可以达 100%.

它的主要原理是另起一个进程在转换的时候你会发现你的进程里多了一个.

```
using System;

using System.Configuration;

/// <summary>

/// Summary description for PublicMethod
```

```
/// </summary>
public class PublicMethod:System.Web.UI.Page
{
public PublicMethod()
{
}
//文件路径
public static string ffmpegtool = ConfigurationManager.AppSettings["ffmpeg"];
public static string mencodertool = ConfigurationManager.AppSettings["mencoder"];
public static string mplayertool = ConfigurationManager.AppSettings["mplayer"];
public static string upFile = ConfigurationManager.AppSettings["upfile"] + "/";
public static string imgFile = ConfigurationManager.AppSettings["imgfile"] + "/";
public static string playFile = ConfigurationManager.AppSettings["playfile"] + "/";
//文件图片大小
public static string sizeOfImg = ConfigurationManager.AppSettings["CatchFlvImgSize"];
//文件大小
public static string widthOfFile = ConfigurationManager.AppSettings["widthSize"];
public static string heightOfFile = ConfigurationManager.AppSettings["heightSize"];
// // //获取文件的名字
public static string GetFileName(string fileName)
{
```

```
int i = fileName.LastIndexOf("\\") + 1;
string Name = fileName.Substring(i);
return Name;
}
//获取文件扩展名
public static string GetExtension(string fileName)
{
int i = fileName.LastIndexOf(".")+1;
string Name = fileName.Substring(i);
return Name;
}
//
#region //运行 FFMpeg 的视频解码，(这里是绝对路径)
/// <summary>
/// 转换文件并保存在指定文件夹下面(这里是绝对路径)
/// </summary>
/// <param name="fileName">上传视频文件的路径（原文件）</param>
/// <param name="playFile">转换后的文件的路径（网络播放文件）</param>
/// <param name="imgFile">从视频文件中抓取的图片路径</param>
/// <returns>成功:返回图片虚拟地址; 失败:返回空字符串</returns>
public string ChangeFilePhy(string fileName, string playFile, string imgFile)
```

```
{
//取得 ffmpeg.exe 的路径,路径配置在 Web.Config 中,如:<add key="ffmpeg"
value="E:\51aspx\ffmpeg.exe" />
string ffmpeg = Server.MapPath(PublicMethod.ffmpegtool);
if ((!System.IO.File.Exists(ffmpeg)) || (!System.IO.File.Exists(fileName)))
{
return "";
}
//获得图片和(.flv)文件相对路径/最后存储到数据库的路径,如:/Web/User1/00001.jpg
string flv_file = System.IO.Path.ChangeExtension(playFile, ".flv");

//截图的尺寸大小,配置在 Web.Config 中,如:<add key="CatchFlvImgSize" value="240x180"
/>
string FlvImgSize = PublicMethod.sizeOfImg;
System.Diagnostics.ProcessStartInfo FilestartInfo = new
System.Diagnostics.ProcessStartInfo(ffmpeg);
FilestartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
FilestartInfo.Arguments = " -i " + fileName + " -ab 56 -ar 22050 -b 500 -r 15 -s " +
widthOfFile + "x" + heightOfFile + " " + flv_file;
//ImgstartInfo.Arguments = " -i " + fileName + " -y -f image2 -t 0.05 -s " + FlvImgSize +
" " + flv_img;
try
{
```

```
//转换
System.Diagnostics.Process.Start(FilestartInfo);
//截图
CatchImg(fileName, imgFile);
//System.Diagnostics.Process.Start(ImgstartInfo);
}
catch
{
return "";
}
//
return "";
}
//
public string CatchImg(string fileName,string imgFile)
{
//
string ffmpeg = Server.MapPath(PublicMethod.ffmpegtool);
//
string flv_img =imgFile+".jpg";
//
```

```
string FlvImgSize = PublicMethod.sizeOfImg;

//

System.Diagnostics.ProcessStartInfo ImgstartInfo = new
System.Diagnostics.ProcessStartInfo(ffmpeg);

ImgstartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;

//

ImgstartInfo.Arguments = " -i " + fileName + " -y -f image2 -ss 2 -vframes 1 -s " +
FlvImgSize + " " + flv_img;

try

{

System.Diagnostics.Process.Start(ImgstartInfo);

}

catch

{

return "";

}

//

if (System.IO.File.Exists(flv_img))

{

return flv_img;

}

return "";
```

```
}
#endregion
//
#region //运行 FFMpeg 的视频解码，(这里是(虚拟)相对路径)
/// <summary>
/// 转换文件并保存在指定文件夹下面(这里是相对路径)
/// </summary>
/// <param name="fileName">上传视频文件的路径（原文件）</param>
/// <param name="playFile">转换后的文件的路径（网络播放文件）</param>
/// <param name="imgFile">从视频文件中抓取的图片路径</param>
/// <returns>成功:返回图片虚拟地址; 失败:返回空字符串</returns>
public string ChangeFileVir(string fileName, string playFile, string imgFile)
{
//取得 ffmpeg.exe 的路径,路径配置在 Web.Config 中,如:<add key="ffmpeg"
value="E:\51aspx\ffmpeg.exe" />
string ffmpeg = Server.MapPath(PublicMethod.ffmpegtool);
if ((!System.IO.File.Exists(ffmpeg)) || (!System.IO.File.Exists(fileName)))
{
return "";
}
//获得图片和(.flv)文件相对路径/最后存储到数据库的路径,如:/Web/User1/00001.jpg
```

```
string flv_img = System.IO.Path.ChangeExtension(Server.MapPath(imgFile), ".jpg");

string flv_file = System.IO.Path.ChangeExtension(Server.MapPath(playFile), ".flv");


//截图的尺寸大小,配置在 Web.Config 中,如:<add key="CatchFlvImgSize" value="240x180"
/>

string FlvImgSize = PublicMethod.sizeOfImg;

System.Diagnostics.ProcessStartInfo FilestartInfo = new
System.Diagnostics.ProcessStartInfo(ffmpeg);

System.Diagnostics.ProcessStartInfo ImgstartInfo = new
System.Diagnostics.ProcessStartInfo(ffmpeg);

FilestartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;

ImgstartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;

//此处组合成 ffmpeg.exe 文件需要的参数即可,此处命令在 ffmpeg 0.4.9 调试通过

//ffmpeg -i F:\01.wmv -ab 56 -ar 22050 -b 500 -r 15 -s 320x240 f:\test.flv

FilestartInfo.Arguments = " -i " + fileName + " -ab 56 -ar 22050 -b 500 -r 15 -s " +
widthOfFile + "x" + heightOfFile + " " + flv_file;

ImgstartInfo.Arguments = " -i " + fileName + " -y -f image2 -t 0.001 -s " + FlvImgSize +
" " + flv_img;

try

{

System.Diagnostics.Process.Start(FilestartInfo);

System.Diagnostics.Process.Start(ImgstartInfo);

}
```

```
catch
{
return "";
}
/**/
///注意:图片截取成功后,数据由内存缓存写到磁盘需要时间较长,大概在 3,4 秒甚至更长;
///这儿需要延时后再检测,我服务器延时 8 秒,即如果超过 8 秒图片仍不存在,认为截图失败;
///此处略去延时代码.如有那位知道如何捕捉 ffmpeg.exe 截图失败消息,请告知,先谢过!
if (System.IO.File.Exists(flv_img))
{
return flv_img;
}
return "";
}
#endregion
#region //运行 mencoder 的视频解码器转换(这里是(绝对路径))
public string MChangeFilePhy(string vFileName, string playFile, string imgFile)
{
string tool = Server.MapPath(PublicMethod.mencodertool);
//string mplaytool = Server.MapPath(PublicMethod.ffmpegtool);
if ((!System.IO.File.Exists(tool)) || (!System.IO.File.Exists(vFileName)))
```

```
{
return "";
}
string flv_file = System.IO.Path.ChangeExtension(playFile, ".flv");

//截图的尺寸大小,配置在 Web.Config 中,如:<add key="CatchFlvImgSize" value="240x180"
/>
string FlvImgSize = PublicMethod.sizeOfImg;
System.Diagnostics.ProcessStartInfo FilestartInfo = new
System.Diagnostics.ProcessStartInfo(tool);
FilestartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
FilestartInfo.Arguments = " " + vFileName + " -o " + flv_file + " -of lavf -lavfopts
i_certify_that_my_video_stream_does_not_use_b_frames -oac mp3lame -lameopts
abr:br=56 -ovc lavc
-lavcopts vcodec=flv:vbitrate=200:mbd=2:mv0:trell:v4mv:cbp:last_pred=1:dia=-
1:cmp=0:vb_strategy=1
-vf scale=" + widthOfFile + ":" +heightOfFile + " -ofps 12 -srate 22050";
try
{
System.Diagnostics.Process.Start(FilestartInfo);
CatchImg(flv_file, imgFile);
}
```

```
catch
{
return "";
}
//
return "";
}
#endregion
}
```

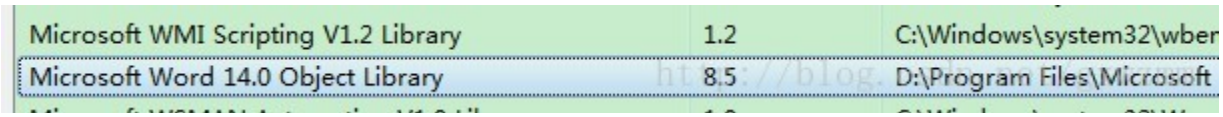
- 文档转换（见图 2）

  - 使用微软 Office 的 COM 组件+iTextShape 进行转化。

主要思路如下：首先通过 COM 组件读取 Word/Excle 文档的相关的内容，然后使用 iTextShape 将独取出来的数据写入到 PDF 文档中。

首先在项目中引入 COM 组件

如图所示：



引入 COM 组件之后，记住要进行引用（using)。然后就可以通过 COM 组件读取 Word 文档的内容

- 打开 Word 程序的对象(word 程序)

```
Microsoft.Office.Interop.Word.Application application = new
Microsoft.Office.Interop.Word.Application();
object nullobj = System.Reflection.Missing.Value;
object fileobj = filePath;
//打开的 word 文档
```

```
            Microsoft.Office.Interop.Word.Document document =
application.Documents.Open(
            ref fileobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj,
            ref nullobj);
            //取得 doc 中的文本
            string file_text = document.Content.Text;
```

```
            //替换一下敏感的字符
            file_text=file_text.Replace('\a',' ');
            file_text = file_text.Replace('\r', '\n');

            //关闭文档
            document.Close(ref nullobj, ref nullobj, ref nullobj);
            //关闭 COM 组件
            application.Quit(ref nullobj, ref nullobj, ref nullobj);

            GC.Collect();
            GC.WaitForPendingFinalizers();
            //将内容写到 Pdf 中
            WriteToPdf(file_text, filePath);
```

- 写入到 PDF 文档代码如如下：

```
            //将内容写入到 Pdf 文档中
        private void WriteToPdf(string file_text, string file_path)
        {
            /*将图片写入到 Pdf 文件中*/
            iTextSharp.text.Document pdfdocument = new
iTextSharp.text.Document(PageSize.A4, 9, 18, 36, 36);   //创建一个文档变量
            string fileName = file_path.Split('\\').Last<string>().Split('.').First<string>();
```

```
        PdfWriter write = PdfWriter.GetInstance(pdfdocument, new
FileStream(@"D:\pdfTest\" + fileName + ".pdf", FileMode.App          end));//创建文档

        pdfdocument.Open();


        //使用 Windows 自带的字体库(确保系统盘在 C 盘)

        BaseFont baseFT = BaseFont.CreateFont("C:\\Windows\\Fonts\\Simsun.TTC,1",
BaseFont.IDENTITY_H, BaseFont.EMBEDDED);


        iTextSharp.text.Font font = new iTextSharp.text.Font(baseFT);

        //写入一个段落, Paragraph

        pdfdocument.Add(new iTextSharp.text.Paragraph(file_text, font));

        //关闭 document

        pdfdocument.Close();

    }
```

- 使用 Aspose.Words 第三方控件来进行转化，Aspose.Words 最新版在我的资料可以进行下载，此版本支持中文。

首先引入 Aspose.Words 控件，添加引用即可。转化的代码及其简单，如下：

```
        string fileName = filePath.Split('\\').Last<string>().Split('.').First<string>();  //获取
文件名

        Aspose.Words.Document document = new Aspose.Words.Document(filePath);

        document.Save(@"D:\pdfTest\" + fileName + ".pdf", SaveFormat.Pdf);
```

- 接下来是 Excle 文档的转换。Excle 文档具有伸缩的特点，所以需要读取每个单元格的信息然后在 PDF 文件中写入表格中，最后将表格插入到 PDF 文档中。

```
/*
        * 将 Xls 文件转化成 Pdf 文件
        * 如果 Xls 文档单元格过多,分页显示
        * */
       private void XlsTransformPdf(string filePath)
       {
           Microsoft.Office.Interop.Excel.Application application = new
Microsoft.Office.Interop.Excel.Application();
           object nullobj = System.Reflection.Missing.Value;
           object fileobj = filePath;

           Microsoft.Office.Interop.Excel.WorkbookClass excleWorkClass= null;
           //打开 Excle 的文档
           excleWorkClass =
(Microsoft.Office.Interop.Excel.WorkbookClass)application.Workbooks.Open(filePath,
nullobj, false, nullobj, nullobj, nullobj, true, nullobj, nullobj, true, nullobj, nullobj,
nullobj, nullobj, nullobj);
           //获取所有的工作表
           Microsoft.Office.Interop.Excel.Sheets sheets = excleWorkClass.Worksheets;
           //循环所有的工作表
           foreach (Microsoft.Office.Interop.Excel.Worksheet sheetItem in sheets)
```

```
        {
            //只转化存在数据的工作表单
            if (sheetItem.UsedRange.Rows.Count > 1)
            {
                //获取 Excle 已经使用的单元格的索引,避免获取大量的空单元格信息
                int columnCount = sheetItem.UsedRange.Columns.Count;
                int rowCount = sheetItem.UsedRange.Rows.Count;
                char endPosition = (char)('A' + columnCount - 1);
                //设置截取的区域的末尾
                string endString = new string(endPosition, 1) + rowCount.ToString();
                //判断工作表的区域,最多支持 36 列
                if (columnCount >= 27)
                {
                    string[] columnArray = { "AA", "AB", "AC", "AD", "AE", "AF", "AG",
"AH", "AI", "AJ" };
                    endString = columnArray[columnCount - 27] + rowCount.ToString();
                }
                Microsoft.Office.Interop.Excel.Range range = sheetItem.get_Range("A1",
endString);
                //将单元格信息保存在数组中
```

```
System.Array array = (System.Array)range.Cells.Value2;

//将数据填充到 PDF 中,使用 iTextShape 将表格插入到 PDF 文件中

iTextSharp.text.pdf.PdfPTable table = new
iTextSharp.text.pdf.PdfPTable(columnCount);  //创建指定列数的表格

//设置中文字体

BaseFont baseFT =
BaseFont.CreateFont("C:\\Windows\\Fonts\\Simsun.TTC,1", BaseFont.IDENTITY_H,
BaseFont.EMBEDDED);

iTextSharp.text.Font font = new iTextSharp.text.Font(baseFT);

//循环读取数组，填充到 PDF 表格中

for (int row = 1; row <= rowCount; row++)

{

   for (byte column = 1; column <= columnCount; column++)

   {

      iTextSharp.text.pdf.PdfPCell cell = new PdfPCell(); //添加单元格



      object text = array.GetValue(row, column);



      Phrase phrase = null;

      //向表格中添加数据

      if (text == null)
```

```
                    phrase = new Phrase();
                else
                    phrase = new Phrase(text.ToString(), font);
                //设置边框
                cell.BorderWidth = 1;
                cell.Padding = 1;
                //向表格中添加数据
                cell.AddElement(phrase);
                table.AddCell(cell);
            }
        }

        //新建 PDF 文档,将表格放入文档中
        iTextSharp.text.Document pdfdocument = new
iTextSharp.text.Document(PageSize.A4, 9, 18, 36, 36);  //创建一个文档变量
        string fileName = filePath.Split('\\').Last<string>().Split('.').First<string>();
        PdfWriter write = PdfWriter.GetInstance(pdfdocument, new
FileStream(@"D:\pdfTest\" + fileName + ".pdf", FileMode.Append));//创建文档
        pdfdocument.Open();
        //写入一个段落, Paragraph
```

```
                pdfdocument.Add(table);

                //关闭 document

                pdfdocument.Close();

                 //*

                // * */

            }

        }

    }
```

- 审计图表（见图 3）
  - 数据模型:

    XYSeries: 最常用的数据结构，主要包括一系列的 double 型（x,y）点对及一个名称（title）。

    XYValueSeries:包括一系列的(x,y,value)点对及一个名称（title）。

    XYMultipleSeriesDataset:包含一系列 XYSeries，是最终的数据结构

    TimeSeries:与 XYSeries 类似，x 变为 Date 型,可以转化为 XYSeries。

    CategorySeries:与 XYSeries 类似，x 变为 string 型，可以转化为 XYSeries。

    MultipleCategorySeries:一系列 CategorySeries。

    不同的数据模型用于不同的图表显示。XYSeries 可以用于折线图、直方图。CategorySeries 可以用于圆饼图。每种数据模型都提供了丰富的操作接口，用于插入删除数据等多种操作。我们可以根据需要将数据存为合适的数据模型。

    绘制器声明了绘制图表的类型。主要包括以下几种。

SimpleSeriesRenderer、 XYSeriesRenderer 用于设置每张图中每一个序列的绘制方法。而 DefaultRenderer 和 XYMultipleSeriesRenderer 用于设置整张图的绘制格式。它们之间的关系类似于 XYSeries 与 XYMultipleSeriesDataset 之间的关系。

要 画一张图，我们需要设置好需要绘制的数据及绘制的方法。确定好 data model 及 renderer 后，再调用合适的绘制方法绘制图形。库提供了两种绘制图形的方法，一种是 get***Intent( ) (如 getLineChartIntent()) 直接新建一个 Activity 全屏显示生成的图表，另一种是 get***View( ) (如 getLineChartView()) 生成一个 View，用户可以自行设置它的显示。

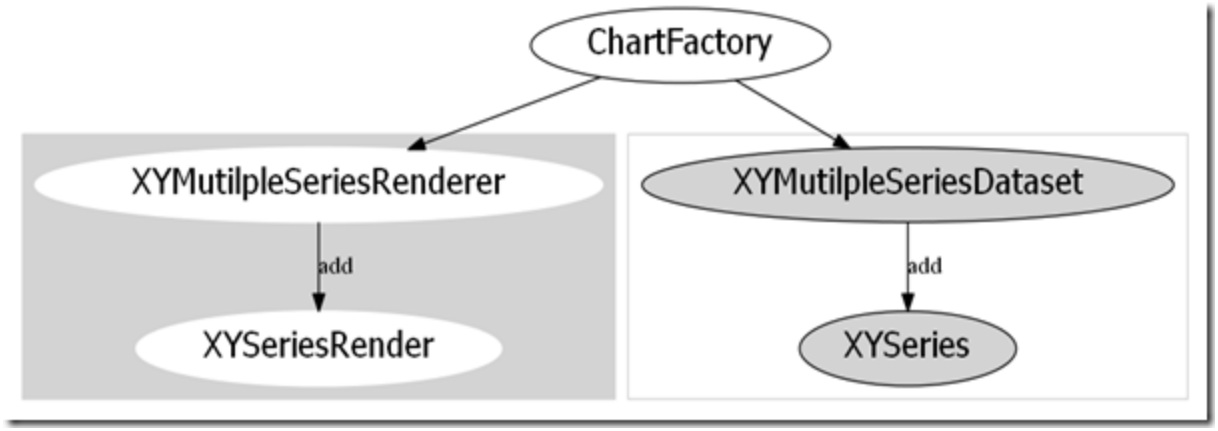
- 主要通过设置几个对象

1、XYSeries 对象：用于存储一条线的数据信息；
2、XYMultipleSeriesDataset 对象：即数据集，可添加多个 XYSeries 对象，因为一个折线图中可能有多条线。
3、XYSeriesRenderer 对象：主要是用来设置一条线条的风格，颜色啊，粗细之类的。
4、XYMultipleSeriesRenderer 对象：主要用来定义一个图的整体风格，设置 xTitle,yTitle,chartName 等等整体性的风格，
可添加多个 XYSeriesRenderer 对象，因为一个图中可以有多条折线。
设置完那些对象之后，可通过 org.achartengine.ChartFactory 调用数据集 XYMultipleSeriesDataset 对象
与 XYMultipleSeriesRenderer 对象来画图并将图加载到 GraphicalView 中，
ChartFactory 有多种 api，通过这些 api 调用来决定是画折线图还是柱状图。

- 绘制的基础

很多,使用这个引擎的同学,大多,会被它提供的例子的给吓到,因为,太多参数了!到最后,都不知道看到那里去了,其实,仔细研究,这个引擎的使用是非常简单...()

首先,我们整理一下思路,一般绘制一个图表需要:如下图所示



从图中,我们可以看出,绘制一个图表我们其实,我们只需要理解三个概念

1,ChartFactory ,传入 XYMutilpleSeriesRenderer,XYMutilpleSeriesDataset,然后,我们只需用 getXXXChartIntent(Context context,XYMutilpleSeriesRenderer,XYMutilpleSeriesDataset,)方法就可以进行图表的显示

2,XYMutilpleSeriesRenderer 用于进行绘制的设置,添加的 XYSeriesRender 对象,用于定义绘制的点集合设置,注意数量要和 XYMutilpleSeriesDataset,添加的 XYseries 一致!!!

3,XYMutilpleSeriesDataset 用于数据的存放,添加的 XYseries 对象,用于提供绘制的点集合的数据

- 数据表格（见图 4）
  - 前台代码：

```
<form id="form1" runat="server">
  <div>
    <telerik:RadScriptManager Runat="server">
    </telerik:RadScriptManager>
    <telerik:RadInputManager ID="RadInputManager1" runat="server">
```

```
        </telerik:RadInputManager>

        <telerik:RadFormDecorator ID="RadFormDecorator1" runat="server" />

        <telerik:RadGrid ID="RadGrid1" runat="server" AllowAutomaticInserts="true"

           AllowAutomaticUpdates="true" onneeddatasource="RadGrid1_NeedDataSource"

          GridLines="None" AllowPaging="true" onitemupdated="RadGrid1_ItemUpdated"  >

        <MasterTableView AutoGenerateColumns="False" DataKeyNames="Id"
CommandItemDisplay="Top" >

        <CommandItemSettings AddNewRecordText="添加" RefreshText="刷新"
></CommandItemSettings>

        <Columns>

        <telerik:GridEditCommandColumn ButtonType="ImageButton" />

        <telerik:GridBoundColumn DataField="Id" HeaderText="Id" />

        <telerik:GridBoundColumn DataField="Price" HeaderText="Value"
DataFormatString="{0:C}" />

        <telerik:GridBoundColumn DataField="Text" HeaderText="Text" />

        </Columns>

        </MasterTableView>

        </telerik:RadGrid>

    </div>

    <div>

    <asp:Label runat="server" ID="output" />
```

```
    </div>

    </form>
```

- 在上述代码中，为 RadGrid 控件的 NeedDataSource 事件指定了一个事件处理程序。顾名思义，当 RadGrid 需要数据源时（如翻页、绑定）就会触发此事件。在此事件中通过编写代码绑定数据源，如下代码所示。

```
protected void RadGrid1_NeedDataSource(object source,
Telerik.Web.UI.GridNeedDataSourceEventArgs e)

{

    buildTestData();

    RadGrid1.DataSource = data;

}

//Grid 所使用的数据源

List<TestClass> data = new List<TestClass>();

/// <summary>

/// 随机生成数据源

/// </summary>

private void buildTestData()

{

    if (data.Count > 0) return;     //如果已经生成过则不再生成

    string alphbet = "abcdefghijklmnopqrstuvwxyz0123456789";

    Random r = new Random();
```

```
    for (int i = 0; i < 80; i++)
    {
        TestClass t = new TestClass();
        t.Id = i;
        t.Price = r.NextDouble() * 100000000;
        StringBuilder sb = new StringBuilder();
        int len = r.Next(15);
        for (int j = 0; j < len; j++)
        {
            sb.Append(alphbet[r.Next(alphbet.Length)]);
        }
        t.Text = sb.ToString();
        data.Add(t);
    }
}
/// <summary>
/// 测试用的数据类
/// </summary>
private class TestClass
```

```
{
    public int Id { get; set; }
    public double Price { get; set; }
    public string Text { get; set; }
}
```

- 鼠标拖放（见图 5）

  ■ 声明性拖放

为了把拖放行为添加到一个 div 标签，第一项任务是使用 Atlas 标记。通过拖放，我仅想实现能够拖放一个对象并且让它位于你想把它放置的地方。当把一个对象放置到一个指定的点时，在实际开发中所表现出的动作将在后面讨论。为了配置你的网页以便使用 Atlas，你需要从微软站点把 Microsoft.Web.Atlas.dll 文件下载到你的 bin 文件夹下并且使用下列入口配置你的 web.config 文件：

```
<system.web>
<pages>
<controls>
<add namespace="Microsoft.Web.UI"
assembly="Microsoft.Web.Atlas" tagPrefix="atlas"/>
<add namespace="Microsoft.Web.UI.Controls"
assembly="Microsoft.Web.Atlas" tagPrefix="atlas"/>
</controls>
</pages>
</system.web>
```

接下来，你需要把一个 Atlas 脚本管理器控件添加到你的.aspx 页面并且使用 AtlasUIDragDrop 库来配置：

```
<atlas:ScriptManager ID="ScriptManager1" runat="server">
<Scripts>
<atlas:ScriptReference ScriptName="AtlasUIDragDrop" />
</Scripts>
</atlas:ScriptManager>
```

然后，添加你想使之可拖放的 div 对象，并且确保它有一个拖放句柄：

```
< div style="background-color:Red;height:800px;width:600px;" >
< div id="draggableDiv"
style="height:100px;width:100px;background-color:Blue;" >
< div id="handleBar"
style="height:20px;width:auto;background-color:Green;" >
< /div >
< /div >
< /div >
```

最后，添加能够使你的 div 成为可拖放的标记脚本：

```
< script. type="text/xml-script" >
< page xmlns:script="http://schemas.microsoft.com/xml-script/2005" >
< components >
< control id="draggableDiv" >
< behaviors >
< floatingBehavior. handle="handleBar"/ >
< /behaviors >
< /control >
< /components >
< /page >
< /script >
```

至此，你应该有了一个可拖放的 div 标签。该示例展示了结合 Atlas 使用声明性方式的简单性和容易性。在 Atlas 所引入的术语中，你仅使用了声明性标记来把漂浮行为添加到一个 HTML 元素。

■ 强制性拖放

为了使用编程方式来实现相同的功能，我们需要进行一些编程，但是不需要较多的编码。你必须明白，当你把一个 Atlas 脚本管理器组件添加到你的页面上时，你实际上是在下命令把 Atlas JavaScript 库加载到你的页面。这个 Atlas 库提供了扩展 DOM 的客户端类，并且提供允许你在一个浏览器中进行编码的工具（尽管现在在 Safari 兼容性方面还存在一些问题）。这些客户端类还允许你把你的 HTML 元素添加到行为。

为了切换到一个强制性模型，你需要用两个 JavaScript 函数来代替 XML 标记。第一个函数是一个普通脚本用于把漂浮行为添加到一个 HTML 元素上。它利用了 Atlas 客户端类来完成此功能：

```
< script. type="text/javascript" >
function addFloatingBehavior(ctrl， ctrlHandle){
//创建新的漂浮行为对象
var floatingBehavior. = new Sys.UI.FloatingBehavior();
```

```
//漂浮行为类具有一个 Handle 属性
floatingBehavior.set_handle(ctrlHandle);
//把对象参考值的为 Atlas 客户端控件
var dragItem = new Sys.UI.Control(ctrl);
//从 Atlas 控件中取得行为集合
//添加我们自己的漂浮行为
dragItem.get_behaviors().add(floatingBehavior);
//运行该漂浮行为的内部javascript
floatingBehavior.initialize();
}
</script>
```

这个函数使用两个参数值：你想要拖放的 HTML 元素和实现该拖放行为的拖放句柄 HTML 元素。然后，你实例化一个新的 Atlas 客户端行为对象。该漂浮行为具有一个 handle 属性-你把 HTML 元素的句柄传递给它。然后，你需要基于你想使之成为可拖放的控件以创建一个新的客户端控件对象。把你的 div 标签转换成一个 Atlas 客户端控件能够使你把 Atlas 行为添加到它上面。你可以使用 get_behaviors()方法来返回一个行为集合，并且使用 add 方法来把一个新行为添加到你的 HTML 对象。最后，你调用行为对象的 initialize()方法以允许在内部配置行为自身。我们将在本文剩下的部分中一直使用这个工具函数。

现在，当页面装载时，你需要调用 addFloatingBehavior 函数。说实话，这是编写这个示例中最有难度的编码部分。脚本管理器并不是简单地创建一个到 Atlas JavaScript 库的引用，我推想它实际把该库脚本装载到 DOM。在任何情况下，这意味着，只有页面中的其它一切都装载后该库才得到加载。这样以来，我们所面临的问题在于，装载该库后，没有标准的方法来使我们的添加漂浮行为的代码运行；并且如果我们在加载该库前运行它，那么我们可以简单地生成 JavaScript 错误-因为我们调用的所有的 Atlas 方法都不能被发现。

其实，存在好几种方法可以来解决这个问题，但是最容易的方法是使用一个定制的 Atlas 事件 pageLoad()-这个事件实际只在装载这些库后才调用它。为了把漂浮行为添加到你的 div 标签中，当第一次加载页面时（但是在库脚本装载后），你仅需要编写如下代码：

```
<script. type="text/javascript">
function pageLoad(){
addFloatingBehavior(document.getElementById('draggableDiv'),
document.getElementById('handleBar'));
}
</script>
```

这可以使用一种 Atlas 脚本速记方式来书写-用"$()"代替"document.getElementById()"：

```
<script. type="text/javascript" >
function pageLoad(){
addFloatingBehavior($('draggableDiv')，$('handleBar'));
}
</script>
```

在此，可以看到，你有一个可拖动的 div，其行为与你使用声明性模型编写的可拖动的 div 完全一致。

■ 动态拖放

既然声明性模型比强制性模型更为清晰，那么为什么你还要编写自己的 JavaScript 来处理 Atlas 行为呢？其实，这种声明性模型的一个限制是，你只能使用一开始就位于该页面上的对象。如果你开始动态地把对象添加到该页面，那么你无法使用声明性模型来把漂浮行为添加到其上。不过，借助于强制性模型，你能够实现。

基于前面的例子，你要用一个据要求创建漂浮 div 的函数来代替 pageLoad()函数。下列 JavaScript 函数会创建一个嵌有另一个 div 标签（用作一个 handlebar）的 div 标签，然后把该 div 标签插入到当前的页面，并且最后把漂浮行为添加到 div 标签：

```
function createDraggableDiv() {
var panel= document.createElement("div");
panel.style.height="100px";
panel.style.width="100px";
panel.style.backgroundColor="Blue";
var panelHandle = document.createElement("div");
panelHandle.style.height="20px";
panelHandle.style.width="auto";
panelHandle.style.backgroundColor="Green";
panel.appendChild(panelHandle);
var target = $('containerDiv').appendChild(panel);
addFloatingBehavior(panel， panelHandle);
}
```

然后，你只需要把一个按钮添加到该调用 createDraggableDiv()函数的页面。现在，新的 HTML 体看上去具有如下形式：

```
<input type="button" value="Add Floating Div" />
<div id="containerDiv" style="background-color:Purple;height:800px;width:600px;"/>
```

这将允许你把很多的可拖放元素添加到你的页面上，这说明了，一旦你理解了在以声明方式使用 Atlas 和以编程方式使用它之间的关系，那么 Atlas 将表现出强大的威力和灵活性。作为参考，下面是动态拖放示例的完整实现代码：

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Imperative Drag and Drop II</title>
<script. type="text/javascript">
function createDraggableDiv() {
var panel= document.createElement("div");
panel.style.height="100px";
panel.style.width="100px";
panel.style.backgroundColor="Blue";
var panelHandle = document.createElement("div");
panelHandle.style.height="20px";
panelHandle.style.width="auto";
panelHandle.style.backgroundColor="Green";
panel.appendChild(panelHandle);
var target = $('containerDiv').appendChild(panel);
addFloatingBehavior(panel， panelHandle);
}
function addFloatingBehavior(ctrl， ctrlHandle){
var floatingBehavior. = new Sys.UI.FloatingBehavior();
floatingBehavior.set_handle(ctrlHandle);
var dragItem = new Sys.UI.Control(ctrl);
dragItem.get_behaviors().add(floatingBehavior);
floatingBehavior.initialize();
}
</script>
</head>
<body>
<form. id="form1" runat="server">
```

```
< atlas:ScriptManager ID="ScriptManager1" runat="server" >
< Scripts >
< atlas:ScriptReference ScriptName="AtlasUIDragDrop" / >
< /Scripts >
< /atlas:ScriptManager >
< h2 > Imperative Drag and Drop Code with javascript.:
demonstrate dynamic loading of behaviors < /h2 >
< input type="button" value="Add Floating Div" / >
< div id="containerDiv" style="background-color:Purple;height:800px;width:600px;"/ >
< /form >
< /body >
< /html >
```

■ 声明性 Dropzone

我们可以在一个页面上拖动 HTML 元素，然后让它们位于其投放位置。然而，为了使该行为真正有用，当投放发生时，应该抛出一个事件。而且，所抛出的事件应该依赖于在何处发生投放。换句话说，需要把行为添加到一个给定 HTML 元素-由它来把这一行为转换成一个"dropzone"或"投放目标"，可以使用相同的方法把漂浮行为添加到一个 HTML div 标签以便把它转换成一个可拖放的元素。

在下列例子中，我将向你展示 Atlas 是如何支持 dropzone 概念的。在它的当前状态中，Atlas 并不支持与其支持漂浮元素一样的方式来提供一种现成的行为以支持创建 dropzone 元素。然而，它确实实现了一个 DragDropList 元素和一个 DraggableListItem 元素的行为（这两个元素联用，允许你创建能够通过拖放重新排序的列表）。如果你想进一步探讨这一功能，你可以在网上找到若干使用 DragDropList 行为的好例子，例如，《Introduction to Drag And Drop with Atlas》。

dragdropzone 行为的主要不利条件是，它仅与具有 DragDropList 行为的项一起工作。为了确定我上面描述的开放端点的 dropzone 功能的类型（它将与预定义的漂浮行为一同使用），你需要用 JavaScript 编写你自己的 dropzone 行为类。幸好，这并不困难。

Atlas 把若干 OOP 扩展添加到 JavaScript 中以使加强其扩展能力，例如命名空间，抽象类和接口。在编写你自己的 dropzone 行为时你还要利用这些工具。如果你分析一下 AtlasUIDragDrop.js 文件的源码（可以使用 Visual Studio 调试器），那么你会发现在那里定义了若干接口，这包括一个相应于 Sys.UI.DragSource 而另一个相应于 Sys.UI.DropTarget。事实上，FloatingBehavior 类和 DraggableListItem 类都实现了 Sys.UI.DragSource 接口，而 Sys.UI.DropTarget 被 DragDropList 类所实现。这两个接口的代码如下所示：

```
Sys.UI.IDragSource = function() {
```

```
this.get_dataType = Function.abstractMethod;
this.get_data = Function.abstractMethod;
this.get_dragMode = Function.abstractMethod;
this.onDragStart = Function.abstractMethod;
this.onDrag = Function.abstractMethod;
this.onDragEnd = Function.abstractMethod;
}
Sys.UI.IDragSource.registerInterface('Sys.UI.IDragSource');
Sys.UI.IDropTarget = function() {
this.get_dropTargetElement = Function.abstractMethod;
this.canDrop = Function.abstractMethod;
this.drop = Function.abstractMethod;
this.onDragEnterTarget = Function.abstractMethod;
this.onDragLeaveTarget = Function.abstractMethod;
this.onDragInTarget = Function.abstractMethod;
}
Sys.UI.IDropTarget.registerInterface('Sys.UI.IDropTarget');
```

为什么你需要实现这些接口而不是简单地编写一些新类来支持拖放和 dropzone 呢？秘密是，在后台，还有一个类 DragDropManager，负责实际协调可拖放元素与 dropzone 元素之间的交互，并且它仅仅知道如何与实现 IDragSource 或 IDropTarget 接口的类一起工作。这个 DragDropManager 类注册对于每一个可拖放的元素来说哪些 dropzone 是合法的目标，并处理 MouseOver 事件以决定何时一个 dropzone 上面具有一个可拖放的元素，以及其它你不需要自己做的大量事情。事实上，它处理得如此完美，以至后面你要编写的 dropzone 行为需要极少的代码。首先，创建一新的 JavaScript 文件 DropZoneBehavior.js。我把我的 JavaScript 文件放到了一个子目录 scriptLibrary 下，但是，这对于实现 dropzone 行为是不必要的。然后，把下列代码复制到你的文件中：

```
Type.registerNamespace('Custom.UI');
Custom.UI.DropZoneBehavior. = function() {
Custom.UI.DropZoneBehavior.initializeBase(this);
this.initialize = function() {
Custom.UI.DropZoneBehavior.callBaseMethod(this， 'initialize');
//把我们自己注册为一个拖放目标.
Sys.UI.DragDropManager.registerDropTarget(this);
}
```

```
this.dispose = function() {
Custom.UI.DropZoneBehavior.callBaseMethod(this， 'dispose');
}
this.getDescriptor = function() {
var td = Custom.UI.DropZoneBehavior.callBaseMethod(this， 'getDescriptor');
return td;
}
//IDropTarget 成员.
this.get_dropTargetElement = function() {
return this.control.element;
}
this.drop = function(dragMode， type， data) {
alert('dropped');
}
this.canDrop = function(dragMode， dataType) {
return true;
}
this.onDragEnterTarget = function(dragMode， type， data) {}
this.onDragLeaveTarget = function(dragMode， type， data) {}
this.onDragInTarget = function(dragMode， type， data) {}
}
Custom.UI.DropZoneBehavior.registerClass('Custom.UI.DropZoneBehavior'，
Sys.UI.Behavior， Sys.UI.IDragSource，
Sys.UI.IDropTarget， Sys.IDisposable);
Sys.TypeDescriptor.addType('script'， 'DropZoneBehavior'，
Custom.UI.DropZoneBehavior);
```

我需要解释一下这个类。首先要注意从第二（以"Custom.UI.DropZoneBehavior.registerClass"开始）到最后一行代码。这是上面定义的 dropZoneBehaviorClass 注册到 Atlas 的位置。registerClass 方法的第一个参数相应于类的名字，第二个参数则相应于基类的名字。第三个参数相应于实现新类的接口。接下来的一行代码使你的类可用于声明性标记脚本。现在，我们回到开始，"Type.registerNamespace"方法允许你注册你的定制命名空间。下一行使用一个匿名方法语法声明我们的新类。这里使用了 JavaScript 面向对象的设计思想，这对于设计 Atlas 行为来说是必要的。在该匿名方法中，类方法 Initialize，Dispose 和 getDescriptor 都是一些简单的标准方法，用于所有的行为类，而且在这个简单实现中，你仅需要

调用基方法（也就是，从这个例子的第二到最后一行代码中所指定的基类的方法）即可。你要做的唯一特别的一点是，使用在 Initialize 方法中的 Sys.UI.DragDropManager 来注册拖放目标。这里是大部分的拖放"魔术"所在。

然后，你实现 IDropTarget 方法。在这个例子中，你仅实现了两个方法：this.canDrop 与 this.drop。对于 canDrop，你只是简单地返回 true。其实，更有趣的逻辑可以放到其中，譬如实现有哪些 div 标签被实际拖放到一个给定的目标上，或者决定相应于不同类型的漂浮 div，在拖放它们时各自的不同行为；但是，在此情况下，你仅想简单地实现 IDropTarget-它允许任何漂浮 div 拖动到其上。你的"drop"方法的实现只是个框架而已。当一个漂浮元素被拖放到你的拖放目标之一时，将显示一条警告消息指示已经发生了一些事情。现在，你已经有了一个拖放行为，它能够与我们在上一个例子中所用的漂浮行为一同工作。

现在你应该创建一个页面来展示你的新定制的 dropzone 行为。为此，你可以在前面示例的基础上来实现。在 Atlas 脚本管理器中，除注册 AtlasUIDragDrop 脚本以外，你还要注册你的新的 DropZoneBehavior 脚本：

```
< atlas:ScriptManager ID="ScriptManager1" runat="server" >
< Scripts >
< atlas:ScriptReference ScriptName="AtlasUIDragDrop" / >
< atlas:ScriptReference Path="scriptLibrary/DropZoneBehavior.js" / >
< /Scripts >
< /atlas:ScriptManager >
```

然后，你要把一个新的 div 标签添加到 HTML 体，这可以被用作一个拖放的目标：

```
< div style="background-color:Red;height:200px;width:200px;" >
< div id="draggableDiv" style="height:100px;width:100px;background-color:Blue;" >
< div id="handleBar" style="height:20px;width:auto;background-color:Green;" >
< /div >
< /div >
< /div >
< div id="dropZone" style="background-
color:cornflowerblue;height:200px;width:200px;" >
Drop Zone
< /div >
```

最后，你需要添加一个声明性标记元素以添加你的定制 DropZone 行为到你计划用作一个 dropzone 元素的 div。该 XML 标记应该有如下所示形式：

```
< script. type="text/xml-script" >
< page xmlns:script="http://schemas.microsoft.com/xml-script/2005" >
```

```
< components >
< control id="dropZone" >
< behaviors >
< DropZoneBehavior/ >
< /behaviors >
< /control >
< control id="draggableDiv" >
< behaviors >
< floatingBehavior. handle="handleBar"/ >
< /behaviors >
< /control >
< /components >
< /page >
< /script >
```

刚才的代码把一个 dropzone 添加到最初声明的拖放示例中。当你在 dropzone 上拖动元素时，将出现一个警告消息框。你可以扩展这些代码以便使你的定制 dropzone 行为的 drop 方法实现一些更为有趣的事情，例如激活当前的页面中的其它 JavaScript 事件，甚至使用 Atlas 调用一个 web 服务-由它来为你处理代码。

■ 强制性 Dropzone

为了使用 JavaScript 代替声明性脚本创建 dropzone，仅需要使用定制的 dropzone 行为添加如下的 JavaScript 函数来初始化你的 dropzone 元素：

```
function addDropZoneBehavior(ctrl){
var dropZone = new Sys.UI.Control(ctrl);
var dropZoneBehavior. = new Custom.UI.DropZoneBehavior();
dropZone.get_behaviors().add(dropZoneBehavior);
dropZoneBehavior.initialize();
}
```

为了"钩住"一切，你可以调用这个来自 Atlas pageLoad()方法的 addDropZoneBehavior 函数（就象你在前面的示例中操作 addFloatingBehavior 函数一样）。这样可以把正确的行为依附到它们各自的 HTML 元素并且复制上面你使用声明性标记所创建的拖放和 dropzone 功能。如果你想使此能够动态工作，那么你只要添加你为上一个示例编写的 createDraggableDiv()函数即可。作为一种参考，下面是创建可编程dropzone 的完整代码：

```
< %@ Page Language="C#" % >
< !DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" >
< html xmlns="http://www.w3.org/1999/xhtml"  >
< head id="Head1" runat="server" >
< title > Imperative Drop Targets < /title >
< script. type="text/javascript" >
function addFloatingBehavior(ctrl ,  ctrlHandle){
var floatingBehavior. = new Sys.UI.FloatingBehavior();
floatingBehavior.set_handle(ctrlHandle);
var dragItem = new Sys.UI.Control(ctrl);
dragItem.get_behaviors().add(floatingBehavior);
floatingBehavior.initialize();
}
function addDropZoneBehavior(ctrl){
var dropZone = new Sys.UI.Control(ctrl);
var dropZoneBehavior. = new Custom.UI.DropZoneBehavior();
dropZone.get_behaviors().add(dropZoneBehavior);
dropZoneBehavior.initialize();
}
function pageLoad(){
addDropZoneBehavior($('dropZone'));
addFloatingBehavior($('draggableDiv') , $('handleBar'));
}
< /script >
< /head >
< body >
< form. id="form1" runat="server" >
< atlas:ScriptManager ID="ScriptManager1" runat="server" >
< Scripts >
< atlas:ScriptReference ScriptName="AtlasUIDragDrop" / >
< atlas:ScriptReference Path="scriptLibrary/DropZoneBehavior.js" / >
< /Scripts >
< /atlas:ScriptManager >
< h2 > Imperative Drop Targets with javacript < /h2 >
< div style="background-color:Red;height:200px;width:200px;" >
```

```
< div id="draggableDiv"
style="height:100px;width:100px;background-color:Blue;" >
< div id="handleBar"
style="height:20px;width:auto;background-color:Green;" >
< /div >
< /div >
< /div >
< div id="dropZone" style="background-color:cornflowerblue;
height:200px;width:200px;" > Drop Zone < /div >
< /form >
< /body >
< /html >
```