ArcGIS Server 开发系列

文章来源：ESRI 中国社区

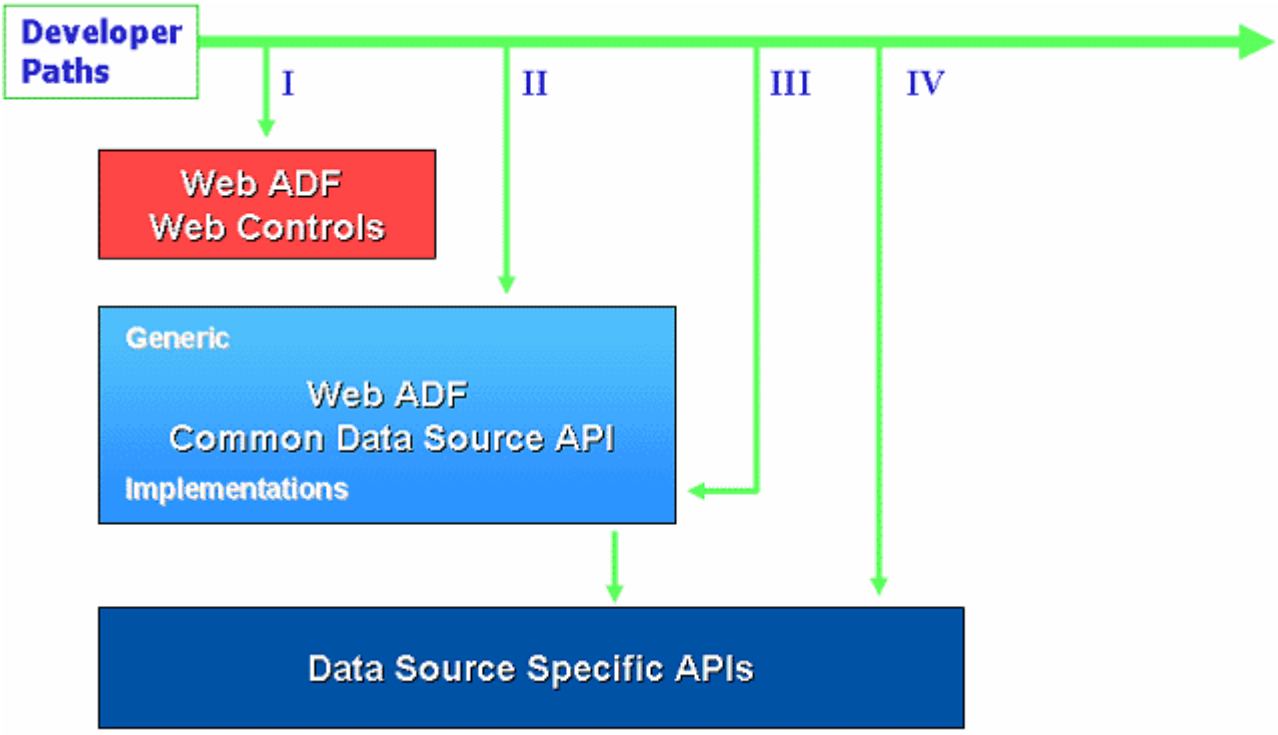
1.编程框架总览 2
2.Web ADF 编程 3
3.漫游 Graphics data sources 8
4.ArcGIS Server data sources 开发 13
5.自定义 Toolbar 工具 19
6.自定义 Tasks 28
ArcGIS 的缓存技术 32
深入浅出 ArcGIS 9.3 Javascript API 37
如何动态添加图层并进行查询 44
ArcGIS Server 常见问题 53
Server JAVA 开发 58
前言------为什么我们选择 JSF，而不是其它 Framework 58
JSF 基础 60
ADF 体系结构 第一部分 62
自定义 Command 68
自定义 Tools 开发 69
实现后台 JSF Bean 和前台 Javascript 的联动 73
AJAX 基础 74
ADF 中的 Ajax 函数 77
如何在用 JSF 在服务器端处理 Ajax 请求 79
用 Ajax 得到地图比例尺(客户端代码) 82
利用 Ajax 得到当前地图的比例尺（服务器端） 84
扩展 Tasks 框架(一) 86
Task 框架扩展之 Taskinfo 87
自定义 Functionality 92
用自定义 Functionality 实现用户权限控制 95
使用图片和 TrueType 字体进行标注 99
空间查询和高亮显示的实现 103
动态添加图层数据 105
Server Java 自定义开发－Network Analysis 107
ArcGIS Server 开发——控制图层是否可见 110
航线查询问题－Server Java 的实现方法 111
ArcGIS 航线查询完全例子 113
Born for SOA 系列 120
前言 120
Web Service 基础 120
ADF 中访问 Web Services 的类 130
ArcGIS Server 实现 SOA 132
Server 对于 SOAP 的支持 137

通过代理类访问 ArcGIS Server Web Service.....139
VS2008 下一个简单的 WebService 142
Flex 调用 Net WebService 149
WSDL 介绍 152
SOAP 介绍 161
SOAP /WSDL/Web Service 关系.....166

1.编程框架总览

ArcGIS Server 是一个发布企业级 GIS 应用程序的综合平台，如果想对 ArcGIS Server 基础有更多的了解，可以先看看《ArcGIS Server 体系结构》、ESRI(中国)和 ESRI 网站。这里及接下来的一系列文章将主要介绍 ArcGIS Server 开发相关的内容（如不加说明，后文中 Server 专指 ArcGIS Server），由浅入深，充分利用 Server 的资源来构建我们的企业级应用，同时也可以了解 Server 的优势及不足，在实际应用中扬长避短。

进行 Server 开发之前，我们首先需要了解 Server 开发的整体架构。下图（截取自官方文档）可以看到，Server 提供了一系列的应用开发途径，从 1 到 4 难度逐步增加，当然功能和可定制性也越来越强，我们可以充分利用 Server 提供的这些资源来构建我们的应用。



Web 应用开发

Server 为 Web 应用开发提供了一系列的开发方式，在 Visual Studio 2005 中可以直接使用 Web Mapping Application template 建立应用，它包含了 Web ADF 框架，是学习 Server 开发的入口。对照上面的示意图，可以这样来理解，如果想使用各种 Web 控件加上少量代码来构建 Web 应用，可以直接使用第 一种途径，这样的应用比较死板，让人感觉有点向搭积木，程序员可能没有任何成就感，好处是开发迅速，适合入门学习或做个简单的 Server 演示。如果想在 Common API 基础上，使用各种通用的 Functionality，通过

各种 Data Source 类型来定制 Web ADF，可以使用第二种开发途径。如果想使用 Common API，并且需要通过 Common API 调用 Specific API 来定制 Web ADF，可以通过第三种途径来开发。

Web Service 开发

Web Service 是什么不用再介绍，ArcGIS 提供了两种类型的 Web Service 创建方法：GIS Web Service 和 Application Web Service。

GIS Web Service 提供了一种将 ArcGIS Server Object（Local data source）发布为 ArcGIS Server Web Service（Internet data source）的 ESRI 标准，GIS Web Service 不用于开发，通常它们是用来发布信息和提供资源，ArcMap 就可以直接使用 GIS Web Service 的资源而不用进行任何开发，另外 Web ADF 控件和 Common API 也可以使用 GIS Web Service 资源。**因为 GIS Web Service 基于标准 web service，它可以作为传统 web service 来使用，ArcGIS Server 提供了 SOAP API 进行相关的开发，以后的学习中会使用到。**

Application Web Service 是基于标准 web service 建立的应用，使用一种 ESRI 的 data source 进行开发。因为 web service 没有用户接口，上图中途径一不适合这样的应用，途径四是最适合的，因为 web service 可充分利用 ESRI 提供的各种 Data Source Specific API 的所有 Functionality。

桌面客户端应用开发

ESRI 在桌面客户端应用中提供了两种实现 GIS 功能和服务的方法：ArcGIS Engine 和 Data Source Specific Developer API，上图中只有第四种途径可用于桌面客户端应用。

ArcEngine 不论是和本地 GIS 资源交互，还是使用远程数据资源，都拥有丰富的控件和众多 API，但它在使用和分发之前必须进行授权注册，而任何 data source specific developer APIs 都可以在桌面客户端应用中编程使用。

移动应用开发

移动应用和 Web 应用、web services 比较类似，但它们是为某一个特定的客户端环境而设计，如 PocketPC。Mobile ADF 是为移动应用定制的开发框架，通常使用途径四来进行开发，这样可以充分利用 ArcGIS Server Web services 和 SOAP API 提供的功能。

在以后的实际开发过程中，我们会逐步了解上述各种开发方式，并不断深入。

2.Web ADF 编程

Web ADF 全名是 Web Application Develope Framework，是 ArcGIS Server 专用的开发框架，9.2 版本的 ArcIMS 也开始提供 ADF 的封装。完全使用 Web 控件编程几乎不能解决我们实际项目中的问题，因此我们从 《ArcGIS Server 开发系列（一）--编程框架总览》中提到的途径二开始，即 Web Controls + Common Datasource API。

目标：

根据查询语句实现图层信息查询

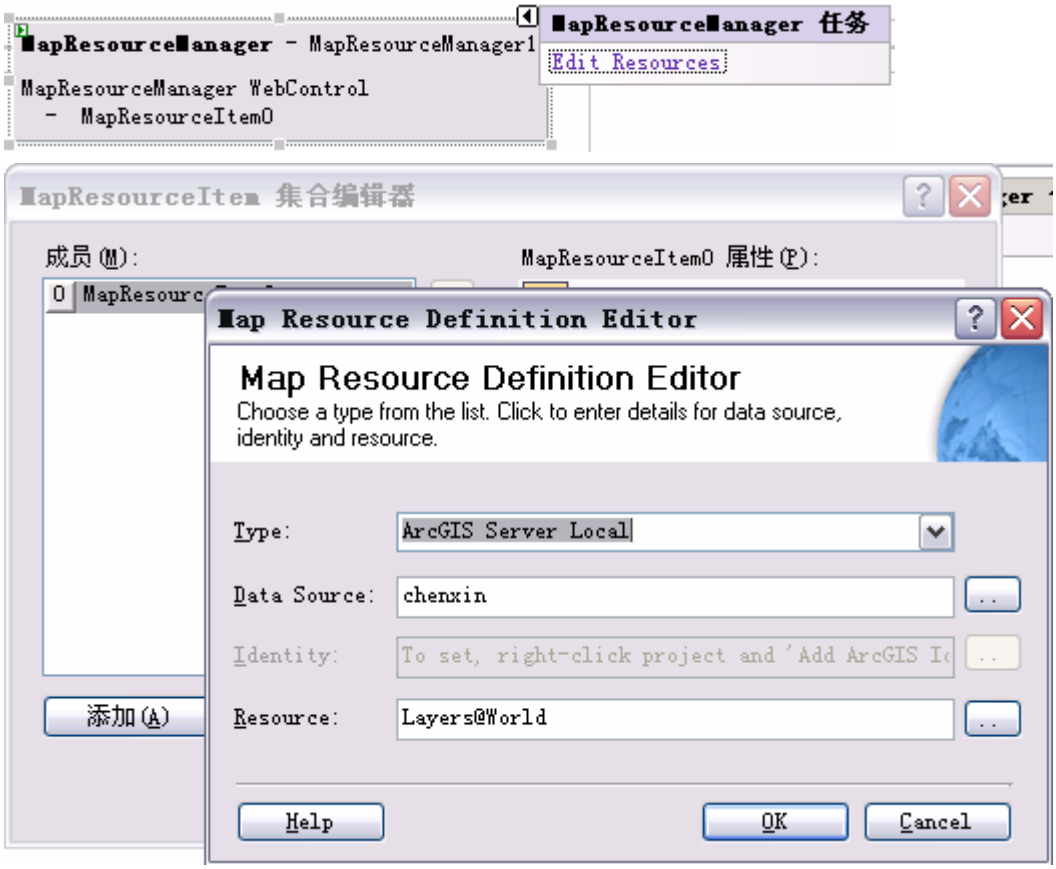
准备工作：

1.Windows XP sp2 中/英文版

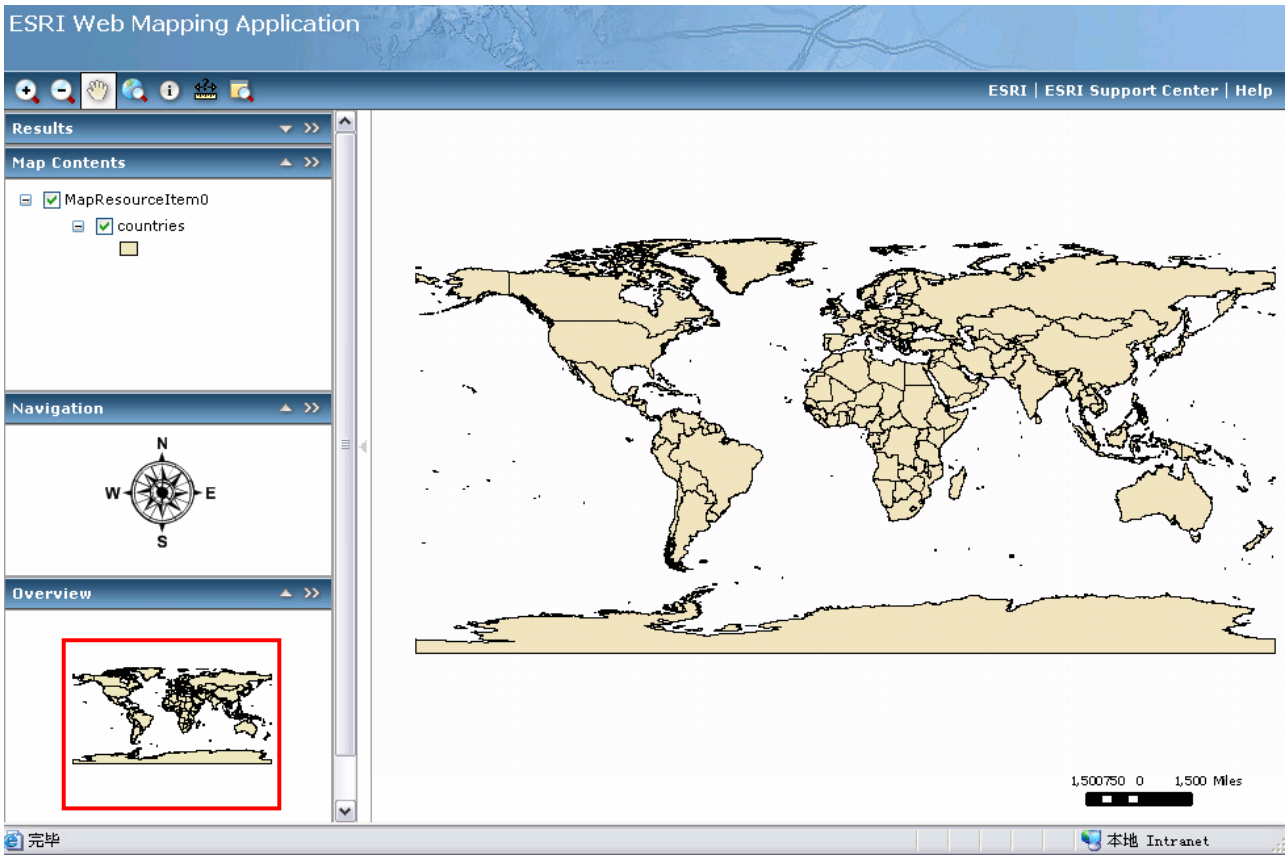
2.Visual Studio 2005 中/英文版

3.ArcGIS Server 9.2（我打上了 sp2 补丁），创建一个 ArcGIS Server 服务，Map Server 属性中更改 Pooling，使用池化连接方式，Progresses 选择“In a seperate process for each instance(high isolation)”，我的服务为“World”

4.利用 vs2005 创建一个模板 server 应用--Web Mapping Application，并更改 MapResourceManager 属性

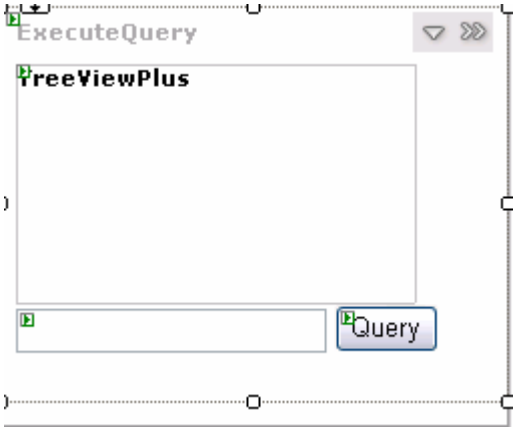


这样我们可以测试程序看是否能够正常运行，启动调试看到如下结果：



代码实现：

我们要完成的工作是对图层属性信息的查询，首先在页面左侧添加一个新的 panel，左侧的 Tasks、Results 等都是放置在 panel 之中，在这个 新的 panel 中分别添加 TreeViewPlus、TextBox、Button 三个控件，TextBox 输入查询条件，Button 确定查询， TreeViewPlus 显示结果，在设计试图中浏览效果如下：



双击“Query”Button，编写点击事件。

```
protected void cmdQuery_Click(object sender, EventArgs e)
   {
        Query(Map1.Extent);
```

```
    }
```

Query 方法实现了对图层属性信息的查询。

```
  protected void Query(ESRI.ArcGIS.ADF.Web.Geometry.Geometry geometry)
{

                IEnumerable func_enum=null;
            //获取当前 map1 控件中所有的 functionality
            func_enum= Map1.GetFunctionalities();

            System.Data.DataTable datatable;
            //对所有的 functionality 进行遍历

foreach (ESRI.ArcGIS.ADF.Web.DataSources.IGISFunctionality gisfunctionalityin func_enum)
        {
                ESRI.ArcGIS.ADF.Web.DataSources.IGISResource gisresource=null;
                //得到该 functionality 的 resource
                gisresource= gisfunctionality.Resource;

                //判断该 resource 是否支持 IQueryFunctionality

             bool supported=false;
                supported=
gisresource.SupportsFunctionality(typeof(ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality));

                if (supported)
        {
                    ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality qfunc;

                        qfunc=
(ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality)gisresource.CreateFunctionality(typeof(ESRI.ArcG
IS.ADF.Web.DataSources.IQueryFunctionality),null);

                  string[] lids;
                  string[] lnames;
                  //获得图层的 layerId 和 layerName，GetQueryableLayers 的重载方法可以指定图层类型
                    qfunc.GetQueryableLayers(null,out lids,out lnames);

                    ESRI.ArcGIS.ADF.Web.SpatialFilter spatialfilter=new
ESRI.ArcGIS.ADF.Web.SpatialFilter();
                  //设置过滤器的过滤条件，txtQuery 就是 panel 中 text box 的 ID
                    spatialfilter.ReturnADFGeometries=false;
                    spatialfilter.MaxRecords=1000;
                    spatialfilter.WhereClause= txtQuery.Text;
```

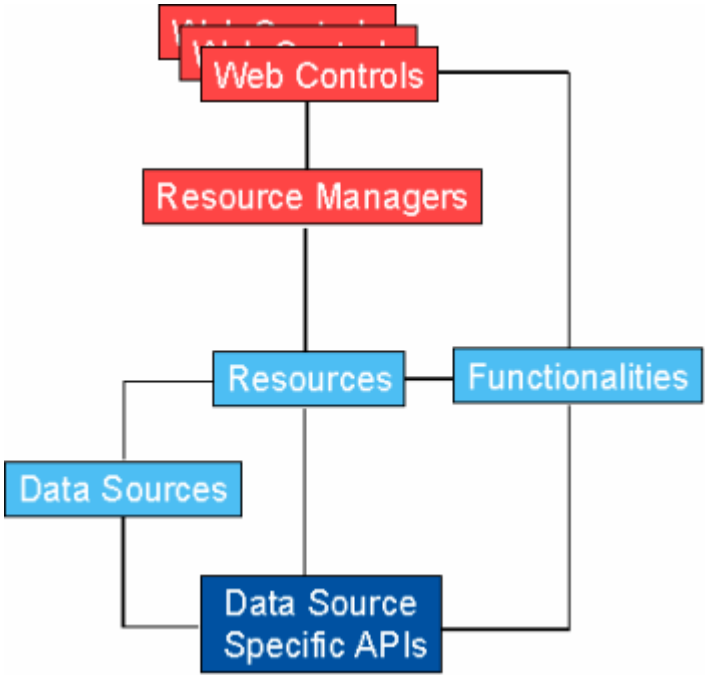
```
                    spatialfilter.Geometry= geometry;
                   //对指定的图层进行查询，查询的结果保存为 DataTable
                    datatable= qfunc.Query(null, lids[0], spatialfilter);

                    TreeViewPlus1.Nodes.Clear();
                   if (datatable!=null)
                    {
                        System.Data.DataSet ds=
new System.Data.DataSet();
                        ds.Tables.Add(datatable);
                        TreeViewPlus1.ShowClearAllButton=false;
                        //将结果绑定到 TreeViewPlus 控件上
                      TreeViewPlus1.BindToDataSet(ds);
                        TreeViewPlus1.Nodes[0].Expanded=true;
                    }
                }
            }
        }
```

运行程序，text box 中输入“CNTRY_NAME LIKE 'A%'”，查询结果：



这样这个小程序就完成了，在 server 中实现了对图层属性信息的查询，现在我们需要进行更多的考虑：

第一，考虑 web 控件、 functionality、datasource、resource、resourcemamager 之间的关系，代码中多次对它们的关系进行了描述，用文档中的一幅图可以概括整个框架，这篇文章对此进行了阐述--《ArcGIS Server .Net Web ADF 体系结构》。



第二，程序中使用的 mxd 仅仅包含一个 world 图层，因此对图层进行遍历的过程中只需获得第一图层的 ID，如果指定图层名，可以在 lnames 得到所需的图层进行查询。

第三，程序实现的查询方式相当简单，在 text box 中输入原始的 sql 来完成 where 语句，只仅仅是一个测试，一般我们会屏蔽掉 sql 相关的关键字，提供多个 text box 或 dropdownlist 供用户选择进行字段查询。

第四，如果查询信息量不大，返回结果数量不多，完全可以考虑 ajax 来完成页面请求，不论是速度还是用户体验都会得到提高。

第五，通过程序可以看到，查询结果保存在一个 datatable 中，这极大的方便了 server 查询和.net 的无缝连接，直接将结果和 TreeViewPlus 绑定，换成其他的 visual studio 控件也没问题。

只要整个框架思路清晰了，查询功能的实现其实很简单，这也是途径 2 开发方式的一个例子，可以看出 ArcGIS Server ADF 给开发提供了很多便利，和 ArcIMS 相比开发难度相当，多看看文档和 ADF OMD 图可以让我们对 ArcGIS Server ADF 开发有更多的了解，记住，这只是 server 开发的起步。

3.漫游 Graphics data sources

和 ArcGIS Server Local、ArcGIS Server Internet 一样，GraphicsLayer 是 ArcGIS Server MapResource 的一种，提供 functionality 给 web controls 使用。本文将在《ArcGIS Server 开发系列（二）--Web ADF 编程》示例基础上，增加查询结果高亮显示的功能，因为高亮显示的结果并不是图层本身所具备的，因此只需将高亮显示的图片存为 graphics 即可。

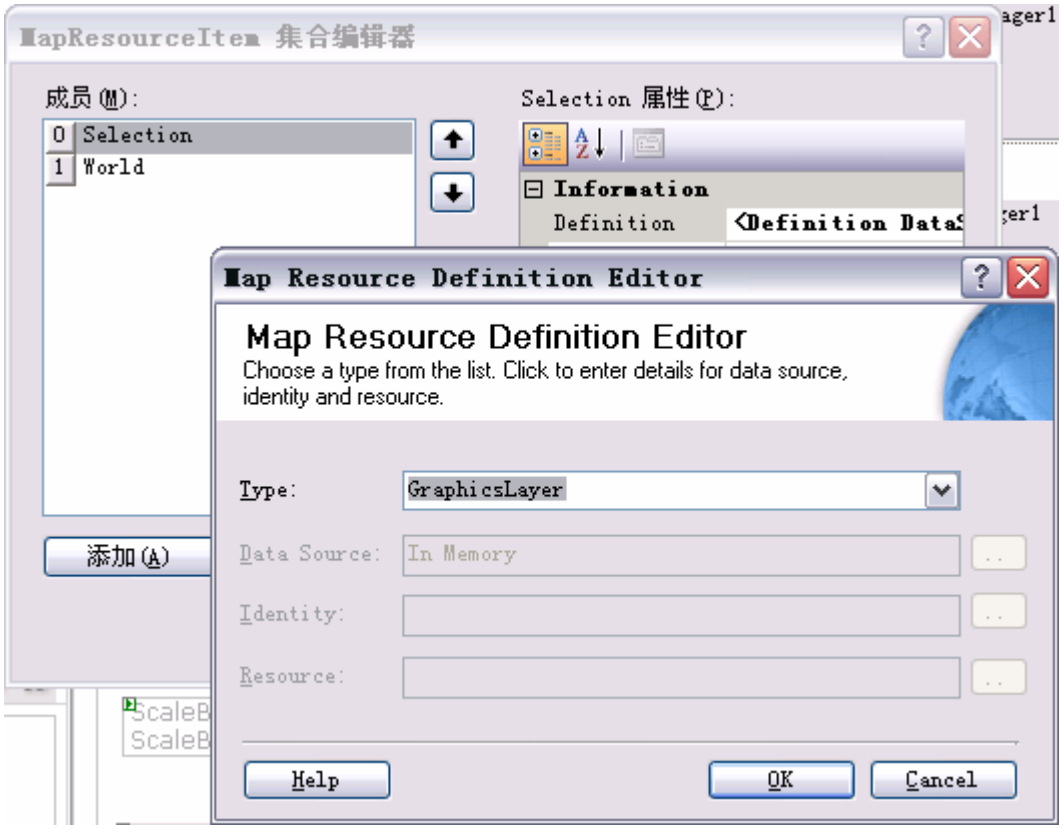
目标：

查询结果的高亮显示

准备工作：

1.以《ArcGIS Server 开发系列（二）--Web ADF 编程》示例配置和代码为基础。

2.MapResourceManager 属性中增加一个名为 Selection 的 MapResource，并将它移动到编号为 0 的位置，即显示在所有 MapResource 最上面。



可以看到 GraphicsLayer 的 datasource 是在内存中的，也就是说是为了临时显示或存储使用的，这样速度比较快。Selection 一定要放在 World 上面，否则就被 World 图层覆盖掉了。 代码实现：

在 UI 界面上，增加一个 command，用来清除 graphics。



双击“Select”生成事件响应方法：

```
protected void cmdSelect_Click(object sender, EventArgs e)
{
    SelectFeatures();
}
```

代码的核心就在 SelectFeature()里，它分为两个步骤，第一步对图层进行属性查询，第二步对查询结果进行高亮显示。首先是图层的属性查询：

```
int resource_index= 1;
```

```
    string targetlayername= "countries";
   System.Data.DataTable datatable=null;
  //直接获取 MapResourceName 为 world 的 MapFunctionality，它的编号为 1

       ESRI.ArcGIS.ADF.Web.DataSources.IMapFunctionality mf=
(ESRI.ArcGIS.ADF.Web.DataSources.IMapFunctionality)Map1.GetFunctionality(resource_index);
  //先得到 functionality，再获取 resource
  ESRI.ArcGIS.ADF.Web.DataSources.IGISResource gisresource= mf.Resource;
  bool supported=
gisresource.SupportsFunctionality(typeof(ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality));
  if (supported)
    {
      ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality qfunc;
      qfunc=
(ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality)gisresource.CreateFunctionality(typeof(ESRI.ArcG
IS.ADF.Web.DataSources.IQueryFunctionality),null);

     string[] lids;
     string[] lnames;
      qfunc.GetQueryableLayers(null,out lids,out lnames);

          ESRI.ArcGIS.ADF.Web.SpatialFilter spatialfilter=
new ESRI.ArcGIS.ADF.Web.SpatialFilter();
      spatialfilter.ReturnADFGeometries=false;
      spatialfilter.MaxRecords=1000;
      spatialfilter.WhereClause= txtQuery.Text;

      datatable= qfunc.Query(null, lids[0], spatialfilter);
}
```

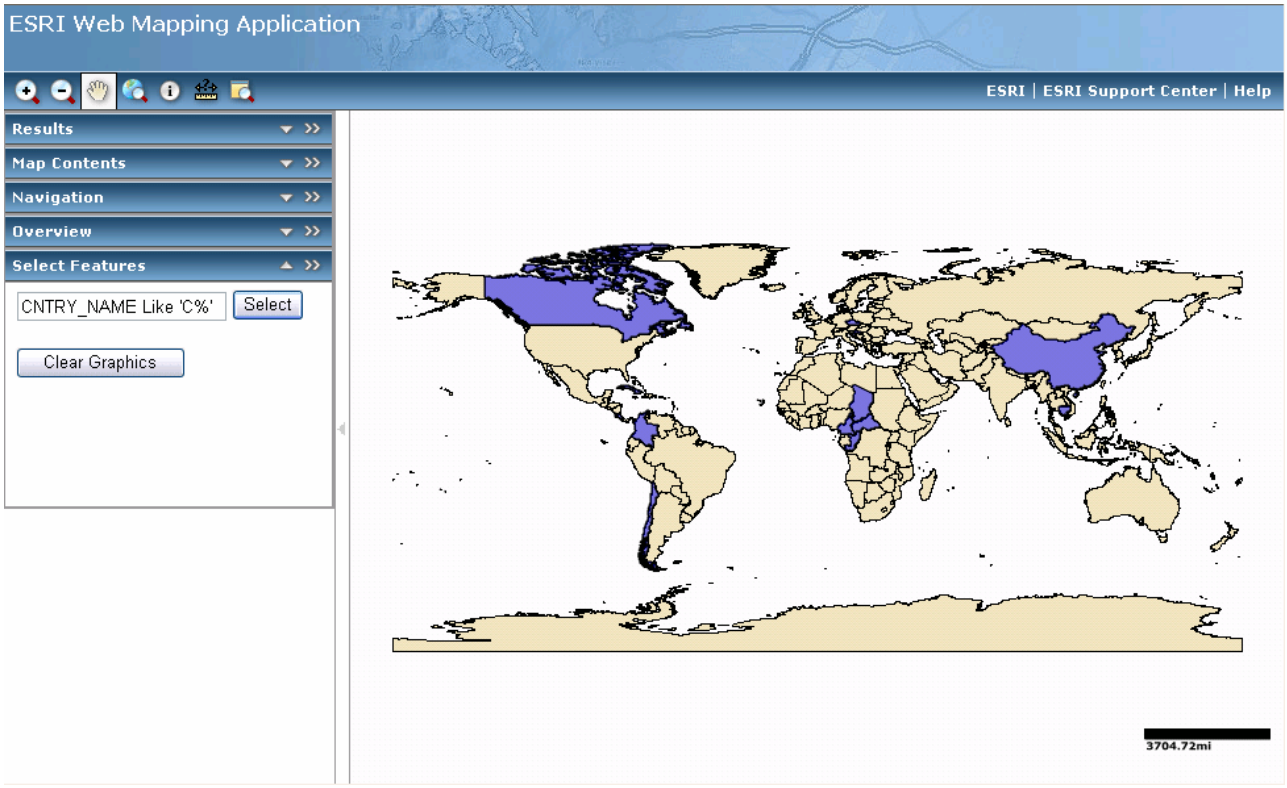
这段代码和《ArcGIS Server 开发系列（二）--Web ADF 编程》示例中的代码相比，没有太多改动的地方，用到了 ESRI.ArcGIS.ADF.Web.DataSources.IMapFunctionality，它继承于 ESRI.ArcGIS.ADF.Web.DataSources.IGISFunctionality 接口。每一个 web control 可能拥有多个 functionality，而 funtionality 是各种 resource 展现出来的，因此可以通过 web controls--funcionalities--resources 这条路线来获得当前的资源，那么如何让查询结果高亮显示呢？

```
  //重新获得 Map1 控件所有的 functionality
  IEnumerable gfc= Map1.GetFunctionalities();
  ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource gResource=null;
  foreach (IGISFunctionality gfuncin gfc)
 {
    //找到名为"Selection"的 MapResource

if (gfunc.Resource.Name=="Selection")    {
          //down cast 到 ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource
```

```
            gResource= (ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource)gfunc.Resource;
        }
    }

  if (gResource==null)
      return;

  ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer glayer=null;

  foreach (System.Data.DataTable dtin gResource.Graphics.Tables)
{
    if (dtis ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer)
   {
            glayer= (ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer)dt;
            break;
        }

      }

  if (glayer==null)
 {
      glayer=new ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer();

      gResource.Graphics.Tables.Add(glayer);
}

      //清除已有数据
  glayer.Clear();
  DataRowCollection drs= datatable.Rows;

  int shpind=-1;
  for (int i=0; i< datatable.Columns.Count; i++)
{
    if (datatable.Columns.DataType==typeof(ESRI.ArcGIS.ADF.Web.Geometry.Geometry))
   {
            //找到 Geometry 字段的序号
            shpind= i;
            break;
        }

      }

  try
{
```

```
    foreach (DataRow drin drs)
  {
            ESRI.ArcGIS.ADF.Web.Geometry.Geometry geom=
(ESRI.ArcGIS.ADF.Web.Geometry.Geometry)dr[shpind];

             //创建一个 GraphicElement
            ESRI.ArcGIS.ADF.Web.Display.Graphics.GraphicElement ge=null;
            ge=new ESRI.ArcGIS.ADF.Web.Display.Graphics.GraphicElement(geom,
System.Drawing.Color.Yellow);
            ge.Symbol.Transparency=50.0;

             //将 GraphicElement 添加到 ElementGraphicsLayer 中
            glayer.Add(ge);
    }
}
  catch (InvalidCastException ice)
{
    throw
new Exception("No geometry available in datatable");
}

  if (Map1.ImageBlendingMode== ImageBlendingMode.WebTier)
    { Map1.Refresh(); }
  else
if (Map1.ImageBlendingMode== ImageBlendingMode.Browser)
{
    //只刷新 Graphics Resource
      Map1.RefreshResource(gResource.Name);
}
```

这次我们没有将搜索到的结果绑定到控件上，只要得到高亮显示的结果，测试一下程序，看看能得到什么样的效果。



搜索出国家名称以"C"开头的国家，最典型的“中国”、“加拿大”已经找到了，这样我们就实现了高亮显示的功能。同样，我们进行开发后的小结，能想到些什么呢？还是按照 CH 风格来进行总结：

第一，GraphicsLayer 有两个子类，ElementGraphicLayer 和 FeatureGraphicLayer，因为程序中只需要暂时 显示查询的结果，因此将查询要素存为 ElementGraphicLayer 就可以，想想在什么情况下使用 FeatureGraphicLayer。

第二，ElementGraphicLayer 继承于 System.Data.DataTable，gResource.graphics 属于 System.Data.DataSet 类型，这样使得我们在开发过程中，可以将 GraphicElement 添加到 ElementGraphicLayer，然后将 ElementGraphicLayer 添加到 gResource.graphics，通过这种途径来向 GraphicsLayer 的 mapresource 中添加数据，这种机制方便了我们能够像操纵 datatable 和 dataset 一样来控制 mapresource 中的数据，既和.Net 无缝整合，也在一定程度上降低了 Server 开发难度，例如代码中 glayer.Clear()调用了 datatable 的 clear()方法，还有后面 GraphicElement 的创建。

第三，Map1.ImageBlendingMode 决定了地图的刷新是刷新整个页面，还是仅刷新当前 mapresource，这样的设计在 web 开发中尽可能的较少了网络数据传输量。

4.ArcGIS Server data sources 开发

ArcGIS Server 开发系列前几篇文章搭建了一个基本的 webgis 开发框架，包括模板应用程序搭建、属性查询、查询结果高亮显示，在 arcims 中，不论是使 用 9.2 之前的 javaconnector、.net_link、htmlviewer、

还是 9.2 里的 adf，这些功能都可以轻松实现，从软件成本上来 看，ArcGIS Server 企业版要比 ArcIMS

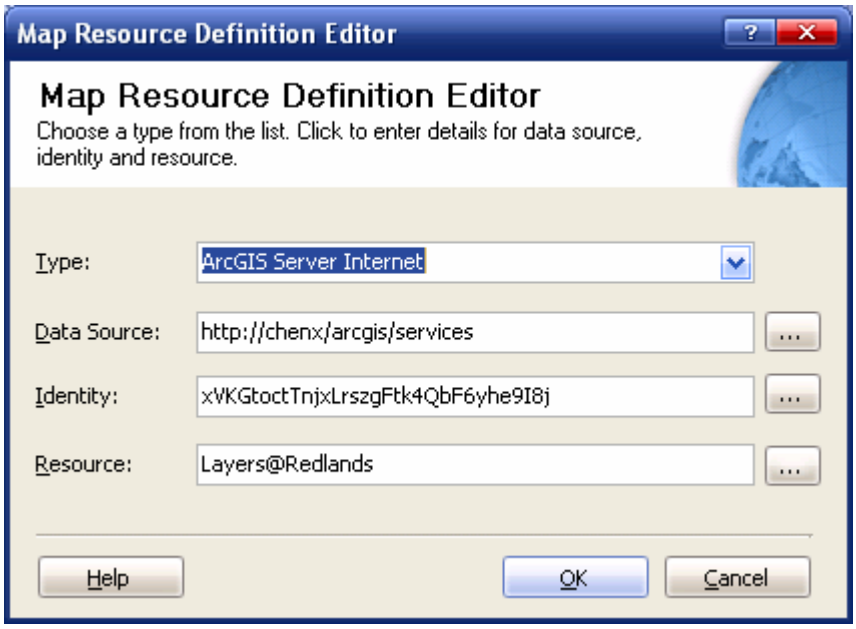
高出不少，如何体现 ArcGIS Server 价值呢？我们就从这一篇开始挖掘 ArcGIS Server 的价值，从 ArcGIS Server data sources 启航……

目标：

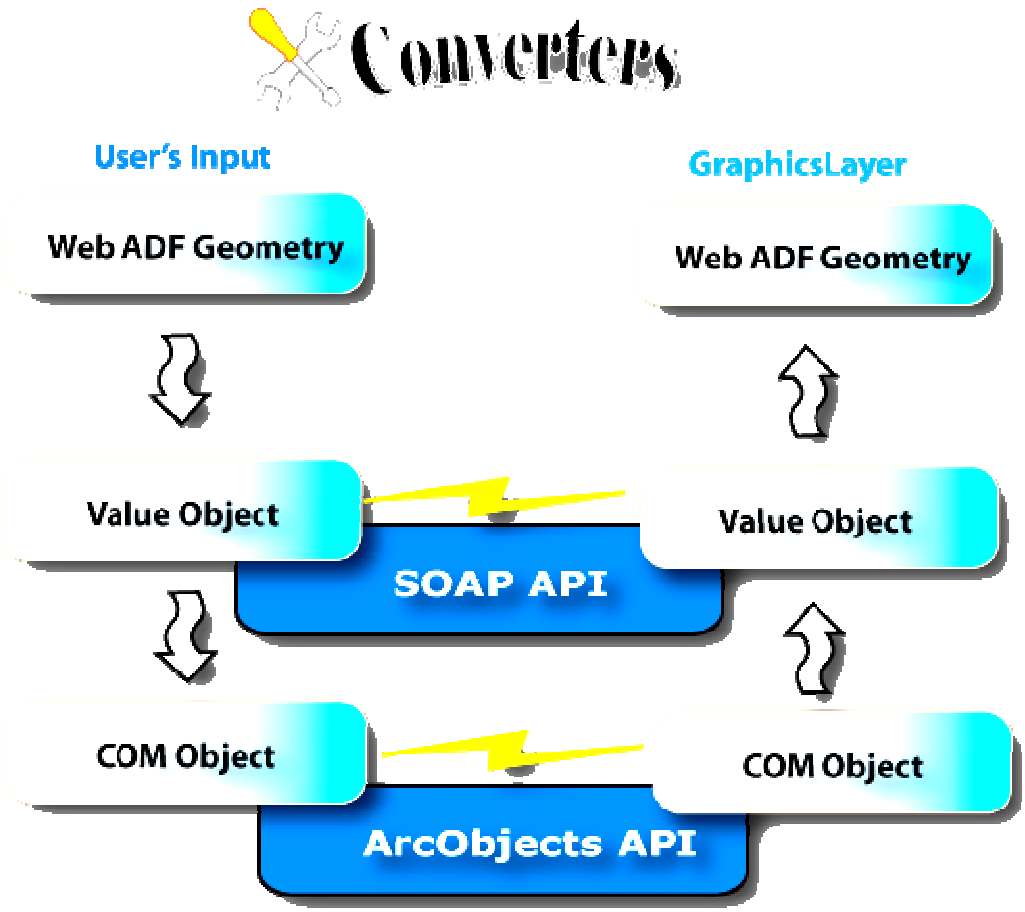
对点要素进行缓冲区分析

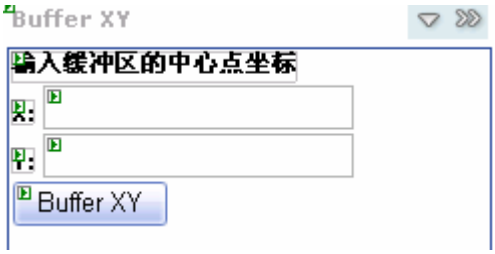
准备工作：

1.了解 ArcGIS Server 中的 ValueObject 和 ComOjbect。
2.AO 接口中缓冲区分析的编程方法。
3.重新回顾第一篇中 ArcGIS Server 开发的四种方法。
4.利用 vs2005 创建一个模板 server 应用--Web Mapping Application，并更改 MapResourceManager 属性。



注意这里使用的 map resource 类型是 ArcGIS Server Internet，在界面上增加一个新的 panel，里面包括两个 textbox 和一个 comand，textbox 对应的分别是缓冲区中心点的 x、y 坐 标，以该点为中心，一定半径做圆形缓冲区。





思路：

缓冲区分析需要在 AO 接口中实现，输入的点应该是一个 COM 对象，而在页面中输入 xy 坐标点是一个 SOAP API valueojbect，valueobject 可以用于 adf web controls，但不能用于 AO 接口，因此需要进行 valueobject 到 comojbect 的转换，在调用 ITopologicalOperator 接口的 Buffer 方法后会得到缓冲区分析的结果，即一个 polygon，同理，该 polygon 是一个 comobject，需要逆转换为 valueobject 才能显示到网页上。看下面的图，详细的说明了不同类型间的转换

代码实现：

新建一个类 XYBuffer，缓冲功能的在其 buffer 方法中实现。首先设置 textbox 中输入点的渲染方式，

该作为一个 ESRI.ArcGIS.ADF.ArcGISServer.PointN 对象，渲染方法如下：

```
  ESRI.ArcGIS.ADF.ArcGISServer.PointN pt=
new ESRI.ArcGIS.ADF.ArcGISServer.PointN();
 pt.X= x;
 pt.Y= y;

  // 设置点的颜色
  ESRI.ArcGIS.ADF.ArcGISServer.RgbColor rgb= new ESRI.ArcGIS.ADF.ArcGISServer.RgbColor();
  rgb.Red=0;
  rgb.Blue=0;
  rgb.Green=20;

  // 设置点的符号
  ESRI.ArcGIS.ADF.ArcGISServer.SimpleMarkerSymbol sms=
new ESRI.ArcGIS.ADF.ArcGISServer.SimpleMarkerSymbol();
  sms.Style= ESRI.ArcGIS.ADF.ArcGISServer.esriSimpleMarkerStyle.esriSMSCircle;
  sms.Color= rgb;
  sms.Size=
20;

  ESRI.ArcGIS.ADF.ArcGISServer.MarkerElement marker=
new ESRI.ArcGIS.ADF.ArcGISServer.MarkerElement();
  marker.Symbol= sms;
  marker.Point= pt;
```

然后用 arcgis server local 方式建立到 datasource 的连接，这点非常重要，主要是为了在这种连接状态下进行 valueobject 和 comobject 之间的转换。建立连接的用户应属于 ArcGIS Server 管理组。

```
  ESRI.ArcGIS.ADF.Identity identity=
new ESRI.ArcGIS.ADF.Identity("user","password","localhost");
  ESRI.ArcGIS.ADF.Connection.AGS.AGSServerConnection agsconnection;
  agsconnection=
new ESRI.ArcGIS.ADF.Connection.AGS.AGSServerConnection("localhost", identity);
  agsconnection.Connect();
  ESRI.ArcGIS.Server.IServerObjectManager som= agsconnection.ServerObjectManager;
  ESRI.ArcGIS.Server.IServerContext serverContext= som.CreateServerContext("Redlands","MapServer");
```

这样我们就可以在 arcgis server local 连接方式下进行对象转换：

```
  // 定义 COM 对象的点
  ESRI.ArcGIS.Geometry.IPoint ipnt;
  // 进行 valueobject 到 comobject 之间的转换
  ipnt=
(ESRI.ArcGIS.Geometry.IPoint)ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ValueObjectToComObject(pt, serverContext);
```

下面是 AO 中缓冲区分析的代码，熟悉 AO 编程的对下面代码应该很了解了：

```
ESRI.ArcGIS.Geometry.ITopologicalOperator topop= (ESRI.ArcGIS.Geometry.ITopologicalOperator)ipnt;
double bufferDistance= map.Extent.Width/
6;
ESRI.ArcGIS.Geometry.IPolygon bufferPolygon;
bufferPolygon= (ESRI.ArcGIS.Geometry.IPolygon)topop.Buffer(bufferDistance);
```

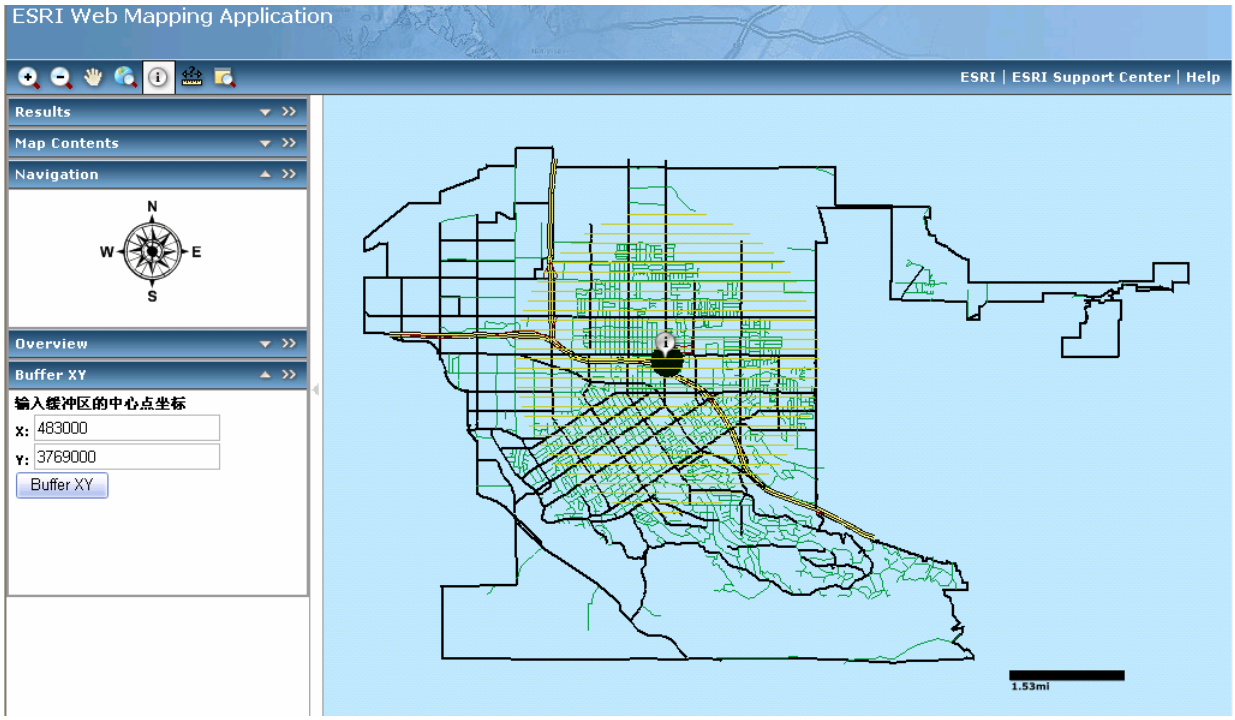
bufferPolygon 就是缓冲区分析的结果，但它还不是我们最后想要的，因为 ESRI.ArcGIS.Geometry.IPolygon 无法在 adf web control 中显示，还需要做一次转换：

```
// 定义 valueobject 的点
ESRI.ArcGIS.ADF.ArcGISServer.PolygonN buffer_polyn;
// 进行 comobject 到 valueobject 之间的转换

buffer_polyn=(ESRI.ArcGIS.ADF.ArcGISServer.PolygonN)ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServ
er.Converter.ComObjectToValueObject(bufferPolygon,
serverContext,typeof(ESRI.ArcGIS.ADF.ArcGISServer.PolygonN));
```

buffer_polyn 就是我们最后想要的结果，定义一种渲染方式：

```
ESRI.ArcGIS.ADF.ArcGISServer.RgbColor rgb1=
new ESRI.ArcGIS.ADF.ArcGISServer.RgbColor();
rgb1.Red=200;
rgb1.Green=200;
rgb1.Blue=20;
// 设置区的填充色
ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol sfs1=
new ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol();
sfs1.Style= ESRI.ArcGIS.ADF.ArcGISServer.esriSimpleFillStyle.esriSFSHorizontal;
sfs1.Color= rgb1;
ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement polyelement1=
new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
polyelement1.Symbol= sfs1;
polyelement1.Polygon= buffer_polyn;
```

最后将 marker 和 polyelement1 添加到 ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement 对象数组 中，传给当前 MapFunctionality 的 CustomGraphics 属性，刷新 map 控件，看看显示的效果图，缓冲区显示的样式为一系列水平的 平行线：



以上在 ArcGIS Server 中实现了缓冲区分析功能，虽然 arcims 也能实现（arcims 分析功能也仅限于此），但是两者的本质有天然的差别，前者是基于 AO，因此类推可以将 AO 中的分析功能全部引入 server 中，实现更多更复杂的分析功能。

最后，我们还有哪些需要考虑的呢？

1.ESRI.ArcGIS.ADF.ArcGISServer 命名空间中我们用到了 PointN 类，但同样可以找到 PointB 类，对于其他几何类型也是如此，如 PolylineN 和 PolylineB，它们之间有什么差别呢？

2.整个开发过程我们用到了 Data Source Specific API，重新回想一下第一篇中提到的途径三和途径四两种开发方式。

3. 例子中我们仅仅是将缓冲区显示出来，如果需要用缓冲区做进一步的分析，如一条街道向两侧拓展 3 米，有哪些房屋或建筑需要拆除或改建呢？这时需要做进一步的 相交分析，同样可以调用 AO 接口实现，最后将结果转换为 valueobject 显示出来，这样可以在 server 中实现决策分析的功能模块。当然我们可以通过这个例子做更多更深入的延伸。

4.如何将显示的结果清除掉？

5.自定义 Toolbar 工具

前面的开发系列均是使用 server 开发模板程序 Web Mapping Application，工具条上的基本工具是已经在模板中定制好的，在实际项目应用中，我们需要的工具远远不仅如此，如何在工具条中增加新的自定义

工具 是开发系列（五）所要描述的，其中使用 ASP.Net 2.0 Callback framework 进行异步刷新地图是重点。

目标：

自定义工具按钮进行矩选查询，高亮显示所选择的地图要素，页面下方的 Gridview 显示所选择要素的属性信息。

准备工作：

1.了解 ESRI.ArcGIS.Server.WebControls.IMapServerToolAction 接口

2.了解 ASP.Net 2.0 Callback framework

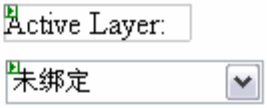
3.新建一个网站，在 ArcGIS Web Controls 控件中拖动如下控件：Toolbar、Map、Toc、MapResourceManager，以及常用控件 Label、DropDownList、Gridview。

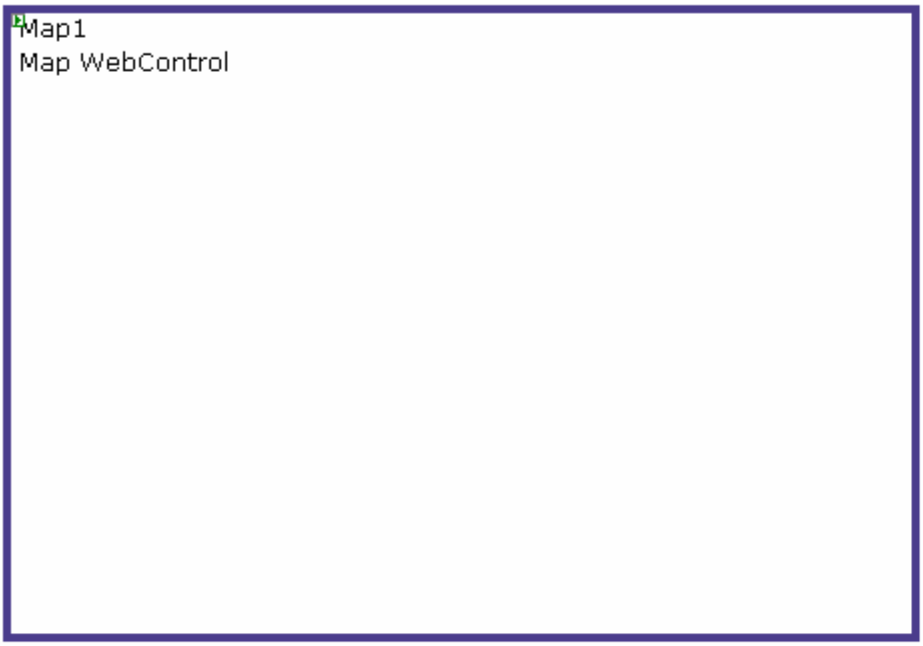
4.设置控件属性，Toolbar、Toc 的 BuddyControls 均为 Map1，Toolbar 的 BuddyControlType 为 Map，Map 控件的 MapResourceManager 为 MapResourceManager1。

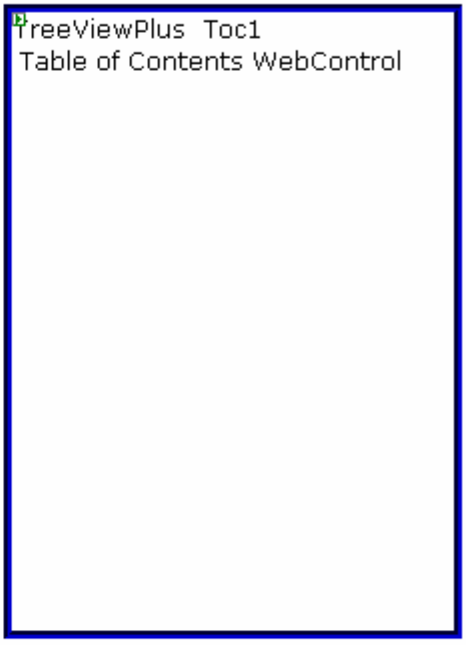
5.更改 MapResourceManager 属性，添加两个 Resource：Selection 和 NorthAmerica，类型分别是 Graphics Layer 和 ArcGIS Server Internet。

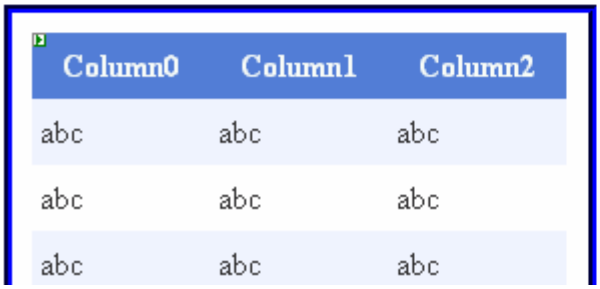
最后视图效果：













思路：

现在重新想想我们要做什么，首先要自定义一个工具按钮，使用该工具后在地图上进行矩形选择，对选择的要素高亮显示，同时 gridview 显示出这些要素的 属性信息。整个过程看似容易，实际上需要在客户端和服务器端之间来回切换，异步调用，这里用到了 ASP.Net Callback framework，其实 Server 中很多地图操作都基于 asp.net callback，或是实现了 ICallbackEventHandler 接口，理解了这一段程序开发有利于深入了解 Server 地图刷新、Task 等组 件的工作机制。

首先在 Toolbar 上新增一个按钮 Select Features，上图其实已经加入了，加入的方法是，选择 Toolbar 控件属性 ToobarItems，添加一个 Tool，设置以下值：

Text:Select Feature
CientAction ragRectangle
Name:SelectTool
ServerActionAssembly:App_Code
ServerActionClass:SelectFeatures

OK，搞定！

除了上述属性外，还可以设置该按钮各种状态下的图片显示、ToolTip 等等，这里就省了，纵观这些属性，可以看出既有 js 脚本的交互（已经封装了，通过 ToolEventArgs 传入），也有服务器端功能的实现，这时我们需新建一个类 SelectFeatures，并实现 IMapServerToolAction 接口，类中实现 IMapServerToolAction 的方法 ServerAction。

```
public class SelectFeatures : IMapServerToolAction{
    public
void ServerAction(ToolEventArgs args)
}
```

代码实现：

1.获取矩形框的屏幕坐标

要查询矩选的地图信息，首先应知道矩形的坐标，在服务器端如何获取呢？

```
Map mapctrl=null;
mapctrl= (Map)args.Control;

 // 获取下拉框中的数据，在后面实现
string targetlayername= (string)mapctrl.Page.Session["TargetLayer"];

RectangleEventArgs rectargs=null;
// 强制类型转换为 RectangleEventArgs
rectargs= (RectangleEventArgs)args;
```

```
// 获取矩形选择框的屏幕坐标
System.Drawing.Rectangle rect= rectargs.ScreenExtent;
ESRI.ArcGIS.ADF.Web.Geometry.Point minpnt=
ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint(rect.Left, rect.Bottom, mapctrl.Extent,
(int)mapctrl.Width.Value, (int)mapctrl.Height.Value);
ESRI.ArcGIS.ADF.Web.Geometry.Point maxpnt=
ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint(rect.Right, rect.Top, mapctrl.Extent,
(int)mapctrl.Width.Value, (int)mapctrl.Height.Value);

ESRI.ArcGIS.ADF.Web.Geometry.Envelope mappoly=null;
// minpnt、maxpnt 分别是左下、右上坐标点
mappoly=new ESRI.ArcGIS.ADF.Web.Geometry.Envelope(minpnt, maxpnt);
```

所有的信息都是通过 args 获取，它是一个 ESRI.ArcGIS.ADF.Web.UI.WebControls.ToolEventArgs 对象，包 含了客户端 Map 控件和当前客户端工具的信息，RectangleEventArgs 是它的子类，强制性转换后得到矩选的矩形坐标，最后得到一个 Envelope，用于 spatialfilter.Geometry 属性。

2.查询所选择的要素并对 Graphics Layer 进行渲染实现高亮

这部分内容完全可以参考《ArcGIS Server 开发系列（三）--漫游 Graphics data sources》，只需要注释掉 WhereClause 属性赋值，再增加一行代码：

```
ESRI.ArcGIS.ADF.Web.SpatialFilter spatialfilter=new ESRI.ArcGIS.ADF.Web.SpatialFilter();
spatialfilter.ReturnADFGeometries=false;
spatialfilter.MaxRecords=1000;
//spatialfilter.WhereClause = txtQuery.Text;
spatialfilter.Geometry= mappoly;
```

3.异步刷新 Gridview 显示地图要素的属性

```
GridView gdview= (GridView)mapctrl.Page.FindControl("GridView1");

object[] oa=new object[1];
string showtable="'visible'";

// datatable 为矩选时所选择的地图要素，绑定到 gridview
gdview.DataSource= datatable;
gdview.DataBind();

string returnstring=null;

using (System.IO.StringWriter sw=new System.IO.StringWriter())
{
    HtmlTextWriter htw=new HtmlTextWriter(sw);
    gdview.RenderControl(htw);
```

```
    htw.Flush();
    returnstring= sw.ToString();
  }

  // innercontent 相当于 innerhtml
  CallbackResult cr=new CallbackResult("div","griddiv","innercontent", returnstring);
  // 通过回调将信息从服务器端传输到客户端
  mapctrl.CallbackResults.Add(cr);

  if (datatable.Rows.Count>1)
{
    showtable="'visible'";
  }
else
{
    showtable="'hidden'";
  }

  string sa="var griddiv = document.getElementById('griddiv');";
  sa+="griddiv.style.visibility ="+ showtable+";";
  oa[0]= sa;

  CallbackResult cr1=new CallbackResult(null,null,"javascript", oa);
  mapctrl.CallbackResults.Add(cr1);
```

这段代码最关键的类是 CallbackResult，它简化了 web adf framework 中客户端回调的处理，不用再创建自己的客户端和服务器端逻辑，使用 CallbackResult 就可以将信息传回客户端，更新客户端页 面的内容、图片或执行 js 脚本。关于 CallbackResult 构造方法第三个参数，下面 js 代码写的很详细：

```
if (action=="content")
{
    o= document.getElementById(actions[1]);
    if (o!=null)
{
    o.outerHTML=actions[3];
  }
  }
else

if (action=="innercontent")…{


      o= document.getElementById(actions[1]);
    if (o!=null)
{
    o.innerHTML=actions[3];
```

```
}
}
  else
if (action=="image")
 {
  o= document.images[actions[1]];
 if (o!=null)
{
 o.src= actions[3];
}
 else alert (actions[1]+" was null");
}
else
if (action=="javascript")
{
 eval(actions[3]);
}
```

4.填充 DropDownList

DropDownList 显示的是 ArcGIS Server Internet 地图数据源所包含的图层名称，选择哪个图层，矩选时就对哪个图层进行查询，DropDownList 的填充在 Page_PreRender 过程中。

```
  if (!IsPostBack)
{
      ESRI.ArcGIS.ADF.Web.DataSources.IMapFunctionality mf=
(ESRI.ArcGIS.ADF.Web.DataSources.IMapFunctionality)Map1.GetFunctionality(1);


          ESRI.ArcGIS.ADF.Web.DataSources.IGISResource gisresource= mf.Resource;
    bool supported=
gisresource.SupportsFunctionality(typeof(ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality));

    if (supported)
 {
           ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality qfunc=
(ESRI.ArcGIS.ADF.Web.DataSources.IQueryFunctionality)gisresource.CreateFunctionality(typeof(ESRI.ArcG
IS.ADF.Web.DataSources.IQueryFunctionality),null);

           string[] lids;
           string[] lnames;
           qfunc.GetQueryableLayers(null,out lids,out lnames);
           for (int i=0; i< lnames.Length; i++)
{
```

```
                    LayerDropDownList1.Items.Add(lnames);
            }
            Session["TargetLayer"]= LayerDropDownList1.Items[0].Value;
    }
}
```

5.实现 ICallbackEventHandler 接口

Default.aspx.cs 的_Default 实现 ICallbackEventHandler 接口，在类中实现 RaiseCallbackEvent 和 GetCallbackResult 两个方法，做 ASP.Net 2.0 的对 ICallbackEventHandler 应该是再熟悉不过了，但下来框显示图层为什么要用到 callback？

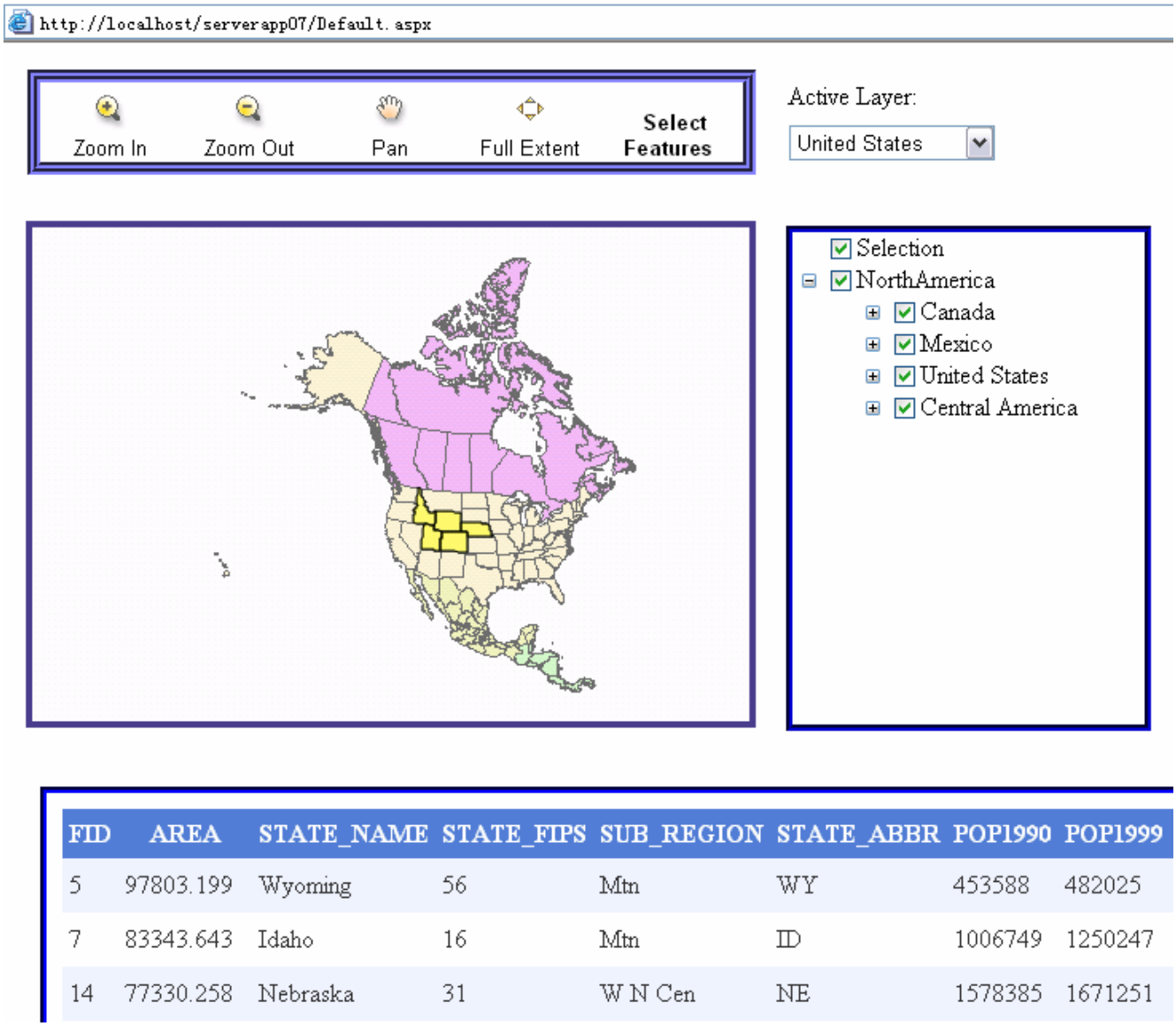
```
  protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
     {
        Session["TargetLayer"]="";
     }
     LayerDropDownList1.Attributes.Add("onchange","ChangeLayer()");
     sADFCallBackFunctionInvocation=
Page.ClientScript.GetCallbackEventReference(this,"message","processCallbackResult","context","postBackErr
or",true);
}

  publicvoid ChangeDropDownListServer(string ea)
{

     char[] parser_char=… {',' };
    string[] messages= ea.Split(parser_char);
    string dll1= messages[1];
     Session["TargetLayer"]= dll1;
}

ICallbackEventHandler 成员#region ICallbackEventHandler 成员
public void RaiseCallbackEvent(string eventArgument)
{
    if (eventArgument.Contains("ddl1"))
 {
     ChangeDropDownListServer(eventArgument);
     }
}
public string GetCallbackResult()
  {
    return returnstring;
```

```
}
```

原因就在这里，改变 Session ["TargetLayer"]的值，SelectFeatures 需要知道是对哪个图层进行查询的，从而对在那个图层选择要素进行高亮及属性显示，这里 callback 仅仅是在做了变量值的处理。最后在页面之间加入 js 脚本 ChangeLayer()。

```
<script type="text/javascript" language="javascript">
var context;
function ChangeLayer()
 {
    var message;
    var ddl1value= document.getElementById('LayerDropDownList1').value;
     message= 'ddl1';
     message+= ','+ ddl1value;
    <%=sADFCallBackFunctionInvocation%>
}
</script>
```

运行程序：



FID	AREA	STATE_NAME	STATE_FIPS	SUB_REGION	STATE_ABBR	POP1990	POP1999
5	97803.199	Wyoming	56	Mtn	WY	453588	482025
7	83343.643	Idaho	16	Mtn	ID	1006749	1250247
14	77330.258	Nebraska	31	W N Cen	NE	1578385	1671251

其中黄色区域就是 Select Features 按钮矩选的要素，下方 gridview 显示了查询到的属性结果。

程序中有两个地方用到了异步刷新，一个是 ASP.Net 2.0 原有接口 ICallbackEventHandler，另一个是 Web ADF framework 的 CallbackResult 类，最初认为简单异步刷新用自己写的 XMLHttpRequest 请求更为简单，如上例中对 session 存储值的改变，不用 ICallbackEventHandler，但是在 server 地图互操作的过程中， ICallbackEventHandler 给我们提供了更多的便利。

继续思考：

1.本例实现了根据地图查询属性，反过来根据属性查询几何图形怎么实现呢？其实前面《ArcGIS Server 开发系列（三）--漫游 Graphics data sources》已经讲到了，只不过需要将条件查询的信息，更改为在 gridview 或其他地方选择的属性信息，然后高亮显示相应的几何要素。

2.这种几何要素图形和属性信息的关联可以应用于各种不同的业务需求中，如图形和属性的同步删

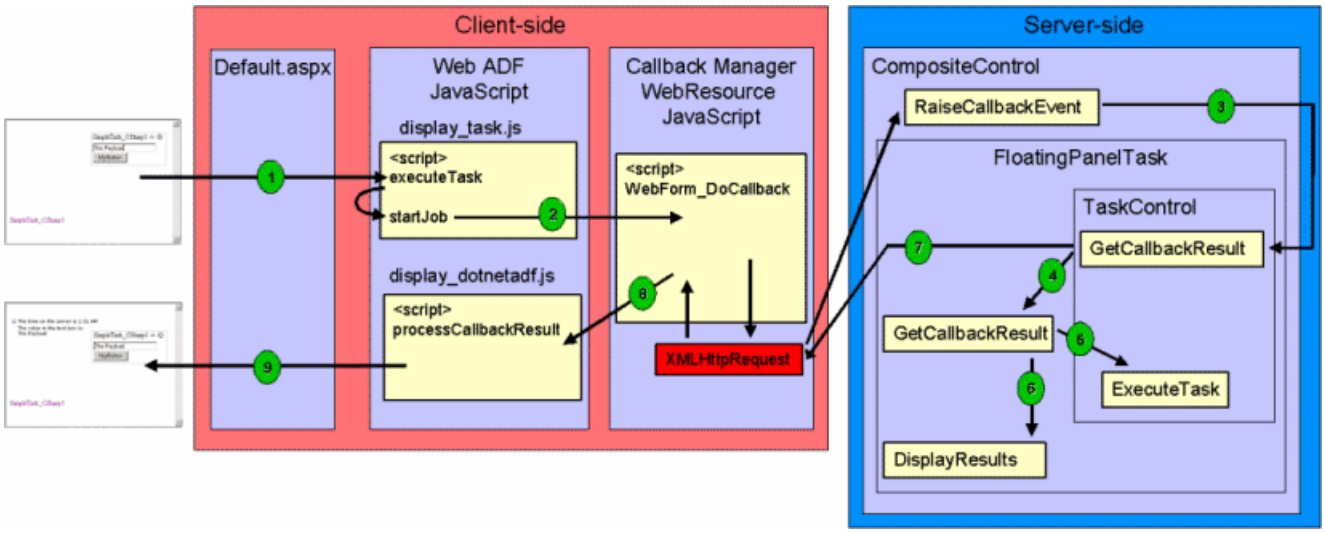
除、位置定位、类似结果查询等等。

3.如何改进或提升这种图形和属性的异步刷新带来的用户体验？

6.自定义 Tasks

Tasks 由一组相互关联的组件、动作组成，并可以展现最后的结果，如 QueryAttributeTask 执行空间数据属性查询，结果返回到 Results 控件中，开发中我们既可以使用.Net Web ADF 已有的 Tasks 控件，也可以自定义 Tasks，构建一些通用的 GIS 功能。使用已有的 Tasks 控件比较容易，所有的 Tasks 均有 TaskManager 管理，编辑 TaskManager 可以找到当前应用中所有的 Tasks，因此，一个基本页面中只需包含 TaskManager、 TaskResults、及一个或多个 Task 控件就可以实现 Task 功能。而实际开发中，我们更多需要的是自定义各种 Task，包括功能、外观，现在我 们就一起来设计一个 custom task。

动工之前，先来一图——Task Runtime Workflow，描述了整个 Task 的工作机制。



目标：

自定义一个功能，对指定的图层进行属性查询，返回查询的地图信息，将该功能封装成一个 Task，添加到 TaskManager 中。

准备工作：

1.熟悉 Web ADF Task 已有控件的使用

2.了解 Task Runtime Workflow

3.了解 ESRI.ArcGIS.ADF.Web.UI.WebControls.ITask 接口

4.了解 ASP.Net 2.0 Callback framework

5.更改 MapResourceManager 属性，添加一个 Resource：MapResourceItem0，类型是 ArcGIS Server Internet

开发分为两部分，一是功能实现，二是封装 Task，发布整个 Web 应用。

首先，创建一个 ClassLibrary 模板的工程 DefQuery，生成的 DefQuery 类实现抽象类 FloatingPanelTask，后者继承 了 FloatingPanel，实现了 ITask 接口，因此 DefQuery 必须实现 FloatingPanelTask 中的抽象方法。编码过程需要根据 Task 的运行流程进行。

1.事件触发 Task

事件触发有多种方式，例如点击 button、textbox 回车等等，事件触发后需要调用 display_task.js 里的 executeTask 函数，通常我们这样写：

```
executeTask('args='+ document.getElementById('TaskManager1_DefQuery1_sellayer').value+':'
+ document.getElementById('TaskManager1_DefQuery1_textBox').value,

"WebForm_DoCallback('TaskManager1$DefQuery1',argument,processCallbackResult,context,postBackError,true)");
```

里面包含两部分，一是传入的参数，一是 WebForm_DoCallback，后者可以去看 ASP.Net2.0 异步调用相关资料。 executeTask 函数在 display_task.js 中，它完成两件事情，分配一个 job id 来跟踪异步请求，和初始化一个回调并触发 TaskResults 控件中的指示器（指示 Task 程序正在运行中）。

2.调用 startJob 函数

startJob 提供了一系列的参数，通过键值对方式传送用户请求：

```
EventArg=startTaskActivityIndicator&taskJobID=1&args=0:TYPE='St'
```

然后通过参数_callbackArg 传入 RaiseCallbackEvent 方法。

3.服务器端处理

客户端的请求传入服务器端在 GetCallbackResult 方法中进行处理，这里分为两个步骤，一是 startTaskActivityIndicator，它会在 TaskResults 控件中指示 Task 任务正在执行，二是 executeTask，即 处理真正的核心业务：

```
public override string GetCallbackResult()
  {
    NameValueCollection keyValColl= CallbackUtility.ParseStringIntoNameValueCollection(_callbackArg);
   if (keyValColl["EventArg"]=="executeTask")
   {
        string sInArgs= keyValColl["args"];

            string delim=":";
        char[] delchar= delim.ToCharArray();
        string[] args= sInArgs.Split(delchar);
        object[] inputs=newobject[2];
        inputs[0]= args[0];
        inputs[1]= args[1];
        Input= inputs;
```

```
    }
    else
if (keyValColl["EventArg"]=="startTaskActivityIndicator")
{
 }
    else
if (keyValColl["EventArg"]=="hidden")
{
}
    return
base.GetCallbackResult();
  }
```

参数为"executeTask"时，会执行 ExecuteTask 重写方法，里面可以处理业务逻辑，将结果保存为一种 Results，可以是 SimpleTaskResult、DataSet 或 TaskResultNode。

4.DisplayResults 显示结果

Task 处理完之后，最终会交给 DisplayResults 将结果显示出来，看看 DisplayResults 代码：

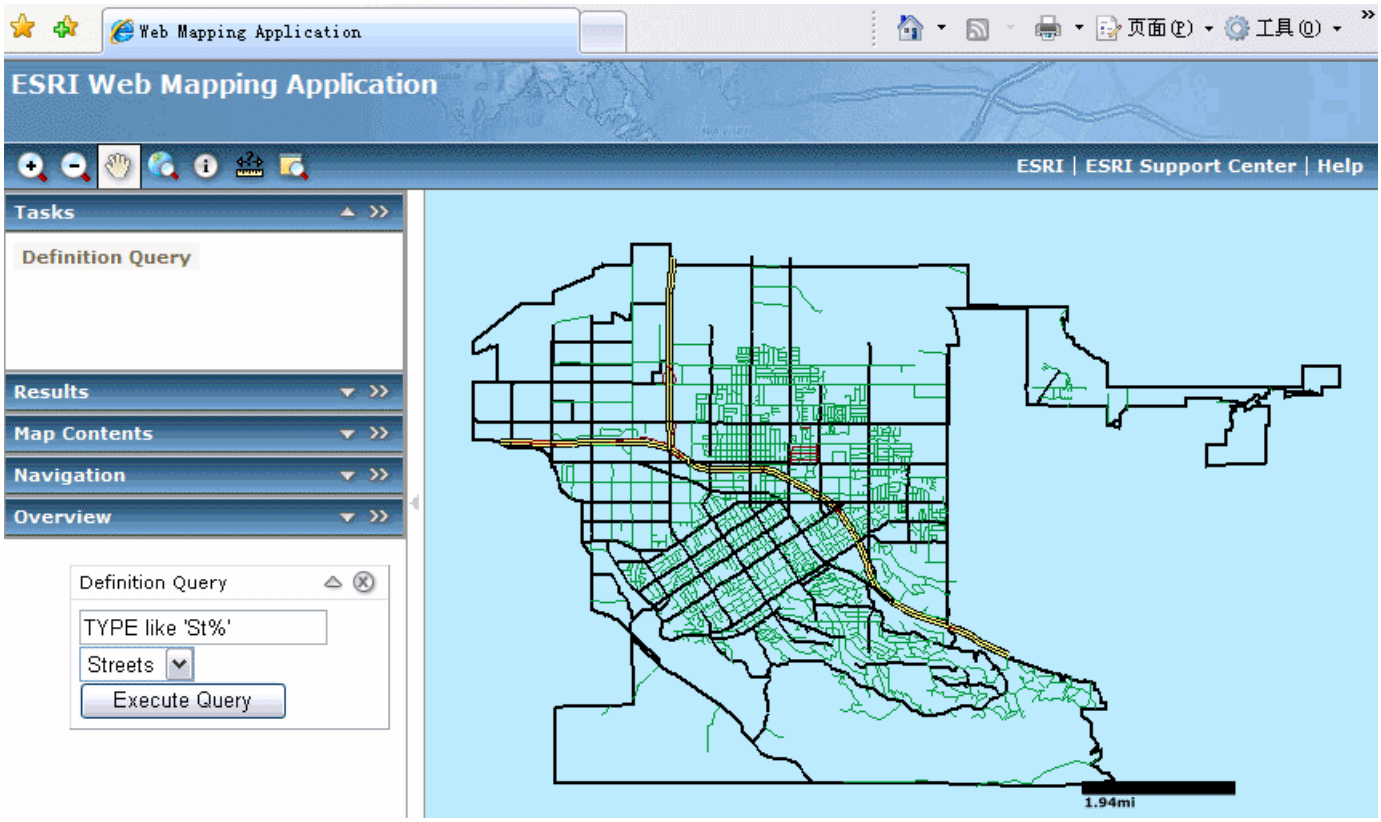
```
  protected virtual void DisplayResults(string jobID,object taskInput,object taskResults)
{
      ITaskResultsContainer resultsContainer=null;
     for (int i=0; i< TaskResultsContainers.Count; i++)
   {
            resultsContainer= Utility.FindControl(TaskResultsContainers.Name, Page)as ITaskResultsContainer;
            if (resultsContainer!=null)
     {
                  resultsContainer.DisplayResults(this, jobID, taskInput, taskResults);
                  CallbackResults.CopyFrom(resultsContainer.CallbackResults);
            }
     }
}
```

5.processCallbackResult

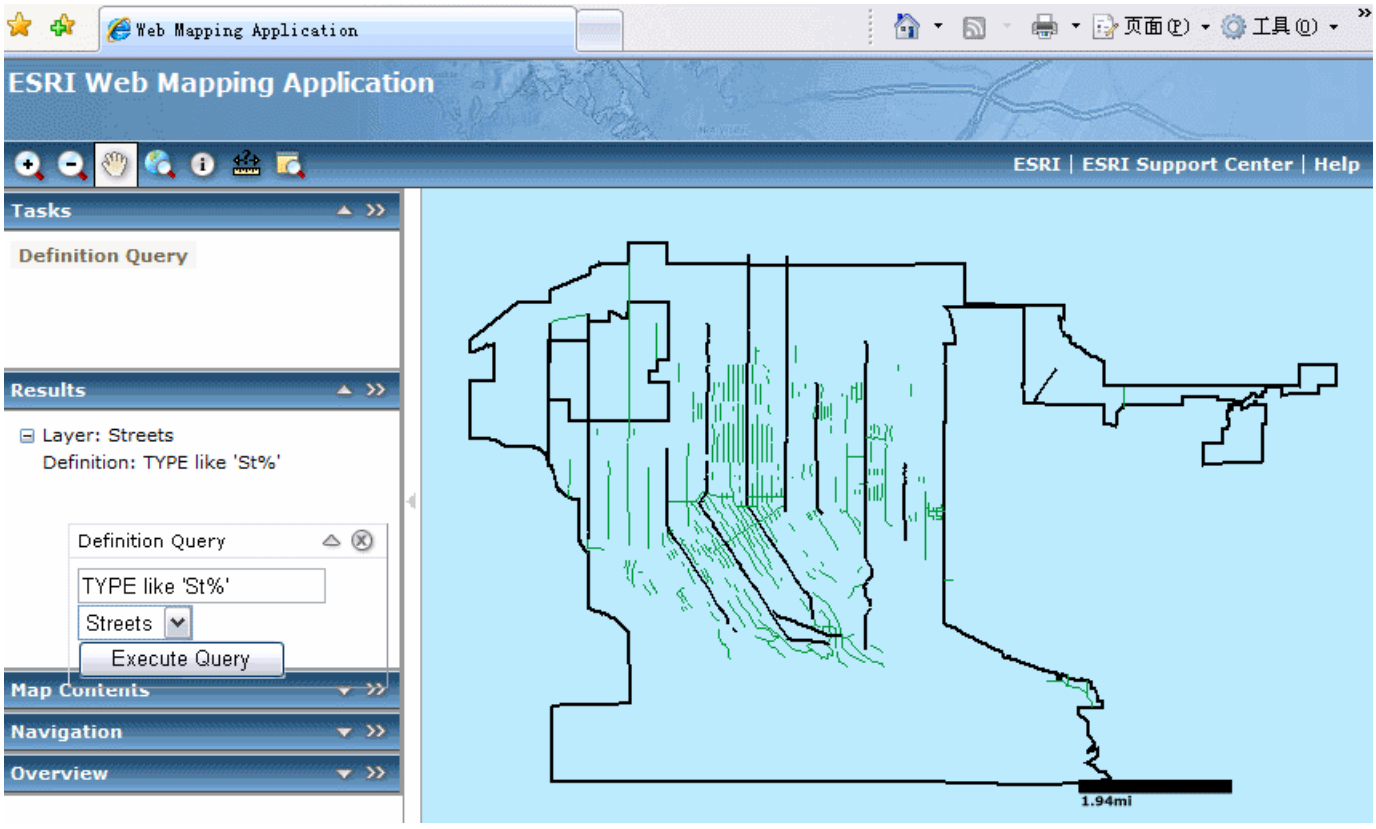
最后所有回调的结果均返回到回调响应函数——processCallbackResult，它专门处理客户端 Web ADF 控件的回调响应，并触发一系列后续动作，更新客户端显示，这样整个 Task 就执行完成了。

Build 整个工程，生成 DefQuery 的 DLL 文件，用模板新建一个 ArcGIS Server 应用，展开工具栏右键空白处，选择之前生成的 dll，此时就可以像 QueryAttributeTask 等原有的 Task 一样来使用 DefQuery 了，添加到 TaskManager 中，发布整个 Web 应用，这样自定义的 Task 就完成了，可以看看效果，下图做了一个简单的查询 Task，根据用户写的属性查询条件，在地图上返回所查询的地图。

Task 查询之前


Web Mapping Application
ESRI Web Mapping Application
ESRI | ESRI Support Center | Help
Tasks
Definition Query
Results
Map Contents
Navigation
Overview
Definition Query
TYPE like 'St%'
Streets
Execute Query
1.94mi

Task 查询之后


Web Mapping Application
ESRI Web Mapping Application
ESRI | ESRI Support Center | Help
Tasks
Definition Query
Results
Layer: Streets
Definition: TYPE like 'St%'
Definition Query
TYPE like 'St%'
Streets
Execute Query
Map Contents
Navigation
Overview
1.94mi

进一步完善应用：

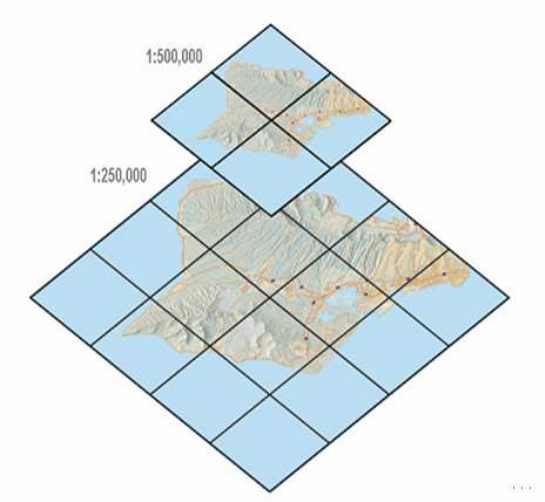
1.以上自定义 Task 的 UI 基于 FloatingPanel，我们可以更改它的现实样式，比如透明度、颜色、布局等等。
2.可以将显示出来的结果存为一个 Graphic 图层，保留原始图层。
3.根据属性查空间信息，考虑查询到结果时，同时将 Map 定位到结果所在的窗口范围。

ArcGIS 的缓存技术

地图缓存是快速访问地图服务的有效方式，目前流行的 Google 地图，MapBar 等在线地图都是通过缓存地图的方式提供高效的地图访问速度。ArcGIS 的缓存技术为企业和公共 GIS 应用提供了基本的技术保证。下面我们看看 ArcGIS 缓存：

一、 ArcGIS 的缓存服务

ArcGIS Server 的缓存服务有2D 和3D 两种，分别为二维和三维地图提供服务。根据应用需求，还可以分为**融合缓存（如图1）、多图层缓存、以及按需缓存**。融合缓存把所有图层群组在一起生成切片块。多层缓存则为每个图层建立切片块，可以支持客户端控制图层显示、标注和要素选取。按需缓存则不是事先生成切片，而是根据客户端首次访问请求创建切片，可以节省空间和缓存创建时间。



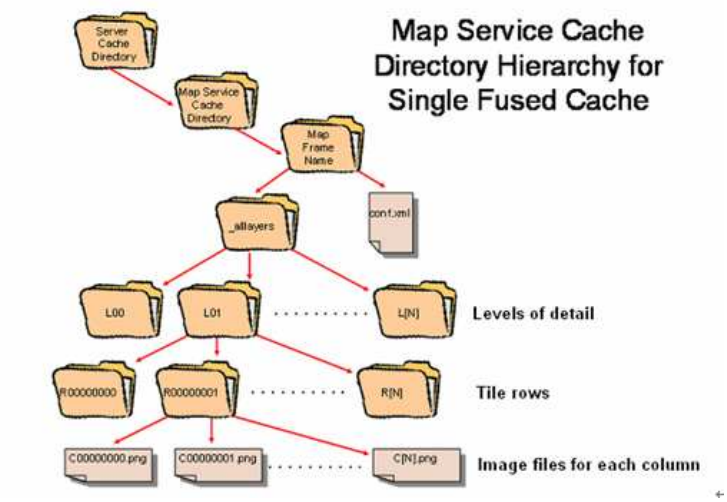


图1：ArcGIS 缓存比例尺和融合缓存结构

ArcGIS Server 缓存由目录和配置文件组成，并根据一定的规则命名，如 LXX,RXX,CXX。ArcGIS 支持与 ArcGIS Online、Google Map、Virtual Earth 等的地图叠加，因此在创建缓存中支持按照 GMAPS 或 VW 的 Scale 的规则来创建缓存，实现地图集成应用。

二、 ArcGIS 缓存工具

ArcGIS 提供了缓存管理的完整的工具。可以在 ArcCatalog 中创建缓存、删除缓存、更新缓存。也可以通过 ToolBox 的工具来管理缓存，如图2所示。

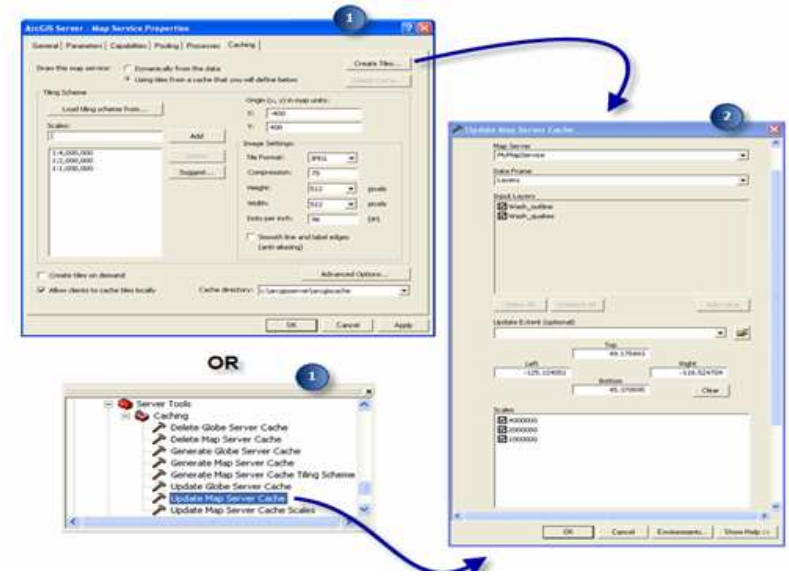


图2：ArcGIS 缓存工具

ArcGIS 支持全图范围创建缓存，也可以制定某个地图范围内创建缓存地图。在9.3中，ArcGIS 提供更多的方法，可以按照某个要素范围，例如某个省范围内进行局部地图缓存的创建或更新。

ArcGIS 可以手动输入所要创建缓存的比例尺，也可以直接输入要创建缓存的级别，例如10级，自动分配比例尺。ArcGIS 可以把创建的缓存级别保存为配置文件，方便下次直接调用，或共享。
对于全国范围再到某个县、乡这样级别的地图，需要创建多个比例尺不同级别的缓存，需要比较大的时间代价来实现。ArcGIS 可以在多台机器上分别创建各自比例尺，并把生成的缓存地图拷贝到同一个结构目录下，节省时间代价。这对于地图更新同样适用，保持地图快速更新。

三、 ArcGIS 缓存关键概念

（1）、Tiling Schema

切片结构包括比例尺等级、切片尺度以及缓存的切片原点。切片结构可以选择几种方式：

从在线地图服务，例如 ArcGIS Online,Google Maps,或 Microsoft Virtual Earth 中载入
从已有 Map Service 中载入
载入 tiling schema 文件
创建自己的 tiling schema

（2）、比例尺

在创建切片结构时候，可以自定义比例尺，也可以让 ArcGIS Server 按照2的幂数推荐一些比例尺。例如，如果当前比例尺是1:121,000,000，定义了3个比例尺级别，那么系统自动推荐了以下三个比例 尺：1：128，000，000；1：64，000，000；1：32，000，000。

（3）、Tiling Schema Origin

切片结构原点是切片结构格网的左上角。大多数情况下，保持系统选择的原点。如果地图文档没有定义坐标参考，系统将选择地图最大范围的两倍范围作为地图原点。

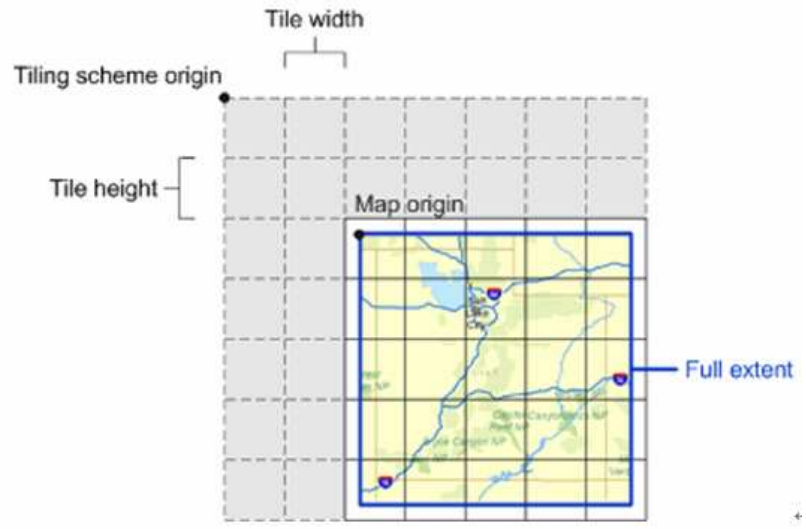


图3：切片结构原点

（4）、块宽度和高度

默认的宽度和高度是512像素。你可以选择它的倍数，如256或128。ArcGIS Online 默认是512x512，Virtual Earth 则采用256x256。Windows 磁盘最小块大小是4k，即同样图片，采用256需要16k 空间，采用512只需要4k 空间。所以推荐使用 512x512。

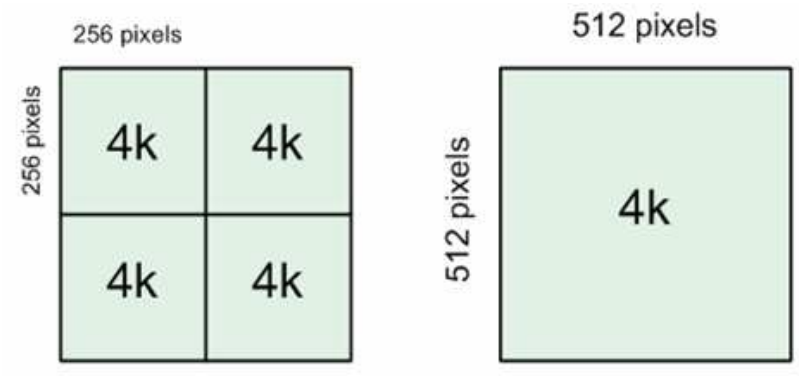


图4：块高度和宽度

（5）、反锯齿

反锯齿是图像处理技术，利用前景和背景的像素接近边缘，让眼睛看起来显得更光滑的边缘。ArcGIS Server 缓存并不是使用真的反锯齿，但是做相似的工作，让标注、未填充的线和多边形要素看起来更光滑。如果你在做矢量图层和影像叠加时候，要小心使用 这个工具。数据框的背景色取决于反锯齿要素的颜色。默认情况下，背景框的颜色为 RGB 253,253,253（几乎是白色的），这会引起反锯齿要素像故意变白，特别是当放置了暗色的栅格图层如正射影像的时候。最有效的方式是，在创建缓存以 前把背景色改成暗色。



图5：反锯齿

（6）、DPI

DPI 指 dots per inch，指服务器产生缓存块时候的分辨率。DPI 依赖于用户的屏幕分辨率（例如 1024x768）和物理大小(如，19”寸)。DPI 在客户端机器上差 别比较大，因为大的屏幕（低 DPI）和高分辨率的屏幕和图形卡（相应的 DPI 通过设置可以增加）。如果客户端频繁的需要打印地图，包括你的缓存图层，你会 发现提高 DPI 是很有帮助的，但是需要注意的是这也将增加缓存文件的大小。

当选择 DPI 时候，默认值是96，通常来说已经足够了，除非你是在网络上工作，而且你的客户端机器有不同的 DPI。

四、 使用缓存

（1）、Web ADF

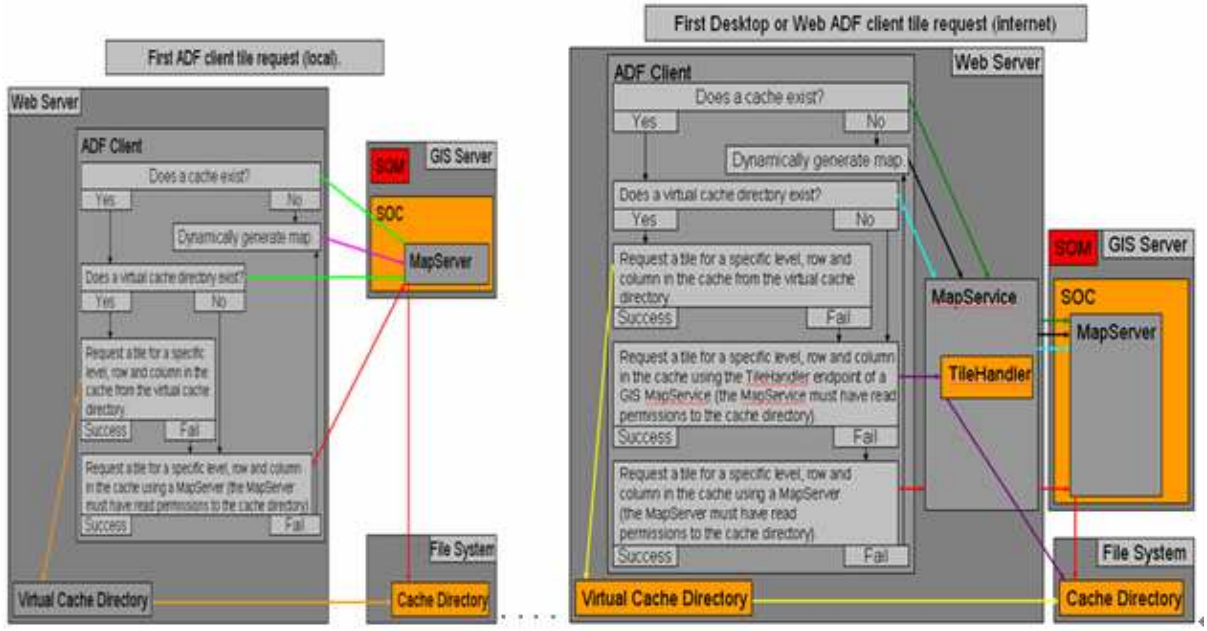


图6：ADF 访问缓存

如图6，使用 ADF 可以判断是否存在缓存，并基于本地（Local）或 Internet 方式访问缓存的地图。

（2）、ArcGIS 应用

ArcMap、ArcReader、ArcGlobe、ArcGIS Explorer、ArcGIS Mobile 应用都可以访问 ArcGIS 的缓存地图服务。

（3）、Open API

通过开源的 API，例如 Flex、OpenLayer、Rest API 等可以访问 ArcGIS 的缓存地图服务。

五、 总结

ArcGIS 缓存技术结合 GIS 全面技术为企业应用提供强大的技术支持，概括起来有以下几点：

（1）、缓存服务类型符合多方位应用需求。ArcGIS 提供二维、三维缓存技术，并结合应用需求，支持融合缓存、多图层缓存、按需缓存，满足实际需要。

（2）、丰富的缓存管理工具。结合 ArcGIS 功能，可按照全图、区域、要素范围，按照制定或推荐比例尺创建缓存。也可以结合 Geoprocessing 功能，定制缓存计划。

（3）、ArcGIS 可实现缓存与非缓存的地图叠加，也可以根据需要按需缓存。

（4）、ArcGIS 缓存支持与 Google Maps、Virtual Earth 的结合，并为其它开源应用提供使用。

深入浅出 ArcGIS 9.3 Javascript API

我只想快速开发简单的 Web GIS 应用，有没有比 ADF 控件再简洁一些的方式？我不想记那些复杂的 AO 接口，可是又喜欢 ArcGIS Server 漂亮的地图和强大的功能，怎么办？好吧，在 9.2 时代，面对这些问题，我们很头疼，但是 ArcGIS Server 9.3 已经插上了 REST 的翅膀！ArcGIS 的 fans 们兴奋吧！期待吧！

虽然有些枯燥，但是技术细节总是绕不开的，ArcGIS Server 9.3以全新的 REST 方式实现了一些 ArcGIS Server 通用功能和服务，便于 ArcGIS Server 客户端开发，如 javascript api。安装了 ArcGIS Server 9.3之后，可以通过"http://<host>/<instance>/services/<folder>" 方式来访问服务的根目录。其中"instance"默认为"arcgis/rest"，里面列举了当前服务器上所有已启动的 server 服务，并且可以将 这些服务列表导入 Google Earth，格式为 kmz。REST 访问地址中，folder 和 service name 是对大小写敏感的，folder 前面的部分则没有影响。

点击根目录列举的一个服务，页面会展示该服务的所有基本信息，包括地图图层、图层编号、空间参考、部分 cache 信息、初始化地图显示范围、地图全图显示范围、地图单位、MXD 文档基本信息。

发现其中的妙处了吗？现在只需要通过 Http 的 URI 就可以轻松的访问到 ArcGIS Server 的地图和服务了，我们要介绍的 Javascript API 就是基于这种方式实现的。REST 本身其实是一个完美的技术理想，并没有这么简单，不过我们并没有必要搞清楚 REST 的所有细节才能享受到它带来的 好处。还是让我们来看看基于 REST 构架的 Javascript API 吧，这才是更值得我们关注的！

第一章 基本框架

首先来看看 JavaScript API 是什么。JavaScript API 是 ArcGIS Server 9.3新增的一套 API 框架，为创建 WebGIS 应用提供了轻量级的解决方案，在客户端可以轻松地利用 JavaScript API 来调用 ArcGIS Server 所提供的服务，实现地图应用和地理处理功能。**最让人兴奋的是这一切操作都是在客户端仅仅用脚本调用服务器端的接口完成的，不需要写任何的服务器端代码。**

熟悉 ArcIMS 的人一定还记得曾经风靡一时的 ArcIMS Blueviewer 示例框架。里面的所有操作都是基于客户端实现，**放大缩小平移都是在 javascript 脚本中完成的，将结果返回服务器得到我们所需的图片。**当时这个 Demo 让很多人耳目一新。熟悉 Blueviewer 的程序员能够更好的把握 WebGIS 开发中的基础流程，并在其基础上写出漂亮的 WebGIS+Ajax 应用。类似的，JavaScript API 也是纯粹的客户端操作，但是它基于更新更炫的 **Dojo 库**，并用面向对象的方式重新封装了功能。

理论上我们可以对 JavaScript API 无限扩展，以实现 ArcGIS Server 所有服务器端功能，目前已经封装好的基础功能有：

1.以用户数据和服务器端服务结合显示交互性的地图
2.执行一个 GIS 空间分析模型并显示结果
3.在 ArcGIS 在线基础地图上显示我们自己的数据
4.对属性或几何图形进行搜索并显示结果
5.搜索地址并显示结果

JavaScript API 包含的资源组件有：

Maps--支持 ArcGIS Server 上 cache 和非 cache 地图的显示，还可以在客户端按需要指定投影参考系，这样就可以轻松实现跟各种标准地图服务的叠加

Graphics--绘图，和通过鼠标、键盘等操作实现属性信息的展示或地理操作

Tasks--和 ADF Tasks 类似，提供 Querying、Finding addresses、Finding attributes、Identifying features、Geoprocessing 等丰富的 GIS 功能

基于 Dojo 和其他库进行扩展--比如利用 Dojo widgets，google maps api，virtual earth api 等等。

ESRI 选择 Dojo 作为 JavaScript API 基础库有如下几点好处：

1.Dojo 宽松的使用许可，基于标准的开放式应用
2.Dojo 功能全面，**事件设计模式**、矢量图形支持、显示效果、widgets、Ajax、JSON 等都是 ArcGIS Server **客户端开发所需要的**，并且 Dojo 利于扩展，方便 Mashup 应用，并且有大量的参考资料
3.IBM、Sun、AOL 等公司的支持，可轻松跟各种大型系统集成

一个 WebGIS 应用的基本内容包括地图显示与放大、缩小、移动、坐标显示等基本操作，那么在 JavaScript API 中是如何实现的？实际上 JavaScript API 在这些基本操作方面提供了非常便捷的开发方法。一般的 WebGIS 需要 ArcGIS Server 服务器支持，即使系统没有配置 Server 服务，也可以使用 ArcGIS Online 在线服务。

基于 Dojo 的 JavaScript API 天生具有 Dojo 的**跨浏览器特性和强大的前端(用户端)制图功能**，由于 Dojo 的开放性和扩展性，JavaScript API 将具有令人期待的发展前景。在 ESRI 的资源站点上提供了丰富的示例(http://resources.esri.com)

使用过 Dojo 的开发人员会很熟悉 JavaScript API 应用的整个页面结构，让我们来看一个例子：

1.使用 Dojo 提供的样式表

```
<style type="text/css">
    @import "http://serverapi.arcgisonline.com/jsapi/arcgis/1/js/dojo/dijit/themes/tundra/tundra.css";
</style>
```

2.引用脚本文件

```
<script type="text/javascript" src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=1">
</script>
```

这里可能会有一个疑问，为什么是这样的目录结构，里面包含了 Dojo 和 JavaScript API 所需要的 js 库吗？其实部署 JavaScript API 时，针对不同类型服务器端代码（Java、.Net、php）会有不同的 Default 页面，以.Net 为例，Default.ashx 引用了三个 文件：

```
context.Response.WriteFile(context.Server.MapPath("js""esri""esri.js"));
context.Response.WriteFile(context.Server.MapPath("js""dojo""dojo""dojo.xd.js"));
context.Response.WriteFile(context.Server.MapPath("js""esri""jsapi.js"));
```

3.接下来，就要对页面中所添加的地图元素与功能进行定制，同样是在脚本中完成，使用 JavaScript API

```
<script type="text/javascript">
    dojo.require("esri.map");
    var myMap, myTiledMapServiceLayer;
     function init() {
        myMap = new esri.Map("mapDiv");
        myTiledMapServiceLayer = new esri.layers.ArcGISTiledMapServiceLayer
("http://server.arcgisonline.com/ArcGIS/rest/services/ESRI_StreetMap_World_2D/MapServer");
        myMap.addLayer(myTiledMapServiceLayer);
    }
    dojo.addOnLoad(init);
</script>
```

dojo.require 引入所需的包，其中的 esri.Map 类实现了图层、Graphics、消息窗口（查询属 性）、其他导航控制的方法，这里在指定的 div 中创建了一个地图。esri.layers.ArcGISTiledMapServiceLayer 的作用 是获取 REST API 所提供的 cache 地图服务。这是一个在线服务，REST 所能提供的其他服务可以查看 REST API 文档。esri.layers.ArcGISTiledMapServiceLayer 继承于 esri.layers.ArcGISMapServiceLayer，该父类下还有另外一个子类 ArcGISDynamicMapServiceLayer，用于获取非 cache 地图服务。

最后通过 esri.Map 的 addLayer 方法将该服务中的图层添加到当前地图中显示。

dojo.addOnLoad(init)等价于<body onload=init>，页面加载的时候将执行 init()脚本。

4.页面 body 区域

```
<body>
    <div id="mapDiv" class="tundra" style="width:900px; height:600px; border:1px solid #000;"></div>
    <h4>Work flow:</h4>
    <ul>
        <li>Create a map.</li>
        <li>Add an ArcGISTiledMapServiceLayer.</li>
    </ul>
</body>
```

最终地图将在 mapDiv 区域绘出。

5.放大缩小平移

esri.Map 类包含地图放大、缩小、平移等基本功能的实现，可以将这些操作和业务应用绑定。通过 esri.Map 构造方法生成的地图对象实际已经包含 一些基础操作的快捷方式，如键盘方向键上下左右完成地图上下左右的固定单位移动，鼠标滚轮实现地图放大缩小，"shift+单击"为居中，"shift+ 双击"为居中放大，"+"放大一级，"-"缩小一级。

6.动态显示当前鼠标位置的经纬度坐标

动态显示鼠标经纬度坐标大体上有两种方式，：一是服务器端生成坐标；二是客户端生成坐标，。孰优孰劣都很清楚了，服务器生成坐标值会消耗大量服务器资 源，onMouseMove 总是在不停向服务器发送请求。这种模式下经常会碰到鼠标移动已经停止了，但是状态栏上的坐标信息还在变化，这就是服务器计算造成的延时造成的。因此对于真 正上线的 WebGIS 网站更多使用客户端方式，通过获取屏幕坐标，在投影参考系下换算地理坐标，显示在页面。

JavaScript API 提供了一套 API 获取地理坐标：

```
function showCoordinates(evt) {
    var mp = evt.mapPoint;
    dojo.byId("info").innerHTML = mp.x + ", " + mp.y;
}
```

mapPoint 是地理坐标，screenPoint 是屏幕坐标。

7.通过 dojo.connect 建立对鼠标事件的监听

```
dojo.connect(map, "onMouseMove", showCoordinates);
dojo.connect(map, "onMouseDrag", showCoordinates);
```

同时监听鼠标拖拽事件，添加到 map.onLoad 中就可以了，当然 body 区域中要加入显示坐标的 div

```
<div id="info" style="padding:5px; margin:5px; background-color:#eee;"></div>
```

完工！显示效果，界面很简单，但包含了鼠标和键盘的快捷键操作：

对于 WebGIS 来说，地图配置是核心问题之一。地图是为业务服务的，围绕对地图基本操作的一些效果进行定制，有的是从程序调试考虑，有的从界面美 观上考虑，有的从服务器响应速度上考虑。JavaScript API 丰富的客户端表达和清凉便捷的开发方式可以非常好的处理这多种要素之间的关系，达到理想效果。

我们来看采用 JavaScript API 模式如何进行程序调试，定制导航、移动、放大、缩小动画等基本地图功能以及制作滑动条标注、刻度、位置、大小等扩展地图功能。

1.程序调试是开发中必不可少的一个重要过程，js 调试在 Firefox 浏览器上有一个大家耳熟能详的工具 Firebug。而在 IE 的环境中，可以利用 Dojo 的 debug 控制台管理调试过程中的信息，帮助开发。

在 head 区域声明：

```
<script type="text/javascript">djConfig = { isDebug:true };</script>
```

然后在客户端事件响应过程中加入对事件的监控，并写入日志：

```
dojo.connect(map, "onExtentChange", function(extent) { console.log("Extent changed : " + dojo.toJson(extent.toJson())); });
```

2.导航按钮的定制很简单，esri.Map 类提供了八个方位角所对应的方法：

```
map.panUpperLeft()
map.panUp()
map.panUpperRight()
```

分别对应西北、北、东北，其他的可以在文档中查找。这些方法可以直接在客户端 js 脚本事件中调用，如按钮的 onclick 事件。

3.移动和放大缩小的动画效果取决于客户的视觉感受，包括地图变化持续时间和移动（放大缩小）速度，但也要考虑服务器的负载，比如服务器访问量比较大。我们可以使用较长的持续时间和较慢的速度来改善用户体验，不至于让用户看到空白等待地图的出现。

移动（默认）：

```
esriConfig.defaults.map.panDuration = 250; //0-1000ms
esriConfig.defaults.map.panRate = 25; //0-100
```

放大缩小（默认）：

```
esriConfig.defaults.map.zoomDuration = 250;
esriConfig.defaults.map.zoomRate = 25;
```

esriConfig 可以用来重写 JavaScript API 的默认配置参数，除了上面列举的移动和放大缩小效果外，还有滑动条样式、标注，、放大框形状，以及连接 REST 连接点的代理等。

4. 滚动条指的是地图默认左上方的放大缩小滑动条，类似于其他大众 WebGIS 应用站点 Googlemap、Mapbar。通过简单的编程，可以修改滚动条的 显示效果，如标注、刻度、位置、大小。创建 esri.Map 对象的时候，可以指定 option 参数，其中 slider 参数为 Boolean 类型，默认为 true，写为 false 则不显示滚动条。

滚动条标注指每一级放大缩小刻度所需显示的标注，通过针对 cache 地图，比如1级、2级、……或每个级别的比例尺

图中显示的是比例尺的大小，通过 layer.tileInfo.lods 获取每个级别比例尺的信息

```
var lods = layer.tileInfo.lods;
  for (var i=0, il=lods.length; i<il; i++) {
      labels[i] = lods[i].scale;
}
```

然后配置 esriConfig.defaults.map.sliderLabel 即可。默认状态和设置了"esriConfig.defaults.map.sliderLabel=false"效果分别为：

位置和大小主要通过"esriConfig.defaults.map.slider"指定，可以让滚动条水平放置，还有设置离 map 边框的距离。以上这 些都是客户端开发和网站界面设计时所需要考虑的，JavaScript API 提供了一套比较方便的方法来实现各种效果。

Graphics 绘制是那个能让死气沉沉的 WebGIS 应用变得精彩有趣的点石成金魔棒。JavaScript API 中拥有更强大的 Graphics 绘制功能。基于 JavaScript API 开发很容易联想到其鼻祖——Dojo 丰富灵活的图片绘制功能。在 Dojo 中，point、multipoint、polyline、polygon 等基本图形可以很简洁的在地图上绘制出来，并且展示效果令人兴奋！

在开始这部分的示例前，首先要引入一个新的库"esri.toolbars.draw"。Toolbar 并不是一个用户接口组件，而是一个 Helper 类用于在地图上绘制图形，通常由客户端的 UI 组件来触发所要绘制的具体图形，如通过按钮触发：

用户绘制完成后，将触发事件来完成具体图形的显示：

```
dojo.connect(tb, "onDrawEnd", addGraphic);
```

了解 AO 和 ArcGIS Server ADF 的开发人员都知道，图形一般由两部分组成，一是所要绘制的几何图形，即 geometry，一是绘制图形的显示效果，如 SimpleMarkerSymbol，所以在 addGraphic 方法要完成这两部分的赋值：

```
function addGraphic(geometry) {
    var symbol = dojo.byId("symbol").value;
     if (symbol) {
        symbol = eval(symbol);
    }
     else {
        var type = geometry.type;
         if (type === "point" || type === "multipoint") {
            symbol = tb.markerSymbol;
        }
         else if (type === "line" || type === "polyline") {
            symbol = tb.lineSymbol;
        }
         else {
            symbol = tb.fillSymbol;
        }
    }
    map.graphics.add(new esri.Graphic(geometry, symbol));
}
```

1.在上文对 Graphics 绘制功能编写基础上，以官方网站上的代码为例，在客户端提供多种基础形状的绘制功能，此时 Graphics 形状由用户客户端通过 toolbar 提供的 helper 方法决定，绘制的显示效果通过一个 DropDownList 进行选择：

```
<option value="new esri.symbol.SimpleMarkerSymbol(esri.symbol.SimpleMarkerSymbol.STYLE_SQUARE, 10, new esri.symbol.SimpleLineSymbol(esri.symbol.SimpleLineSymbol.STYLE_SOLID, new dojo.Color([255,0,0]), 1), new dojo.Color([0,255,0,0.25]))">Square</option>
```

Toolbar 初始化及事件监听在 map.addLayer 之前完成：

```
function init() {
    map = new esri.Map("map");
    dojo.connect(map, "onLoad", initToolbar);
    map.addLayer(new esri.layers.ArcGISTiledMapServiceLayer("http://server.arcgisonline.com/ArcGIS/rest/services/ESRI_StreetMap_World_2D/MapServer"));
}

function initToolbar(map) {
    tb = new esri.toolbars.Draw(map);
    dojo.connect(tb, "onDrawEnd", addGraphic);
}
```

构建界面 UI 之后，可以直接浏览网页，Demo 演示地址：
http://resources.esri.com/help/9.3/arcgisserver/apis/javascript/arcgis/demos/Graphics/graphics.html

2.框选点要素，在很多行业应用中都涉及到，如对银行 POI 网点选择、电线杆塔统计、烟草销售点业态分析等等，在 JavaScript API 实现这样的功能，需要分两步实现，第一是框选查询，第二是对查询的结果渲染并以 Graphics 方式显示。

查询功能在后续文章中会详细介绍，这里仅轻笔带过，引入"esri.tasks.query"，查询功能实现：

```
var queryTask = new esri.tasks.QueryTask("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Specialty/ESRI_StatesCitiesRivers_USA/MapServer/0");
var query = new esri.tasks.Query();
query.where = "STATE_NAME = 'Washington'";
query.returnGeometry = true;
query.outFields = ["CITY_NAME"];
queryTask.execute(query, addPointsToMap);
```

上面 URL 指向的是另外一个 REST 服务，addPointsToMap 将所有符合条件的点都添加到 Map 中作为 Graphics 显示，接下来就是要在这些点中框选点并用不同颜色高亮显示。

```
function findPointsInExtent(extent) {
    var graphics = map.graphics.graphics;
    var results = [];
    var graphic;
```

```
        for (var i=0, il=graphics.length; i<il; i++) {
            graphic = graphics[i];
            // 选择框所包含的点用 highlightSymbol 高亮显示
             if (extent.contains(graphic.geometry)) {
                graphic.setSymbol(highlightSymbol);
                results.push(graphic.getContent());
            }
            // 之前已经 highlightSymbol 高亮显示的点恢复为原来 defaultSymbol 显示
             else if (graphic.symbol == highlightSymbol) {
                graphic.setSymbol(defaultSymbol);
            }
        }
    }
```

功能开发就完成了，剩下的就是加上对事件的监听以及客户端 UI 编辑。

```
  function initToolbar(map) {
      var tb = new esri.toolbars.Draw(map);
      dojo.connect(tb, "onDrawEnd", findPointsInExtent);
      tb.activate(esri.toolbars.Draw.EXTENT);
  }
```

篇幅所限，如果想了解更多 JavaScript API 请关注 Flyingis 的 blog，地址：http://www.cnblogs.com/flyingis。

基于 REST 构架的 JavaScript API 丰富了 ArcGIS Server 的开发方式，使得没有接触过 GIS 的 Web 开发者也可以通过纯粹的客户端脚本技术轻松地调用复杂的 GIS 功能。这种轻量级便捷的开放方式让人 兴奋，但是 REST 的迷人魅力才刚刚开始展现，JavaScript API 刚刚打了个前站，基于 Dojo 的开放方式使得 JavaScript API 具有丰富的客户端展示功能和令人期待的发展潜力。另外，ESRI 不久将会推出基于 REST 模式的 Flex API。基于 Flex，将强大的 GIS 功能和炫目的 Flash 结合在一起将成为现实。

如果您关注 ArcGIS Server 的最新动态和精彩演示，请访问：http://resources.esri.com/arcgisserver/

JavaScript API 官方资源站点：http://resources.esri.com/arcgisserver/index.cfm?fa=jsapis

如何动态添加图层并进行查询

软件：ArcGIS Server for Java platform

版本：9.2

平台：Solaris 9.0, 10； Linux Red Hat AS/ES 3.0, AS/ES 4.0； Linux-SUSE Server 9

在 ArcGIS Server 9.2 的应用程序中，如果使用的资源类型是 com.esri.adf.web.ags.data.AGSLocalMapResource，就可以在该资源上动态添加新的图层，图层数据可以来自不同的地方，比如本地的 shapefile、file geodatabase 或者 SDE，也可以来自 WMS 等 web 服务。本文以本地的 file geodatabase 为例，为 AGSLocalMapResource 动态添加一个新的图层。

首先，要得到地图的 IMap 接口才能对地图的图层进行操作。

清单1.获取 IMap 接口

```
1.  AGSLocalMapResource res =
(AGSLocalMapResource)webContext                                    .getResourc
es().get("ags0");

2.  com.esri.arcgis.carto.MapServer mapServer = res.getLocalMapServer();

3. try {

4.      IMap map = mapServer.getMap(mapServer.getDefaultMapName());

5.      map.addLayer(ILayer layer);

6.  }catch(Exception e){

7.  }
```

动态添加图层的核心代码就是第5行，它接收的参数是实现了 ILayer 接口的对象。因此，接下来我们需要把 file geodatabase 里的 feature class 组装成一个 ILayer 对象。

清单2.从 Feature Class 到 ILayer

```
1. IServerContext sctx = res.getServerContext();
```

```
2.  IWorkspaceFactory pWorkspaceFactory = (IWorkspaceFactory)
sctx                                    .createObject(FileGDBWorkspace
Factory.getClsid());

3.   IFeatureWorkspace pFWS = (IFeatureWorkspace)
pWorkspaceFactory                                .openFromFile("E:\\data\\
qixiang\\qixiang.gdb", 0);

4.   IFeatureClass fc = pFWS.openFeatureClass("Road");

5.  IFeatureLayer fLayer = (IFeatureLayer)
sctx                                              .createObject(Feature
Layer.getClsid());



6.  fLayer.setFeatureClassByRef(fc);

7.  fLayer.setName("Road");
```

Server 中的 AO 对象，必须在一个 IServerContext 上下文中使用，因此，AO 对象的创建跟一般的 Java 类有所不同。因为数据是保存在 filegeodatabase，所以在第2行代码中我们创建了一个 FileGDBWorkspaceFactory，接下来的两行利用这个工厂类打开了一个 IFeatureWorkspace，然后读取了其中的一个 feature class。最后，用这个 feature class 设置了一个 IFeatureLayer（ILayer 的一个子类）。

有了 IMap 和 IFeatureLayer 对象以后，其实就已经能动态添加图层了。但是这样添加的图层有一点欠缺的地方——它的渲染方式是随机的。就像我们用 ArcMap 直接打开一个 polygon feature class 时，ArcMap 会随机选择一个填充色。为了保证每次用户添加图层以后能够看到同样的效果，我们需要为动态图层设置一个渲染方式。当然，我们可以在程序中为不同类型的要素分别设置一个 Renderer，但是本文将探讨另一种方式。先用 ArcMap 设置好渲染方式，然后保存成一个*.lyr 文件，动态添加图层的时候从 lyr 文件中读取 Renderer 信息，然后添加到 IFeatureLayer。毕竟，用 ArcMap 设置 Renderer 比写代码要简单多了！而且，也更易于分发。

清单3.从*.lyr 文件中读取 Renderer 信息

```
1.  IMapDocument doc =
            (IMapDocument)sctx.createObject(MapDocument.getClsid());
2.  doc.open(lyrFilePath,null);
3.    ILayer lyr = doc.getLayer(0, 0);//get the first map's first layer
4.    IFeatureRenderer renderer =null;
5.   if(lyrinstanceof IGeoFeatureLayer){
6.                  IGeoFeatureLayer geoLyr = (IGeoFeatureLayer)lyr;
7.                  renderer = geoLyr.getRenderer();
8.   }
```

在清单3中我们创建了一个 MapDocument 对象来读取 lyr 文件（API 说明：The MapDocument CoClass encapsulates map document files (*.mxd, *mxt, *.pmf) and layer files (*.lyr)）。第2行代码中的 lyrFilePath 就是 lyr 文件存放的路径。由于打开的是 lyr 文件，只有一个图层，所以第3行代码的两个参数都为零，表示打开的是第一个地图的第一个图层。接下来从图层文件中读取 Renderer 信息。

接下来，我们就可以把 Renderer 信息赋给图层，然后将图层添加到地图上。

清单4.设置渲染方式，添加图层

```
1. if(renderer !=null){
2.          IGeoFeatureLayer geoFeatureLyr = (IGeoFeatureLayer)fLayer;
3.          geoFeatureLyr.setRendererByRef(renderer);
```

```
4.      }

5.      fLayer.setVisible(true);

6.      map.addLayer(fLayer);

7.     map.moveLayer(fLayer, 1);
```

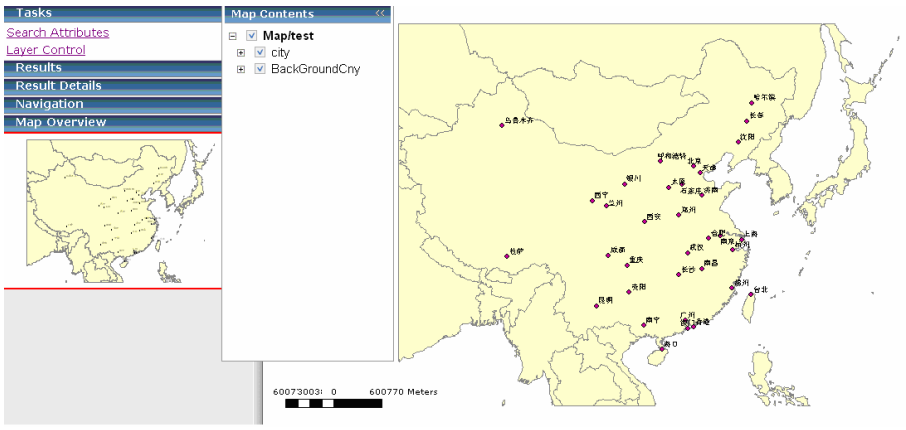
至此，我们通过 map service 提供的 AO 接口完成了动态添加图层的工作，但是如果这时候我们去刷新浏览器，并不会看到新增加的图层！这是由于 Web ADF 层还不知道图层信息发生变化了。这与 Web ADF 的初始化有关，当一个 session 启动的时候，WebContext 会根据 map service 的信息完成初始化，包括一些列的 functionalities 以及 attributes，这些初始化信息中就包含图层信息（保存在 com.esri.arcgisws.MapDescription 和 com.esri.arcgisws.MapLayerInfo 等类中）。在 WebContext 初始化完成以后，如果 map service 的信息发生改变，必须由程序员自己去刷新 Web ADF 中的相关对象，实现 Web ADF 对象与 map service 同步。有趣的是，并不是所有对 map service 的修改都要自己去通知 Web ADF，比如修改图层的 Renderer 信息就不需要。我觉得如果 com.esri.arcgisws 包中的与 AO 同名的对象，很可能就需要手工修改，因为这些对象都是初始化的时候创建的；如果没有，说明没有必要为其单独创建 Web ADF 对象，需要的时候就直接访问 AO 接口了。

清单5.刷新 ADF 对象

```
1.  mapServer.refreshServerObjects();


2.  IMapServerInfo serverInfo =
            mapServer.getServerInfo(mapServer.getDefaultMapName());


3.  agsServerInfo =
            (com.esri.arcgisws.MapServerInfo)AGSUtil.createStubFromArcObject(ser
```

```
            verInfo,com.esri.arcgisws.MapServerInfo.class,sctx);



4.      com.esri.arcgisws.MapServerInfo si = mapFun.getMapServerInfo();

5.      si.setMapLayerInfos(agsServerInfo.getMapLayerInfos());

6.    mapFun.setMapDescription(agsServerInfo.getDefaultMapDescription());
```

在清单5中，我们先刷新了 map server object，然后获取了新的 IMapServerInfo 对象。第3行代码中我们完成了从 AO 对象到 ADF 对象的转换，这样，ADF 就有了一个反映当前 map service 状态的 com.esri.arcgisws.MapServerInfo。4到6行代码用新的 com.esri.arcgisws.MapServerInfo 刷新了 AGSMapFunctionality，从而完成了 ADF 与 map service 的同步。下面的截图显示了动态添加图层的效果。



由于 WebContext 以及它管理的一系列 ADF 资源只在 session 开始时进行一次初始化，因此动态添加的图层就无法用 WebQuery 来进行查询和高亮显示。在清单5中已经介绍了如何刷新 ADF 端的对象，因此，我们可以用更新以后的 com.esri.arcgisws.MapServerPort 对动态添加的图层进行查询。

接下来的例子中，让用户在地图上拖一个矩形框，然后查询所有图层中与该矩形框相交的要素，并将它们高亮显示。首先需要将用户在地图上绘制的矩形框传递到服务器端，并创建一个 com.esri.arcgisws.PolygonN 对象。

清单6.捕获屏幕操作并构造服务器端几何要素

```
1. WebExtent ext =
 (WebExtent)event.getWebGeometry().toMapGeometry(webContext.getWebMap());

2. EnvelopeN env = (EnvelopeN)AGSUtil.toAGSGeometry(ext);
```

接下来可以根据该 PolygonN 新建一个 com.esri.arcgisws.SpatialFilter 对象。

清单7.创建 SpatailFilter 对象

```
1.  SpatialFilter spatialFilter =new SpatialFilter();

2.  spatialFilter.setSpatialRel(EsriSpatialRelEnum.esriSpatialRelIntersects);

3.  spatialFilter.setWhereClause("");

4.  spatialFilter.setSearchOrder(EsriSearchOrder.esriSearchOrderSpatial);

5.  spatialFilter.setSpatialRelDescription("");

6.  spatialFilter.setGeometryFieldName("");

7.  spatialFilter.setFilterGeometry(env);
```

com.esri.arcgisws.MapServerPort 的 queryFeatureData()方法只能对某一个图层进行查询，所以，如果要查询所有的图层，需要对所有图层做一次循环。

清单8.空间查询

```
1.  AGSMapFunctionality mapFunc =
```

```
            (AGSMapFunctionality)res.getFunctionality("map");
2. int layerCount = mapFunc.getLayerDescriptions().length;
3.    MapServerPort svrPort = res.getMapServer();
4.   try{
5.              for(int i=0;i<layerCount;i++){
6.          RecordSet rs = svrPort.queryFeatureData(“”,i, spatialFilter);
7.       }
8.   }catch(Exception e){
9.   }
```

第 8 行代码对每一个图层都做了一次空间查询，将查询结果保存在 com.esri.arcgisws.RecordSet 中。查询结果中包含了图层要素的所有信息，包括属性信息和空间信息，可以根据需要进行提取。这个例子中需要对查询结果做高亮显示，所以接下来我们关心的焦点就是获取查询结果的空间信息。

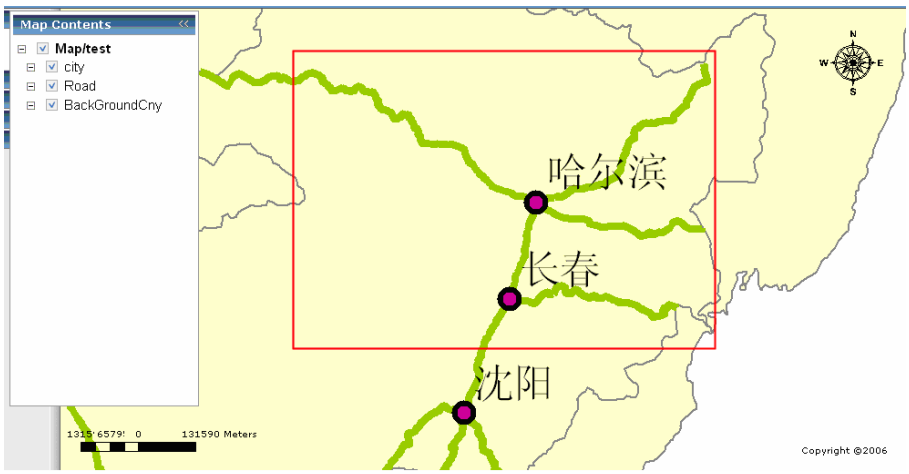
清单9.高亮显示查询结果

```
1.   Record[] records = rs.getRecords();
2. for(int j=0;j<records.length;j++){
3.              Record item = records[j];
4.              int n = item.getValues().length;
5.              for(int k=0;k<n;k++){
```

```
6.                    Object obj = item.getValues()[k];

7.                    if(objinstanceof com.esri.arcgisws.Geometry){

8.                          WebGeometry geom =
          AGSUtil.fromAGSGeometry((com.esri.arcgisws.Geometry)obj);

9.                          query.addDisplayGeometry(geom);

10.                   }

11.             }

12. }
```

com.esri.arcgisws.Record 对应的是某个图层的一条记录，它的 getValues()方法获取该记录属性表中的所有信息（包括 shape 字段）。第6行开始我们读取每一个字段的值，判断该字段是否记录了 shape 信息，如果是，就将 shape 信息转成 WebGeometry 对象，然后用 WebQuery 进行高亮显示。

下面两幅是查询然后高亮显示的效果图。





ArcGIS Server 常见问题

错误描述

在 ArcCatalog 中，当成功的创建了 server object 之后，server object 不能被预览。ArcCatalog 显示下面的错误信息：
“你的选择不能在当前视图中显示.”
导致原因
server 属性中被指定的输出值和 http 位置可能不正确。
解决办法
所有的创建的 ArcGIS Server Objects 需要被停止。
1 在 ArcCatalog 中，右击已经添加的 ArcGIS Server，并选择 Server 属性。
2 在目录页中，选中已经添加的输出目录并点击编辑。
3 输出目录字段应该指向一个有效的目录，比如 C:\ArcGIS\Output。
如果一个虚拟目录已经被指定，确认它是有效的，并且看上去和下面的格式一样，
http://<servername>/output
4 如果有错请更改，然后重新创建 server object 来确认更改是否有效。

错误描述
当在 Visual Studio.NET 2003 上使用 ArcGIS Server .NET 的 ADF 模板创建一个项目时，返回如下错误信息：
“在服务器<主机名>上的 Web 站点不可用（Web Site on server <host> is not available）”
导致原因
该模板包含了调用'http://localhost'这一 URL 的属性。如果在 IIS 中禁用了本地主机的服务，则显示该错误信息。禁用本地服务可能是因为你的 Web 站点已经为其指定了一个 IP 地址或者是因为安全方面的考

虑。
解决办法
启用对 http://localhost 的访问决定于您安装的是哪种类型的操作系统，是服务器还是非服务器。
对于非服务器的操作系统：
1.点击 Internet 信息服务控制台，右键选择默认站点的属性，打开属性对话框，并选择网站标签页。

2.修改 IP 地址栏为"(全部未分配)"。

3.重启 IIS。

在 IE 浏览器中浏览刚刚启用的 http://localhost 服务，在 Visual Studio .NET 2003 中用模板创建项目，现在应该是成功的了。
对于 Windows Server 操作系统：
1. 在 Internet 信息服务控制台上创建一个新的 Web 站点。

2. 为其分配 IP 地址：127.0.0.1。意味着只能在本地计算机上启用该站点。

3. 浏览 Web 站点的根目录，比如 C:\Inetput\wwwroot 。

此时在 Visual Studio .NET 2003 中用模板创建项目应该没有问题。

错误描述
当启动一个服务对象时，出现下述错误信息：“在机器 XXX 上创建 Server 环境（Context）失败。拒绝访问 output 文件夹。”
导致原因
ArcGIS Server Object Container 的帐号必须拥有虚拟文件夹的写权限。例如虚拟文件夹为:
c:\inetpub\wwwroot\temp
解决办法
执行如下步骤：
右键点击虚拟文件夹如: c:\inetpub\wwwroot\temp

1.选择属性->安全标签页.
2.添加帐号并赋予写权限，使其可以读写图片.
3.点击 OK，关闭属性对话框.

错误描述
使用下面的代码，来进行使用字体中的符号进行点要素渲染的时候，会出现地图消失的问题。
下面的代码是用来生成需要在渲染过程中使用的点符号的，使用的是宋体中 index 为 21 的字作为符号。

```
private static ICharacterMarkerSymbol
GetMarkerSymbol(IServerContext pServerContext,PowerNet.GISCommon.SymbolClass pSymbolClass)
{
```

```
System.Drawing.Font drawFont = new
System.Drawing.Font("宋体", 21);
ICharacterMarkerSymbol charMarker=
pServerContext.CreateObject("esriDisplay.CharacterMarkerSymbol") as
ICharacterMarkerSymbol;
charMarker.Font = (stdole.IFontDisp)
OLE.GetIFontDispFromFont(drawFont);
charMarker.CharacterIndex = pSymbolClass.CharacterIndex;
IRgbColor pRGB = pServerContext.CreateObject("esriDisplay.RgbColor") as IRgbColor;
pRGB.Red=pSymbolClass.SymbolColorR;
pRGB.Green=pSymbolClass.SymbolColorG;
pRGB.Blue=pSymbolClass.SymbolColorB;
charMarker.Color = pRGB as IColor;
charMarker.Size = pSymbolClass.SymbolSize;
return charMarker;
}
```

导致原因

使用 ArcGIS Engine 进行开发时，对于点要素类，是可以使用指定字体中的符号作为渲染的符号的，但是在 arcgis server 的应用中来实现这个过程，就稍微有点不同，导致的原因还是本地对象和远程对象进行了混用？对于 ao 对象，可能大家都已经有这个概念，就是一般不在本地创建对象，但是对于其它的对象，比如.net 中的对象，可能就没有太注意了，所以才导致了这个问题的出现。

解决办法

这个问题的解决办法是这样的，为了避免在程序中使用下面这句代码：

```
System.Drawing.Font drawFont = new
System.Drawing.Font(pSymbolClass.FontSymbolName, 21);
```

可以采用先在*.Style 文件中把符号都配好，然后转成*.ServerStyle 文件，然后使用下面的代码得到相应的符号：

```
private IMarkerSymbol createSymbol(IServerContext pServerContext,string index)
{
IStyleGallery pSGallery = pServerContext.CreateObject("esriDisplay.ServerStyleGallery") as IStyleGallery ;
IStyleGalleryStorage pSGStorage = pSGallery as IStyleGalleryStorage ;
pSGStorage.TargetFile = @"D:\xjw.ServerStyle";
pSGStorage.AddFile (@"D:\xjw.ServerStyle");
IEnumStyleGalleryItem pEnumGItem = pSGallery.get_Items("MarkerSymbols",@"D:\xjw.ServerStyle",index);
pEnumGItem.Reset ();
IStyleGalleryItem pSGItem = pEnumGItem.Next();
IMarkerSymbol pMSymbol = pSGItem.Item as IMarkerSymbol;
IRgbColor pRGB = pServerContext.CreateObject("esriDisplay.RgbColor") as IRgbColor;
pRGB.Red=255;
pRGB.Green =0;
pRGB.Blue=0;
pMSymbol.Color=pRGB;
pMSymbol.Size=18;
pEnumGItem=null;
```

```
pSGItem=null;
return pMSymbol;
}
```

错误描述
这个错误可能发生在.net 中的 ArcGIS Primary Interop Assemblies(PIA)。在错误信息中的 CLSID 可能会变化：
"COM object with CLSID {XXXX} is either not valid or not registered"
导致原因
在 ArcGIS 安装中选择安装.NET 的支持，PIAs 就会为所有的 ArcGIS 库而安装上。
当在尝试使用一个对象库的 PIA 时，而这个对象库根本就没有安装时，错误就会发生，因为 Com 对象不能被初始化。
例如，3DAnalyst 扩展模块没有被安装，而尝试使用 ESRI.ArcGIS.Analyst3D 命名空间中的对象，使用如下代码：

```
ESRI.ArcGIS.Analyst3D.IAnimationTrack aTrack = new ESRI.ArcGIS.Analyst3D.AnimationTrackClass();
```

一个错误就会发生：
"Error number -2147221164: COM object with CLSID {4FEDC9CB-A7BE-11D5-B2A0-00508BCDDE28} is either not valid or not registered."
解决办法
编程只能使用安装了的对象库。为了开发使用一个特殊的库，那就需要重新安装 ArcGIS 产品，确保所需的扩展模块都被安装，以及所有的.NET 的支持选项。额外的许可会被需要。
1 识别使用的对象，以及这些对象所属的命名空间。
2 识别命名空间所属的产品。例如，为了使用 ESRI.ArcGIS.Analyst3D 命名空间中的对象，在开发帮助中浏览库总体说明那一页。会有注解说明这个命名空间是有 ArcGIS Engine 产品的 3D 扩展模块，ArcGIS desktop 3D 扩展模块，以及 ArcGIS Server 的 3D 扩展模块支持。
3 重新安装产品，确保所需的扩展模块得到安装，且保证.Net 支持选项被选中。

错误描述
当使用 ArcGIS 开发者控件开发应用程序、控件拖放于窗体上的时候，出现以下错误信息：

"该控件需要一个 ESRI Designer 许可。请从工程中移除该控件。"
导致原因
出现这个信息是因为 ArcGIS Engine Developer Kit 这个产品未在机器上安装并授权。

ArcGIS Desktop 安装了以下控件：

-MapControl
-PageLayoutControl
-GlobeControl
-ReaderControl
-SceneControl
-TOCControl

-ToolbarControl

ArcGIS Desktop 许可仅允许使用 MapControl 和 PageLayoutControl 进行开发或设计应用程序。

剩下的其它控件许可给 ArcGIS Engine，并仅能在 ArcGIS Engine Developer Kit 已安装并授权时用于开发。
解决办法
购买 ArcGIS Engine Developer Kit 产品，并在机器上授权以下任意控件：

-GlobeControl
-ReaderControl
-SceneControl
-TOCControl
-ToolbarControl

有关如何使用这些控件的详细信息，以及哪些控件需要对应哪些许可，请参考 ArcGIS Developer Help。例如， 'TOCControl CoClass' 帮助主题提到，该控件只在有 ArcGIS Engine 产品时才可用。

错误描述
安装出错 1935
当安装 ArcGIS Server, ArcGIS Desktop, including ArcView, ArcEditor, ArcInfo; or ArcIMS, 都可能发生如下错误:

"Error 1935: An error occured during the installation of assembly component {303994BA-6487-47AE-AF1D-7AF6088EEBDB}. HRESULT: -2147024894."
之后点击'OK' 安装回滚退出.
导致原因
导致原因
这个错误发生在系统安装（Microsoft XML Parser 4 ）更新.这现同样的错误信息将发生在试图安装（Microsoft XML Parser 4.0）的时候； 这也可能修改，但是经常会因为其他产品的安装而影响系统文件破坏。
解决办法
步骤：
1. 尽可能停止一些后台程序,例如 adware 或 spyware. 有可能也停止 Adwatch，在机器上删除所有的 adware.；
2. 尽可能停止一些 Windows 服务，控制面板 >管理员工具 >服务；
3. 检查 Internet Explorer 6.0 的安装. 检验 Microsoft service 安装包和安装更新；
4. 在安装产品前请先安装 Microsoft XML 4.0 SP2，这有可能下载其相关联的一部分；
5. 接着安装 ESRI 软件产品在第 4 步安装成功；

错误描述
在操作系统是 Windows XP SP2、CPU 为 AMD Athlon 64 系列或 Pentium 4 和 Celeron-D 的机器上安装 ArcGIS 9.0 时，出现 1904 错误，系统无法注册一些 dll 文件。
导致原因
在 Windows XP SP2 中, DEP（数据执行保护）选项默认情况下是打开的， AMD 和 Intel 新的 CUP 系列中包含这个选项。数据执行保护是一套硬件和软件技术用于执行对内存的额外检查，以保护电脑遭受恶意代码的破坏。
解决办法
方法一：移除 Windows XP 的 SP2，然后安装 ArcGIS 9.0。

方法二：关闭 DEP（数据执行保护），安装 ArcGIS 9.0，安装完再激活数据执行保护。具体如下：
1. 以管理员权限登陆 Windows XP。
2. 鼠标邮件点击“我的电脑”，选择“属性”菜单
3. 点击“高级”页。
4. 选择“启动与恢复”下的“设置”按钮。
5. 选择默认操作系统，如下:
"Microsoft Windows XP Professional" /fastdetect /NoExecute=OptIn
6. 点击“编辑” 按钮打开 boot.ini 文件，将其中的“”/NoExecute=OptIn”改为“/NoExecute=AlwaysOff”。
7. 保存文件，重新启动计算机。
8. 安装 ArcGIS，此时应该就不会出现 1904 错误了
9. 安装完 ArcGIS 以后，将 boot.ini 文件设置改回来，以重新激活数据执行保护的安全选项。

本篇文章来源于 GIS 空间站 转载请以链接形式注明出处 网址：http://www.gissky.net/Article/561.htm

Server JAVA 开发

前言------为什么我们选择 JSF，而不是其它 Framework

在正式开讲之前，我们来探讨一下为什么要使用 JSF 来实现 SERVER ADF，这也是很多开发商跟我讨论时经常会问到的问题。

世界上有 300 种 Java 的 framework，这个数字是我几个月前听到的，现在可能已经不止这个数了。我曾经用过的 framework 包括 Structs，Hibernate，Spring。这些 framework 不但能加速开发，而且会带来全新的程序设计理念。我当时的感觉是：原来程序还可以这样写！ 当然，这些 framework 都是开源免费，你不用担心自己的良心受到谴责，也不用担心比尔盖茨半夜敲门收钱。

在这 300 多种 framework 中，ESRI 为什么选择了 JSF？各个 framework 有各个 framework 的好处。JSF 最大的优点就是：像 Swing 和 AWT 一样，JSF 是一个可以提供一组标准的、可重用的 GUI 组件的开发框架。

原因之一： **行为与表示的完全分离**。

行为和表现不分离的一个典型例子是普通的 JSP 技术，这一技术很好，因为很容易在 HTML（和类 HTML）页面中混合 Java 代码。这样我们就可以使用强大的 Java API 的功能。当然，很明显的坏处是行为表现黏合在一起。而且代码不容易重用。这让我想起刚使用 JSP 技术时，在文本编辑器下写 JSP 页面，读写数据库，并且在同一个页面上显示出来的痛苦回忆。

Model1 架构的出现使开发人员可以用 javabean 的方式进行开发，舒服多了。当然，还有更好的。

这就不得不提到著名的 MVC 结构，(MVC) model-view-controller 架构提供了一系列的设计模式，这些模式让你可以轻松地应用于程序地开发，特别是有界面程序的开发。现在又有多少的程序是没有界面的呢？

Model 包装了商业逻辑，也包含了数据持久的代码，包括存取数据库或者存取文件。Model 最好不要跟 View 有任何关系，它最好不要知道谁将显示结果，是一个 Swing 界面，还是 JSF 界面，还是普通的 JSP 界面。为了做到这一点，事先的规划要花一些时间，但是相信我，这些花的时间带来的价值将超过你的想象。如果设计得当，许多用 ArcGIS Engine 开发的代码可以轻松在 Server 中使用，不需要修改任何代码。.net 也是一样。

View 只用来显示显示逻辑，不能有任何商务逻辑。controller 是 view 和 model 的桥梁，它和 model 进行交互，并且把结果显示在 Model 中。显示页面的选择也由 controller 进行选择。我会在后面的讲座中详细解释 JSF 中的 Controller 的实现。

Structs，Spring 都实现了 MVC 的架构。但是它们都没有像 JSF 这样，提供一个事件驱动的组件对象模型。大家可以想一想，一个平常在 HTML 中使用的标签， <input type="text" name="q">，在 JSF 框架中变成了一个组件，这个组件还可以有事件，而且可以对这个事件可以进行编程。这是多么得激动人心啊！ GIS 程序需要和用户进行良好地交互，选择一个事件驱动的 framework，可以更加容易编程和实现。

这是 ESRI 选择 JSF 的重要原因之一。

原因之二: **事件容易捆绑到服务器端代码**。

JSF 提供了多个事件驱动的 GUI 控件，比如最最常见的 command button。这些 command button 会产生事件，我们有机会对事件进行编程；我在以前的讲座中曾经讲过，ArcGIS Server 的 MapControl 和 command button 是兄弟，他们都继承自 javax.faces.component.UICommand；是不是感觉很奇妙啊？继承自 UICommand 的原因是要利用 UICommand 的很多事件，对鼠标在 Map Control 上的动作，比如放大，缩小，需要产生事件，这些事件提交到相应的监听器，监听器会对事件进行处理。有关于 MapControl 的问题先提这些，在后面的讲座中我会详细解释。

举一个简单的例子吧，一个最简单不过的 fullextent 工具，如果我自己来写，可以写成这样：

```
<h:commandButton id="redisplayCommand" type="submit" value="Full Extent" actionListener="#{getMapBean.getFullMap}" />
```

这个按钮，用来缩放到全图，它的监听器是绑定到 getMapBean 的 getFullMap 方法，在该方法里面，我会将地图的大小缩放到全图范围。代码我会稍后贴上来。这就是 JSF 的事件捆绑到服务器代码的功能。

普通的一个 button，点击事件可以映射到后台 Bean 的某一个方法，在后台方法里面执行商业逻辑，进行数据存取。这个概念跟普通的 Swing 程序的事件和监听器机制十分类似，Java 程序员们看了肯定倍感亲切。注意，刚才这个 button 没有利用任何 ADF 的类，在我们的 ADF 中，所有要建大厦的材料都已经给您准备好，fullExtent 功能只需要三行代码即可实现。
事件容易捆绑到服务器端代码，这是 ESRI 选择 JSF 作为我们 ADF 底层的另外一个重要的原因。

JSF 基础

有很多开发人员问我，如果我已经有了一个 JSP 的 Application，为了使用 JSF，为了使用我们新版的 ArcGIS Server 的 Java ADF，我需要重新修改写我的程序以让它们适应整个 JSF 框架么？这是一个很有意思的问题，我可以想象出大家头皮发麻，左右为难的感觉，这也引出了今天的话题，Servlet 容器是如何对 JSF 的程序进行处理的呢？

随便找一个 JSF 的 Application，可以是我们的 ArcGIS Server 的 Sample，也可以是任何您从网上下载的 JSF 例子，打开 web.xml 文件，你就可以看到该文件里面有如下的 servlet 映射：

复制内容到剪贴板

代码:

```
<servlet>
   <servlet-name>Faces Servlet</servlet-name>
   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
  </servlet>
<servlet-mapping>
   <servlet-name>Faces Servlet</servlet-name>
   <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
```

这是 Java 程序员最熟悉不过的 Servlet 配置了，该 Servlet 映射是什么意思呢？将所有对后缀名为 jsf 的请求，都交由 javax.faces.webapp.FacesServlet 来处理。哦，另外插一句，JSF 的标准是公开的，如果你觉得这个 FacesServlet 不好，你大可以自己写一个 JSF 处理 Servlet。将所有对 jsf 的请求自己来处理。事实上，已经有人这么做了，这就是 Myfaces，可以参考 Apache 的网站。
言归正转，如何将 JSF 的应用集成到已经有的应用程序中呢，您一定已经想到了。事实上事情很简单，在访问某个 JSF 页面的时候，映射到某个目录即可。这个目录在 web.xml 进行了设置，比如下面的配置就可以把所有的对 faces 的请求，让 Face Servlet 来处理：

代码:

```
<servlet-mapping>
```

```
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

所以，如果你在根目录了放了一个 hello.jsp 页面，这个 jsp 页面使用了 JSF 框架，那么你在访问这个 jsp 页面时用 http://servername/faces/hello.jsp 就可以了，你的容器会把这样的请求提交 Face Servlet 来处理，这个时候，就轮到 Face Servlet 来大显身手了，它会忠实地执行你的请求。

JSF 的 Managed Bean 是保存在 session 中的，所以你在 JSF 页面和普通 JSP 之间切换页面的时候，不会丢失信息，反过来也一样。JSF 自己有自己的 Page Navigation 系统，但是它仍然可以跟普通的 JSP 页面互相切换，我们在后面的讲座中会讲到。

所以如果你已经存在的一个项目，想上我们的 ArcGIS Server，利用其基于 JSF 的 ADF 进行开发，也是完全可以的。当然，新的应用最好用 JSF 框架开发，开始时可能需要一点时间去学习，对于你后面会用到的强大的 JSF 的功能，这些投资是完全值得的。

欢迎大家进入美妙的 JSF 的世界。

了解了 Servlet 是如何处理你的 JSF 请求的，那么接下去我们来看看，让 JSF 程序跑起来需要什么必要条件，一个 Servlet 容器是必需的，这个一般取决于公司给我们配什么，或者客户要求什么。一般免费的好选择是 Tomcat，价格便宜量又足，我们大家都用它。当然如果项目大，经费充足，IBM 的 J2EE 容器 Websphere 和 Bea 公司的 Weblogic 也是一个好选择。选用大公司 J2EE 容器有一个好处，即使三更或者五更出了问题，你照样可以拍着桌子让他们的工程师过来帮你解决问题，如果你用我们 ESRI 的软件，你也可以这样；而用免费的软件三更出了问题只有我们自己出马了，当然，google 和百度在这个时候看在点击率的份上，还是可以拉我们一把的。除了一个好的容器，还需要一个 JSF 的实现，用 SUN 的 Reference Implementation 总是不会错的，人家是标准的制定者嘛，名字也起做“参考实现”，写来让你参考的。也可以选择功能更加强大的 Myfaces。ESRI 的 ArcGIS Server ADF 遵循标准的 JSF，所以 ADF 可以跑在 Sun 的 JSF 实现，或者 Myfaces 上。我们这次所有的教程都是基于 sun 的标准的 JSF 实现。还有其它的 JSF 实现，大家可以根据自己的爱好，择优选择之。如果你觉得都不好用，大可以参考标准自己写一个实现，除了你的老板（要投入更多的开发经费）和你的 team (要干更多的活) ，是没有人有意见的。

如果您像我一样选择了 SUN 的参考实现，那么去 SUN 的网站上去下载一个 JSF 的开发包吧，如果不想下载，随便找一个 Server Java ADF 的例子，在 WEB-INF/lib 目录里面有 JSF 的包，名为 jsf-api.jar 和 jsf-impl.jar 的两个包就是；注意那个 JSP 的标准标签库 jstl.jar，把它跟 jsf 的包放在一起，你不一定用标准标签库，但是 JSF 用到了它。

我把 JSF 的 doc 放在这里给大家下载，因为这个东西实在不好找，有人知道好的下载 URL，可以在这里贴一个。

把这些包放在一个 web application 的 lib 目录里面有，你的程序你可以使用 JSF 的强大

功能了。让我也来俗气一下，写一个简单的 hello world 作为今天的结束吧。注意在你的 web application 里面的 web.xml 配置文件添加了 Face Servlet 的声明和 URI 映射。

代码:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page c%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<meta http-equiv="Content-Type" c>

<f:view>
  <html>
    <head>
      <title>
        欢迎来到ArcGIS Server ADF 世界
      </title>

    </head>
    <body>
      <h:form id="welcomeForm">
        <h:outputText id="welcomeOutput" value="欢迎来到ArcGIS Server Java
ADF 教程!"  style="color: red;"/>
        <p>
         <h:message id="errors" for="helloInput" style="color: red"/>
        </p>
        <p>
          <h:outputLabel for="helloInput">
            <h:outputText id="helloInputLabel"
                        value="123"/>
          </h:outputLabel>
      </h:form>
    </body>
  </html>
</f:view>
```

保存后访问 Http://servername:port/faces/hello.jsp 即可，如果页面能够出来，那么恭喜您，您已经迈入了 JSF wonderland 的大门。

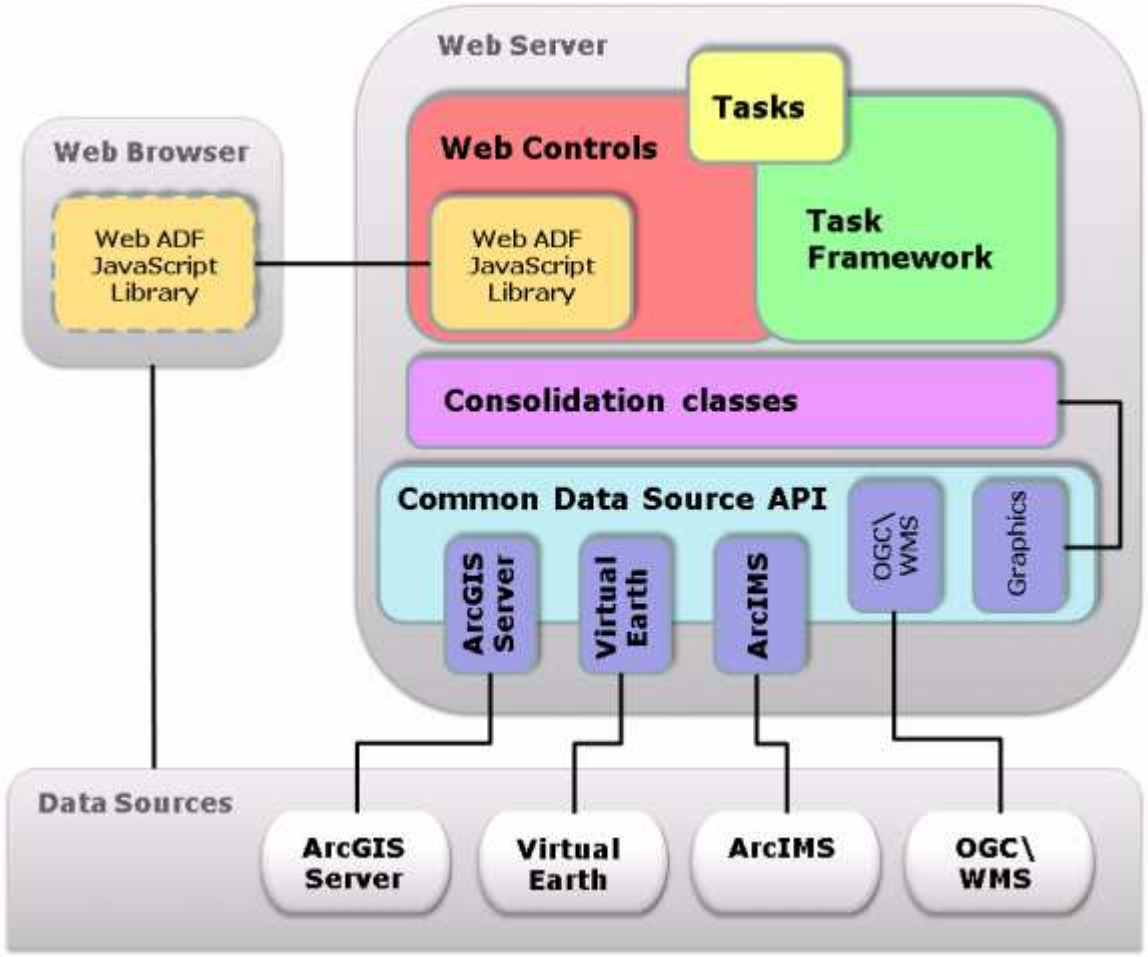
ADF 体系结构 第一部分

这两天一直在等待9.2 Engine 到货，好调试我写的例子，但是看来还得等一下，所以今天的这个讲

座将向前跨越一大步。等 Enging 到货后把前面的部分，也就是如何自己写一个 ADF 补上。

有很多人看了前面的讲座问我，怎么讲了半天，还是没有讲到如何对 server 进行开发，就像吃浙大的豆沙包，咬了一大口，发现离馅还有300里。这句话提醒了我：那么今天我们直接吃豆沙馅了。

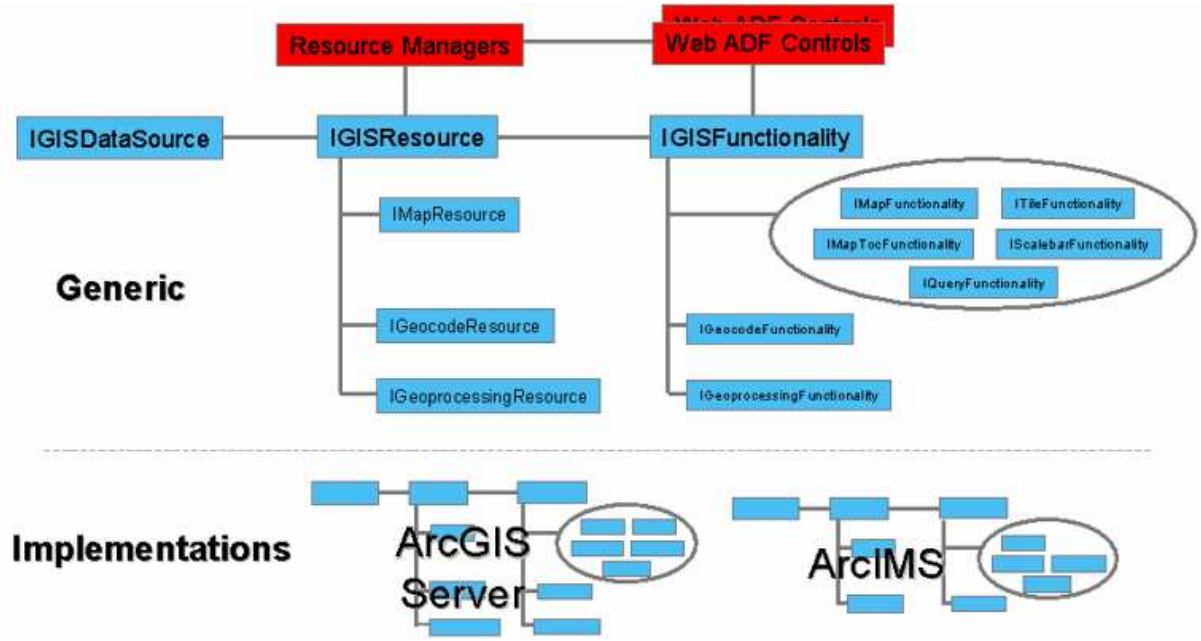
在讲一些 sample 之前，我们来看看整个 ADF 的体系架构，ADF 里面有大部分类是纯 JAVA 的。



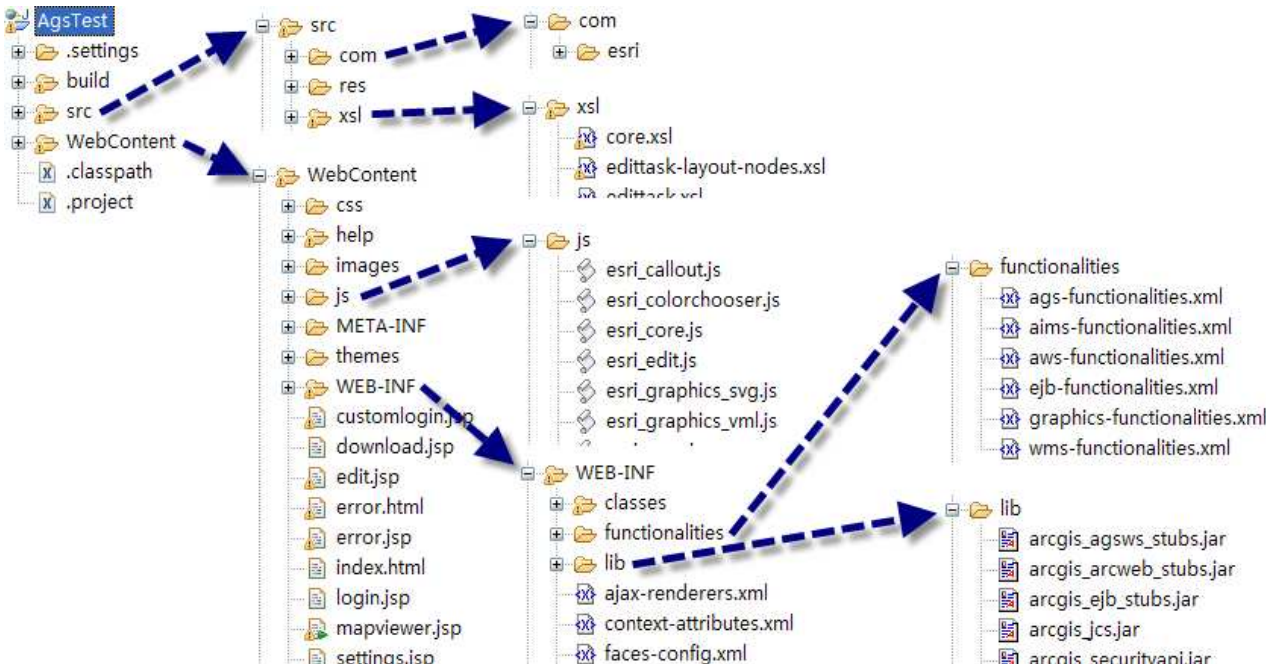
大概来讲，ADF 主要是由一些 JSF 组件、实现 GIS 相关功能的 Java 类库、支持 AJAX 的 JavaScript 脚本库咀一些帮劣文档及例子的集皿。

在上图中，首先你需要把最下面的那些“Data Sources”理解成服务，这个词语我是理解了半天，最后猜测是服务，但是又不确认，问了好几个朋友，载着，在帮助中看到这样一句（**When working with the Web ADF programmatically, a service is referred to as a data source.**）。不管是 ArcGIS Server 还是 ArcIMS 等发布的服务都可以作为 ADF 的数据源。在这些服务之上，ADF 内部通过丌同的 API（“Common Data Source API”）去使用这些服务，当然，这些 API 上层会有相应的类库（“Web ADF consolidation classes”）进行支持。在往上就是和开发者关系最直接的 ADF 的“Web Controls”（Map、TOC 等基本 GIS 功能）和“Task Framework”（完成比如查询、分析等定制的任务），事实上，我们可以把这两部分看成一个整体——Web 组件。返些 Web 组件包含了 JSF 组件以及相应的 AJAX 操作所需的 JavaScript 脚本库（“Web ADF JavaScript”）；它们都会在用户请求的时候被 ADF 通过一定的规则渲染成对应的 DHTML 后输出到浏觅器，事实上在浏览器中已已经基本看不到 ADF 的影子了，因为用户看到的页面不过是一些 HTML 和 JavaScript 脚本的组合，页面会和服务器上的某些 ADF 对象进行交互，但是这对浏览器来说基本是透明的。

在 ADF 中，我们 Common Data Source API 还有针对专门数据源的 API(Specific API),公共数据源通过抽象定义了一个框架，这也使得我们可以利用多数据源，专有 API 通过继承这个抽象框架，类似于一个插件一样，插在这个框架中。



看一下 ADF 的物理结构



ADF 是构建在 JSF 之上的，所以它严格地遵守 JSF 的规范，所有的 Bean 都放在 faces-config.xml 和 web.xml 这样的配置文件中。我们今天吃豆沙馅，所以就讲讲这两个文件里面的配置；

Faces-config.xml 文件中有如下配置：

代码:

```
<managed-bean>
```

```
<managed-bean-name>esriWebSession</managed-bean-name>
<managed-bean-class>com.esri.adf.web.data.WebSession</managed-bean-cla
ss>
<managed-bean-scope>session</managed-bean-scope>
<managed-property>
<property-name>webApplication</property-name>
<value>#{esriWebApplication}</value>
</managed-property>
< /managed-bean>
```

这里定义了一个 managed bean 指向了 WebSession 的类，这个 WebSession 类实现了一个接口，这个接口是 HttpSessionListener，看看这个接口的 Javadoc 吧；这个接口做两件事情：**每当往 Session 中存入一个对象（setAttribute）或从 Session 中删除一个对象的时候，如果这个对象实现了此监听器接口，应用服务器将会自动调用接口相应的方法**。你有机会在这两个方法里面做一些事情，最好的莫过于初始化存放在 session 中的变量。如果你一定要问我，Managed Bean 的 scope 如果是 session，那么在每个新 session 中，都会初始化这个 managed bean，调用它的 constructor，这样不是也可以实现 managed bean 地初始化么？如果你一定要这样写，也可以，但是系统的可扩展性很差啦。

在 ADF 中，如果我们写了一个类，并且希望这个类能够被 ADF 初始化，我们直接实现 WebContextInitialize 接口就可以了，ADF 会自动初始化这个类，那么 WebContextInitialize 接口是什么呢？

首先，来看看我们的核心中的核心 WebContext，Adf 的广大 Managed bean 们都团结在 WebContext 周围，错了，是团结在 WebContex 里面；**这个 WebContext 是这个 Server ADF 的容器**。Context 这个词翻译成上下文实在是不合适，我们还是叫它叫容器吧。

这个容器里面有什么？随便举个例子吧，**它有很多的 Attribute，比如有一个 WebMap**,所有容器里面的 Attribute 都需要实现一个接口，这个接口的名字叫做 **WebContextInitialize**，这个伟大的接口有两个方法：init 和 destroy。看看这两个方法，我不说你都能想到它们是用来做什么的。初始化这个 attribute，和销毁这个 attribute。看个帮助里面的例子吧：

代码:

```
public class MyAttribute implements WebContextInitialize {
  WebContext context;
  public void init(WebContext context) {
    this.context = context;
    //初始化其它资源或者变量
  }
  public void destroy() {
    context = null;
    //释放其它资源
  }
}
```

OK，这个 attribute 也将作为一个 managed bean 部署在 faces-config.xml 文件里面。这里引出的问题是，谁来调用这个 init 方法和 destroy 方法呢？对 JSF 来说，这两个只是普通的方法，JSF 不会调用这两个方法来进行资源的初始化和释放。答案是 WebContext 来调用这些初始化和释放的方法，我们可以看 WebContext 的 doc 原文：

WebContext is responsible for making the callback methods implemented by the resources and attributes at appropriate junctures of the ADF application. The callback methods are declared in the WebContextInitialize, WebContextObserver and WebLifecycle interfaces.

原来是 WebContext 负责调用这三个接口的方法，初始化和释放资源。

我们看一下 WebContext，发现它也是需要有人来初始化它的，那么 WebContext 是谁来负责的呢？是 WebSession。WebSession 里面存放了所有的 WebContext，并且保证这些 WebContext 们能够被正常的初始化和 destroy，什么时候初始化，什么时候 Destroy，当然由 WebSession 实现的，大名鼎鼎的 HttpSessionBindingListener 接口来实现，每次有一个用户访问或者用户退出时，该监听器会自动调用两个方法，用来初始化和摧毁 WebContext。另外要说一句地时：**一个 session 可以有多个 webContext**，你可以用 getWebContexts()得到所有的 WebContext。目前我们看到的所有的配置文件都只使用了一个 WebContext，多个 WebContext 会使用在什么地方呢，这个问题大家先考虑一下，我在后面的讲座中会揭晓答案。

如果有多个用户访问时，就会产生多个并发用户，我们的 Web Server 会为每个用户都分配一个 WebSession，这些 Websession 会放在哪里，答案是放在 WebApplication 中，我们打开 faces-config.xml 文件可以看到里面有一个 webApplication 的定义

代码:

```
<referenced-bean>
<referenced-bean-name>esriWebApplication</referenced-bean-name>
<referenced-bean-class>com.esri.adf.web.data.WebApplication</reference
d-bean-class>
</referenced-bean>
```

再去看 WebSesstion 的定义，有指向 esriWebApplication 的引用。

现在的问题是，是谁，初始化 WebApplication，我们挤出最原始也是最强大的武器了：ServletContextListener 出马了，看 web.xml 中下面的顶定义，

代码:

```
  <listener>

<listener-class>com.esri.adf.web.util.ADFServletContextListener</liste
ner-class>
  </listener>
```

ADFServletContextListener 实现 ServletContextListener 接口，ServletContextListener 有两个方法：contextInitialized()和 contextDestroyed(); 分别在 web application 启动的时候和结束的时候调用，你要是觉得好玩，可以写一个 listener 玩玩，我这里写一个简单的 listener 现现丑：

代码:

```
public class ScheduleListener implements ServletContextListener {
```

```
public void contextInitialized(ServletContextEvent arg0) {
System.out.println("context object inited.");
}
public void contextDestroyed(ServletContextEvent arg0) {
System.out.println("context object destroyed.");
}
}
```

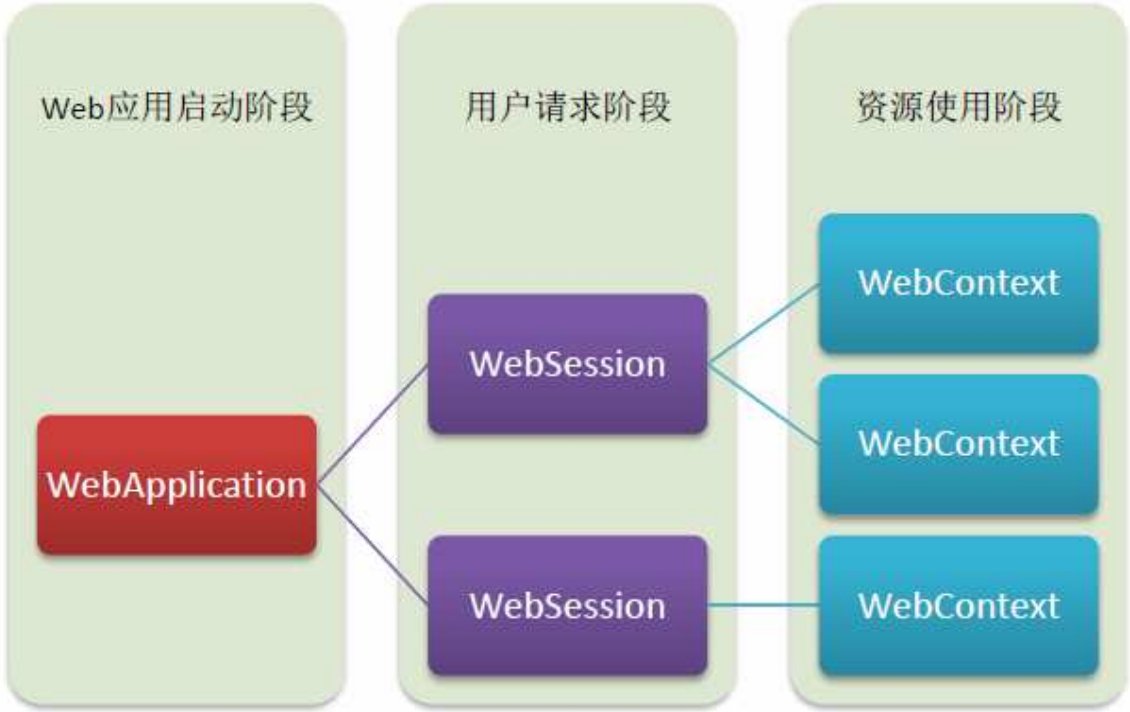
ESRI 使用了 ADFServletContextListener，它实现了 ServletContextListener 接口，用来启动和 destroy WebApplication。看看 Javadoc 的原文吧：

The ADFServletContextListener initializes a WebApplication object and sets it as an attribute on the ServletContext. When the ServletContext is destroyed, it destroys the associated WebApplication and attempts to clean up all server hosted objects in the current thread.

This class must be registered as a listener-class (in the web.xml file) for the web application.

现在大家都明白了吧，整个调用过程。Servelt 容器启动的时候，会发送启动消息给 ADFServletContextListener，ADFServletContextListener 负责创建 WebApplication，WebApplication 负责创建和存放 WebSession，WebSession 负责创建 WebContext，WebContext 负责创建所有的 Attribute，对资源的使用等等。这一下大家都满意了吧。

谨以此文，纪念在浙大吃过的一千个豆沙包。 下面看一幅图



自定义 Command

这个文档大家如果喜欢看英文，在 ESRI 的帮助里面就有。本文和 ESRI 的帮助文档类似，当然，我写的会加入自己的一些东西。

咱开发人员拿到一个开发平台，当然是要用来开发的，而且我们最好把它改得面目全非，才能显出开发人员我的水平所在。对于那些拿着模版生成一个 web gis 站点就去问客户要钱的事情，对于一个有自尊的开发人员，一般是不屑做的。

所以，我们今天就讲讲如何自定义命令和工具，先说说自定义命令吧。

命令事实上是 command，也就是平常所在 html 页面里面的 command，这些 command 里面就是命令按钮。在我们的 ArcGIS Server 的 Web ADF 里面，command 被用来做一些不用跟地图交互的工作；比如缩放到全图：你不需要跟地图做任何交互，只要一按按钮，地图就缩放到全图。其它用处，就靠大家去发挥想象了，相信大家在以后的项目中会经常用到。我们以一个例子作为说明：

代码:

```
package com.demo;
import javax.faces.event.ActionEvent;
import com.esri.adf.web.data.WebContext;
import com.esri.adf.web.data.geometry.WebExtent;
import com.esri.adf.web.faces.event.MapEvent;
import com.esri.adf.web.faces.event.MapToolAction;
public class MyFullExtent {
WebContext context;

public WebContext getContext() {
return context;
}
public void setContext(WebContext context) {
this.context = context;
}

public void setFullExtent(){
try{
WebExtent ex=context.getWebMap().getFullExtent();
context.getWebMap().setCurrentExtent(ex);
context.refresh();

}catch(Exception ex){
ex.printStackTrace();
}
```

```
}
}
```

这是无比简单的一个 command，简单到我都不好意思发出来给大家看了。注意这个类没有实现任何接口，也没有扩展任何现有的类，除了祖宗 Object。它有一个 WebContext，从上面一个讲座我们可以知道，这个类可是我们整个 Web ADF 的关键，有了它，可以得到所有的东西了。
这个类关键的方法是 setFullExtent，里面的几行代码用来将地图缩放到全图，大家看看代码应该就可以明白。

怎么进行部署呢？就像普通的 managed bean 一样进行部署就可以了，打开 JSF Application 的心脏 faces-config.xml 文件，在里面添加如下配置：

代码:

```
<managed-bean>
       <managed-bean-name>myFullExtent</managed-bean-name>
       <managed-bean-class>com.demo. MyFullExtent </managed-bean-class>
       <managed-bean-scope>session</managed-bean-scope>
       <managed-property>
         <property-name>context</property-name>
         <value>#{mapContext}</value>
       </managed-property>
  </managed-bean>
```

注意我们把 web context 作为一个属性，用来初始化 MyFullExtent，#{mapContext}指向了一个在 faces-config.xml 文件中定义的另外一个 Bean，这个 bean 我们上次讲座已经详细解释过了。这里说一句废话，算是提醒一下，有请求过来的时候，先实例化这个类，然后再设置属性，有的工程师想在实例化里面进行一些操作，就必要用 webContext，会报告空指针错误，因为这个时候 webContext 属性还没有被设置。
类写好了，也配置好了，怎么用呢，再给个例子：

代码:

```
<a:command id="fullExtent" action="#{myFullExtent. setFullExtent }"
toolText="全图显示" />
```

自定义 command 就这样完成了。

自定义 Tools 开发

今天进行整理，发现自定义 tools 的教程忘记写了。今天把这个补全吧。

什么是 tools，tools 就是工具，它跟 command 的区别就是 tools 是要跟地图进行交互后再执行某个命令，

而 command 是直接执行某个命令。最简单的 tools 和 command 的对比是放大是一个工具，而全图显示就是一个命令。

ESRI 已经包含的工具包括：放大（zoom in），缩小（zoom out），平移（Pan）。

前面我们都是先说服务器端的处理代码，现在先来看看客户端的处理。跟地图交互，就会产生怎么交互的问题。拉框放大的时候是在地图上画的是一个长方形，测距的时候在地图上画的是一条线。如何控制这些客户端的功能呢？

查看 com.esri.adf.web.faces.event.MapEvnet 的帮助，里面详细说明了如何把客户端的操作如何和服务器端的代码建立联系。已有的客户端支持的操作包括：
EsriMapCircle
EsriMapContinuousPan
EsriMapLine
EsriMapOval
EsriMapPan
EsriMapPoint
EsriMapPolygon
EsriMapPolyline
EsriMapRectangle

如果我们要自己写一个工具，我们可以新建一个普通的类，这个类可以实现 MapToolAction 接口，也可以不实现任何接口。我们先来看一个实现接口的类。

```
public class CJZoomInTool implements MapToolAction {

        public void execute(MapEvent event)  {

WebContext ctx = event.getWebContext();
   WebGeometry screenGeom = event.getWebGeometry();
   WebGeometry mapGeom = screen.toMapGeometry(ctx.getWebMap());



}

}
```

我们可以看到，通过 MapEvent 的 getWebGeometry 方法可以得到客户端所画的多边形，此时得到的是屏幕坐标，然后再通过 toMapGeometry 方法转换为地图坐标。
MapEvent 的 getWebGeometry 方法事实上是指向到 ClientActionArgs 的 getWebGeometry 方法，你可以用下面的代码得到另外一个 WebGeometry 进行比较：

```
ClientActionArgs clientarg=event.getClientActionArgs();
WebGeometry anotherGeom=clientarg.getWebGeometry();
```

我们还可以用 ClientActionArcgs 的 getRequestParameter 查看传送过来的参数是什么，在客户端指定长方形的情况下，得到的 RequestParamerter 里面有如下参数：

Map0_maxx:246
Map0_miny:147
Map0_minx:199
Map0_maxy:199

你一定想看看客户端是多边形的情况下，参数是什么情况,我们满足一下您的愿望：

Map0_coords:220:168|264:220|296:186|292:136|258:125

出现了以|分隔的参数集合，这个意思大家都明白了吧，这就是多边形的情况。

ClientActionArgs 是一个基类，它的所有的子类对应于每一个客户端的操作。每个积累需要实现它的抽象方法。最重要的两个方法是 init()和 getWebGeometry，init 从 request 里面得到所有的参数，getWebGeometry 从根据不同的参数，构建不同的 geometry 返回。ADF 会根据不同的客户端操作，创建不同的 ClientActionArgs 类。如果你一定要问我 ADF 是如何根据不同的客户端操作创建 ClientActionArgs 的，我们可以看到 ClientActionArgs 有一个静态方法：

getClientActionArgs(java.lang.String clientAction, java.util.Map requestParameters, java.lang.String controlId)

ADF 就是用这个静态方法，根据不同的 clientAction 可以得到的 ClientActionArgs。

这一下，把来龙去脉都讲了，既让大家看看整个客户端和服务器端操作的来龙去脉，也为大家增加自己的客户端操作热个身。

当然，自定义一个 tool 并不需要这么多的知识。事实上，自定义一个 tools 非常得简单，我们来看看帮助中的一段代码，计算选中了多少个要素。

```
package com.cj.ucdemo;

import java.rmi.RemoteException;

import com.esri.adf.web.ags.ADFAGSException;
import com.esri.adf.web.ags.data.AGSMapResource;
import com.esri.adf.web.data.WebContext;
import com.esri.adf.web.data.geometry.WebExtent;
import com.esri.adf.web.faces.event.MapEvent;
import com.esri.adf.web.faces.event.MapToolAction;
import com.esri.arcgisws.EnvelopeN;
import com.esri.arcgisws.EsriSearchOrder;
import com.esri.arcgisws.EsriSpatialRelEnum;
import com.esri.arcgisws.MapServerPort;
import com.esri.arcgisws.SpatialFilter;
```

```
public class CountFeatureTool implements MapToolAction {

    WebContext context=null;
    int countedFeatures;

    private void countFeatures(WebExtent extent){

        //Get the MapServerPort so we can execute methods through ArcGIS Server API

        AGSMapResource agsMap = ((AGSMapResource)context.getResources().get("ags1"));
        MapServerPort mapServer = agsMap.getMapServer();
        //Make a new envelope from the web extent
         EnvelopeN env = new EnvelopeN(extent.getMinX(), extent.getMinY(), extent.getMaxX(),
extent.getMaxY(),
           null, null, null, null, null);

        //Setup a spatial filter for an Intersection relationship
        SpatialFilter spatialFilter = new SpatialFilter();
        spatialFilter.setSpatialRel(EsriSpatialRelEnum.esriSpatialRelIntersects);
        spatialFilter.setWhereClause("");
        spatialFilter.setSearchOrder(EsriSearchOrder.esriSearchOrderSpatial);
        spatialFilter.setSpatialRelDescription("");
        spatialFilter.setGeometryFieldName("");
        //Set the envelope as the geometry
        spatialFilter.setFilterGeometry(env);

        //MapServer::queryFeatureCount() executes on the server and can throw a RemoteException
        try{
          //Count features in 4th layer which intersect with the envelope
          int layerId = 1;
          this.countedFeatures =
            mapServer.queryFeatureCount(mapServer.getDefaultMapName(), layerId, spatialFilter);

          System.out.println("你选择了 "+countedFeatures+" 要素");

        }catch(RemoteException rme){
          //Rethrow this as ADFAGSException so that it can participate in the exception framework
          throw new ADFAGSException("Could not execute MapServer::queryFeatureCount()",rme);
        }
       }


    public void execute(MapEvent arg0) throws Exception {
        // TODO Auto-generated method stub
```

```
            try{
                this.context=arg0.getWebContext();
            WebExtent ex=(WebExtent)arg0.getWebGeometry();
            ex=(WebExtent)ex.toMapGeometry(arg0.getWebContext().getWebMap());
            this.countFeatures(ex);
            }catch(Exception ex){
                ex.printStackTrace();
            }

        }

}
```

大家注意这里使用了很多 com.esri.arcgisws package 里面的类，这个 package 是通过 axis 调用远程 webservice 的方式去调用服务器端的方法的。我们会在后续的讲座中说明。编译这个类后，直接在 jsp 页面里面用下面的代码调用即可：

```
<a:tool id="countFeature" defaultImage="images/selection.gif" hoverImage="images/selectionU.gif" selectedImage="images/selectionD.gif" clientAction="EsriMapRectangle" serverAction="com.cj.ucdemo.CountFeatureTool" clientPostBack="true"/>
```

实现后台 JSF Bean 和前台 Javascript 的联动

很多开发人员都梦寐以求的方案，我在论坛里面也看到很多人提出这样的问题，今天集中对这些问题做一个解答，也是作为 Server JAVA 讲座的一个部分。另外，要说明的是，今天的讲座不包括 AJAX 技术。AJAX 技术会另开一个讲座。

先以一个问题开始，如果我们的 webapplication 中对 geodatabase 进行了编辑，大家都知道，这个时候需要一个提交编辑并保存，或者提交编辑并取消的过程，大家都用过 ArcMAP，如果没有用过 ArcMAP，word 总是用过的，编辑过的东西，一关闭，肯定会给你警告。这个问题在桌面程序中简直就不是问题，在 web 上，我们倒是可以来说一说了，因为 server 和你的 browser 可能相距几万公里，你这边要关闭 browser，在美国的服务器怎么能够知道你要关闭呢？服务器上运行的是 JSF 的 Bean，而客户端运行的是 Javascript，怎么让 Javascript 去感知后台的情况，给用户一个提示，这样就会很好地显示我们程序的友好性，对建设以人为本的和谐社会也大有好处。那么我们来看看如何做这个事情。

Browser 关闭时会产生 onUnload 事件，有了这个事件，使我们有机会做一些事情，比如下面的语句，显示一个信息

代码:

```
<body onUnload="javascript: alert('你要关闭我么？')">
```

在 body 中添加了这一句后，是不是已经看到了那个温柔的问句呢？如果是，那么你的 Browser 是支持这个 onUnload 事件的。下面来看这段代码：

代码:

```
function checkTime(){
  var isModified=<c:out value="${editBean.edited }"/>;
   if(isModified ==true)
   alert("更新没有保存");
}
```

这个代码很有意思，其它的好说，<c:out value="${editBean.edited }"/>不好理解，这是 JSTL，如果没有见过 JSTL，google 或者 baidu 一下看看吧。editBean 是一个标准的 JSF 的 bean，它在 faces-config.xml 文件中部署，它有一个属性和属性的 getter,setter 方法，如下：

代码:

```
private boolean edited=false;

public boolean isEdited() {
return edited;
}
public void setEdited(boolean edited) {
this.edited = edited;
}
```

你在开始编辑的时候，设置 edited 为 true 就可以了。JSTL 能够用过<c:out>标签访问到这个 bean 的值。这样，就实现了前台 Javascript 和后台 JSF bean 的联动。

是不是看了这个文章，马上摩拳擦掌，准备去修改您的程序呢？

AJAX 基础

咱也来跟跟潮流，讲一讲 AJAX 和 JSF 如何结合。我相信，这也是将来 SERVER JAVA 的开发人员都会问的一个问题。我们来看看如何能够让 AJAX 能够在服务器上和客户端跑起来。要应用 AJAX 的条件是：

1、 一个支持 javascript 的浏览器，当然，现在主流的浏览器都是支持 Javascript 的。

2、 浏览器必须支持 XMLHTTP 或者 XMLHttpRequest 对象。

3、 能够以 XML 发送响应的服务器端。服务器端可以有很多的技术来实现，不管你是用 ASP，JAVA，都可以实现。当然，本文要讨论的局限于 JSF 技术。

对于熟悉大多数服务器端程序编写的工程师，比如我，Javascript 完全是另外一个世界。但是了解 Javascript，将使你的程序更加强大，开发起来更加容易。一个很有意思的功能是如果用户在编辑地图的时候，编辑到中途，忘记保存了，这提醒我们可以做一个比较有意思的功能，自动保存。在 google mail 中，你是不是已经感觉到过这个自动保存的功能了呢？反正我当时看到的时候吓了一跳。以一个服务器端程序员的思路去思考，会觉得不可思议。事实上这个功能就可以用 Ajax 技术来实现，用一个定时器，每隔5分钟自动发送一个保存的请求就可以实现这样的功能。

再看一个自动查找的例子，比如一个输入一个用户后自动出来和该用户有关的信息，可以通过这样的方法来实现：

代码:

```
<input type="text" id="query" size="20" onkeyup="search('query');">
```

Search('query')方法可以发送一个 Ajax 请求到客户端，进行查询后，将结果以 XML 请求返回，这样就不需要刷新整个页面而进行部分更新。就这么一个小小的技术，让用户在继续 Web 浏览的时候感觉更好，也成为了各个网站争相模仿的对象，一些开发人员也发现了宝贝一样言必称 Ajax，甚至成为项目选型的关键，这大概也是最初的开发者想象不到的。
来看看 Javascript 如何来发送一个 XMLHttpRequest 请求的。来看一下下面的代码：

代码:

```
if (window.XMLHttpRequest) {
   req = new XMLHttpRequest();
}
else if (window.ActiveXObject) {
   req = new ActiveXObject("Microsoft.XMLHTTP");
}
```

这是再简单不过的实例化 XMLHttpRequest 的一段代码，实例化一个之后，你就可以像操作一个普通的对象一样操作它，它有很多的方法，比如下面的方法是建立和服务器端的链接：

代码:

```
req.open("GET", url, true);
```

第一个参数是 HTTP 请求的方式：”Get”或者”Post”，第二个参数是服务器的 URL，第三个参数是是否进行非同步的请求。大家如果已经仔细研究了我们 ESRI 的工具标签，你会发现

这些标签里面有一个 clientPostBack 属性可以设置是否进行异步请求。我们可以知道，Web ADF 在后面肯定用了 open 方法的第三个参数来进行。要小心的是，如果你要设置 clientPostBack 为 true，你的 javascript 代码可能需要重新构建一下。不然这些代码可能不会被执行。因为异步请求更新的是部分页面。

异步请求的意思是在请求时浏览器可以做其它的事情，那么如何得知服务器端的处理已经结束，可以更新页面上的数据了呢？我们来看看 XMLHttpRequest 对象的方法

代码:

```
req.onreadystatechange = processXMLResponse;
```

当服务器端的处理结束时，就可以调用 processXMLResponse 方法，我们就可以通过对这个方法进行编程来处理请求，当然可以是任何的名字的方法，只要将方法注册到 onreadystatechange 方法即可。

XMLHttpRequest 初始化完成后就可以发送到请求到服务器端了,用一个很简单的方法即可：

代码:

```
req.send(null);
```

当然，一般 get 用的实在太少，我们最关心的是 Post，看看 Post 应该怎么做：

代码:

```
req.open("POST", "/agsviewer", true);
req.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded";
req.send("__ADFPostBack__=true&formId=f"+"&mapId=Map0&ajaxdemo=ajaxdem
o ");
```

Post 可以发送一些数据到服务器端，这正是我们所需要的。我们来继续看看一个请求当前比例尺的代码：

代码:

```
function updateInfoResponse(xmlHttp) {
```

```
  if (xmlHttp != null && xmlHttp.readyState == 4 && xmlHttp.status == 200)
{
    var xml = xmlHttp.responseXML;
   document.getElementById("scale").value        =        "1:"        +
xml.getElementsByTagName("scale").item(0).firstChild.nodeValue;
  }
}
```

看一下判断了(xmlHttp != null && xmlHttp.readyState == 4 && xmlHttp.status == 200，判断 4 是表示处理已经结束，判断 200 表示 HTTP 请求正常，相信大家都见过 404 的 HTTP 请求，令人讨厌的“page not found”。
判断结束后，就用 xml.getElementsByTagName("scale").item(0).firstChild.nodeValue;语句把比例尺取过来。

相信大家都大致知道 AJAX 是怎么回事了，下一次我会写一个简单的例子。

IBM 网站上有一篇很好的介绍 AJAX 的文章，大家可以去看看：
http://www-128.ibm.com/developerworks/cn/xml/wa-ajaxintro1.html

ADF 中的 Ajax 函数

ESRI 为了使 Server ADF 能够支持 Ajax，做了大量的工作，Sever ADF 中有很多可以利用的代码，我们来看看：
代码1：创建 XMLHttpRequest，可以看一下下面的代码：

代码:

```
this.createXmlHttpObject = function() {
   if (this.isIE) {
     try {
       return new ActiveXObject("Msxml2.XMLHTTP");
     }
     catch (exception) {
       return new ActiveXObject("Microsoft.XMLHTTP");
     }
   }
   return new XMLHttpRequest();
  }
```

这个代码写得很妙吧，前面两个处理 IE 浏览器，最后一个支持 Firefox，XMLHttpRequest 是 Firefox 的。

发送请求，用如下命令：

代码:

```
EsriUtils.sendAjaxRequest(url,      params,      true,      function()
{ updateInfoResponse(xmlHttp); });
```

解释一下，怎么得到这个 url，ESRI 也提供了办法：

代码:

```
var url = EsriUtils.getServerUrl(formId);
```

getServerUrl 的代码如下：

代码:

```
this.getServerUrl = function(fId) {
   if (document.forms[fId].action.indexOf(";jsessionid=") != -1)
     return document.forms[fId].action;
   else if (this.getJSessionId())
     return    document.forms[fId].action    +    ";jsessionid="    +
this.getJSessionId();
   else
     return document.forms[fId].action;
  }
```

嗯，url 已经解释了，我们看看第二个参数 params，这个参数包含了我们需要传递到服务器端的参数。看一下源代码就可以知道，xh.send(params);将这些参数发送到服务器端。

doGet 参数就比较简单了，用来说明这个请求是 get 还是 post。看看源代码，发送的方法是不一样的，如果是 get 方法，那些参数用?进行连接，传送到服务器端。它返回的仍然是 XMLHttpRequest 对象。

Callback 函数是我们自己写的函数，用来在服务器数据处理完成后，处理服务器端传送过来的数据。大家看着是不是一目了然了呢？

sendAjaxRequest 的源代码

代码:

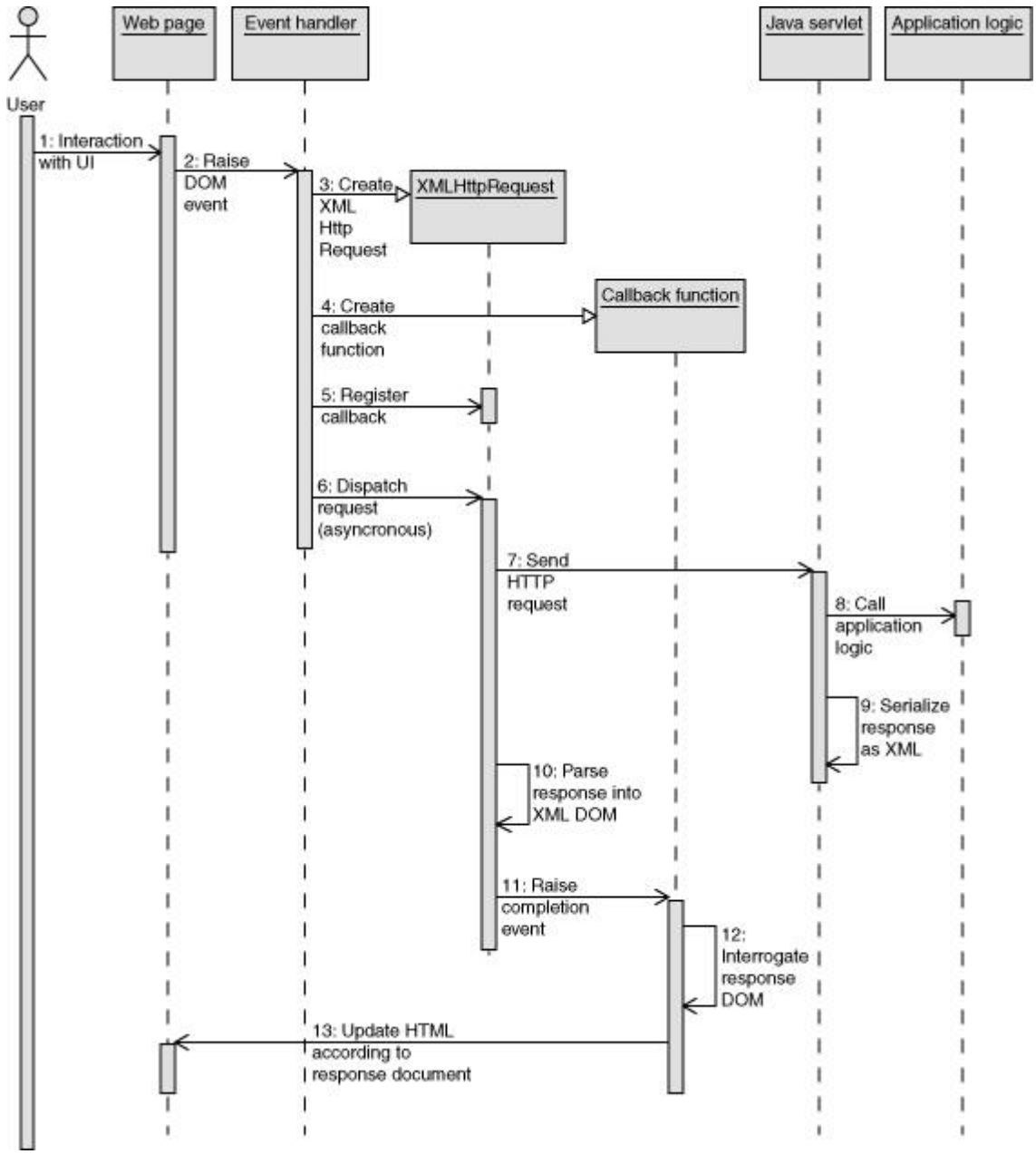
```
this.sendAjaxRequest = function(url, params, doGet, callback) {
   try {
     var xh = this.createXmlHttpObject();
     xh.onreadystatechange = callback;
     if (doGet) {
       xh.open("GET", url + "?" + params, true);
       xh.send(null);
```

```
      }
      else {
        xh.open("POST", url, true);
        xh.send(params);
      }
      return xh;
    }
    catch (exception) {
      return null;
    }
  }
}
```

有了这几个强大的函数，我们就可以利用 ESRI 的成果，来进行 Ajax 的开发了。说了半天客户端的东西，下一篇我们来讲服务器端的处理，服务器端的处理需要比较大的篇幅来说。

如何在用 JSF 在服务器端处理 Ajax 请求

服务器端的处理才是整个事件的 Key，那么我们来看看整个请求，回复的过程，我们通过下面的图看看我们服务器端的代码怎么样才可以 plug in 进去：



看了这个图您肯定已经明白了，我们有机会得到客户端发送过来的 HTTP 请求，得到这个请求，得到请求里面的参数，我们就可以捋起袖子，大干一番了。
我们用什么方式去处理呢，最简单的一个方法是自己写一个 servlet，在 servlet 的 Dopost 方法里面，写上处理这个请求的代码，并且把结果以 XML 的方式返回回去，那么我们这里用一个简单的例子：

代码:

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
                    throws java.io.IOException {

  String action = req.getParameter("action");
  String item = req.getParameter("item");
```

```
  if ((action != null)&&(item != null)) {

    // Add or remove items from the Cart
    if ("add".equals(action)) {
      cart.addItem(item);

    } else if ("remove".equals(action)) {
      cart.removeItems(item);

    }
  }

  // Serialize the Cart's state to XML
  String cartXml = cart.toXml();

  // Write XML to response.
  res.setContentType("application/xml");
  res.getWriter().write(cartXml);
}
```

这对于了解服务器端编程的人来说，是最简单不过的了。当然，我们用了 JSF 框架，肯定不会再通过 Servlet 技术去做这件事情。我们这里用 JSF 的 PhaseListener 进行处理。什么是 PhaseListener，我们来看一下 PhaseListener 的 JavaDoc：

代码:

```
public interface PhaseListener


extends java.util.EventListener, java.io.Serializable


An interface implemented by objects that wish to be notified at the beginning
and ending of processing for each standard phase of the request processing
lifecycle.
```

原来，这个接口捕捉到每个 Request 的生命周期，在请求开始处理和请求处理结束时我们可以插入我们自己的代码。这个接口的两个最重要的方法如下：

代码:

```
void afterPhase(javax.faces.event.PhaseEvent event)
```

```
        Handle a notification that the processing for a particular phase
has just been completed.  void beforePhase(javax.faces.event.PhaseEvent
event)
        Handle a notification that the processing for a particular phase
of the request processing lifecycle is about to begin.
```

处理结束和处理开始两个方法，我们可以在这两个方法里面写入我们自己的代码，对用户过来的请求进行处理。

比如，我们可以在 afterPhase 方法里面，插入下面的代码对数据进行处理：

代码:

```
FacesContext facesContext = phaseEvent.getFacesContext();
   ExternalContext externalContext = facesContext.getExternalContext();
Map paramMap = externalContext.getRequestParameterMap();
```

通过 paramMap 里面可以得到所有的参数，当然，我们并不是对每一个请求都会进行处理，所以，在客户端 Javascript 发送 Ajax 请求得时候，可以添加一个自定义的参数，比如我们前面文档中的 ajaxdemo=ajaxdemo 参数，这样我们可以判断一下当前是不是我们要处理的请求：

代码:

```
private final static String AJAX_DEMO = "ajaxdemo";

if (! AJAX_DEMO.equals(paramMap.get(AJAX_DEMO)))
     return;
```

接下去的代码，就可以处理我们的各种各样的请求了，我们在这里要举的例子是如何得到当前地图的比例尺。下次讲座再深入探讨吧。

用 Ajax 得到地图比例尺(客户端代码)

上个讲座我们讲到，服务器端已经得到了当前地图的比例尺，我们需要在客户端做一个处理，把这个比例尺数据给用户看。我们在页面上放一个文本框，用来显示比例尺数据，如下面的代码所示：

代码:

```
 <tr>
   <td>Scale</td>
   <td><input type="text" id="scale" size="36" /></td>
 </tr>
```

我们如何将服务器传过来的数据更新这个文本框呢？来看看前面讲座中提到的往服务器端发送请求时候的那个回调函数：

var xmlHttp = EsriUtils.sendAjaxRequest(url, params, true, function() { updateInfoResponse(xmlHttp); });

我们把回调函数的代码也贴在这里：

代码:

```
function updateInfoResponse(xmlHttp) {
  if (xmlHttp != null && xmlHttp.readyState == 4 && xmlHttp.status == 200)
{

var xml = xmlHttp.responseXML;
    var
scale=xml.getElementsByTagName("scale").item(0).firstChild.nodeValue;
    document.getElementById("scale").value = "1:" + scale

  }
}
```

这个代码相比大家都能看明白了，先判断服务器端是不是已经处理结束，并且服务器的状态一切正常，然后得到服务器返回的 xml，通过 getElementsByTagName 把里面包含着的比例尺数据提取出来，组合中我们熟悉的文本，然后把这个文本填充到 ID 为 scale 的文本框中。

我们来总结一下，我们发送了一个请求，这个请求包含了 MapID 参数，和一个标志参数 ajaxdemo，MapID 参数可以让服务器得到 MapControl 的实例和 WebMap，进而得到比例尺。Ajaxdemo 让服务器端的处理程序可以正确的辨识这个请求是我们这个 PhaseListener 需要处理的请求，处理结束后，返回 XML 到客户端。客户端的回调函数处理 XML 文档，提取出比例尺信息，写入到页面的文本框中。撩起 Ajax 神秘的面纱后，发现原来 Ajax 不过如此，老酒装新瓶，重新包装了一下。
讲到这里，大家心中想必还有两个疑点，第一个是说了半天，我们是怎么触发这个请求的呢？通过定时器触发么？不是。我们是通过地图的变化进行触发的，这样是最最合理的，每当地图更新了，客户端会自动发送一个请求，要求地图变化后的比例尺。非常合理，但是怎么进行设置呢？我们经常说，我们的控件是支持 Ajax 的，如何支持？我们支持通过监听器的方式，监听地图的变化，如下面的代码所示：

代码:

```
var map = EsriControls.maps["Map0"];
  map.addUpdateListener("request", updateInfoRequest);
```

这样，每次地图更新时，都会触发 updateInfoRequest 函数，该函数用来向服务器端发送请求。

另外一个疑问是 Phaselistener 怎么设置，看一下 JavaDoc 可以知道，我们可以在 faces-config.xml 文件中，用下面的语句进行设置：

代码:

```
<lifecycle>
   <phase-listener>demo.AjaxDemoPhaseListener</phase-listener>
 </lifecycle>
```

恩，已经完美了。

Ajax 的讲座到此告一段落了，欢迎大家讨论。

利用 Ajax 得到当前地图的比例尺（服务器端）

现在我们进入实战阶段，看看我们怎么可以在服务器端得到当前地图的范围，这个代码中你将学到如何从 faces Context 里面得到当前 form 中的控件，以及如何利用 ESRI 提高的工具，简化我们响应 AJAX 请求的过程：

代码:

```
UIComponent form = facesContext.getViewRoot().findComponent((String)
paramMap.get(FORMID));
```

首先我们得到页面上的 form，form 是一个 container，通过它我们可以得到 form 中的其它组件，比如我们的地图控件：

代码:

```
<a:map id="Map0" value="#{mapContext.webMap}" width="600" height="600"
/>
```

我们可以用下面的方法得到这个 Mapcontrol 和 WebMap：

代码:

```
   MapControl mc = (MapControl) form.findComponent((String)
paramMap.get(MAPID));
  WebMap wm = mc.getWebMap();
```

得到了这个 WebMap，哈哈，可以得到 Everything 了，因为你可以从 WebMap 中得到 WebContext。这个时候，想得到那个比例尺，是再简单也不过的事情了，用下面的代码即可：

代码:

```
wm.getMapScale();
```

得到了比例尺了，接下去怎么办呢？把它返回到客户端，我们的 Ajax 客户端认的是 XML 的流，咋办呢？最笨的办法当然是是一个一个 String 的组装成 XML，然后调用 Response 的 Write 写回到客户端，比如下面的代码：

代码:

```
public String toXml() {
    StringBuffer xml = new StringBuffer();
    xml.append("<?xml version=\"1.0\"?>\n");
```

```
   xml.append(“<scale>”);
   xml.append(“<value>100</value>”);
  xml.append("</scale>\n");
   return xml.toString();
 }
```

生成了 XML 代码，然后用下面的代码把结果返回给客户端：

代码:

```
Response res=(Response)externalContext.getResponse();
res.setContentType("application/xml");
res.getWriter().write(scaleXml);
```

当然，各位看官想来都是 JAVA 高手，对于这样的写法实在感觉不舒服，正待想办法用标准的 XML 工具来写时，你会发现，ESRI 已经为您想到了，写好了几个工具类来供你调用了。我们来看看怎么用这些工具：用下面的代码创建一个标准的 XML 文档：

代码:

```
Document doc = XMLUtil.newDocument();
```

用下面的代码新建一个 element：

代码:

```
Element responseElement = XMLUtil.createElement(doc, "response", null,
null);
```

用下面的代码将我们的比例尺数据写到刚才新建的 Element 里面：

代码:

```
XMLUtil.createElement("scale",       String.valueOf(wm.getMapScale()),
responseElement);
```

然后用下面的语句将 XML 文档写入到客户端：

代码:

```
AJAXUtil.writeResponse(facesContext, doc);
```

很可惜的是，ESRI 这些类的代码是不公开的，不过你可以猜一猜，这些方法里面到底发生了什么事情。OK，皮球又被提到了客户端了，我们下一个讲座来看看，如何解析服务器端返回的 XML 文档，并且更新页面上的输入框。

扩展 Tasks 框架(一)

Due to popular demands，今天讲一讲 Tasks 框架的扩展。这是 tasks 系列讲座1。

Tasks 框架是一个很有意思的东西。如果你对 Tasks 框架还没有概念，你可以参考一下 web map application 那个 Sample，左边浮动的放大工具，查询工具都是用 Tasks 框架构成。有人会问，既然已经有了 command 和 tools，为什么还需要 tasks 呢？事实上 tasks 的目的是不同的；我们可以看一下 web map application，可以看到，相同类型的 button 和 toos 组合在一起，组成一个逻辑单元，这就是 tasks 的选择条件。

我们来看看如何编写 Tasks，编写 Tasks 是非常有意思的事情。Tasks 跟 tools 一样，也是一个普通的 Java 类即可，这个 Java 类可以从 Object 直接继承。请看下面的 Task 的例子：

代码:

```
public class JSTasks

{
public void doSomething()

 {
System.out.println("Task button clicked.");
  }
}
```

这个 tasks 实在是简单吧，看看怎么部署上去，我们在 xml 文件中做如下定义：

代码:

```
  <managed-bean>
       <managed-bean-name>jsTask</managed-bean-name>
       <managed-bean-class>com.cj.ucdemo.JSTasks</managed-bean-class>
       <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
```

这个配置文件也非常简单，那么看来所有的诀窍在与如何在 JSP 页面里面使用它。我们来看看：

代码:

```
<a:task mapId="Map0" value="#{jsTask}"   rendered="true" />
```

添加一个 task 标签，把这个标签的 mapId 值指向地图控件"Map0"，讲它的值指向我们刚才定义的 Managed Bean。访问一下，大家看看下面的浮动窗口结果：



是不是很令人惊奇啊？事实上我们这个 Bean 中只有一个方法，但是 Tasks 框架聪明地把这个方法名给提取出来，作为 button 的名字。你点击一下这个按钮，在后台就可以看到执行了这个方法，打印出了一条信息。简单的说：方法就是按钮!!

是不是准备马上试试呢？

Task 框架扩展之 Taskinfo

有了上个 Task 的基础，我们来看其它的 Task 就驾轻就熟了。事实上，整个 Task 就是一个类，Task 的 Caption 就是这个类的类名。里面的每一个 Public 方法都是 Task 浮动条上的一个按钮。那么参数怎么办呢？我们来加一个参数，并且加上一个它的 get 方法和 set 方法：

代码:

```
String parameter;

public String getParameter() {

return parameter;

}

public void setParameter(String parameter) {

this.parameter = parameter;

}
```

看看我们的 task 工具条发生了什么变化，Task 工具条如下图所示：



可以看到，多了一个 Parameter 的输入框，前面还有一个 Label，很有意思吧。那个 DoAnything 是我加的另外一个方法。Task 框架的扩展已经很明显了吧：把相同的功能集合在一个类里面，这个类可以接受参数。例如我们可以利用这个 task 进行 buffer 操作，这个输入框就可以用来输入 buffer 的距离。

看到这里，大家肯定想到了另外的几个问题，比如：怎么把这些参数，按钮的名字修改成中文，上次给人培训，有人说：用中文的变量名？也是一个办法，但是考虑的 JDK 对中文变量的处理，似乎有问题。还有其它的问题包括怎么跟地图进行交互？怎么样设置客户端的动作（画多边形还是画 Polyline）。我们当然有另外的方法。

使用 Taksinfo 建立对 Task 的描述，所有的 Taskinfo 都需要继承自 SimpleTaskInfo 类。大家可以打开 doc 看看 SimpleTaskInfo 的子类有那些。事实上，那些子类都是已经存在的 task 的 taskinfo，如果你需要做汉化，继承这些类并且改写其中的几个方法即可。

我们来看看我们自己的类需要改写 SimpleTaskInfo 的哪几个方法：
TaskActionDescriptorModel[]getActionDescriptors();

此方法用来修改 command 按钮的描述；command 的意思是不需要和地图进行交互而是直接在服务器端执行一个命令。

TaskParamDescriptorModel[]getParamDescriptors()

此方法修改参数的描述；

TaskToolDescriptorModel[]getToolDescriptors()；

此方法修改工具的描述，工具的意思是需要和地图进行交互；

那么我们来写一个类，直接说明如何使用这几个方法，修改 task 的表现:

代码:

```
public TaskDescriptor getTaskDescriptor() {

TaskDescriptor td=new TaskDescriptor(JSTasks.class);
```

```
td.setDisplayName("我的任务");

return td;

}
```

此方法修改了 task 的标题，将此类作为一个 Managed-bean 添加到 faces-config.xml 文件中，并且在.jsp 页面中修改 task 的使用如下：

代码:

```
<a:task     mapId="Map0"          value="#{jsTask}"     rendered="true"
taskInfo="#{jsTaskInfo}"/>
```

修改后的 Task 如下所示：



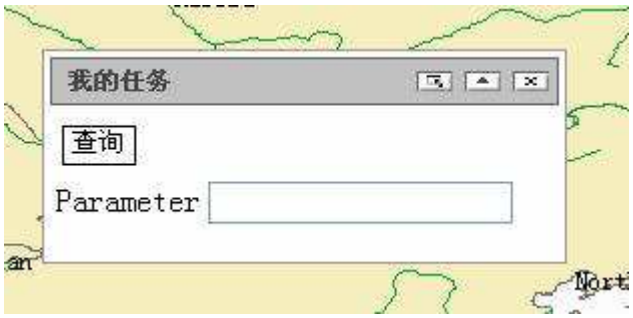
可以看到 task 的标题已经改变。下面我们来看修改 Action 的标题，代码如下：复制内容到剪贴板

代码:

```
public TaskActionDescriptorModel[]

getActionDescriptors() {

TaskActionDescriptorModel[] td=new TaskActionDescriptorModel[1];

TaskActionDescriptorModel                                    actionDesc=new
TaskActionDescriptor(JSTasks.class,"doSomething","查询");

td[0]=actionDesc;

return td;
```

```
}
```

需要说明的是 TaskActionDescriptor(JSTasks.class,"doSomething","查询")构造函数的三个参数，第一个是 task 类，第二个是方法名，第三个是修改后的方法名；构造后放到数组中返回即可，如果有多个方法，可以如法炮制，都放到数组中。修改后的 task 有如下表现：



你会发现另外一个 action 没有了，别着急，因为我们还没有把它放到数组中，而原来默认的显示方式已经被我们修改了。

我们来修改 doAnything 工具，doAnything 工具需要和地图交互，注意它的参数是 MapEvent 而不是 TaskEvent，它完整的代码如下所示：

代码:

```
public void doAnything(MapEvent te){

//通过和地图交互进行放大

WebContext ctx=te.getWebContext();

WebGeometry geom=te.getWebGeometry().toMapGeometry(ctx.getWebMap());

WebExtent ext = (WebExtent)geom;

ctx.getWebMap().setCurrentExtent(ext);

ctx.refresh();

}
```

如果跟地图交互，就要涉及到客户端执行的操作，我们这里没有任何地方指定客户端的操作，这是在 taskinfo 里面指定的，我们来看下面的代码：

代码:

```
public TaskToolDescriptorModel[]
```

```
getToolDescriptors() {

TaskToolDescriptor [] toolDesAry=new

TaskToolDescriptor[1];

TaskToolDescriptor toolDesc=new

TaskToolDescriptor(JSTasks.class,   "doAnything",   "  交  互  放  大
",ClientActions.MAP_RECTANGLE);

toolDesAry[0]=toolDesc;

return toolDesAry;

}
```

注意 TaskToolDescriptor 构造函数的4个参数，第一个是 task 类，第二个是原方法名，第三个是替换后的方法名，第四个是客户端的动作。构造完成后放到数组中，如果有多个 tools，如法炮制即可。修改后的 task 如下所示：



有了上面的解释，我们来看参数的 taskinfo 代码，就非常容易了：复制内容到剪贴板

代码:

```
public TaskParamDescriptorModel[] getParamDescriptors() {

// TODO Auto-generated method stub

TaskParamDescriptorModel[] td=new TaskParamDescriptorModel[1];

TaskParamDescriptorModel                                    paraDesc=new
TaskParamDescriptor(SearchTask.class,"searchText","查询文本");

td[0]=paraDesc;

return td;
```

```
}
```

修改后的 task 如下所示：

Task 框架的基本构成就是这样了，祝大家使用愉快。下节将介绍 TaskResult 的使用。

自定义 Functionality

今天我们来讲讲如何自定义 Functionality，自定义 Functionality 有很多用处。它最大的用处是可以访问数据源。我们来看看已经存在的 Functionality：
AGSMapFunctionality, AIMSMapFunctionality, AWSMapFunctionality, EJBMapFunctionality, WMSMapFunctionality 等等等等，查看 GISFunctionality 接口的子类你可以看到 ADF 中更多的 Functionality。
这些 Functionality 无一不跟后台的资源有关。我们来看看 Functionality 接口的定义：

代码:

```
public void initFunctionality(GISResource arg0) {

   }
public void destroyFunctionality() {
}
public GISResource getResource() {
return null;
}
```

每个 functionality 都需要实现 Functionality 这个接口，这个接口里面有三个方法，初始化方法，destroy 方法和得到资源的方法。GISResource 对应于一个特定的资源，比如我们最常用的资源就是 **AGSLocalMapResource**，当然也可以是 **ArcIMS** 的 **Resource**，取决于你把这个 **functionality** 注册到哪个 **Resource** 中。我们来看一个例子，虽然 **Functionality** 一般跟资源有关，但是也可以做其它的事情，比如我们这里举一个非常简单的例子，**FixedZoomin**：

代码:

```
public class DemoFunction implements GISFunctionality
 {
```

```
WebContext ctx;

public void zoomInFunction()
{
       WebMap wmap = ctx.getWebMap();
       WebExtent ext = wmap.getCurrentExtent();
       ext.expand(0.5);
       wmap.setCurrentExtent(ext);
       ctx.refresh();
}

public void initFunctionality(GISResource arg0)
 {
   ctx=arg0.getWebContext();
   System.out.print("Demo Function inited");
   }

   public void destroyFunctionality()
     {
   System.out.print("Demo Function destroyed");
    }

   public GISResource getResource()
{
   return null;
    }

   }
```

非常简单的一个例子，从 GISResource 里面得到 WebContext，然后得到 WebMap，修改 WebMap 的 Extent，放大功能就实现了。你说非得跟资源有关系，那么确实也有点关系，因为我们还是使用了后台的数据来进行操作。需要说明的是，我们这个代码对于所有的 Resource 都是适用的，不管你后台使用了 ArcIMS，还是其它的 Resource。ADF 的妙处也在这里。
写好了这个 Funcionality，那么我们来看看如何部署的，每一个 Functionality 都需要部署到某个 Resource 中，那么我来看一下，这个 functionality 如何来部署：

代码：

```
<managed-bean>

<managed-bean-name>ags1</managed-bean-name>

<managed-bean-class>com.esri.adf.web.ags.data.AGSLocalMapResource</man
aged-bean-class>
<managed-bean-scope>none</managed-bean-scope>

<managed-property>

<property-name>functionalities</property-name>
<map-entries>
   <!--……略去其它的 functionality->

<map-entry>
      <key>demoFunction</key>
<value>#{demoFunction}</value>
   </map-entry>
</map-entries>
</managed-property>
</managed-bean>
```

部署完成了，如何使用呢？随便在你的一个 tools 里面，或者 task 的运行代码里面，添加下面的代码：

代码:

```
GISResource
rec=(GISResource)evt.getWebContext().getResources().get("ags1");
DemoFunction df=(DemoFunction)rec.getFunctionality("demoFunction");
df.zoomInFunction();
```

就可以了。这只是一个简单的例子，而且是绕了一个远的例子。大家可以发掘一下 Functionlity 的使用方法。在下一次讲座中，我们将详细解释一个对 Resource 进行操作的例子。

用自定义 Functionality 实现用户权限控制

有了上此讲座的基础，我想理解我们这个功能应该就很容易了。刚刚我们经理问我有用户希望控制 web 登陆用户的权限，那么我们正好利用这个讲座来看看如何实现这个功能。我们新建一个类，functionality 当然要实现 GISFunctionality 接口了：

代码:

```
<?xml:namespace prefix = o ns = "urn:schemas-microsoft-comfficeffice" />
public class SecurityCheckFunctionality implements GISFunctionality {
private AGSMapResource resource;
  public void initFunctionality(GISResource resource) { //empty }
public void destroyFunctionality() { //empty  }
  public GISResource getResource() { //empty  }
}
```

我们在这里放了一个 AGSMapResource 类型的变量，因为我们需要对 Resource 进行很多操作，我们来看一下：

代码:

```
public void initFunctionality(GISResource resource) {
this.resource = (AGSMapResource)resource; //得到资源
    //查看用户是否属于"petroEmployee" 角色
   if(!WebUtil.getExternalContext().isUserInRole("pretroEmployee"))
{
   //从 Resource 中得到 MapFunctionality
  AGSMapFunctionality                    mapFunc                    =
(AGSMapFunctionality)resource.getFunctionality("map");

   //得到对 MapServerInfo 的引用
   MapServerInfo serverInfo = mapFunc.getMapServerInfo();
   //得到 "Pipeline_Network" 层
MapLayerInfo                  pipelineLayerInfo                    =
AGSUtil.getLayerInfo("PipeLine_Network",layerInfos);
    if(pipelineLayerInfo==null)
     return; //层不存在，return.
     //从 TOC 中移除
     MapLayerInfo[]                  newLayerInfos                    =
removeLayer(pipelineLayerInfo,layerInfos);
     serverInfo.setMapLayerInfos(newLayerInfos);
     //从 MAP 中移除
     LayerDescription[]                              layerDescriptions
```

```
=serverInfo.getDefaultMapDescription().getLayerDescriptions();
    //新建一个层描述，替换原来的层描述
     LayerDescription[]        newLayerDescriptions        =        new
LayerDescription[newLayerInfos.length];

     for(int i=0;i<newLayerInfos.length;i++){

newLayerDescriptions=AGSUtil.getLayerDescription(newLayerInfos.getLaye
rID(),layerDescriptions);
5
serverInfo.getDefaultMapDescription().setLayerDescriptions(newLayerDes
criptions);
   }
  }
```

上面的代码中用 removeLayer(pipelineLayerInfo,layerInfos);函数移除了要移除的层，然后更新了 TOC 和 MAP，我们来看看这个函数怎么来写：

代码:

```
  //Logic to remove a layer and all its descendants
  private  MapLayerInfo[]  removeLayer(MapLayerInfo  unwantedLayerInfo,
MapLayerInfo[] oldLayerInfos) {
       //新建一个 MapLayerInfo
   MapLayerInfo[]            newLayerInfos            =            new
MapLayerInfo[oldLayerInfos.length-1];
   ArrayList descendantLayers = new ArrayList();
   for(int i=0,j=0;i<oldLayerInfos.length;i++){
     if(oldLayerInfos.getLayerID()!=unwantedLayerInfo.getLayerID())
       newLayerInfos[j++] = oldLayerInfos;

if(oldLayerInfos.getParentLayerID()==unwantedLayerInfo.getLayerID())
       descendantLayers.add(oldLayerInfos);
  }
   for(int i=0;i<descendantLayers.size();i++){
newLayerInfos                                                          =
removeLayer((MapLayerInfo)descendantLayers.get(i),newLayerInfos);
   }
   return newLayerInfos;
  }
```

我们注意到这是一个递归函数，用于把该层下面所有的其它附属层都删除:

这个 functionality 写完了，我们来看看如何部署，部署的时候，先把它作为一个 Managed Bean 部署，用下面可代码即可，可以在 faces-config.xml 文件里面，也可以在 ags-functionality.xml 里面：

代码:

```
<managed-bean>
   <managed-bean-name>securitycheck</managed-bean-name>

<managed-bean-class>com.mypackage.SecurityCheckFunctionality</managed-
bean-class>
   <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
```

将它作为一个 managedBean 部署后，下面就把它部署到 resource 中，注意部署的时候，要将其部署到 map 之后，toc 之前，因为 ADF 初始化是按顺序初始化，我们的权限控制需要从 Map 中读出层的信息，然后修改 MapDesc，从而影响 TOC 的信息，所以，进行如下的部署配置：

代码:

```

<managed-bean>
   <managed-bean-name>ags1</managed-bean-name>
   <managed-bean-class>
   com.esri.adf.web.ags.data.AGSLocalMapResource
  </managed-bean-class>
   <managed-bean-scope>none</managed-bean-scope>
  ...
  <managed-property>
   <property-name>functionalities</property-name>
   <map-entries>
     <map-entry>
<key>map</key>
<value>#{agsMap}</value>
     </map-entry>
     <map-entry>
<key>query</key>
<value>#{agsQuery}</value>
     </map-entry>
     <map-entry>
<key>tile</key>
<value>#{agsTile}</value>
     </map-entry>
     <map-entry>
```

```
<key>overview</key>
<value>#{agsOverview}</value>
     </map-entry>
     <map-entry>
<key>pipeline_security_check</key>
<value>#{securitycheck}</value>
     </map-entry>
     <map-entry>
<key>toc</key>
<value>#{agsToc}</value>
     </map-entry>
   </map-entries>
  </managed-property>
</managed-bean>
```

OK，这个安全控制的 functionality 已经完成了。但是我们要对整个 WebApplication 加上安全控制，需要在 web.xml 做修改，添加如下的配置：

代码:

```
<login-config>
<auth-method>DIGEST</auth-method>
  <realm-name>My_WebApplication</realm-name>
</login-config>
//声明两个组用户
<security-role>
  <role-name>petroEmployee</role-name>
</security-role>

<security-role>
  <role-name>stateEmployee</role-name>
</security-role>

//声明什么资源将收到安全控制，这里我们把整个目录都控制，当然你也可以控制胆敢目录
<security-constraint>
  <web-resource-collection>
<url-pattern>*<url-pattern>
  </web-resource-collection>
  <auth-constraint>
<role-name>petroEmployee</role-name>
<role-name>stateEmployee<role-name>
</auth-constraint>
</security-constraint>
```

这两个组在什么地方定义呢？如果是 Tomcat，我们可以在$TOMCAT_HOME/conf/tomcat-users.xml 中定义，定义如下：

代码:

```
<tomcat-users>  ...
  <user name="joe" password="joe" roles="petroEmployee">
  <user name="daisy" password="daisy" roles="stateEmployee">
  ...
</tomcat-users>
```

这样我们就控制了整个 Web Application，并且对于不同的用户组，赋予了不同的层的查看权限。每次用户访问这个网站的时候，都会被提示要求输入用户名和密码，按照我们在 Tomcat-users 里面的定义进行输入即可。

还有用户会问道，如果我希望对属性字段的编辑进行控制怎么办；事实上原理是类似的，你让用户修改属性字段时，肯定要传输一个字段列表，你可以根据不同的用户组，对这个字段列表进行控制。

另外，这里我们用了 WEB 容器的用户和权限进行控制，事实上你不必依赖于 Web 容器，你可以在数据库中存放你自己定义的用户名和组，进行控制。
权限控制简单的讲就是这样子，.net 里面也类似，有机会的话，我会写一个.net 版本的给大家看看。

注：本讲座内容参考了并翻译了 ESRI 的 JAVA 帮助的部分内容。

使用图片和 TrueType 字体进行标注

直奔主题吧，接上个主题的讲座的内容，讲讲如何用图片进行标注。在前面一个讲座我们已经说明，如何对选中的物体进行高亮显示。而且这一部分工作是在 ADF 这一端完成的。那么有的时候，我们进行高亮显示的时候，不仅仅希望只是设置颜色，我们希望能够用图片或者 truetype 字体进行标注。比如在犯罪地点放一个坏人之类的功能。

我们来看看实现原理，关键是两个类，WebTrueTypeMarkerSymbol 和 WebPictureMarkerSymbol，没啥好说的，直接用代码来说明吧：

先来看看普通的点标注：

代码:

```
WebPointpt=(WebPoint)arg0.getWebGeometry().toMapGeometry(arg0.getWebCo
ntext().getWebMap());
```

```
WebSimpleMarkerSymbol markers =null;

markers = new WebSimpleMarkerSymbol();

markers.setAntialiasing(true);

markers.setColor("255,0,0");

markers.setWidth(8);

markers.setOutlineColor("255,0,0");

markers.setMarkerType(WebSimpleMarkerSymbol.CIRCLE);

markers.setPicture(bytInput);

GraphicElement ge=new GraphicElement();

ge.setGeometry(pt);

ge.setSymbol(markers);

WebGraphicsgraphics=arg0.getWebContext().getWebGraphics();

graphics.addGraphics(ge);

arg0.getWebContext().refresh();
```

注意 webgraphicsymbol 的 setPicture 方法的参数不是图片目录，而是图片的二进制数组，所以需要用文件 IO 把图片读取进来。当然，如果用户访问量很大，线程就不安全了，大家可以在 application 启动时进行读取，放在 context 的某个 attribute 里面。我原来认为是通过设置路径方式实现，这样又可能可以搞定 gif 图形的闪烁，但是现在实验结果是不行。设置图片标注的代码如下：

代码:

```
WebPointpt=(WebPoint)arg0.getWebGeometry().toMapGeometry(arg0.getWebCo
ntext().getWebMap());

//图片在 servelet 容器里面目录

String
picPath=FacesContext.getCurrentInstance().getExternalContext().getRequ
estContextPath()+"\\images\\angle.gif";
```

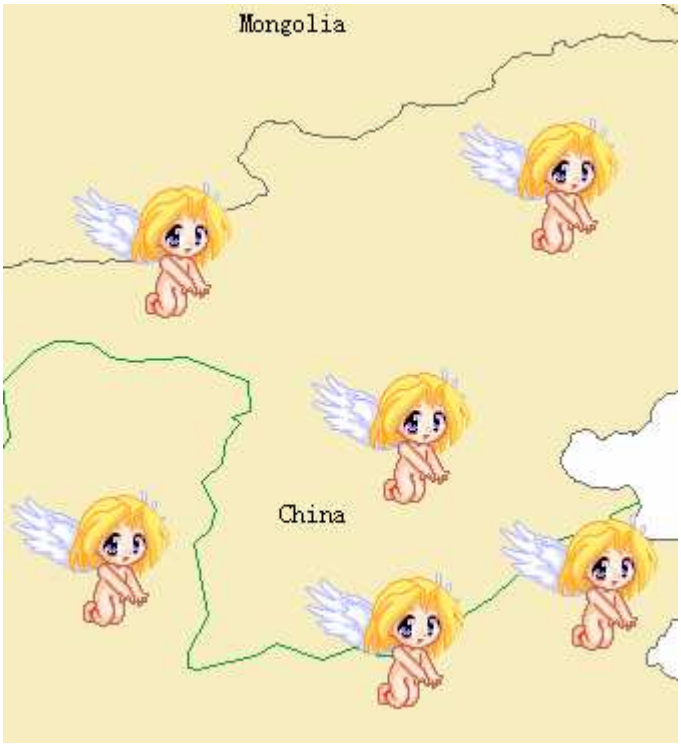
```
File myFile = new File(picPath);

FileInputStream myStream= newFileInputStream(myFile);

BufferedInputStream buf = newBufferedInputStream(myStream);

byte[] bytInput = newbyte[(int)myFile.length()];

buf.read(bytInput, 0, (int) myFile.length());

buf.close();

myStream.close();

WebPictureMarkerSymbol markers=newWebPictureMarkerSymbol();

markers.setPicture(bytInput);

GraphicElement ge=new GraphicElement();

ge.setGeometry(pt);

ge.setSymbol(markers);

WebGraphics graphics=arg0.getWebContext().getWebGraphics();

graphics.addGraphics(ge);

arg0.getWebContext().refresh();
```

在 jsf 文件里面添加如下代码，调用图片标注工具，进行测试：

代码:

```
<a:tool
id="pointTest"defaultImage="images/point.gif"hoverImage="images/pointU
.gif"

selectedImage="images/pointD.gif"clientAction="EsriMapPoint"
serverAction="com.cj.ucdemo.GifTestTool"clientPostBack="true"/>
```

图片标注的结果如下：



用图片做 markSymbol

我们也可以使用 TrueType 字体里面的矢量字体对图形进行符号化显示，这也是我们经常使用的方法，具体代码如下：

代码:

```
webMap=arg0.getWebContext().getWebMap();
WebPointpt=(WebPoint)arg0.getWebGeometry().toMapGeometry(webMap);
WebTrueTypeMarkerSymbol trueMarkerSymbol=new WebTrueTypeMarkerSymbol();
// 注意使用系统里面已经安装的字体
trueMarkerSymbol.setFontName("ESRITransportation & Civic");//设置索引
trueMarkerSymbol.setCharacterIndex(8);
trueMarkerSymbol.setFontColor("255,0,0");
trueMarkerSymbol.setFontSize(20);
trueMarkerSymbol.setFontStyle(WebTrueTypeMarkerSymbol.BOLD);

GraphicElement ge=new GraphicElement();

ge.setGeometry(pt);

ge.setSymbol(trueMarkerSymbol);
```

```
WebGraphicsgraphics=arg0.getWebContext().getWebGraphics();
graphics.addGraphics(ge);arg0.getWebContext().refresh();
```

把上面的代码放在一个点击工具里面，在 jsf 文件中用如下代码进行调试：

代码:

```
 <a:toolid="trueMarkerTest"
defaultImage="images/point.gif"
hoverImage="images/pointU.gif"
selectedImage="images/pointD.gif"
clientAction="EsriMapPoint"
serverAction="com.cj.ucdemo.TrueTypeMarkerTest"
clientPostBack="true"/>
```

TrueType 字体标注的结果如下图所示：



空间查询和高亮显示的实现

ESRI 在 JAVA ADF 中做了很多的工作，其中很重要的一个部分是 WebQuery。WebQuery 用来对

ArcGISServer 和 ArcIMS 进行查询，然后在 ADF 这一端对地图进行渲染。比如说高亮显示，用图片进行渲染等等。
我们来看一个简单的选中，并把选中结果进行高亮显示的开发过程。

事实上选中可以用两种方法，第一种是空间选择，第二种是文本查询，这两种选中都可以。ESRI 也提供了进行这两种查询的两个类，这两个类都继承自同一个接口，我们来看一下：
这个类是 IdentifyCriteria, TextCriteria，接口是 QueryCriteria，QueryCriteria 可以设置一些比如最大返回记录，是否返回记录的详细信息等设置。如名所示，IdentifyCriteria 用来对数据进行空间查询，我们来看一下，如何使用这个类：

```
IdentifyCriteria ic = new IdentifyCriteria(geom);
```

geom 是一个 WebGeometry 类的实例，它可以从客户端触发的 MapEvent 类中得到。

创建了这个 IdentifyCriteria 之后，就可以进行查询了，查询是由 WebQuery 来完成的，所以先要得到 WebQuery：

```
WebQuery query = (WebQuery) context.getAttribute("query");
```

大家打开 faces-config.xml，mapContext 里面由没有 query，如果有，用上面的代码就可以得到 WebQuery。

得到了 WebQuery，就可以大干一场了，可以用下面的办法进行空间查询：

```
List results=query.query(ic,lyrList);
```

ic 就是 IdentifyCriteria，lyrList 是一个查询目标层的列表，目标层可以是一个，也可以是多个，比如可以用下面的方法得到所有可以查到的图层：

```
public    ArrayList getQueryLayers(){
if(layers != null) return layers;
layers= new ArrayList();
WebQuery wQuery = this.context.getWebQuery();
WebLayerInfo layerInfo = null;
if(wQuery != null){
List layerList = wQuery.getQueryLayers();
for(Iterator iter = layerList.iterator(); iter.hasNext(); ) {
      Object item = (Object) iter.next();
       if(item instanceof WebLayerInfo){
            layerInfo = (WebLayerInfo)item;
           layers.add(new SelectItem(layerInfo, layerInfo.getName()));
}
        }
      }
return layers;
   }
```

当然，如果你不想查所有的图层，也可以让用户进行选择，选择哪个需要查询的图层。

现在来说说查询返回的结果，我们可以看到结果是一个 List，List 里面是什么？List 里面的对象是 QueryResult.

你可以循环得到里面的每一个 QueryResult，并把它们高亮显示：

```
for(int ii=0;ii<results.size();ii++){
      result = (QueryResult) results.get(ii);
    result.highlight();
}
```

Highlight 采用什么颜色，什么标志进行高亮显示，也是我们可以自己控制的，我个人的喜好是用红色，跟张艺谋一个水准，我们可以把显示符合加到 query 里面：

```
WebSimpleMarkerSymbol markers = query.getPointGraphicSymbol();
if (markers == null ) {
      markers = new WebSimpleMarkerSymbol();
  query.setPointGraphicSymbol(markers);
}
markers.setAntialiasing(true);
markers.setColor("255,0,0");
markers.setOutlineColor("255,0,0");
markers.setMarkerType(WebSimpleMarkerSymbol.CIRCLE);
```

同样，你可以设置线的颜色和多边形的颜色。这样就可以按照你需要的颜色进行高亮显示了。

当然 QueryResult 的功能不止能高亮显示，你还可以得到查询结果集的每一个字段的值，用 result.getDetails()方法即可得到一个 Map，里面有所有结果的字段名和字段值。

最后有说明的是，以上所有的代码对于 ArcGIS Server 和 ArcIMS 都同样适用，Enjoy。

动态添加图层数据

有的时候，特别是在做分析时，需要经常添加新的图层到当前的地图中。在这个时候，我们一般采用非池化的 Server Object，以保证当前用户的修改不会影响到其他用户。添加图层的代码如下：

首选，得到 Resource 和 mapFunctionality 和 MapServerInfo，以备后面使用：

代码:

```
AGSLocalMapResource
gisrec=(AGSLocalMapResource)webContext.getResources().get("ags1");
AGSMapFunctionalitymapfunction=(AGSMapFunctionality)gisrec.getFunction
alities().get("map");
MapServerPortmapServer=gisrec.getMapServer();
MapServerInfoserverInfo=mapServer.getServerInfo(mapServer.getDefaultMa
pName());
```

得到 MapServer 的 IMap 接口

代码:

```
IMaptargetMap=gisrec.getLocalMapServer().getMap(gisrec.getLocalMapServ
er().getDefaultMapName());
```

IMap 接口可以用来添加数据到 ServerObject。下面的代码可以用来从文件数据库中读取数据并把它添加到 IMap 中：

代码:

```
IWorkspacewk=this.connectToFileDB("D:\\data\\sapdata.gdb",gisrec.getSe
rverContext());
IFeatureClassfc=this.getFeatureclass(wk,"MSA_Intersect");
FeatureLayerflayer=(FeatureLayer)gisrec.getServerContext().createObjec
t(FeatureLayer.getClsid());
flayer.setFeatureClassByRef(fc);
targetMap.addLayer(flayer);
```

里面用到的两个 function 的代码如下：

代码:

```
public IWorkspace connectToFileDB(StringfileGDB,IServerContext sc)
throws java.net.UnknownHostException,java.io.IOException{
com.esri.arcgis.datasourcesGDB.FileGDBWorkspaceFactoryfactory =
(FileGDBWorkspaceFactory)sc.createObject(FileGDBWorkspaceFactory.getCl
sid());
com.esri.arcgis.geodatabase.IWorkspaceworkspace =
factory.openFromFile(fileGDB,0);
return workspace;

}

publicIFeatureClass getFeatureclass(IWorkspace workspace,String fName)
throwsjava.net.UnknownHostException, java.io.IOException{
com.esri.arcgis.geodatabase.IEnumDatasetenumDataset
=workspace.getDatasets(com.esri.arcgis.geodatabase.esriDatasetType.esr
iDTFeatureClass);
com.esri.arcgis.geodatabase.IDatasetds = null;
ds= enumDataset.next();
while(ds!= null){
if(ds.getName().endsWith(fName)){
return
new IFeatureClassProxy(ds);
}
ds= enumDataset.next();

}

retur nnull;

}
```

添加图层到 ServerObject 后，调用下面的代码，让修改生效。调用下面的代码后，将会使 ServerObject 的状态发生改变。

gisrec.getLocalMapServer().refreshServerObjects();

Server Object 状态改变后，需要重新得到 serverInfo，并且设置 MapFunctionality 里面的 LayerDescription，把新的 layer 加入进去。这里我们就不手工添加，而是直接把整个 MapDescription 设置一遍，如下面的代码所示：

代码:

```
serverInfo=mapServer.getServerInfo(mapServer.getDefaultMapName());
mapfunction.getMapServerInfo().setDefaultMapDescription(serverInfo.get
DefaultMapDescription());
mapfunction.setMapDescription(serverInfo.getDefaultMapDescription());
```

最后，当然要加上一句 refresh context 的代码:

代码:

```
webContext.refresh();
```

把此代码放在一个 tool 或者 command 中运行即可。这应该是最简洁明了的添加一个图层的代码了。

Server Java 自定义开发 - Network Analysis

一、目的

Server 9.2 下用户不用写任何代码，就可以用 Manager 或者 Eclipse 的工程模板创建很漂亮的 mapviewer 应用，但是没有可以即拿即用的 network Analysis 的模板。通过 Eclipse 的 Web Samples 中一个 Route Finder 的模板，这个模板创建的 network 应用需要有两个服务，一是带 network 图层的地图文档服务，另一个是 geocode 服务。为什么会同时需要两个服务呢？就是因为 Route Finder 是让用户在输入框中输入起点和终点的地址，然后通过 geocode 转换为地图上的空间点，然后才进行路径查询。

我们可能已经习惯了在 ArcMap 中的操作方式，在地图上添加 stop 和 barrier 点，然后进行路径生成。这种方式也更能为用户接受，毕竟可视化操作的用户体验更好，再说了，谁会愿意记一大堆地址呢？因此我们希望可以通过修改 Route Finder 模板的代码，来实现以下功能：

1、让用户通过点击工具按钮在地图上选择起点、终点以及路障，提交以后生成路径；这样就不需要 geocode 服务了。

2、用户选择的点能够以自定义的符号显示在地图上，达到交互的目的。

就是这两个简单的功能，下面就开始吧 javascript:window.open(this.src); alt=图片点击可在新窗口打开查看 src="/article/UploadPic/2007-6/2007626142947642.gif" onload=imgresize(this); border=0 smilieid="3">

二、数据的准备

这个比较简单，就用 ArcTutor\Network_Analyst\Exercise2\Paris.gdb 中的 streets 数据，因为我觉得 sfo 的图不好看。要先进行一些预处理，这不是本文重点，详细情况可以参考 Documentation 下的文档。

三、修改 faces-config.xml

1、在 Eclipse 下新建一个 ArcGIS Web Samples 工程，选择的模板是 ArcGIS Server Route Finder。

2、修改 faces-config.xml 文档，找到下面这段代码，删之，并且找到定义#{agswsgeo1}的那段<managed-bean>代码，一并删除。这两段代码在 WebContext 中注册一个 Geocode 服务，我们不需要。

```
        <value>#{agswsgeo1}</value>
    </map-entry>
```

3、找到下面这段代码，大家有兴趣可以观察一下它所在的位置，上下打望一番。把其中红色字体部分替换为 Map Server 的 URL 地址，其实就是我们发布的地图服务的地址，我的是 http://wangj:8399/arcgis/services/network/paris/MapServer。其实这个带中括号的部分应该在编译的时候就被替换掉的（有兴趣的可以去看看 sample 里面的那些 build.xml，其中主要功能就是用用户的输入参数替换这些方括号的内容），但实际上需要我们自己改，呵呵，自己动手，丰衣足食。

```
            <!-- AGSWS Map Server URL -->
        <value>[AGSWS_MAPSERVER_ENDPOINT]</value>
    </managed-property>
```

4、好了，最后修改一个地方，在 WebContext 中注册的两个 resources，其中一个是我们熟悉的 ags0，另一个是 agswsExt1，分别是地图服务和 NA 扩展。细心的人也许已经发现了，它们都注册了自己的 functionalities，但是用的是两套变量。ags0 用的是

```
        <value>#{agsMap}</value>
    </map-entry>
```

而 agswsExt1 用的是

```
        <key>map</key>

        <value>#{agswsMap}</value>

    </map-entry>
```

看到区别了吧，其实它们指向的是同一个变量，这个变量是在 WEB-INF/functionalities/ags-functionalities.xml 中定义的：

```
    <managed-bean-class>
      com.esri.adf.web.ags.data.AGSMapFunctionality
    </managed-bean-class>
    <managed-bean-scope>none</managed-bean-scope>
  </managed-bean>
```

看到了吧，好大一个坑，被我发现了 javascript:window.open(this.src); alt=图片点击可在新窗口打开查看 src="/article/UploadPic/2007-6/2007626142951155.gif" onload=imgresize(this); border=0 smilieid="9">。原来 ags0 引用错了，要把它改成与 ags-functionalities.xml 中定义的一样。当然，您也可以不修改，先运行看看会有什么样的 Exception 出现，很有意思。

```
        selectedImage="images/tasks/routing/addstopD.gif" clientAction="EsriMapPoint"
```

```
serverAction="com.wj.adf.web.route.AddOriginPoint" clientPostBack="true"

toolTip="Add start point"/>
```

看这段代码大家应该就知道这个 tool 执行什么任务了 javascript:window.open(this.src); alt=图片点击可在新窗口打开查看 src="/article/UploadPic/2007-6/2007626142951237.gif" onload=imgresize(this); border=0 smilieid="1"> 不错，目的就是让用户选择这个工具在地图上进行选点，执行了 EsriMapPoint 行为。服务器端代码需要执行两个操作，一是保存这个点的坐标（它是一个 WebPoint 类型，下面就称为 originPoint）；二是在地图上用自定义的符号绘制这个点，why？因为默认情况下 EsriMapPoint 这个行为不会在地图上做任何标记，那用户怎么能知道起点是否选中了呢？！没关系，看我的。

先定义一个标记的样式：

```
//定义标记的样式

originMarker = new WebTrueTypeMarkerSymbol();
originMarker.setAntialiasing(true);
originMarker.setMarkerColor("72,145,91");
originMarker.setOutlineColor("5,5,5");
originMarker.setMarkerType(WebTrueTypeMarkerSymbol.CIRCLE);
originMarker.setGlowingColor("125,125,125");
```

下面揭开 com.wj.adf.web.route.AddOriginPoint 这个类的神秘面纱

```
private WebTrueTypeMarkerSymbol originMarker;
protected GraphicElement stopsElement;
 public AddOriginPoint(){
     //实例化 originMarker 和 stopsElement
    ......
}
public void execute(MapEvent event) throws Exception {
     WebContext ctx = event.getWebContext();


     WebPoint point = (WebPoint)event.getWebGeometry();
     //将屏幕坐标转换为地图坐标
     point = WebPoint.toMapPoint(point, ctx.getWebMap().getCurrentExtent(),
                 ctx.getWebMap().getWidth(), ctx.getWebMap().getHeight());
     WebGraphics graphics = ctx.getWebGraphics();
     if(graphics == null){
         System.out.println("WebGraphics is null");
         return;
     }
     //将这个点显示在地图上
     stopsElement.setGeometry(point);
     //给该点一个编号，因为是起点，所以编号始终为 1
     ArrayList pList = new ArrayList();
     pList.add("1");
```

```
            originMarker.setTextValues(pList);
            stopsElement.setSymbol(originMarker);
            graphics.addGraphics(stopsElement);
             //保存 point，保存的策略大家可以自由选择，我是将 point 都存到了 WebSession 中
            ......
        }
}
```

哈哈，这下大家看到 WebGraphics 的妙处了吧，贴个效果图

javascript:window.open(this.src); alt= 图 片 点 击 可 在 新 窗 口 打 开 查 看 src="/article/UploadPic/2007-6/2007626142951817.gif" onload=imgresize(this); border=0>图片附件: [图标 1 就是用 WebGraphics 画的] 44.JPG (2006-12-20 03:57 PM, 14.15 K)

javascript:window.open(this.src); alt= 图 片 点 击 可 在 新 窗 口 打 开 查 看 src="/article/UploadPic/2007-6/2007626142951712.jpg" onload=imgresize(this); border=0>

在取得 networkAnalystFunc 之后，随便写个类如 RouteFind,注册 faces-config.xml，

RouteFind 里面有这个方法 getSearched，放到 toolbar 里面，

```
public void getSearched(MapEvent event,boolean click){
                    if(!doubleClick){
                            pointStrart= (WebPoint) event.getWebGeometry().toMapGeometry(webMap);
                            networkAnalystFunc.setStopSymbol(originMarker);
                        networkAnalystFunc.addStop(pointStrart);
                    }else{
                            pointEnd = (WebPoint) event.getWebGeometry().toMapGeometry(webMap);
                            networkAnalystFunc.clearRoute();
                            networkAnalystFunc.setRouteSymbol(routeLine);
                            networkAnalystFunc.setStopSymbol(destinationMarker);
                        networkAnalystFunc.addStop(pointEnd);
                        results = networkAnalystFunc.solve();
                    }
            }
```

ArcGIS Server 开发——控制图层是否可见

当需要控制图层的可见性时，可以使用

```
IMapDescription mapdescription = webMap.MapDescription;
webMap.ManageLifetime(mapdescription);

        ILayerDescriptions layerdec = mapdescription.LayerDescriptions;

    for(int i=0;i < mapdescription.LayerDescriptions.Count; i++)
{
    ILayerDescription onelayerdesc = layerdec.get_Element(i);
   onelayerdesc.Visible = true;
}
```

航线查询问题 - Server Java 的实现方法

数据准备

1、航线的相关数据，google 了一部分南航的数据，只要有几十条线路做演示就足够用了。将它们保存在数据库表中，主要有出发城市和到达城市两个字段。

2、用作地图显示的行政区划图以及城市的点要素图。为了简化，只选择了涉及上述航线的若干城市。城市数据是一个 point 的 shapefile 文件，其中有一个属性列为城市名称“name”，为了方便检索，name 属性中保存的城市名称与数据库中航线信息所保存的城市名称是匹配的。

实现方法

ADF 提供了很好用的 task framework，因此这个例子以自定义 task 的形式实现。先来简单了解一下整个操作的流程：

用户选择了自定义工具栏提供的一个工具，与地图进行一个点查询的交互，选择一个起点城市。在服务器端根据用户的操作将得到一个点的坐标，根据这个坐标进行查询，得到这个坐标点所表示的点要素的相关信息，这里我们所关心的只是这个城市的名称。知道了城市名称以后，以它作为起点城市，到数据库中检索所有从该城市出发的航线的目的地城市。这是一个字符串类型的数组，接下来可以根据这些城市名到地图中查找它们对应的点要素，并获得这些点要素的坐标。这样，我们就有了一个起点和若干个终点的坐标，可以绘制航线了。好了，基本思路就是这样，来看看怎么实现

先创建一个自定义 task 类，它有一个方法：

```
public class SearchAirlinesTask{
    public void getAirlines(MapEvent event){
    }
}
```

声明了这个类以后，task framework 就会根据 getAirlines(MapEvent event)方法的参数来判断这个工具的类型。由于 MapEvent 涉及的地图交互操作有很多种，而这里我们需要的是点查询操作，所以接下来我们需要再构造一个类来给自定义 task 加一些说明：

```
public class SearchAirlinesTaskInfo extends SimpleTaskInfo{
    private TaskToolDescriptor[] taskTool = new TaskToolDescriptor[1];
    public SearchAirlinesTaskInfo(){
   taskTool[0] =
            new TaskToolDescriptor(SearchAirlines.class,"getAirlines","选择起点
",ClientActions.MAP_POINT);
            taskTool[0].setToolTip(“从地图上选择一个起点”);
    }
    public TaskToolDescriptorModel[] getToolDescriptors(){
            return taskTool;
    }
}
```

然后将这个 taskInfo 类添加到我们的自定义 task 中：

```
public class SearchAirlinesTask{
```

```
    private SearchAirlinesTaskInfo taskInfo = new SearchAirlinesTaskInfo();
    public void getAirlines(MapEvent event){
    }
    public SimpleTaskInfo getTaskInfo(){
        return taskInfo;
    }
}
```

完成了上面的准备步骤以后，我们深入到 getAirlines(MapEvent event)方法的内部，看看它是怎么运行的：

```
public getAirlines(MapEvent event){
    WebContext ctx = event.getWebContext();
    WebGraphics graphics = ctx.getWebGraphics();
    WebQuery query = ctx.getWebQuery();
}
```

所有的查询操作都是通过 WebQuery 来进行的，查询的结果将通过 WebGraphics 绘制到地图上。查询的时候需要提供两个信息，点的坐标以及目标图层：

```
WebPoint point =
    (WebPoint)event.getWebGeometry().toMapGeometry(ctx.getWebMap());
IdentifyCriteria ic = new IdentifyCriteria(point);
ic.setMaxRecordCount(1);
//只需要查询 city 图层，该图层保存的是城市的相关信息
List layers = query.getQueryLayers();
List<WebLayerInfo> queryLayer = new ArrayList<WebLayerInfo>();
for(Iterator iter=layers.iterator(); iter.hasNext();){
    Object item = iter.next();
    if(item instanceof WebLayerInfo){
        WebLayerInfo layerinfo = (WebLayerInfo)item;
        if(layerinfo.getName().equals(“city”)){
            queryLayer.add(layerinfo);
        }
    }
}

List rs = query.query(ic, queryLayer);
```

ok，用户与地图交互操作的查询就完成了，接下来我们需要从查询结果中得到这个城市的城市名。

```
if(rs.size() > 0){
    Iterator iter = rs.iterator();
    QueryResult item = (QueryResult)iter.next();
    Object obj = item.getDetails().get(“name”);
    String startCity = obj.toString();
    WebPoint start = (WebPoint)item.getHighlightGeometry();
}
```

这里需要解释一下的是最后一句，也许有人会问，刚才我们不是已经获取了一个 point 坐标了吗，那个不就是起点城市么？事实上，我们在地图上点击查询的时候不会恰好就点在表示城市的那个点的中心，WebQuery 查询的时候是有一个距离容错的。而绘制航线的时候要求更精确，所以我们以查询得到的点

要素的坐标为准。
费了半天劲总算把城市名给找出来了，擦一把汗，接着查，我们还需要终点城市的坐标呢！刚才我们用的是 WebGeometry 来查询，接下来我们将使用文本进行查询。

```
DbAirlineSearch dboper = new DbAirlineSearch();
String[] destinations = dboper.getDestinations(startCity);
//终点城市的坐标
WebPoint[] ends = new WebPoint[destinations.length];
//航线
WebPath[] airlines = new WebPath[destinations.length];
//在 WebGraphics 中绘制航线所需的要素
GraphicElement[] linesElements =
        new GraphicElement[destinations.length];
for(int i=0;i<destinations.length;i++){
    //设置文本查询的条件
    TextCriteria tc = new TextCriteria();
    List<String> searchFields = new ArrayList<String>();
    searchFields.add("name");
    tc.setSearchFields(searchFields);
    tc.setSearchText(destinations[i]);
    List rs2 = query.query(tc,queryLayer);
    //处理文本查询的结果
    if(rs2.size() > 0){
        QueryResult destination = (QueryResult)rs2.iterator.next();
        ends[i] = (WebPoint)destination.getHighlightGeometry();
        //得到一个终点坐标以后就可以绘制一条航线了
        List<WebPoint> pointList = new ArrayList<WebPoint>();
        pointList.add(start);
        pointList.add(ends[i]);
        airlines[i] = new WebPath(pointList);
        WebPolyline polyline = new WebPolyline();
        polyline.addPath(airlines[i]);
        linesElement[i] = new GraphicElement();
        linesElement[i].setGeometry(polyline);//这里需要说明一下，如果这里用的是 WebPath，将不会绘制出航线，换成 WebPolyline 以后就可以了。
        linesElement[i].setSymbol(lineSymbol);
        graphics.addGraphics(linesElement[i]);
    }
}
```

到这里，航线的绘制功能就算完成了。后续的文章中将介绍如何将航线的其他信息添加到 WebResults 中，以及高亮显示某一条航线的功能

ArcGIS 航线查询完全例子

主要功能

1、可以在地图上点击一个起点城市，自动画出从该城市出发的所有航线；也可以通过文本框输入起点城市进行查询，然后绘制航线；
2、将查询结果显示给用户。当用户选择一条具体的航线时，列出该航线的相关信息，包括起点城市、终点城市、出发时间、到达时间、票价等；
3、根据需要，用户可以高亮显示某条航线，另外还有 ZoomTo 等功能。

数据准备
1、航线的相关数据，google 了一部分南航的数据，虽然少，但是做 demo 也差不多了。其中票价、起飞时间和到达时间是为了演示需要添加的三个字段。

	出发城市	到达城市	票价	起飞时间	到达时间
1	海口	北京	1201.00	8:35	14:35
2	海口	广州	1202.00	8:35	14:35
3	海口	深圳	1203.00	8:35	14:35
4	海口	南京	1204.00	8:35	14:35
5	海口	贵阳	1205.00	8:35	14:35
6	海口	成都	1206.00	8:35	14:35
7	海口	上海	1207.00	8:35	14:35
8	海口	郑州	1208.00	8:35	14:35
9	海口	长沙	1209.00	8:35	14:35
10	海口	西安	1210.00	8:35	14:35
11	海口	昆明	1211.00	8:35	14:35
12	海口	温州	1212.00	8:35	14:35
13	海口	大连	1213.00	8:35	14:35
14	海口	武汉	1214.00	8:35	14:35
15	海口	沈阳	1215.00	8:35	14:35
16	海口	哈尔滨	1216.00	8:35	14:35
17	海口	重庆	1217.00	8:35	14:35
18	海口	南昌	1218.00	8:35	14:35
19	海口	乌鲁木齐	1219.00	8:35	14:35
20	北京	重庆	1220.00	8:35	14:35
21	北京	成都	1221.00	8:35	14:35
22	北京	南昌	1222.00	8:35	14:35
23	北京	长沙	1223.00	8:35	14:35
24	北京	乌鲁木齐	1224.00	8:35	14:35
25	北京	三亚	1225.00	8:35	14:35
26	北京	延吉	1226.00	8:35	14:35
27	北京	昆明	1227.00	8:35	14:35
28	西安	榆林	1228.00	8:35	14:35
29	西安	银川	1229.00	8:35	14:35
30	西安	南京	1230.00	8:35	14:35
31	西安	长沙	1231.00	8:35	14:35
32	西安	海口	1232.00	8:35	14:35
33	西安	深圳	1233.00	8:35	14:35

34 | 西安 | 广州 | 1234.00 | 8:35 | 14:35
35 | 深圳 | 成都 | 1235.00 | 8:35 | 14:35
36 | 深圳 | 重庆 | 1236.00 | 8:35 | 14:35
37 | 深圳 | 武汉 | 1237.00 | 8:35 | 14:35
38 | 深圳 | 宜昌 | 1238.00 | 8:35 | 14:35
39 | 深圳 | 襄樊 | 1239.00 | 8:35 | 14:35
40 | 温州 | 南京 | 1240.00 | 8:35 | 14:35
41 | 温州 | 烟台 | 1241.00 | 8:35 | 14:35
42 | 温州 | 厦门 | 1242.00 | 8:35 | 14:35
43 | 温州 | 赣州 | 1243.00 | 8:35 | 14:35

把这些数据导入到数据库的 airlines 表中。

2、用作地图显示的行政区划图以及城市的点要素图。为了简化，只选择了涉及上述航线的若干城市。城市数据是一个 point 的 shapefile 文件，其中有一个属性列为城市名称“name”，为了方便检索，name 属性中保存的城市名称与数据库中航线信息所保存的城市名称是匹配的。

实现方法

ADF 提供了很好用的 task framework，因此这个例子以自定义 task 的形式实现。先来简单了解一下整个操作的流程：

用户选择了自定义工具栏提供的一个工具，与地图进行一个点查询的交互，选择一个起点城市。在服务器端根据用户的操作将得到一个点的坐标，根据这个坐标进行查询，得到这个坐标点所表示的点要素的相关信息，这里我们所关心的只是这个城市的名称。知道了城市名称以后，以它作为起点城市，到数据库中检索所有从该城市出发的航线的目的地城市。这是一个字符串类型的数组，接下来可以根据这些城市名到地图中查找它们对应的点要素，并获得这些点要素的坐标。这样，我们就有了一个起点和若干个终点的坐标，可以绘制航线了。好了，基本思路就是这样，来看看怎么实现 J

先创建一个自定义 task 类，它有一个方法：

```
public class SearchAirlinesTask{
public void getAirlines(MapEvent event){
}
}
```

声明了这个类以后，task framework 就会根据 getAirlines(MapEvent event)方法的参数来判断这个工具的类型。由于 MapEvent 涉及的地图交互操作有很多种，而这里我们需要的是点查询操作，所以接下来我们需要再构造一个类来给自定义 task 加一些说明：

```
public class SearchAirlinesTaskInfo extends SimpleTaskInfo{
private TaskToolDescriptor[] taskTool = new TaskToolDescriptor[1];
public SearchAirlinesTaskInfo(){
        taskTool[0] =
new TaskToolDescriptor(SearchAirlines.class,"getAirlines","选择起点",ClientActions.MAP_POINT);
        taskTool[0].setToolTip(“从地图上选择一个起点”);
}
public TaskToolDescriptorModel[] getToolDescriptors(){
        return taskTool;
}
}
```

然后将这个 taskInfo 类添加到我们的自定义 task 中：

```
public class SearchAirlinesTask{
private SearchAirlinesTaskInfo taskInfo = new SearchAirlinesTaskInfo();
public void getAirlines(MapEvent event){
}
public SimpleTaskInfo getTaskInfo(){
    return taskInfo;
}
}
```

完成了上面的准备步骤以后，我们深入到 getAirlines(MapEvent event)方法的内部，看看它是怎么运行的：

```
public getAirlines(MapEvent event){
WebContext ctx = event.getWebContext();
WebGraphics graphics = ctx.getWebGraphics();
WebQuery query = ctx.getWebQuery();
}
```

所有的查询操作都是通过 WebQuery 来进行的，查询的结果将通过 WebGraphics 绘制到地图上。查询的时候需要提供两个信息，点的坐标以及目标图层：

```
WebPoint point =
(WebPoint)event.getWebGeometry().toMapGeometry(ctx.getWebMap());
IdentifyCriteria ic = new IdentifyCriteria(point);
ic.setMaxRecordCount(1);
//只需要查询 city 图层，该图层保存的是城市的相关信息
List layers = query.getQueryLayers();
List<WebLayerInfo> queryLayer = new ArrayList<WebLayerInfo>();
for(Iterator iter=layers.iterator(); iter.hasNext();){
Object item = iter.next();
if(item instanceof WebLayerInfo){
    WebLayerInfo layerinfo = (WebLayerInfo)item;
    if(layerinfo.getName().equals(“city”)){
        queryLayer.add(layerinfo);
    }
}
}
List rs = query.query(ic, queryLayer);
```

ok，用户与地图交互操作的查询就完成了，接下来我们需要从查询结果中得到这个城市的城市名。

```
if(rs.size() > 0){
Iterator iter = rs.iterator();
QueryResult item = (QueryResult)iter.next();
Object obj = item.getDetails().get(“name”);
String startCity = obj.toString();
WebPoint start = (WebPoint)item.getHighlightGeometry();
}
```

这里需要解释一下的是最后一句，也许有人会问，刚才我们不是已经获取了一个 point 坐标了吗，那个不就是起点城市么？事实上，我们在地图上点击查询的时候不会恰好就点在表示城市的那个点的中心，WebQuery 查询的时候是有一个距离容错的。而绘制航线的时候要求更精确，所以我们以查询得到的点

要素的坐标为准。

费了半天劲总算把城市名给找出来了，擦一把汗，接着查，我们还需要终点城市的坐标呢！刚才我们用的是 WebGeometry 来查询，接下来我们将使用文本进行查询。

```
DbAirlineSearch dboper = new DbAirlineSearch();
String[] destinations = dboper.getDestinations(startCity);
//终点城市的坐标
WebPoint[] ends = new WebPoint[destinations.length];
//航线
WebPath[] airlines = new WebPath[destinations.length];
//在 WebGraphics 中绘制航线所需的要素
GraphicElement[] linesElements =
new GraphicElement[destinations.length];
for(int i=0;i<destinations.length;i++){
//设置文本查询的条件
TextCriteria tc = new TextCriteria();
List<String> searchFields = new ArrayList<String>();
searchFields.add(“name”);
tc.setSearchFields(searchFields);
tc.setSearchText(destinations[i]);
List rs2 = query.query(tc,queryLayer);
//处理文本查询的结果
if(rs2.size() > 0){
        QueryResult destination = (QueryResult)rs2.iterator.next();
        ends[i] = (WebPoint)destination.getHighlightGeometry();
      //得到一个终点坐标以后就可以绘制一条航线了
        List<WebPoint> pointList = new ArrayList<WebPoint>();
        pointList.add(start);
        pointList.add(ends[i]);
        airlines[i] = new WebPath(pointList);
        WebPolyline polyline = new WebPolyline();
        polyline.addPath(airlines[i]);
        linesElement[i] = new GraphicElement();
        linesElement[i].setGeometry(polyline);//这里需要说明一下，如果这里用的是 WebPath，将不会绘制出航线，换成 WebPolyline 以后就可以了。
        linesElement[i].setSymbol(lineSymbol);
        graphics.addGraphics(linesElement[i]);
}
}
```

好了，getAirlines(MapEvent event)方法大致就是这样，完成以后就可以在地图上进行交互查询了，选择某个城市以后就可以自动绘制出该城市出发的航线。

但是玩了几次以后，发现这个 demo 还可以有些扩展。比如可以将查询的结果保存在 WebResults 中，可以让用户点击某个航线记录，然后显示出该航线的相关信息，像起飞时间、到达时间、票价等等；另外，用户可能还有兴趣高亮显示某条记录，甚至用用 ZoomTo 功能啊。

仔细考虑一下，其实刚才我们已经做了很多准备工作了呀。首先，已经查询得到了起点和终点城市，根

据这两个城市的名称我们就可以从数据库中查询到这条航线的其他相关信息；其次，绘制航线时创建了 GraphicElement 来表示线要素，这个线要素中包含了 WebPolyline 的信息，可以用来进行高亮和 ZoomTo 等操作。哈哈，现在我们来充分利用这些已有数据。

首先要创建一个辅助类，用来记录每次查询操作的结果：

```
public class SearchAirlinesTaskResult{
private static WebSimpleLineSymbol highlightSymbol;
private WebContext ctx;
private GraphicElement originElement; //航线的原始符号
private WebPolyline polyline;
private GraphicElement highlightElement; //高亮线的符号
private String startCity;
private String endCity;
private Map<String,String> details; //航线的详细信息
static{
        highlightSymbol = new WebSimpleLineSymbol();
        highlightSymbol.setWidth(2);
        highlightSymbol.setColor(“234,244,34”);
        highlightSymbol.setLineType(WebSimpleLineSymbol.SOLID);
}
public SearchAirlinesTaskResult(WebContext context, GraphicElement originElement){
      this.ctx = context;
      this.originElement = originElement;
      this.polyline = (WebPolyline)originElement.getGeometry();
}
public Map getDetails(){
if(details == null){
        DbAirlineSearch dboper = new DbAirlineSearch();
        details = dboper.getDetails(startCity, endCity);
  }
return details;
}
//startCity 和 endCity 的 getter 与 setter 方法
……
public String getDescription(){
return startCity + “ 至 ” + endCity;
}
//highlight 的实现方法就是从 WebGraphics 中删除原有的线型符号，用新的高亮线型符号重绘一次
public void highlight(){
        WebGraphics graphics = ctx.getWebGraphics();
        graphics.removeGraphics(originElement);
        highlightElement = new GraphicElement();
        highlightElement.setGeometry(polyline);
        highlightElement.setSymbol(highlightSymbol);
        graphics.addGraphics(highlightElement);
```

```
        ctx.refresh();
}
public void clearHighlight(){
        WebGraphics graphics = ctx.getWebGraphics();
        if(highlightElement != null){
              graphics.removeGraphics(highlightElement);
        }
        graphics.addGraphics(originElement);
        ctx.refresh();
}
//根据 polyline 获取其外包矩形，然后刷新地图
public void zoomTo(){
        WebExtent ext = getExtent(polyline);
        ctx.getWebMap().setCurrentExtent(ext);
        ctx.refresh();
}
}
```

有了这个辅助类以后，只需要在前面的 getAirlines(MapEvent event)方法中添加以下代码：

```
……
ArrayList<SearchAirlinesTaskResult> results =
new ArrayList<SearchAirlinesTaskResult>();
for(int i=0;i<destinations.length;i++){
//设置文本查询的条件
TextCriteria tc = new TextCriteria();
List<String> searchFields = new ArrayList<String>();
searchFields.add(“name”);
tc.setSearchFields(searchFields);
tc.setSearchText(destinations[i]);
List rs2 = query.query(tc,queryLayer);
//处理文本查询的结果
if(rs2.size() > 0){
        QueryResult destination = (QueryResult)rs2.iterator.next();
        ends[i] = (WebPoint)destination.getHighlightGeometry();
        //得到一个终点坐标以后就可以绘制一条航线了
        List<WebPoint> pointList = new ArrayList<WebPoint>();
        pointList.add(start);
        pointList.add(ends[i]);
       airlines[i] = new WebPath(pointList);
        WebPolyline polyline = new WebPolyline();
        polyline.addPath(airlines[i]);
        linesElement[i] = new GraphicElement();
        linesElement[i].setGeometry(polyline);//这里需要说明一下，如果这里用的是 WebPath，将不会绘制出航线，换成 WebPolyline 以后就可以了。
        linesElement[i].setSymbol(lineSymbol);
```

```
        graphics.addGraphics(linesElement[i]);
          //将查询结果添加到 QueryResult 中
        SearchAirlinesTaskResult searchRs =
                                    new SearchAirlinesTaskResult(ctx,linesElement[i]);
                searchRs.setFromcity(fromcity);
                searchRs.setDestination(destinations[i]);
        results.add(searchRs);
}
}
HashMap<String,String> actions = new HashMap<String,String>();
actions.put("高亮显示", "highlight");
actions.put("取消高亮", "clearHighlight");
actions.put("ZoomTo", "zoomTo");
      ctx.getWebResults().addResultsWithActionMap("航线查询结果", results,
  "getDescription", "getDetails", actions);
ctx.refresh();
```

最终的效果图如下：

Born for SOA 系列

前言

在这一章里面，我们将用大量的篇幅，阐述 ArcGIS Server 对 SOA 的支持，我们在这里所指的 SOA，是基于 Web Serivces 的 SOA，其他架构的 SOA 不在我们这个系列的讨论之列。本章内容包括 Web Service 基础；如何在 J2EE 下环境下构建和部署 Web Service；如何使用 ArcGIS Server 构建 Web Service；如何在 SOA 系统中利用这些 Web Service 等。最后会以一个实际的例子为例，如何在利用 ArcGIS Server 编写 Web Service，并且部署到 SOA 环境中，解决实际中的问题。本章所有的例子都是基于 JDK 1.5, AXIS2, ArcGIS Server 9.2 JAVA ADF 开发。本章也是 CJ 编写的 ESRI 内部教材 Server Java 开发教程的第十五章。

Web Service 基础

Web Service 是描述一些操作（利用标准化的 XML 消息传递机制可以通过网络访问这些操作）的接口。Web 服务是用标准的、规范的 XML 概念描述的，称为 Web 服务的服务描述。这一描述囊括了与服务交互需要的全部细节，包括消息格式（详细描述操作）、传输协议和位置。该接口隐藏了实现服务的细节，允许独立于实现服务基于的硬件或软件平台和编写服务所用的编程语言使用服务。这允许并支持基于 Web 服务的应用程序成为松散耦合、面向组件和跨技术实现。Web 服务执行一项特定的任务或一组任务。Web 服务可以单独或同其它 Web 服务一起用于实现复杂的功能或商业交易。

听起来很复杂，但是我们用一个简单的例子加以说明。这个例子跟 GIS 无关，但是可以帮助大家了解在

J2EE 环境中，整个 Web Service 的编写，部署过程。并且如何在.Net 环境和 Java 环境中调用这个 Web Service。

以下的需求来自于一个我原来导师的项目，这虽然不是一个典型的 WebService 充满前景的一个例子，但是确实用 web service 来解决问题的一个好的开始。项目的背景是要帮用户部署多个网站，为了节约成本，两个网站都是使用租用的空间，结果有一个租用的 Linux 操作系统的空间没有安装 XWindows，导致一个使用 JAVA 写的图像缩放程序无法执行，当然，没有 Xwindows，基本上需要访问 AWT 包的程序都会无法执行。怎么解决呢，刚好另外一台服务器上有 Xwindows，可以使用这个功能，所以我们可以在另外一台服务器上编写一个 WebService，发送一个图像 Resize 请求，请求参数是被需要被 Resize 的图片的 URL，以及 Resize 的比例，而返回的结果是 Resize 后的图像。压缩图像的代码如下所示：

代码:

```
package edu.zju;

import java.awt.image.BufferedImage;

import java.io.FileOutputStream;

import java.io.InputStream;

import java.net.URL;

import javax.imageio.ImageIO;

import com.sun.image.codec.jpeg.JPEGCodec;

import com.sun.image.codec.jpeg.JPEGImageEncoder;

/**

*

* @author CJ

*

*/

public class CompressLocal {

publicCompressLocal() {
```

```
super();

}

/**

*

* @param filename 要被缩放的文件名，如果是 URL，请设置 isURL 为 true

* @param newname 缩放后的新的文件名

* @param std_img_w 目标文件的宽度

* @param std_img_h 目标文件的高度

* @param stretch 是否拉伸，如果1，则按照参数比例，如果是2，则按照宽度的比例；如果3，则取小的那个比例尺

* @param isURL 是否是 URL

*/

public void doScaleOnImage(String filename,String newname,int std_img_w,intstd_img_h,int stretch,boolean isURL)

{

BufferedImageoldimage=null;

try{

intnew_w=-1;int new_h=-1;

if(isURL){

InputStreamin=(new URL(filename)).openStream();

oldimage= ImageIO.read(in);

in.close();
```

```
}else{
oldimage = javax.imageio.ImageIO.read(newjava.io.File(filename));
}
intold_w = oldimage.getWidth(null); //原图像的宽度
intold_h = oldimage.getHeight(null);//原图像的高度
if(stretch==1){
new_w=std_img_w;
new_h=std_img_h;
}elseif(stretch==2){
if((old_w > std_img_w) || (old_h > std_img_h)) {
doubletagSize = (double)old_w / std_img_w; // 按宽度比例确定缩放比例
new_w= (int)(old_w / tagSize); //得到一个新的比例
new_h= (int)(old_h / tagSize);
}
}
else if(stretch==3){
new_w= old_w;
new_h = old_h;
if((old_w > std_img_w) || (old_h > std_img_h)) { //如果原图像高度和宽度都大于目标高度和宽度
floattmpdouble = std_img_w;
```

```
floattagSize = old_w / tmpdouble; // 按宽度比例确定缩放比例

tmpdouble= std_img_h;

if(tagSize < (old_h / tmpdouble)) //按高度确定缩放比例

tagSize= old_h / tmpdouble;

new_w= Math.round(old_w / tagSize); //得到一个新的比例

new_h= Math.round(old_h / tagSize);

}

}

BufferedImagetag = new
BufferedImage(new_w,new_h,BufferedImage.TYPE_INT_RGB);

tag.getGraphics().drawImage(oldimage,0,0,new_w,new_h,null);

oldimage.flush();

FileOutputStream newfile = new FileOutputStream(newname);

JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(newfile);

encoder.encode(tag);

tag.flush();

newfile.close();

}catch (Exception e) {

e.printStackTrace();

}

}
```

```
/**

* @param args

*/

public static void main(String[] args) {

CompressLocal cl = new CompressLocal();

cl.doScaleOnImage("D:\\temp\\photo\\0016.jpg","c:\\temp\\cj.jpg", 535,
312,2,false);

}

}
```

缩放图片的代码已经完成，下面来写一个方法，如何调用图片缩放代码： package edu.zju;

```
/**

* 作为一个 application scope 的 web service，供大家使用
*

* @author CJ

*

*/

public class CompressWebService {

publicCompressWebService() {

super();

}

/**
```

```
*

* @param url 源 jpg 文件的 URL

* @param width 目标 jpg 文件的宽度

* @param length 目标 jpg 文件的高度

* @return 返回目标 jpg 文件的 url

*/

publicString doCompress(String url,int width,int height){

StringrltURL="c:/temp/aa.jpg"; //例子程序使用，正式代码根据 url 生成文件名

CompressLocalcomp=new CompressLocal();

comp.doScaleOnImage(url,rltURL,width,height,1,true);

returnrltURL;

}

/**

* @param args

*/

public static void main(String[] args) {


}



}
```

代码编写完毕后，可以直接部署了，由于有了 AXIS，部署变得非常的简单，我们使用下面的 service.xml 进行部署，具体的部署方法可以参考 AXIS 的部署文档：

代码:

```
<service name="CompressService"scope="application">

<description>

ScaleJPEG Service

</description>

<messageReceivers>

<messageReceiver

mep="http://www.w3.org/2004/08/wsdl/in-only"

class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>

<messageReceiver

mep="http://www.w3.org/2004/08/wsdl/in-out"

class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>

</messageReceivers>

<parameter name="ServiceClass">

edu.zju.CompressWebService

</parameter>

</service>
```

部署完成后，可以使用下面的 URL 对 WebService 进行访问。

http://chunjie:8088/axis2/services/CompressService/doCompress?url=http://chunjie:8088/jpg/10/deans/0037.JPG&width=55&height=55

返回结果为

代码:

```
<ns:doCompressResponse xmlns:ns="http://zju.edu/xsd">

<ns:return>c:/temp/aa.jpg</ns:return>

</ns:doCompressResponse>
```

如果访问 http://chunjie:8088/axis2/services/CompressService?wsdl，可以得到描述当前 WebService 的 wsdl 文件。第一个 WebService 就部署完成了，是不是简单的不可思议？当然，得益于 axis 的伟大工作，隐藏了大量的细节，从而使工作变得如此简单。

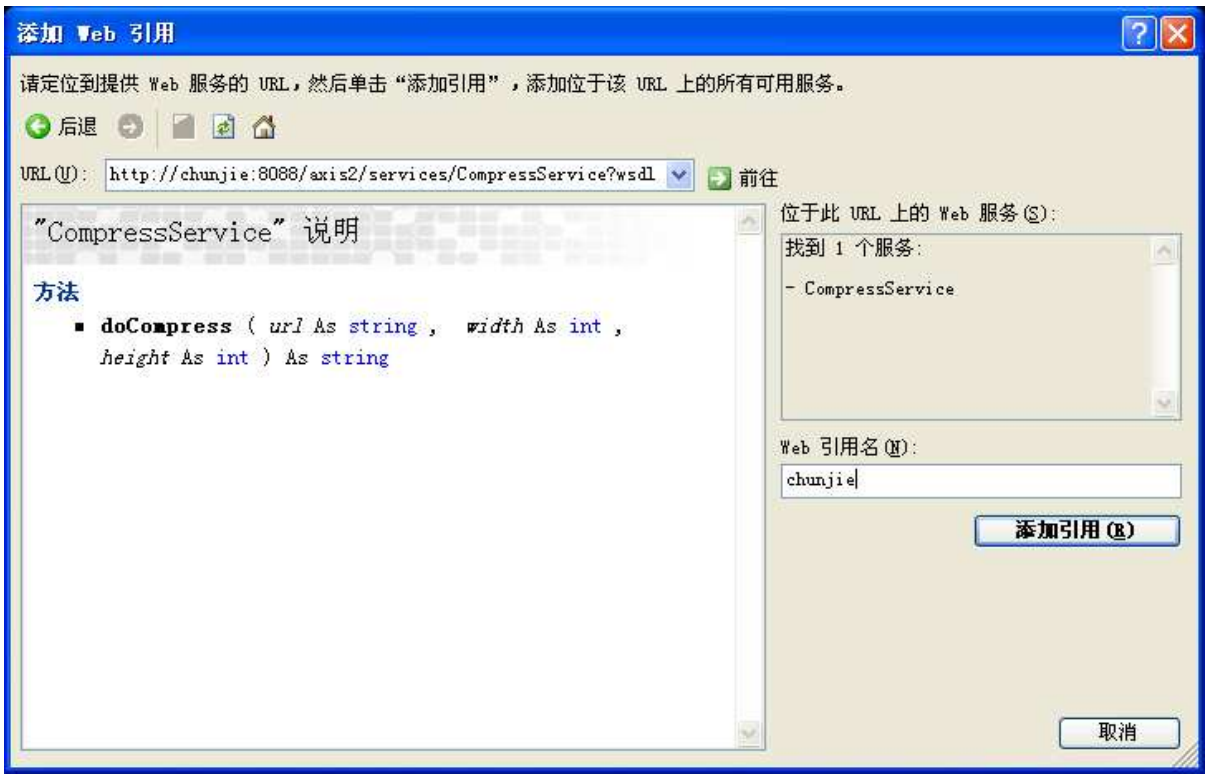
刚才我们直接通过 URL 已经可以访问 webservice 了。但是我们一般在 javacode 里面访问，访问后还需要做一些其他的操作，比如马上把图片文件保存到服务器上的某个目录中。如果需要在 Javacode 中访问该 web service，我们需要根据 wsdl 文件生成一个本地的 Java 代理类，通过这个代理类用来访问 WebService 的方法。**Axis 提供了 wsdl2java 的工具可以根据 wsdl 的描述生成 java 代理类**，可以使用命令行的工具，或者使用 Axis 为 Eclipse 提供的工具生成代理类。生成代理后，就可以用下面的方法调用该 WebService：

代码:

```
try {

DoCompress dc = new DoCompress();

dc.setUrl("http://chunjie:8088/jspsamples/jpg/thanks.jpg");

dc.setHeight(100);

dc.setWidth(100);

CompressWebServiceCompressWebServiceSOAP11PortStubstub = new
CompressWebServiceCompressWebServiceSOAP11PortStub(

null,"http://chunjie:8088/axis2/services/CompressService");

DoCompressResponsedcr=stub.doCompress(dc);
```

```
System.out.println(dcr.get_return());

} catch (AxisFault e) {

e.printStackTrace();

} catch (RemoteException e) {

e.printStackTrace();

}
```

当然，我们也可以用.Net 来调用这个 WebService，这正是 Web Service 的妙处所在，在 VisualStudio 里面新建一个项目，然后添加 Web 引用，在引用地址里面填入 service 的 URL，如下图所示：



添加引用后，就可以用下面的代码来进行访问：

代码:

```
consumeWS.chunjie.doCompress dc = new consumeWS.chunjie.doCompress();

dc.url = "http://chunjie:8088/jspsamples/jpg/thanks.jpg";
```

```
dc.height = 160;
dc.width = 120;
consumeWS.chunjie.CompressService sc = new  consumeWs.chunjie.CompressService();

consumeWS.chunjie.doCompressResponse dr= cs.doCompress(dc);
```

现在我们已经知道如何编写，部署，在 JAVA 和.NET 中调用 WebService。在下一节中，我们来看看可以通过怎样的方式对 ArcGIS Server 提供的 WebService 的进行访问。

ADF 中访问 Web Services 的类

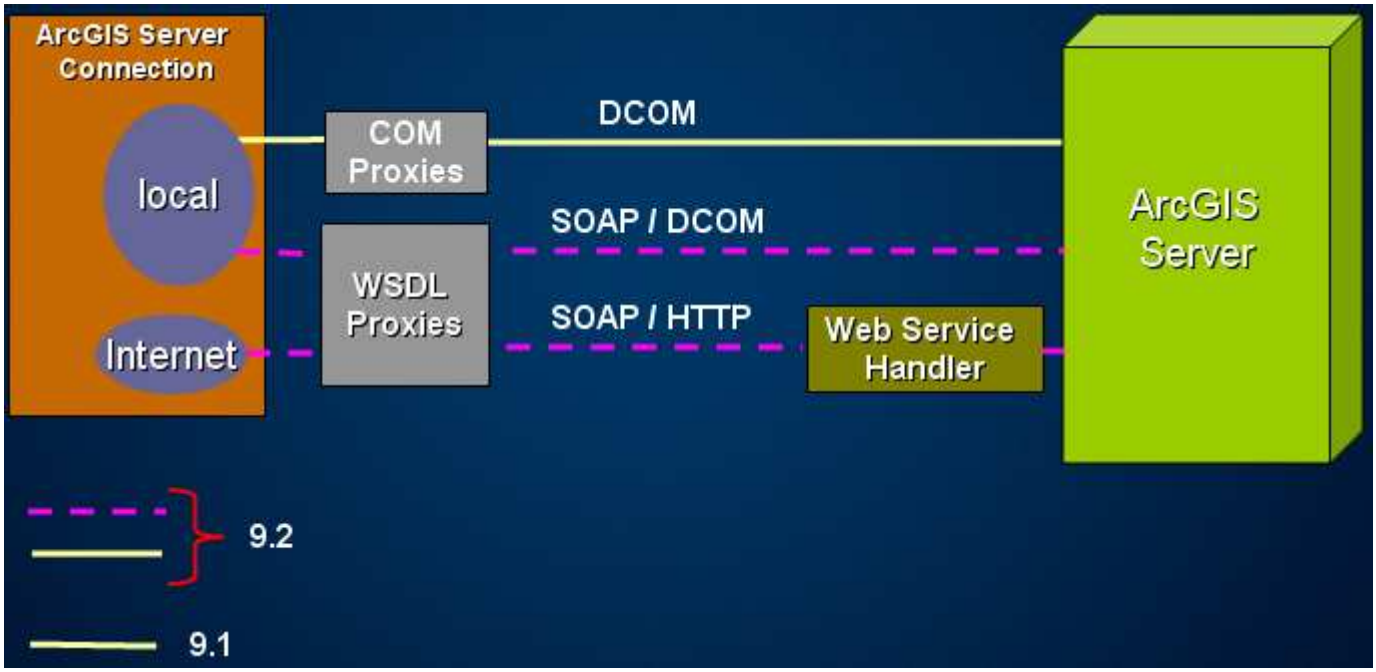
事实上细心的您肯定已经发现了，在 ADF 的类库中，很多类在两个 Package 中都会有，比如我们曾经在前面的例子中提到的类：MapDescription。

在9.2以前的服务中，此类都是老老实实的呆在 com.esri.arcgis.carto.MapDescription 包中，但是在9.2以后，有了一个新的 package，此 package 就是大名鼎鼎的 com.esri.arcgisws。这两个类是不同之处在于：**com.esri.arcgis.carto.MapDescription 类是指向 com 的一个代理，你在创建这个类的时候，必须创建在 ServerContext 里面**；而 com.esri.arcgisws. MapDescription 则是一个标准的 Java 类，你可以用创建普通类的方法创建一个 com.esri.arcgisws.MapDescription 实例。

在上面一节中已经知道，MapDescription 是通过 AXIS 的 WSDL2JAVA 工具转换而来的，事实上，这个包里面所有的类都是这么来的。注意 com.esri.arcgisws.MapDescription 的父类是 Object，创建 com.esri.arcgisws. MapDescription 类的时候，**它在本地的 JVM 中运行（比如 Tomcat 的 JVM 中），所以不消耗 ArcGIS Server 那一端的资源，而且很容易进行调试**。另外，com.esri.arcgisws. MapDescription 被称为 **Value Object**，它在向服务器发送请求前，被实例化并且设置一些值，作为参数发送到服务器端；从服务器端返回结果后，该类的属性又被设置成相应的值，从而得到服务器端返回的结果。如果想要看 MapDescription 的源代码，可以用 AXIS 根据 WSDL 生成一个 JAVA 类来查看。**跟 Server 进行交互的是8个特别的接口，这些接口的命名都很特殊:…port**，比如 com.esri.arcgisws.MapServerPort，这个接口唯一的实现类是 MapServerBindingStub ，这个接口可以通过 com.esri.adf.web.ags.data.AGSMapResource 类得到，而 AGSMapResource 是 adf 中的类，这样也就把我们所需要的一些功能跟 adf 连接起来了。如果您使用的是 AGSGPResource，那么对应的 port 就是 GPServerPort。而 network Analysist 比较奇怪，似乎现在 ESRI 更愿意让我们通过 Functionality 来使用它，看看它将来的情况吧。所以，所有的参数的创建，设置都在本地 JVM 中，从服务器端返回的结果也在 JVM 中，只有在使用 MapServerPort 进行查询或者其它操作时才跟 Server 进行交互。所以大大减少了和 Server 交互的次数和时间，大大提高了性能，同样的情况下，9.2比9.1可以服务更多的用户。这也可以让我们回想起在学习分布式系统编程的时候的一个规则：一次性传输1000 bytes 的数据，比传输1000次的1 byte 数据又快又好。

所以，adf 访问 ArcGISServer 的时候，推荐使用 SOAP 协议对 Server 进行访问，也就是我们提到的 com.esri.arcgisws 包中的类，这里 arcgisws 包中 ws 的含义就是 web services。这样效率会更高，速度也会更快。可以查看下面的图，了解协议请求的关系。事实上我们 ArcIMS 一开始就是使用 ArcXML 进行数据的交互；Server 现在也开始吸取 ArcIMS 的长处，在本地构建类后，通过 SOAP 协议，把 XML

请求发送到服务器端，如图16.3所示：



看看例子里面一个 CountFeature 的功能吧，在9.1下我们如果要计算某个层中选中的要素，方法是在 **ServerContext 里面建一个，SpatialFilter**，设置它的过滤要素和关系。然后对层进行 query，然后再得到返回记录集的数目，完成一个空间查询并得到选中要素的过程。在9.2环境下，我们不需要在 ServerContext 里面建 SpatialFilter，**而可以使用 arcgisws 包里面的 SpatialFitler，在本地创建空间过滤器，如下面的代码所示：**

代码:

```
EnvelopeN env =new EnvelopeN(extent.getMinX(), extent.getMinY(),
extent.getMaxX(),

extent.getMaxY(), null, null, null, null, null);

//新建一个空间过滤器

SpatialFilter spatialFilter = new SpatialFilter();

//设置空间相交关系

spatialFilter.setSpatialRel(EsriSpatialRelEnum.esriSpatialRelIntersect
s);

spatialFilter.setWhereClause("");

spatialFilter.setSearchOrder(EsriSearchOrder.esriSearchOrderSpatial);
```

```
spatialFilter.setSpatialRelDescription("");

spatialFilter.setGeometryFieldName("");

//设置和层相交的要素

spatialFilter.setFilterGeometry(env);

int layerId = 1;

this.countedFeatures =

mapServer.queryFeatureCount(mapServer.getDefaultMapName(),
layerId,spatialFilter);
```

在本地创建 SpatialFitler，设置好参数，**调用 mapServerPort 的 queryFeatureCount 方法，在调用这个方法的时候，SpatialFitler，layerId 等参数都作为 XML 封装在 SOAP 中，发送到服务器端，查询完成后再以 XML 返回结果。**

上面我们说到的一直是 ADF 这一端的对象，那么如果我们有的时候确实需要使用 AO 对象，怎么办呢？ESRI 的 JAVA 开发人员事实上早就想到了这些，比如用如下的方法实现对 AO 对象和 ADF 中 ws 包对象中的转换：

ADF 对象转向 AO 对象：

代码:

```
IGeometry    igeom    =(IGeometry)AGSUtil.createArcObjectFromStub(geom,
serverContext);
```

AO 对象转向 ADF 对象：

代码:

```
com.esri.arcgisws.Polygon                                          wPoly
=(com.esri.arcgisws.Polygon)AGSUtil.createStubFromArcObject(bufferedGe
om,com.esri.arcgisws.Polygon.class,serverContext);
```

这就是我们 ArcGIS Server 中无处不在的 Web Services，ArcGIS Server 有.net 版本和 Java 版本，事实上你完全可以用 Java 版本的 ADF 去访问.net 版本 Server。如果你要构建大型的 SOA，也完全可以用我们 server 的 API，去通过 SOAP 协议访问我们的 Server。

ArcGIS Server 实现 SOA

我们看到，ArcGIS Server 本身提供了大量的 Web Service，供各种使用者进行调用，这些使用者可以

是各种各样服务。包括地理编码服务，地图提供服务，空间查询服务，属性查询服务等等。
Web 服务是用于实现 SOA 的最常见技术标准。这虽然不是可以用于开发 SOA 的各个部分的唯一技术，但 Web 服务正迅速成为用于支持 SOA 的事实标准。

SOA 作为一种 IT 体系结构，支持将企业的业务重新集成，可在需要时通过网络访问这些服务和任务。这个网络可能完全包含公司总部，也可能分散于各地且采用不同的技术，通过对来自纽约、伦敦和香港的服务进行组合，可让最终用户感觉似乎这些服务就安装在本地桌面上一样。需要时，这些服务可以将自己组装为按需应用程序，以完成特定的任务，使公司的业务能够适应不断变化的情况和需求。

我们来看一个美国的城市系统管理平台，有点类似于在中国大地轰轰烈烈的网格化管理平台，只是该系统采用 SOA 的系统进行架构，以满足不同的用户需求。



该系统包含的 Web Service 包括地图服务，地理编码服务，事件管理服务等。先来看事件管理服务，可以用的事件包括：井盖丢了，广告牌倒了等市政管理项目。
1、 用户可以自己上网填写这些信息：输入地址，利用地理编码服务得到坐标，根据坐标调用点在多边形内服务，得到该事件所在的行政区域和管理区域，即可找到相应的管理人员。
2、 不是所有的用户喜欢上网，所以也可以通过打电话，接电话的人员采用桌面系统来输入，当然桌面系统也都是调用后面的 Web Service。流程类似。事件处理人员在处理完该事件后，可以在他们的系统中，调用事件管理服务，标记该事件为已经处理。
3、 地图生成服务，可以动态查看有哪些事件未处理，哪些事件已经处理；另外在内部系统的人员考核系统中，可以查看人员关联的事件情况。这里需要提到的是池化对象，无状态调用的时候，过

滤掉其他人员的事件是一个难点。我在下一节中会演示如何编写这样的 Web Service。

4、 当用户的系统扩展后，有其他的应用程序需要这些服务时，那些应用程序也可以方便的利用已经有的服务，或者编写新的服务，并且和老的服务进行组装，极大的提高服务和代码的可重用性，并且使系统易于扩展。

这是一个简单的例子，但是这是一个在实际系统中使用的例子。ArcGIS Server 内在的对 Web Service 的良好支持使它在 SOA 系统中如鱼得水。利用 ArcGIS Server 开发 GIS Web Serivce，简单且高效。

城市管理系统部署后，基于内部考核的需要，需要为每个地块管理人员生成地图，显示详细的事件处理情况。我们这里采用 2 种方式编写这样一个地图生成服务，1、采用 ADF 中 ws 包中的类；2、采用我们自己用工具生成的类。事实上，与其说编写这个 Web 服务，不如说是对已有的 ArcGISServer 提供的地图服务进行组装。我们编写的新的 Web Service，调用的恰恰是 ArcGIS Server 已经发布的 Service。

```
package com.cj.soa;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import org.apache.axis.AxisFault;
import com.esri.adf.web.ags.util.AGSUtil;
import com.esri.arcgisws.EsriImageFormat;
import com.esri.arcgisws.EsriImageReturnType;
import com.esri.arcgisws.ImageDescription;
import com.esri.arcgisws.ImageDisplay;
import com.esri.arcgisws.ImageType;
import com.esri.arcgisws.LayerDescription;
import com.esri.arcgisws.MapDescription;
import com.esri.arcgisws.MapImage;
import com.esri.arcgisws.MapLayerInfo;
import com.esri.arcgisws.MapServerBindingStub;
import com.esri.arcgisws.MapServerInfo;

/**
* 使用层的 definition 的方式，过滤记录集，生成最终的地图
* @authorCJ92
*
*/
public class CreateMapWithLayerDef {

                /**
        *
        * @param sqlStr 层的过滤 SQL
        * @param mapWidth 地图宽度
        * @param mapHeight 地图高度
        * @return 生成的地图的 URL
        */
public StringgetMapWithFilter(String sqlStr,int mapWidth,int mapHeight){
```

```
        Stringendpoint = "http://192.168.0.57:8399/arcgis/services/world/MapServer";

        MapServerBindingStubmapserver;
        try {
                mapserver= new MapServerBindingStub(new java.net.URL(endpoint), null);


        MapServerInfomapinfo = mapserver.getServerInfo(mapserver.getDefaultMapName());
        MapDescriptionmapdesc = mapinfo.getDefaultMapDescription();

        MapLayerInfo[]layerInfos=mapinfo.getMapLayerInfos();
    MapLayerInfo[] newLayerInfos=new MapLayerInfo[layerInfos.length];
     for(intii=0;ii<layerInfos.length;ii++){

                newLayerInfos[ii]=layerInfos[ii];
        }

          LayerDescription[] newLayerDescriptions = newLayerDescription[newLayerInfos.length];
          LayerDescription[]
layerDescriptions=mapinfo.getDefaultMapDescription().getLayerDescriptions();

          for(int i=0;i<newLayerInfos.length;i++){
newLayerDescriptions=AGSUtil.getLayerDescription(newLayerInfos.getLayerID(),layerDescriptions);
if(newLayerInfos.getName().equalsIgnoreCase("country")){

newLayerDescriptions.setDefinitionExpression(sqlStr);

};
}
mapinfo.getDefaultMapDescription().setLayerDescriptions(newLayerDescriptions);


ImageTypeimgtype = new ImageType();
imgtype.setImageFormat(EsriImageFormat.esriImageJPG);
imgtype.setImageReturnType(EsriImageReturnType.esriImageReturnURL);

ImageDisplayimgdisp = new ImageDisplay();
imgdisp.setImageHeight(mapHeight);
imgdisp.setImageWidth(mapWidth);
imgdisp.setImageDPI(96);


ImageDescriptionimgdesc = new ImageDescription();
imgdesc.setImageDisplay(imgdisp);
```

```
imgdesc.setImageType(imgtype);

MapImagemapimg = mapserver.exportMapImage(mapdesc, imgdesc);

returnmapimg.getImageURL();




}catch (AxisFault e) {

e.printStackTrace();
returnnull;
}catch (MalformedURLException e) {

e.printStackTrace();
returnnull;
}catch (RemoteException e) {

e.printStackTrace();
returnnull;
}



}
}
```

该类的代码我们就不在这里详细解释了。我这里直接对层名为 country 的层进行过滤。将该类直接部署到 AXIS2 的 Service 目录，部署的文件和目录结构如下：

```
├─com
│ └─cj
│ └─soa
│ CreateMapWithLayerDef.class
│
├─lib
│ arcgis_agsws_stubs.jar
│ arcgis_webcontrols_ags.jar
│ axis.jar
│
└─META-INF
MapService.wsdl
services.xml
```

放了一个 axis.jar，这个是早期版本的 axis.jar，因为 ESRI 使用了里面的一个异常类。MapService.wsdl 是用 Java 的工具生成。大家一定想看看 services.xml 里面的内容，services.xml 是我手工编写的，的内容如下：

```
<service name="MapService"scope="application">
<description>
Getthe Map by Layer Deffinition
</description>
<messageReceivers>
<messageReceiver
mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
<messageReceiver
mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</messageReceivers>
<parameter name="ServiceClass">
com.cj.soa.CreateMapWithLayerDef
</parameter>
</service>
```

在 AXIS2 的首页，可以看到这个服务的状态如下：



Server 对于 SOAP 的支持

ArcGIS Server 的服务可以通过 Local 或者 Internet 两种方式进行连接，Local 方式直接连接到 SOM 上，本地的对象如何和 SOM 进行交互呢？事实上是通过 AO 进行交互的，所以你必须在本地有 AO 对象才可以进行连接，我们平常使用 ArcCatlog 就可以这样连接，或者通过 Engine 也可以进行这样的连接。而 Internet 方式直接连接到 Web Service 的引用地址，它是通过本地对象连接的，**对于 Java ADF 而言，本地对象表示连接 ArcGIS Server 的类存在于本地 JRE 中**。Local 连接可以改变 Server Object 的状态，而 Internet 连接由于只是通过 SOAP API 进行访问，所有的交互都是无状态的。

对于 SOAP 的支持是在 AO 这一级，所以处理每一次 SOAP 请求，都会需要 AO 的介入。有两个接口特别要注意一下，一个是 IServiceCatalogAdmin，一个是 IRequestHandler，IRequestHandler 我们已经在上面一节提到过，而 IserviceCatalogAdmin 是用来得到 Server Object 所提供服务的 WSDL 的。

SOM 用来向客户端提供 WSDL，所有支持的服务类型的 WSDL 保存在 ArcGIS Server 的 XMLSchema 目录下。

我们可以使用 IServiceCatalogAdmin 接口得到所有服务类型的 WSDL，如下面的代码所示：

代码:

```
private void getWSDL(String domain,String username,String
password,String myserver){
ServerInitializer initializer = new ServerInitializer();
initializer.initializeServer(domain,username, password);
ServerConnection gisconnection;
try {
      gisconnection = new ServerConnection();
      gisconnection.connect(myserver);
// Get reference to ServerObjectManager class.

IServerObjectManager som =gisconnection.getServerObjectManager();
IServerContext serverContext = som.createServerContext("usa",
"MapServer");

byte[] bitscatalog = isc.getCatalogDescriptionDocument("Catalog",
"http://localhost");

//      Catalog WSDL
byte[] bitscatalog = isc.getCatalogDescriptionDocument("Catalog",
"http://localhost");

String catalog_wsdl = new String(bitscatalog,"UTF8");
System.out.println(catalog_wsdl);
//      Service WSDL
byte[] bitsservice = isc.getDescriptionDocument("usa", "MapServer",
"http://localhost");
String service_wsdl = new String(bitsservice, "UTF8");
System.out.println(service_wsdl);
} catch (UnknownHostException
e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}

}
```

事实上，ArcGISServer 每一个 ServerObject 都实现了 IRequestHandler 接口，都可以响应 SOAP 的请求，可以使用下面的方法得到 ServerObject 的 IRequestHandler，并且得到当前地图的 Default Map Name.

代码:

```
private void getMapName(IServerContext serverContext){
                IRequestHandler irh;
                try {
                        irh
=(IRequestHandler)serverContext.getServerObject();
                      String   soap_request   ="<?xml   version='1.0'
encoding='utf-8' ?>";
                     soap_request                    +="<soap:Envelope
xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:tns='http://www.esri.com/schemas/ArcGIS/9.2'>";
                      soap_request +="<soap:Body>";
                   soap_request +="<tns:GetDefaultMapName>";
                   soap_request +="</tns:GetDefaultMapName>";
                   soap_request +="</soap:Body>";
                   soap_request +="</soap:Envelope>";
                 String                                    soap_response
=irh.handleStringRequest("map,query,data", soap_request);
                } catch (AutomationException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
```

所以，ArcGIS Server 处理 SOAP 请求的能力是与生俱来的，从 Server Object 诞生那一天起，它就能处理 SOAP 请求。从而使通过 Web Service 的方式调用 GIS 功能变得异常简单。当然，我们一般都不会向 Server Object 直接发送请求，根据 WSDL 的描述，通过发送和接受 SOAP 请求来访问 ArcGIS Server 工作量十分巨大，那么我们来看看如何在 ADF 中使用 Web Service。

通过代理类访问 ArcGIS Server Web Service

我们现在知道，ArcGIS Server 的处理 SOAP 请求的能力在 Server Object 级别就已经实现了，所以我们可以不通过 Web Service 的方式，而通过直接发请求到 Server Object 的方式实现开发，只是这样太过麻烦。

好在 ArcGISServer 在发布每一个 Service 的时候，也同时发布了一个 Web Service，使我们可以通过 Web Services 的方式来直接访问 Service。图16.1就演示了一个 Map Service 的 WebService 入口：



这是ArcGIS Server提供的Web Service的引用地址，在第一节里面我们已经知道，我们可以通过在JAVA中创建代理类和.NET 中添加 Web 引用的方式访问这个 Web 服务。当然，最后对于 SOAP 的请求的处理还是通过 Server Object 来执行的。

在使用 ESRI 提供的类包之前，我们不妨自己来试试，通过特定的工具来根据 WSDL 生成本地代理类。AXIS2提供的 WSDL2JAVA 是一个很好的工具，生成的 JAVA 类的代码多达7万行。里面包含了所有需要访问这个 Web Service 所需要的一切类和方法。这些方法包括 Identify，Find，QueryFeatureCount 等，这些方法的参数和返回值，也作为内部类包含在该方法所在类中。需要特别指出的是：在访问 ArcGIS Server 的 Web Service 的时候，在本地 JRE 环境中，只需要有 AXIS 的类库和标准的 JRE 库即可，不需要任何其他的 AO 类或者代理类，这极大地提高了部署的灵活性。那么我们使用代理类来调用该 Web Services 的一个 QueryFeatureCount 的一个功能：

代码:

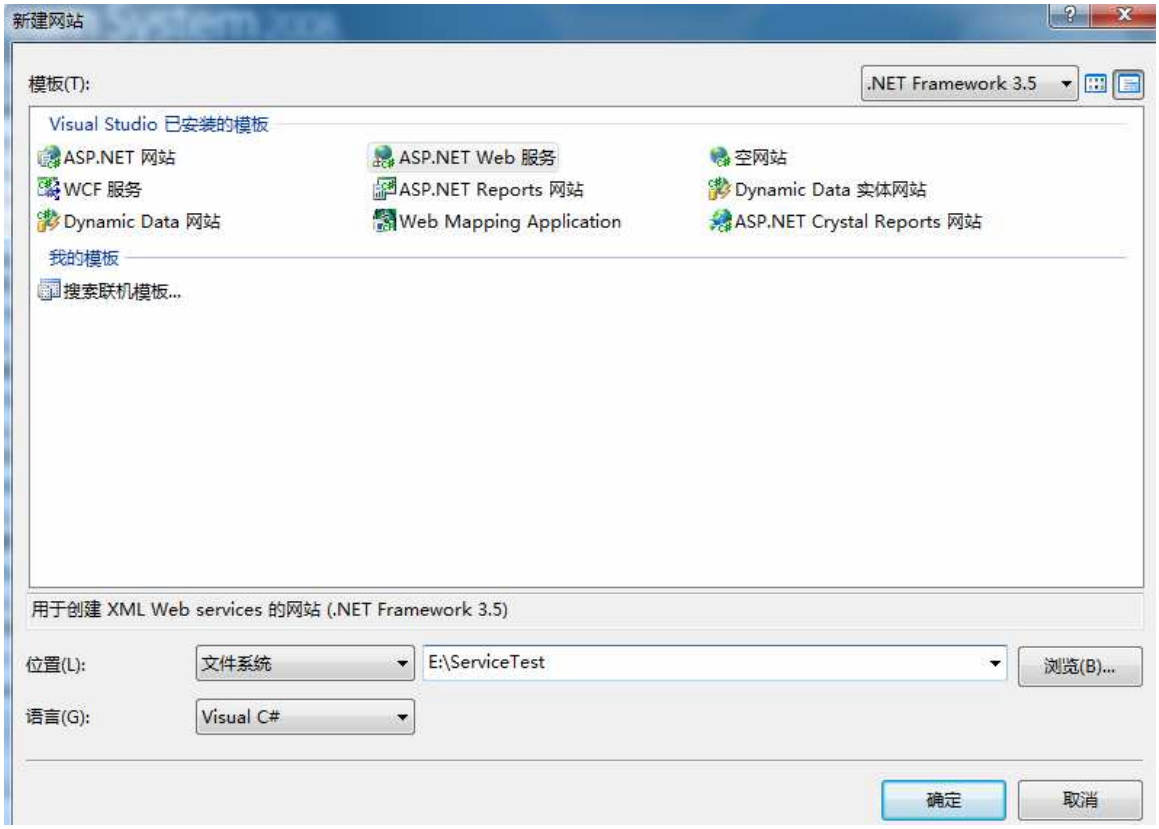
```
packageedu.zju.esrisvs;

importjava.rmi.RemoteException;

import org.apache.axis2.AxisFault;

import edu.zju.esrisvs.World_MapServerStub.QueryFeatureCount;

import edu.zju.esrisvs. World _MapServerStub.QueryFeatureCountResponse;

import edu.zju.esrisvs. World _MapServerStub.QueryFilter;

public classWebSvsTest {

public WebSvsTest() {

super();

}

/**



        * @param args
```

```
         */


        public static void main(String[] args){
try {
World _MapServerStubstub=new World _MapServerStub();
QueryFilter qf=newQueryFilter();
QueryFeatureCountqfc=new QueryFeatureCount();
qfc.setLayerID(0);
qfc.setMapName("Layers");
qf.setWhereClause("1=1");
qfc.setQueryFilter(qf);
QueryFeatureCountResponseqfr=stub.QueryFeatureCount(qfc);
System.out.println(qfr.getResult());
} catch (AxisFault e) {
e.printStackTrace();
} catch (RemoteException e) {
e.printStackTrace();
}
}
}
```

这就是使用代理类来访问该 Web Service 的过程，用工具生成的代理类非常长，大概有几万行代码，我就不贴在这里了，大家可以用工具生成。大家可能注意到，很多类的名字如 QueryFitler，和方法，如

QueryFeatureCount，都跟 ADF 里面的一样，事实上 ADF 里面访问 Web Service 的包同样也是用 AXIS 生成的。你也可以在.NET 里面，通过添加 Web 引用的方式来访问该 ServerObject 的 Web Service。

VS2008 下一个简单的 WebService



在这个例子里面，我们添加一个 Add 方法，用于求两个数的和，代码如下：

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Services;



[WebService(Namespace = "http://Esri.org/")]
```

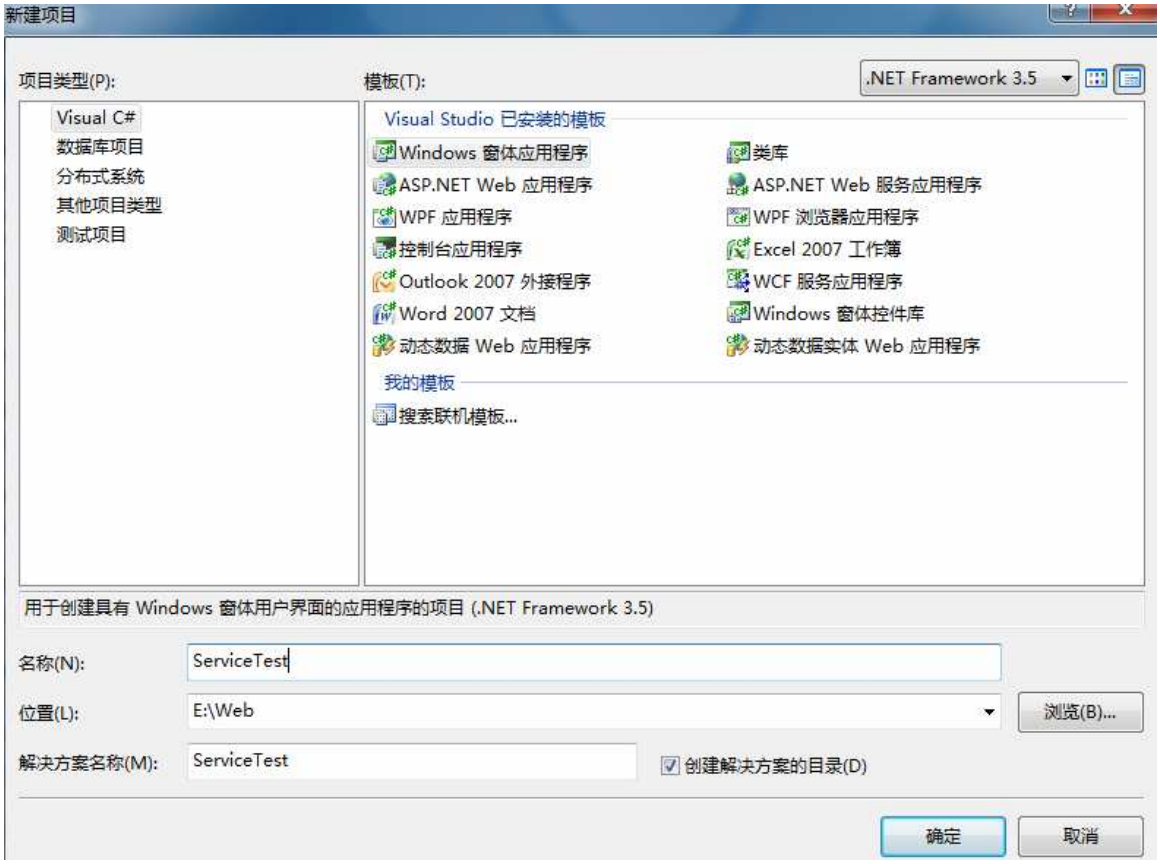
```
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
// [System.Web.Script.Services.ScriptService]
public class Service : System.Web.Services.WebService
{
    public Service () {

        //如果使用设计的组件，请取消注释以下行
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }

    [WebMethod]

    public double Add(double x, double y)
    {
        return x + y;
    }
```

}

我们在VS下调试，出现如下界面



我们可以清楚的看到这两个方法就是我们的WebService中的，现在我们通过客户端调用这这个Add方法。
我们创建一个新的Windows窗体应用程序，如下图：



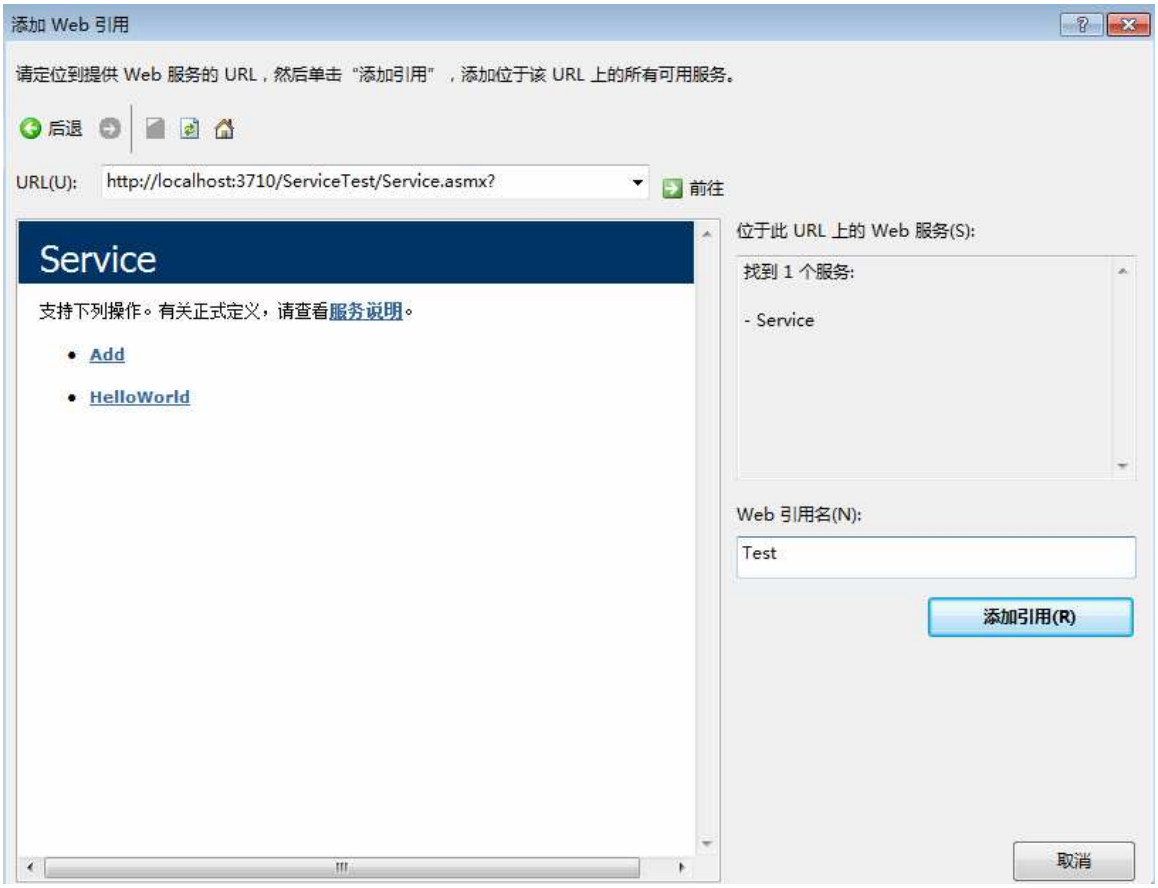
界面如下：



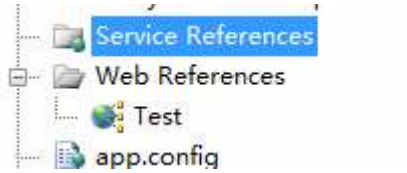
现在我们就调用我们刚才写的求和函数，在解决方案中，找到引用，右键，如下：



点添加服务引用，然后高级，找到添加Web引用

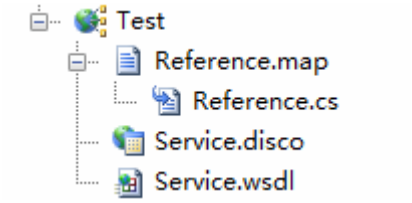


这时会在资源解决方案右边增加几个东西，如下图：



上面说过，我们要用代理类才能调用函数，而这里除了添加一个Web References的目录，我们没看见代理类？

单击工程中的解决方案资源管理器中的“显示所有文件”按钮，这时就可以看到那个代理类了，名叫Reference.cs。在代码中使用这个代理类基本上和使用本地普通类一样。如下图：



现在我们调用这个Add函数，代码如下：

```
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;



namespace ServiceTest

{

    public partial class ServiceTest : Form

    {

        public ServiceTest()

        {
```

```
            InitializeComponent();
        }

        private void Btn_Add_Click(object sender, EventArgs e)
        {
            //然后在后台写调用的代码，调用之前和使用其它的对象一样，要先实例化，
            //实例化的方法是  Test.Service ServiceAdd = new Test.Service();;
            //然后就可以通过ServiceAdd来访问WebService里面提供的方法了
            Test.Service ServiceAdd = new Test.Service();

            double   a = Convert .ToDouble (textBox1.Text);

            double   b =   Convert.ToDouble (textBox2.Text);
            //调用Add方法
            double   c = ServiceAdd.Add(a, b);

            textBox3.Text = c.ToString();

        }
    }
}
```

执行我们的Windows引用程序，得到如下的结果



从上面的这个例子可以看出，Web Service 没有什么神秘的，它就是一个应用程序，向外界暴露出一个能够通过Web进行调用的API。

Flex 调用 Net WebService

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
<mx:WebService  id ="TestService"
wsdl="http://localhost:3710/ServiceTest/Service.asmx?wsdl"
showBusyCursor="true" useProxy="false" >
   <mx:operation name ="Add" result="succeed(event)"/>
</mx:WebService>
    <mx:Text id = "BText" x="124" y="33" text="Text" width="80"/>
    <mx:Button x="124" y="82" label="计算" click="Result()"/>
    <mx:Script >
    <![CDATA[
        import mx.rpc.events.FaultEvent;
           import mx.rpc.events.ResultEvent;
         internal function Result(): void
          {
          TestService.add(5,6);
          }
```

```
        internal function succeed(event :ResultEvent):void
         {
         BText.text=event.result.toString();
         }
    ]]>
   </mx:Script>
</mx:Application>
```

wsdl属性指定到要访问的WebService的wsdl地址既可。我们在WebService 组件内部定义了操作标签，其中 name属性必须和WebService中的函数名相同，result属性标记访问WebService方法成功后的处理函数。同理还可以指定WebService调用失败的函数，在这里我省略了，人比较懒，哈哈……，这个WebService就是我们上次Net中的。使用代理(不在同一主机或域) useProxy="true"属性直接调用服务，不使用代理(在同一主机或域)，useProxy="false"属性直接调用服务。终端服务必须有 crossdomain.xml 文件。

下面一个是官方文档给出的，可以参照

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns="*" xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical">
    <mx:WebService
        id="nwCL"
        wsdl="http://localhost/data/CustomerList.asmx?wsdl"
        result="onResult(event)"
        fault="onFault(event)" />
    <mx:Button label="Get Single DataTable" click="nwCL.getSingleDataTable()"/>
    <mx:Button label="Get MultiTable DataSet"
click="nwCL.getMultiTableDataSet()"/>
    <mx:Panel id="dataPanel" width="100%" height="100%" title="Data Tables"/>

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.controls.DataGrid;
            import mx.rpc.events.FaultEvent;
            import mx.rpc.events.ResultEvent;

            private function onResult(event:ResultEvent):void {
                // A DataTable or DataSet returned from a .NET webservice is
                // automatically converted to an object with a "Tables" property,
                // which contains a map of one or more dataTables.
                if (event.result.Tables != null)
                {
                    // clean up panel from previous calls.
                    dataPanel.removeAllChildren();

                    for each (var table:Object in event.result.Tables)
                    {
```

```
                    displayTable(table);
                }

                // Alternatively, if a table's name is known beforehand,
                // it can be accessed using this syntax:
                var namedTable:Object = event.result.Tables.Customers;
                //displayTable(namedTable);
            }
        }

        private function displayTable(tbl:Object):void {
            var dg:DataGrid = new DataGrid();
            dataPanel.addChild(dg);
            // Each table object from the "Tables" map contains two properties:
            // "Columns" and "Rows". "Rows" is where the data is, so we can
set
            // that as the dataProvider for a DataGrid.
            dg.dataProvider = tbl.Rows;
        }

        private function onFault(event:FaultEvent):void {
            Alert.show(event.fault.toString());
        }
    ]]>
</mx:Script>

</mx:Application>
```

The following example shows the .NET C# class that is the backend web service implementation called by the Flex application; this class uses the Microsoft SQL Server Northwind sample database:

:

```
<%@ WebService Language="C#" Class="CustomerList" %>
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Data;
using System.Data.SqlClient;
using System;

public class CustomerList : WebService {
```

```
    [WebMethod]
    public DataTable getSingleDataTable() {
        string cnStr = "[Your_Database_Connection_String]";
        string query = "SELECT TOP 10 * FROM Customers";
        SqlConnection cn = new SqlConnection(cnStr);
        cn.Open();
        SqlDataAdapter adpt = new SqlDataAdapter(new SqlCommand(query, cn));
        DataTable dt = new DataTable("Customers");

        adpt.Fill(dt);
        return dt;
    }

    [WebMethod]
    public DataSet getMultiTableDataSet() {
        string cnStr = "[Your_Database_Connection_String]";
        string query1 = "SELECT TOP 10 CustomerID, CompanyName FROM Customers";
        string query2 = "SELECT TOP 10 OrderID, CustomerID, ShipCity,
        ShipCountry FROM Orders";
        SqlConnection cn = new SqlConnection(cnStr);
        cn.Open();

        SqlDataAdapter adpt = new SqlDataAdapter(new SqlCommand(query1, cn));
        DataSet ds = new DataSet("TwoTableDataSet");
        adpt.Fill(ds, "Customers");

        adpt.SelectCommand = new SqlCommand(query2, cn);
        adpt.Fill(ds, "Orders");

        return ds;
    }
}
```

WSDL 介绍

一：WSDL 定义

WSDL 是一个用于精确描述 Web 服务的文档，**WSDL** 文档是一个遵循 **WSDL** XML 模式的 XML 文档。**WSDL** 文档将 Web 服务定义为服务访问点或端口的集合。在 **WSDL** 中，由于服务访问点和消息的抽象定义已从具体的服务部署或数据格式绑定中分离出来，因此可以对抽象定义进行再次使用：**消息，指对交换数据的抽象描述；而端口类型，指操作的抽象集合**。用于

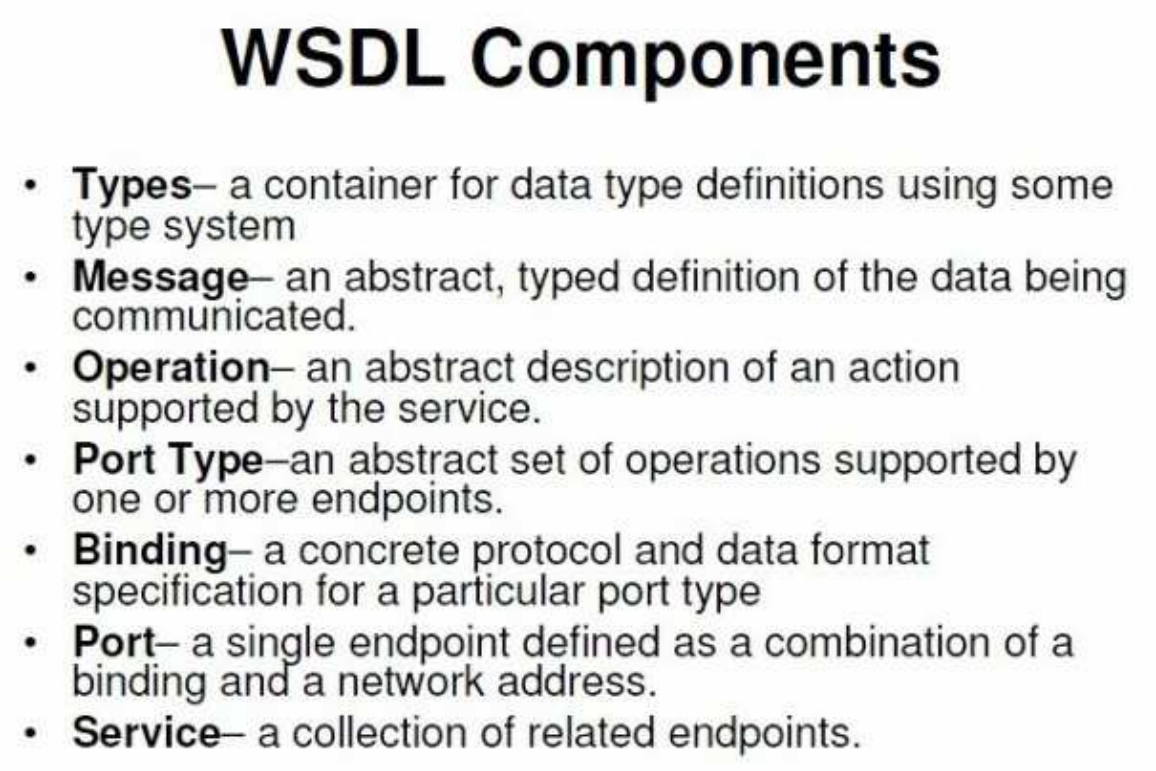
特定端口类型的具体协议和数据格式规范构成了可以再次使用的绑定。将 Web 访问地址与可再次使用的绑定相关联，可以定义一个端口，而端口的集合则定义为服务。

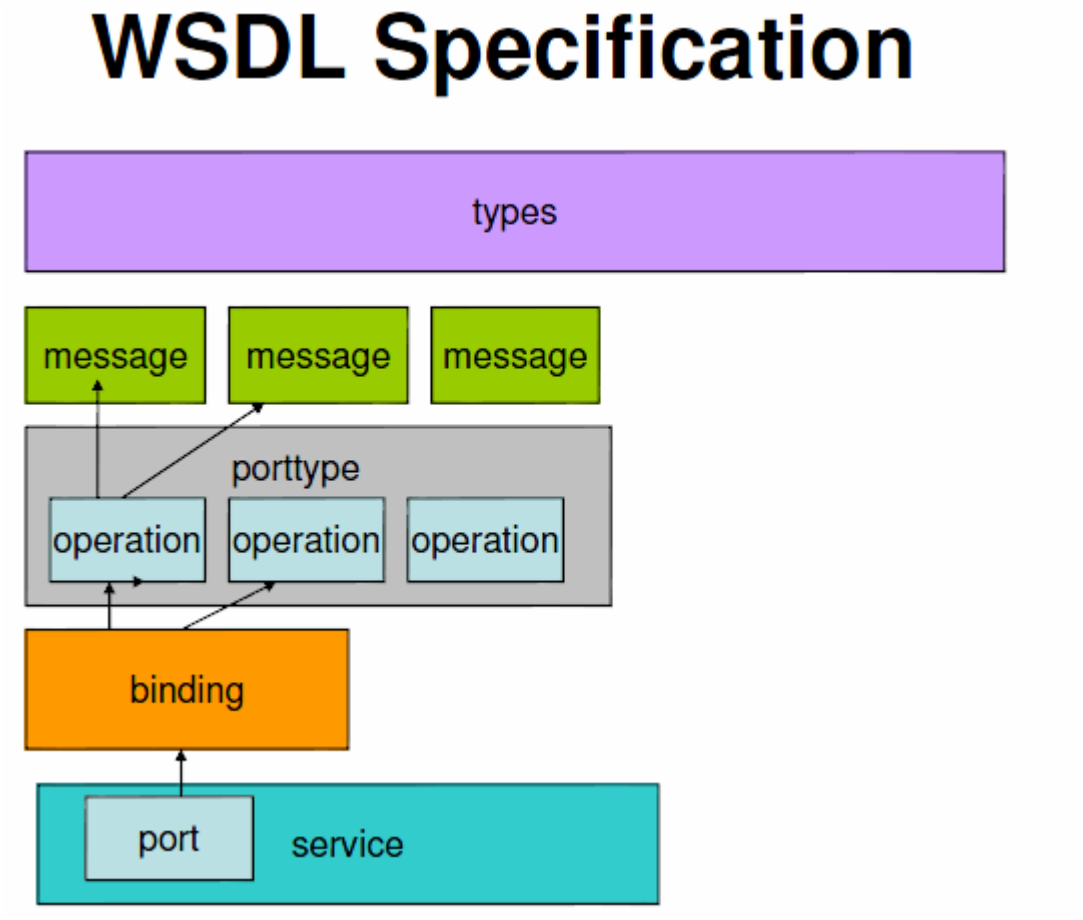
A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.

一个 **WSDL** 文档通常包含 7 个重要的元素，即 types、import、message、portType、operation、binding、 service 元素。这些元素嵌套在 definitions 元素中，definitions 是 **WSDL** 文档的根元素。文章的下一部分将会详细介绍 **WSDL** 的基本**结构**。

二：**WSDL** 的基本**结构**--概述

如第一部分最后描述的那样，一个基本的 **WSDL** 文档包含 7 个重要的元素。下面将分别介绍这几个元素以及他们的作用。





Types – 数据类型定义的容器，它使用某种类型系统(一般地使用 XML Schema 中的类型系统)。

Message – 通信消息的数据**结构**的抽象类型化定义。使用 Types 所定义的类型来定义整个消息的数据**结构**。

Operation – 对服务中所支持的操作的抽象描述，一般单个 Operation 描述了一个访问入口的请求/响应消息对。

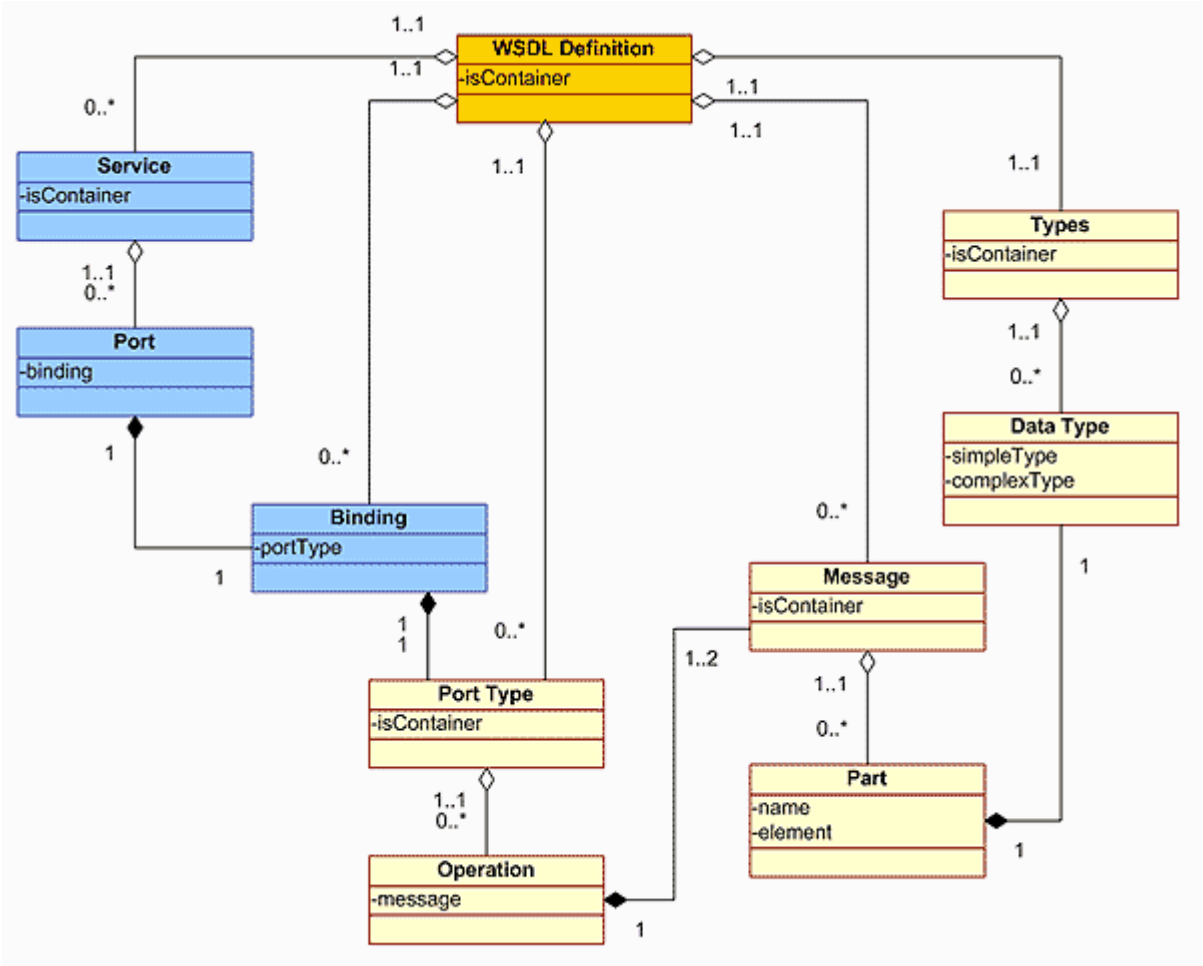
PortType – 对于某个访问入口点类型所支持的操作的抽象集合，这些操作可以由一个或多个服务访问点来支持。

Binding – 特定端口类型的具体协议和数据格式规范的绑定。

Port – 定义为协议/数据格式绑定与具体 Web 访问地址组合的单个服务访问点。

Service– 相关服务访问点的集合。

可以参考下图来理解一下 WSDL 的文档**结构**图：



WSDL 的 xml schema 可以参照如下网址：http://schemas.xmlsoap.org/**wsdl**/

三：WSDL 的基本**结构**--详述

本节将通过一个例子详细描述 WSDL 文档每个元素的作用。下面一个例子是一个简单的 WSDL 文档的内容，该文档的产生可以参见我的另外一篇文章：xfire 开发实例--HelloWorld 篇 。

一个简单的 Web Service 的 WSDL 文档，该服务支持名为 sayHello 的唯一操作，该操作通过在 http 上运行 SOAP 协议来实现的。该请求接受一个字符串 name，经过处理后返回一个简单的字符串。文档如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
   targetNamespace="http://com.liuxiang.xfireDemo/HelloService"
   xmlns:tns="http://com.liuxiang.xfireDemo/HelloService"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      attributeFormDefault="qualified" elementFormDefault="qualified"
      targetNamespace="http://com.liuxiang.xfireDemo/HelloService">
      <xsd:element name="sayHello">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1"
              name="name" nillable="true" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="sayHelloResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1"
              name="out" nillable="true" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="sayHelloResponse">
    <wsdl:part name="parameters" element="tns:sayHelloResponse" />
  </wsdl:message>
  <wsdl:message name="sayHelloRequest">
    <wsdl:part name="parameters" element="tns:sayHello" />
  </wsdl:message>
  <wsdl:portType name="HelloServicePortType">
    <wsdl:operation name="sayHello">
      <wsdl:input name="sayHelloRequest"
        message="tns:sayHelloRequest" />
      <wsdl:output name="sayHelloResponse"
        message="tns:sayHelloResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HelloServiceHttpBinding"
    type="tns:HelloServicePortType">
    <wsdlsoap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="sayHello">
```

```
        <wsdlsoap:operation soapAction="" />
        <wsdl:input name="sayHelloRequest">
            <wsdlsoap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="sayHelloResponse">
            <wsdlsoap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HelloService">
    <wsdl:port name="HelloServiceHttpPort"
      binding="tns:HelloServiceHttpBinding">
        <wsdlsoap:address
          location="http://localhost:8080/xfire/services/HelloService" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

types 元素使用 XML 模式语言声明在 WSDL 文档中的其他位置使用的复杂数据类型与元素；

import 元素类似于 XML 模式文档中的 import 元素，用于从其他 WSDL 文档中导入 WSDL 定义；

message 元素使用在 WSDL 文档的 type 元素中定义或在 import 元素引用的外部 WSDL 文档中定义的 XML 模式的内置类型、复杂类型或元素描述了消息的有效负载；

portType 元素和 operation 元素描述了 Web 服务的接口并定义了他的方法。portType 元素和 operation 元素类似于 java 接口和接口中定义的方法声明。operation 元素使用一个或者多个 message 类型来定义他的输入和输出的有效负载；

Binding 元素将 portType 元素和 operation 元素赋给一个特殊的协议和编码样式；

service 元素负责将 Internet 地址赋给一个具体的绑定；

1、definitions 元素

所有的 WSDL 文档的根元素均是 definitions 元素。该元素封装了整个文档，同时通过其 name 提供了一个 WSDL 文档。除了提供一个命名空间外，该元素没有其他作用，故不作详细描述。

下面的代码是一个 definitions 元素的结构：

```
<wsdl:definitions
  targetNamespace="http://com.liuxiang.xfireDemo/HelloService"
```

```
  xmlns:tns="http://com.liuxiang.xfireDemo/HelloService"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
</wsdl:definitions>
```

2、types 元素

WSDL 采用了 W3C XML 模式内置类型作为其基本类型系统。types 元素用作一个容器，用于定义 XML 模式内置类型中没有描述的各种数据类型。当声明消息部分的有效负载时，消息定义使用了在 types 元素中定义的数据类型和元素。在本文的 **WSDL** 文档中的 types 定义：

```
<wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      attributeFormDefault="qualified" elementFormDefault="qualified"
      targetNamespace="http://com.liuxiang.xfireDemo/HelloService">
      <xsd:element name="sayHello">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1"
              name="name" nillable="true" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="sayHelloResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1"
              name="out" nillable="true" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
```

上面是数据定义部分，该部分定义了两个元素，一个是 sayHello，一个是 sayHelloResponse：

sayHello：定义了一个复杂类型，仅仅包含一个简单的字符串，将来用来描述操作的参数传入部分；

sayHelloResponse：定义了一个复杂类型，仅仅包含一个简单的字符串，将来用来描述操作的返回值；

3、import 元素

import 元素使得可以在当前的 WSDL 文档中使用其他 WSDL 文档中指定的命名空间中的定义元素。本例子中没有使用 import 元素。通常在用户希望模块化 WSDL 文档的时候，该功能是非常有效果的。

import 的格式如下：

```
<wsdl:import namespace="http://xxx.xxx.xxx/xxx/xxx" location="http://xxx.xxx.xxx/xxx/xxx.wsdl"/>
```

必须有 namespace 属性和 location 属性：

namespace 属性：值必须与正导入的 WSDL 文档中声明的 targetNamespace 相匹配；

location 属性：必须指向一个实际的 WSDL 文档，并且该文档不能为空。

4、message 元素

message 元素描述了 Web 服务使用消息的有效负载。message 元素可以描述输出或者接受消息的有效负载；还可以描述 SOAP 文件头和错误 detail 元素的内容。定义 message 元素的方式取决于使用 RPC 样式还是文档样式的消息传递。在本文中的 message 元素的定义，本文档使用了采用文档样式的消息传递：

```
<wsdl:message name="sayHelloResponse">
    <wsdl:part name="parameters" element="tns:sayHelloResponse" />
  </wsdl:message>
  <wsdl:message name="sayHelloRequest">
    <wsdl:part name="parameters" element="tns:sayHello" />
  </wsdl:message>
```

该部分是消息格式的抽象定义：定义了两个消息 sayHelloResponse 和 sayHelloRequest：

sayHelloRequest：sayHello 操作的请求消息格式，由一个消息片断组成，名字为 parameters,元素是我们前面定义的 types 中的元素；

sayHelloResponse：sayHello 操作的响应消息格式，由一个消息片断组成，名字为 parameters,元素是我们前面定义的 types 中的元素；

如果采用 RPC 样式的消息传递，只需要将文档中的 element 元素应以修改为 type 即可。

5、portType 元素

portType 元素定义了 Web 服务的抽象接口。该接口有点类似 Java 的接口，都是定义了一个抽象类型和方法，没有定义实现。在 WSDL 中， portType 元素是由 binding 和 service 元素来实现的，这两个元素用来说明 Web 服务实现使用的 Internet 协议、编码方案以及 Internet 地址。

一个 portType 中可以定义多个 operation，一个 operation 可以看作是一个方法，本文中 WSDL 文档的定义：

```
    <wsdl:portType name="HelloServicePortType">
      <wsdl:operation name="sayHello">
        <wsdl:input name="sayHelloRequest"
          message="tns:sayHelloRequest" />
        <wsdl:output name="sayHelloResponse"
          message="tns:sayHelloResponse" />
      </wsdl:operation>
    </wsdl:portType>
```

portType 定义了服务的调用模式的类型，这里包含一个操作 sayHello 方法，同时包含 input 和 output 表明该操作是一个请求 / 响应模式，请求消息是前面定义的 sayHelloRequest，响应消息是前面定义的 sayHelloResponse。input 表示传递到 Web 服务的有效负载，output 消息表示传递给客户的有效负载。

6 、binding

binding 元素将一个抽象 portType 映射到一组具体协议(SOAO 和 HTTP)、消息传递样式、编码样式。通常 binding 元素与协议专有的元素和在一起使用，本文中的例子：

```
    <wsdl:binding name="HelloServiceHttpBinding"
      type="tns:HelloServicePortType">
      <wsdlsoap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="sayHello">
        <wsdlsoap:operation soapAction="" />
        <wsdl:input name="sayHelloRequest">
          <wsdlsoap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="sayHelloResponse">
          <wsdlsoap:body use="literal" />
        </wsdl:output>
      </
```



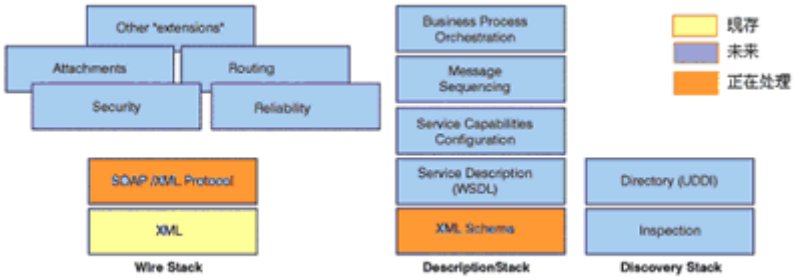

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/ws
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/
  xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap1
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://tempuri.org/" xmlns:wsdl="
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      - <s:element name="SearchUserName">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="useID" type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="SearchUserNameResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="SearchUserNameResult" type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
      + <s:element name="AddUser">
      + <s:element name="AddUserResponse">
      </s:schema>
    </wsdl:types>
  + <wsdl:message name="SearchUserNameSoapIn">
  + <wsdl:message name="SearchUserNameSoapOut">
  + <wsdl:message name="AddUserSoapIn">
  + <wsdl:message name="AddUserSoapOut">
  + <wsdl:portType name="SearchUserServiceSoap">
  + <wsdl:binding name="SearchUserServiceSoap" type="tns:SearchUserServiceSoap">
  + <wsdl:binding name="SearchUserServiceSoap12" type="tns:SearchUserServiceSoap">
  + <wsdl:service name="SearchUserService">
  </wsdl:definitions>
```

SOAP 介绍

随着计算机技术的不断发展，现代企业面临的环境越来越复杂，其信息系统大多数为多平台、多系统的复杂系统。这就要求今天的企业解决方案具有广泛的兼容能力，可以支持不同的系统平台、数据格式和多种连接方式，要求在

Internet 环境下，实现系统是松散耦合的、跨平台的，与语言无关的，与特定接口无关的，而且要提供对 Web 应用程序的可靠访问。

随着异种计算环境的不断增加，各种系统间的互操作性就愈显得必要，要求系统能够无缝地进行通信和共享数据，从而在 Internet 环境下，消除巨大的信息孤岛，实现信息共享、进行数据交换，达到信息的一致性。Web services 希望实现不同的系统之间能够用“软件-软件对话”的方式相互调用，打破了软件应用、网站和各种设备之间的格格不入的状态，实现“基于 WEB 无缝集成”的目标。



什么是 SOAP？

SOAP（Simple Object Access Protocol ）简单对象访问协议是在分散或分布式的环境中交换信息的简单的协议，是一个基于 XML 的协议，它包括四个部分：SOAP 封装(envelop)，封装定义了一个描述消息中的内容是什么，是谁发送的，谁应当接受并处理它以及如何处理它们的框架；SOAP 编码规则（encoding rules），用于表示应用程序需要使用的数据类型的实例；SOAP RPC 表示(RPC representation)，表示远程过程调用和应答的协定;SOAP 绑定（binding），使用底层协议交换信息。

虽然这四个部分都作为 SOAP 的一部分，作为一个整体定义的，但他们在功能上是相交的、彼此独立的。特别的，信封和编码规则是被定义在不同的 XML 命名空间(namespace)中，这样使得定义更加简单。

SOAP 的两个主要设计目标是简单性和可扩展性。这就意味着有一些传统消息系统或分布式对象系统中的某些性质将不是 SOAP 规范的一部分。比如：分布式垃圾收集 (Distributed garbage collection)、成批传送消息(Boxcarring or batching of messages)、对象引用 (Objects-by-reference（which requires distributed garbage collection）)、对象激活 (Activation（which requires objects-by-reference）)。

SOAP 消息举例：

1．第一个例子阐明了 SOAP 中一个简单的通信信息，包括了两个不是 SOAP 定义而是应用程序定义的元素：头块元素 alertcontrol 和体块元素 alert。头块元素包括两个参数：priority 和 expires。体块元素包括的是实际传送的信息。

（EXAMPLE 1）

```
<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

2．SOAP 通信与底层的不同协议和不同的交换格式有关，下面的例子 SOAP 使用 HTTP 作为底层通信协议，从而可以很好的使用 request/response 机制来传送信息。 SOAP/HTTP 请求包括一个 GetLastTradePrice 的块元素，该请求携带一个字符串参数和 ticker 符号，在 SOAP 应答中返回一个浮点数。XML 名域用来区分 SOAP 标志符和应用程序特定的标志符。

（EXAMPLE 2）

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://example.org/2001/06/quotes"

<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope" >
 <env:Body>
  <m:GetLastTradePrice
        env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
        xmlns:m="http://example.org/2001/06/quotes">
    <symbol>DIS</symbol>
  </m:GetLastTradePrice>
 </env:Body>
</env:Envelope>
```

3. 例 3 展示的是 StockQuote SOAP 服务信息，是对例 2 的请求作出的一条应答消息。（EXAMPLE 3）

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope" >
 <env:Body>
  <m:GetLastTradePriceResponse
        env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
        xmlns:m="http://example.org/2001/06/quotes">
   <Price>34.5</Price>
  </m:GetLastTradePriceResponse>
 </env:Body>
</env:Envelope>
```

SOAP 与 CORBA，COM/DCOM 的区别？

在 SOAP 刚刚提出来的时候，许多人就提出了疑问：SOAP 与 CORBA 和 DCOM 的区别何在？

• CORBA（Common Object Request Broker Architecture）公共对象请求代理体系结构是由 OMG 组织制订的一种标准的面向对象应用程序体系规范。由对象请求代理 ORB、对象服务、公共设施、域接口和应用接口这几个部分组成。其核心部分是对象请求代理 ORB（Object Request Broker）。ORB 提供了一种机制，通过这种机制，对象可以透明的发出请求和接收响应。分布的、可以互操作的对象可以利用 ORB 构造可以互操作的应用。ORB 可看作是在对象之间建立客户/服务关系的一种中间件。基于 ORB，客户可以透明的调用服务对象提供的方法，该服务对象可以与客户运行在同一台机器上，也可以运行在其他机器上通过网络与客户进行交互。ORB 截取客户发送的请求，并负责在该软件总线上找到实现该请求的服务对象，然后完成参数、方法调用，并返回最终结果。CORBA 1.1 由对象管理组织在 1991 年发布。定义了接口定义语言（IDL）和应用编程接口（API），从而通过实现对象请求代理（ORB）来激活客户/服务器的交互。CORBA 2.0 于 1994 年的 12 月发布定义了如何跨越不同的 ORB 提供者而进行通讯。

• COM/DCOM(Component Object Model / Distributed Component Object Model)是微软公司提出的分布式组件对象模型标准，支持在局域网、广域网甚至 Internet 上不同计算机的对象之间的通讯。DCOM 基于 COM 的应用程序、组件、工具等的基础之上，处理网络协议的低层次的细节问题，而不必关心太多的网络协议细节，从而使用户能够集中精力解决用户所要求的问题。DCOM 位于应用程序的组件之间，将组件以不可见的方式胶合在一起组成具有完整功能的应用程序。

SOAP 与 CORBA，DCOM/COM 的比较。

首先指出的是 SOAP 不会取代 CORBA，COM/DCOM，三者的概念有所区别。COM/DCOM 是个组件模型标准，CORBA 是分布式应用的服务标准。CORBA 和 DCOM 为分布式应用程序建立服务，服务对象来执行客户端调用的服务。而 SOAP 是基于 XML 和 HTTP 的分布式对象的通信协议，是 COM/DCOM 和 CORBA 对象进行通讯的协议。实际上，利用 SOAP 的互操作性和 CORBA 强大的执行能力，两者可以很好的结合在一起。 OMG （Object Management Group responsible for the CORBA specification）正在关注这方面的发展。

CORBA 应用程序和 DCOM 应用程序不能实现互操作，两者不能在一起协作。因为在 ORPC(Object RPC)协议中，用 ObjRef 代表了一个正在运行对象的引用；在 CORBA/IIOP(Internet Inter-Orb Protocol)中，用交换可互操作对象引用 IOR（Interoperable Object Reference）代表一个服务器的对象引用。不幸的是，IOR 与 ObjRef 不能够关联起来。然而，使用 SOAP 可以实现在垂直应用层面上 CORBA ，DCOM 技术的水平整合，能够更好的集成 CORBA，DCOM 为一个整体。

SOAP 并没有定义信息的语义，服务质量，基于 INTERNET 的事务处理。而是采用 XML 进行消息编码，正确的处理需要服务器和客户端本身来执行，理解和执行彼此使用的信息格式（ONE-TO-ONE，REQUEST/REPLY，BROADCAST，ETC），应用程序本身在语义解析中扮演着十分重要的角色。而 CORBA，DCOM 表示了传送信息的语义，对参数和返回值使用二进制编码。可对诸如参数名称或类型的任何元信息都不编码，但使中介很难处理消息。又因为每个系统使用不同的二进制编码，系统间的互操作的很难实现。

尽管 CORBA 可以在不同的平台上执行，DCOM 可以在微软的各种平台上运行，但是基于 CORBA 和 DCOM 的解决方案必须依赖于单一的应用程序。比如说，假如运行的是 DCOM 服务器程序，所有的分布式的客户端不得不运行于微软的操作平台上。CORBA 虽然可以运行于不同的平台，但 CORBA 的互操作性并没有在更高层的服务上进行扩展，如安全性和事务处理，在这种情况下，许多提供的服务没有得到很好的优化。DCOM 和 CORBA 适合于服务器--服务器间的通讯，但是对于客户端--服务器的通讯十分脆弱，尤其当客户程序分布在 INTERNET 上更是如此。

SOAP 不象 DCOM 一样试图定义分布式系统的所有元素，SOAP 没有提供分布式类库，类型安全检查，版本控制等等，SOAP 比它处于一个更低的层次，有点类似于 IIOP 在 CORBA 的作用，DCOM 却提供了一些额外的协议功能，是 IIOP 或者 SOAP 所不具备的。然而，许多. DCOM 的额外功能只有在服务器－－服务器间通信时才会用到，对于客户端－－服务器之间的通信则是多余的。

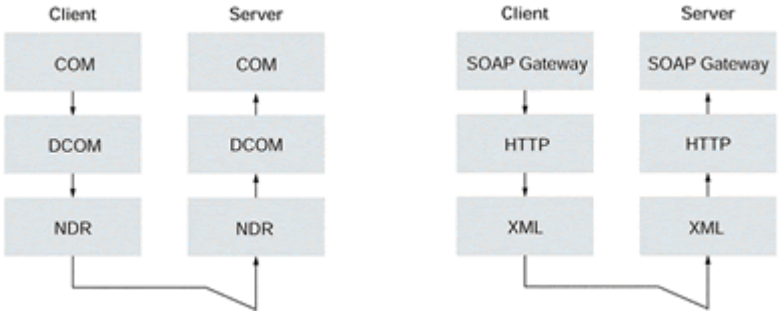
SOAP=RPC+HTTP+XML

SOAP 简单的理解，就是这样的一个开放协议 SOAP=RPC+HTTP+XML：采用 HTTP 作为底层通讯协议；RPC 作为一致性的调用途径，ＸＭＬ作为数据传送的格式，允许服务提供者和服务客户经过防火墙在 INTERNET 进行通讯交互。RPC 的描叙可能不大准确，因为 SOAP 一开始构思就是要实现平台与环境的无关性和独立性，每一个通过网络的远程调用都可以通过 SOAP 封装起来，包括 DCE（Distributed Computing Environment ） RPC CALLS，COM/DCOM CALLS, CORBA CALLS, JAVA CALLS，etc。

SOAP 使用 HTTP 传送 XML，尽管 HTTP 不是有效率的通讯协议，而且 XML 还需要额外的文件解析（parse），两者使得交易的速度大大低于其它方案。但是 XML 是一个开放、健全、有语义的讯息机制，而 HTTP 是一个广泛又能避免许多关于防火墙的问题，从而使 SOAP 得到了广泛的应用。但是如果效率对你来说很重要，那么你应该多考虑其它的方式，而不要用 SOAP。

为了更好的理解 SOAP,HTTP,XML 如何工作的，不妨先考虑一下 COM/DCOM 的运行机制，DCOM 处理网络协议的低层次的细节问题，如 PROXY/STUB 间的通讯，生命周期的管理，对象的标识。在客户端与服务器端进行交互的时候，DCOM 采用 NDR（Network Data Representation）作为数据表示，它是低层次的与平台无关的数据表现形式。

DCOM 是有效的，灵活的，但也是很复杂的。而 SOAP 的一个主要优点就在于它的简单性，SOAP 使用 HTTP 作为网络通讯协议，接受和传送数据参数时采用 XML 作为数据格式，从而代替了 DCOM 中的 NDR 格式，SOAP 和 DCOM 执行过程是类似的，如下图，但是用 XML 取代 NDR 作为编码表现形式，提供了更高层次上的抽象，与平台和环境无关。



客户端发送请求时，不管客户端是什么平台的，首先把请求转换成 XML 格式，SOAP 网关可自动执行这个转换。为了保证传送时参数，方法名，返回值的唯一性，SOAP 协议使用了一个私有标记表，从而服务端的 SOAP 网关可以正确的解析，这有点类似于 COM/DCOM

中的桩（STUB）。转化成 XML 格式后，SOAP 终端名（远程调用方法名）及其他的一些协议标识信息被封装成 HTTP 请求，然后发送给服务器。如果应用程序要求，服务器返回一个 HTTP 应答信息给客户端。与通常对 HTML 页面的 HTTP GET 请求不同的是，此请求设置了一些 HTTP HEADER，标识着一个 SOAP 服务激发，和 HTTP 包一起传送。例如：对于一个询问股票价格的应用程序，服务器端具有组件提供某股票当前的价格，组件是 COM 或 CORBA 在服务器上建立的。客户端发送一个 SOAP 请求给服务器询问股票价格。服务器依赖于服务器上的 SOAP 网关，使用内嵌的 HTML 对象调用合适的方法，然后把得到的价格通过 SOAP 应答传给客户端。

SOAP 的前景

- W3C 于 2000 年 5 月 8 日发表了 Simple Object Access Protocol (SOAP) 1.1 版本，具体规范发布在下列站点上(http://www.w3.org/TR/SOAP/)。又于 2001 年 7 月 9 号推出了 SOAP Version 1.2 版本的建议草案，具体规范发布在下列站点上（http://www.w3.org/TR/soap12/）。编写 SOAP Version 1.1 版本的工作小组的成员包括：DevelopMentor, International Business Machines Corporation, Lotus Development Corporation, Microsoft, UserLand Software。

SOAP 的推出是令人兴奋的。可以相信，随着网络服务的的不断发展，它将极大的改变我们的思考模式和开发模式。现在，已有许多大公司着手支持 SOAP 的开发，2000 年 IBM 公司 和 Microsoft 公司 都发行了实现 SOAP 的第一批版本。 IBM 公司启动了 Apache SOAP 项目计划，微软最近又推出了 SOAPtoolkit2.0 的正式版，主要包括如下的一些特征：SOAP 的高层接口和低层接口，消息对象接口，完全支持 WSDL 1.1 标准，支持用户自定义类型映射，并且提供了丰富和完整的开发文档以及应用实例。而且，两家公司正在互操作性方面努力研究。可以乐观的估计，不用多久，SOAP 互操作性的时代就将来临。

SOAP /WSDL/Web Service 关系

http://www.cnblogs.com/zernyree/archive/2009/05/27/1491179.html

以前也曾经写过简单的 WebService，但是并没有深入的研究，这两天看了园子里的一些文章，又请教了身边的高人，把 SOAP、Web Service 和 WSDL 的关系大概搞明白了，举例说明如下：

X 局有两个副局长 A 和 B，A 副局长分管财务，B 副局长分管计划生育，但是 A 副局长是上海人，B 副局长是广东人，两个人又都只会说自己家乡的方言，不会说普通话，这让下面的工作人员在请示汇报的时候非常困难，为了解决这个问题，局里的科员小 C 发明了一个表格，表格列出了需要向局长请示的问题以及说明这个问题所需要的数据等等（这个是 SOAP 的雏形），采用这种方式之后，**工作人员就可以在需要向领导请示工作的时候直接填写表格**，然后把表格交给相应的局长就可以了。后来 X 局的直属上级机关 Y 部的领导发现了这个方式，决定推广这种方式，将这种“表格描述问题”的方式升级为规范（或者称为“协议”），在所有 Y 部下属单位中执行，这个时候就是 SOAP 了。

后来 X 局又出现了新的问题，科员小 D 到分管计划生育的 B 局长那里领 TT（嘿嘿，YY 一下），但是 TT 没有了，需要进行采购，这个时候就需要与分管财务的 A 局长打交道了，由于小 D 对工作流程不太了解，不知道 A、B 副局长在这个问题上分别承担什么职责，又产生了困难。这时小 C 再次发挥了他的聪明才智，**他又发明了一种表格，这个表格专门说明某位领导分管什么工作，都有哪些职责（Web Service），在请示每个问题上应该提供什么样的材料和数据，会得到什么样的反馈，这个就是 WSDL 了。**
关于 Web Service 我们可以通过上面的例子清楚知道，就是一个做事情的在这里不再累赘。
通过上面的例子，对 SOAP/ WSDL/Web Service 关系说明如下：
Web Service 实现业务诉求：Web Service 是真正“办事”的那个。
WSDL 提供“能办的事的说明”：我想帮你的忙，但是我要告诉你我都能干什么，以及干这些事情需要的参数类型。
SOAP 提供“请求”的规范：你想让人家办事，总得告诉人家你想干什么吧，SOAP 就是定义这个“请求”的格式的，按照 SOAP 定义的“请求”格式“书写”请求就可以保证 Web Service 能够正确的解读你想让它

干什么以及你为它提供了什么参数。在这个请求中，你需要描述的主要问题有：向哪个 Web Service 发送请求，请求的参数类型、参数值、返回值类型。这些都“填写”完毕，也就完成了符合 SOAP 规范的 SOAP 消息。

以上是本人暂时对 SOAP/Web Service/WSDL 三者关系到目前为止的浅显理解，最后把两个缩写名词的英文给出来吧：

SOAP：Simple Object Access Protocol

WSDL：Web Services Description Language

用户大会在即，我抽出时间整理了些关于Server的东西，而这些东西是我在ESRI实习期间的一部分内容，可惜我只在ESRI实习了14天，在整理的过程中，除了后面几篇是我写的（也没有什么大的功能，只是为了说明一些问题而已），以及添加了几幅我觉得很有代表性的图片，其余全部来自互联网，很感谢那些愿意将自己的心得和我们分享的人，对文中的一些我认为重要的地方，不管是名词还是细节，我用红字强调，RIA技术的发展，也许ADF将成为过去，就在我做最后的整理的时候，我在听同事的讲座，她用HTML5演示了一个三维的东西，并且做了三维下的网络分析，很炫。 HTML5的出现也许会在Server这里引起新的技术，我们期待吧。上面这些Server的开发时给予9.2的，我手中有9.3.1的PDF和Flex的教程，我会将这些一并打包。祝大家进步，同时也祝我自己进步。

醉意人间

2009年9月8日