

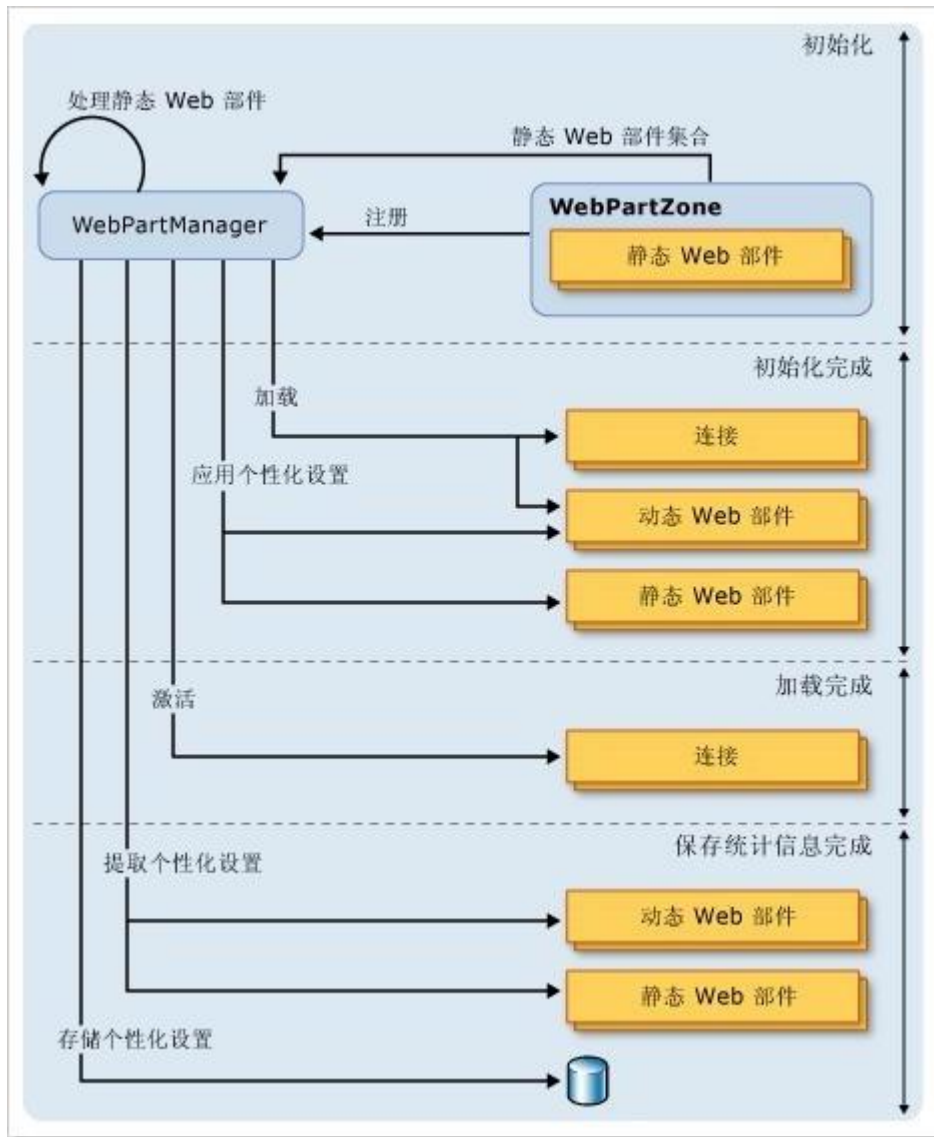
05 MOSS 中的 WebPart 开发 .

由于在 asp.net1.1 的时候 asp.net 中还没有 webpart 的概念，所以在 sps2003 中的 webpart 都是依赖于 Microsoft.SharePoint.dll 的，必须继承自命名空间为 Microsoft.SharePoint.WebPartPages 下的基类。到了 asp.net2.0，asp.net 将 sharepoint 的 webpart 集成到 asp.net 框架中，新的 ASP 风格 WebPart 依赖于 System.Web.dll，继承自不同的一个由 ASP.NET 2.0 定义的 WebPart 基类，其命名空间为 System.Web.UI.WebControls.WebParts。在 moss 中推荐使用 asp.net2.0 的 webpart。

在一个 ASP.NET 2.0 的应用中使用 WebPart，我们必须创建一个.aspx 页面并包括一个

WebPartManager 控件和至少一个 WebPartZone 控件。在 MOSS 中的 WebPart 是架构在一个名为 SPWebPartManager 的控件基础上的，他重写了 asp.net 中的 WebPartManager 的许多标准方法，同样 WebPartZone 我们也需要使用 Microsoft.SharePoint.WebPartPages 命名空间下的，这些其实都已经在默认的母版页都中了。

MOSS 本身也提供了很多开箱即用的 WebPart，例如 Content Editor WebPart, Data view WebPart, List View WebPart, Image WebPart, Members WebPart, Page viewer WebPart。下图为 WebPart 的声明周期。



然后我们就通过几个例子来说明 MOSS 中 WebPart 的相关开发：

一：上传附件到文档库

使用 asp.net 中的 FileUpload 控件，设置我们要上传的网站地址和列表的标题，代码如下：

```

using System;

using .....

namespace CaryWebPart
{
    [Guid("288802c4-4dfe-45b6-bb28-49dda89ec225")]

    public class FileUploadWP : System.Web.UI.WebControls.WebParts.WebPart

```

```

{
    FileUpload objFileUpload = new FileUpload();

    protected override void CreateChildControls()
    {
        Controls.Add(objFileUpload);

        Button btnUpload = new Button();

        btnUpload.Text = "Save File";

        this.Load += new System.EventHandler(btnUpload_Click);

        Controls.Add(btnUpload);
    }

    private void btnUpload_Click(object sender, EventArgs e)
    {
        using (SPSite objSite = new SPSite(SiteCollectionUrl))
        {
            using (SPWeb objWeb = objSite.OpenWeb(SiteUrl))
            {
                SPList objList = objWeb.Lists[ListName];

                if (objFileUpload.HasFile)
                {
                    try
                    {
                        objList.RootFolder.Files.Add(objFileUpload.FileName,
                            objFileUpload.PostedFile.InputStream, true);
                    }
                    catch(Exception ex)
                    {
                        string a = ex.Message;
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}

private string _strSiteCollectionUrl;

[Personalizable(PersonalizationScope.Shared), WebBrowsable(true),
 WebDisplayName("网站集 URL"), WebDescription("请输入网站集 URL")]

public string SiteCollectionUrl
{
    get { return _strSiteCollectionUrl; }
    set { _strSiteCollectionUrl = value; }
}

private string _strSiteUrl;

[Personalizable(PersonalizationScope.Shared), WebBrowsable(true),
 WebDisplayName("网站 URL"), WebDescription("请输入网站 URL")]

public string SiteUrl
{
    get { return _strSiteUrl; }
    set { _strSiteUrl = value; }
}

private string _strListName;

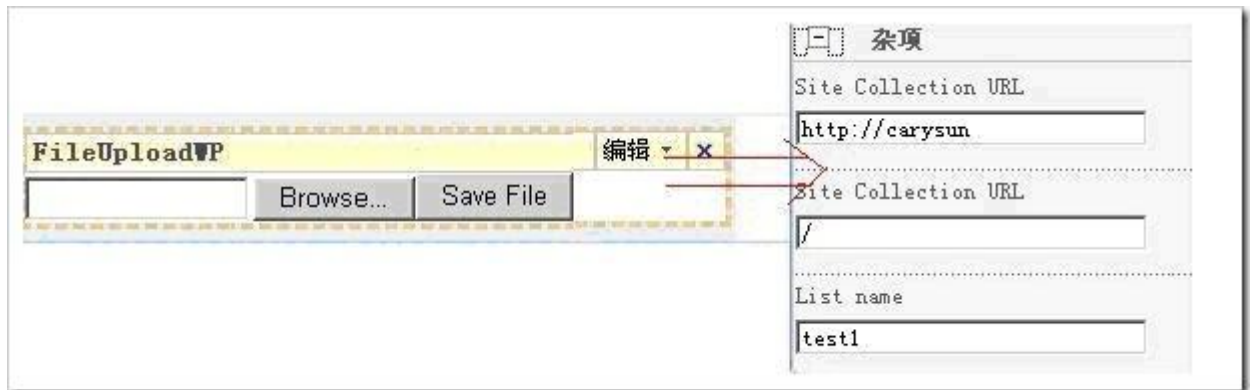
[Personalizable(PersonalizationScope.Shared), WebBrowsable(true),
 WebDisplayName("列表名称"), WebDescription("请输入列表名称")]

public string ListName
{
    get { return _strListName; }
    set { _strListName = value; }
}
}

```

}

下面是部署后的截图：



说明：

1. Personalizable 说明该属性是可配置的。
2. PersonalizationScope.Shared 设置该属性的是否所有人都可见，可以设置 Shared 和 User 两种。
- WebBrowsable(true) 表示在编辑部件里是可见的。
3. SPSite 的 OpenWeb 方法的一个重载方法的 URL 参数，是相对地址。
4. 例子中需要将站点的 web.config 的 `<trust level="Full" originUrl="" />` 设置为 Full 才可以上传附件。

二：Webpart 中使用资源

有两种方式来访问资源一种就是放在 _layouts 下，另一种就是 web 资源，它可以编译到程序集中去，我们主要说 Web 资源。首先新建一个项目，在项目中添加一个图片，并且在 AssemblyInfo.cs 中添加如下代码：

```
[assembly: WebResourceAttribute("WebResource.icons.recycbin.gif","image/gif")]
```

image/gif 代表资源的类型，如果是 js 则写成 text/javascript。

然后还需要创建一个虚拟的类，这个类不做任何动作，只是为了获取到资源的引用而用，下面是该类的代码：

```
using System;

using .....

namespace WebResource
{
    public class Placeholder
```

```
    {}  
}
```

用下面代码来获取 web 资源:

```
string strResource = "WebResource.icons.recycbin.gif";  
  
string strUrl = Page.ClientScript.GetWebResourceUrl(typeof(WebResource.PlaceHolder),  
strResource);
```

获取到的 url 如下形式 “/WebResource.axd?d=[Assembly key]&t=[last write time of resource assembly]”

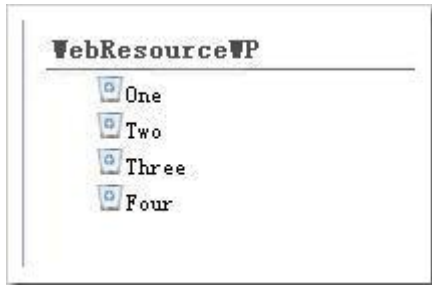
```
/WebResource.axd?d=A34Cgk8tSLs0aGW2Mtx1-iMMZgj-CEJ77updNPilqoda3ab0CWc1mJqVyolRohMp0&t=6  
33457672629813944
```

下面是使用 BulletedList 控件，并且将控件项的图标设置为我们的 web 资源中的图片，代码如下:

```
namespace CaryWebPart  
{  
    public class WebResourceWP : System.Web.UI.WebControls.WebParts.WebPart  
    {  
        protected override void CreateChildControls()  
        {  
            string strResource = "WebResource.icons.recycbin.gif";  
            string strUrl =  
Page.ClientScript.GetWebResourceUrl(typeof(WebResource.PlaceHolder), strResource);  
  
            BulletedList objBullist = new BulletedList();  
            objBullist.BulletStyle = BulletStyle.CustomImage;  
            objBullist.BulletImageUrl = strUrl;  
            objBullist.Items.Add("One");  
            objBullist.Items.Add("Two");  
            objBullist.Items.Add("Three");  
            objBullist.Items.Add("Four");  
            Controls.Add(objBullist);  
        }  
    }  
}
```

```
}  
}
```

下图是效果图:



三: WebPart 动作菜单

有三种类型:

1. Client-side verbs: 在客户端执行, 使用 `webpartverb` 类, 构造函数需要传两个参数, 第一个是 `verbs` 的 `id`, 第二个是点击的时的处理程序, 是 `js` 的程序。代码如下:

```
WebPartVerb objFirst = new WebPartVerb("FirVerbId", "javascript:alert('Hello Verb!');");  
  
objFirst.Text = "第一个 verb";  
  
objFirst.Description = "第一个 verb 信息描述";  
  
objFirst.ImageUrl = "_layouts/images/test/recycbin.gif";
```

2. Server-side verbs: 在服务器端执行。参数中的事件处理程序使用服务器端的代码。

```
WebPartVerb objSecond = new WebPartVerb("SecVerbId", new  
WebPartEventHandler(SecondVerbHandler));  
  
objSecond.Text = "第二个 verb";  
  
objSecond.Description = "第二个 verb 信息描述";
```

3. Both: 这种在服务器端和客户端都执行动作。可以使用 `WebPartVerbCollection`。

```
WebPartVerb[] objVerbs = new WebPartVerb[] {objFirst, objSecond};  
  
WebPartVerbCollection objVerbCollection = new WebPartVerbCollection(base.Verbs, objVerbs);
```

下面是完整代码:

```

using System;

using .....

namespace CaryWebPart
{
    [Guid("288802c4-4dfe-45b6-bb28-49dda89ec225")]
    public class VerbsWP: System.Web.UI.WebControls.WebParts.WebPart
    {
        public override WebPartVerbCollection Verbs
        {
            get
            {
                WebPartVerb objFirst = new WebPartVerb("FirVerbId",
"javascript:alert('Hello Verb!');");
                objFirst.Text = "第一个 verb";
                objFirst.Description = "第一个 verb 信息描述";
                objFirst.ImageUrl = "_layouts/images/test/recycbin.gif";

                WebPartVerb objSecond = new WebPartVerb("SecVerbId", new
WebPartEventHandler(SecondVerbHandler));
                objSecond.Text = "第二个 verb";
                objSecond.Description = "第二个 verb 信息描述";

                WebPartVerb[] objVerbs = new WebPartVerb[] { objFirst, objSecond };
                WebPartVerbCollection objVerbCollection = new
WebPartVerbCollection(base.Verbs, objVerbs);

                return objVerbCollection;
            }
        }
    }
}

```

```

protected void SecondVerbHandler(object sender, WebPartEventArgs args)
{
}
}
}

```

如下图:



四：创建编辑部件

所有的编辑部件都集成自 `EditorPart` 类，需要编辑的 `webpart` 的代码如下：

```
using System;
```

```
using .....
```

```

namespace CaryWebPart
{
    [Guid("288802c4-4dfe-45b6-bb28-49dda89ec225")]
    public class ForEditWP: System.Web.UI.WebControls.WebParts.WebPart
    {
        protected override void RenderContents(HtmlTextWriter writer)
        {
            writer.Write("TestValue: " + TestValue);
        }

        public override EditorPartCollection CreateEditorParts()
        {
            EditWP objEditor = new EditWP();

```

```

objEditor.ID = ID + "testEditor1";

objEditor.Title = "Test Editor Title";

objEditor.ToolTip = "Test Editor tooltip";

objEditor.TabIndex = 100;

objEditor.GroupingText = "Test editor grouping text";

ArrayList objEditorParts = new ArrayList();

objEditorParts.Add(objEditor);

EditorPartCollection objEditorPartsCollection = new
EditorPartCollection(objEditorParts);

return objEditorPartsCollection;
}

private string strNormalValue = String.Empty;

[Personalizable(PersonalizationScope.Shared), WebBrowsable(false),
WebDisplayName("Normal value"), WebDescription("Normal value description")]

public string TestValue
{
    get { return strNormalValue; }
    set { strNormalValue = value; }
}
}
}

```

CreateEditorParts 方法：创建要与 **WebPart** 控件关联的自定义 **EditorPart** 控件的实例。当用户单击 **WebPart** 控件上的编辑动作时，将调用该方法。然后就是开发上面 **Webpart** 对应的编辑部件，代码如下：

```

using System;

using .....

namespace CaryWebPart
{

```

```

public class EditWP : EditorPart
{
    TextBox txtNormalBox;

    protected override void CreateChildControls()
    {
        txtNormalBox = new TextBox();
        txtNormalBox.ID = "txtNormalBox";
        txtNormalBox.Text = "[Custom editor part]";
        txtNormalBox.TextMode = TextBoxMode.MultiLine;
        txtNormalBox.Rows = 5;
        Controls.Add(txtNormalBox);
    }

    public override bool ApplyChanges()
    {
        ForEditWP objNormal = (ForEditWP)WebPartToEdit;
        objNormal.TestValue= txtNormalBox.Text;

        return true;
    }

    public override void SyncChanges()
    {
        EnsureChildControls();

        ForEditWP objNormal = (ForEditWP)WebPartToEdit;
        txtNormalBox.Text = objNormal.TestValue;
    }
}
}

```

ApplyChanges 方法：将用户输入到 **EditorPart** 控件中的值保存到 **WebPartToEdit** 属性中引用的 **WebPart** 控件的相应属性中。

SyncChanges 方法和上面的方法是对应的，他负责 EditorPart 控件中的值始终与关联 WebPart 控件中的值保持同步。

如下图：



五：部署

MOSS 中部署一个 webpart 的大致流程：

1.搭建好开发环境，建立 webpart 工程，写代码。

2.修改 assembly.cs 文件

在部署前，需要修改 assembly 文件，增加以下两句：

```
using System.Security;
```

```
[assembly: AllowPartiallyTrustedCallers]
```

如果不进行以上修改，在安装 WebPart 时，会提示失败。

3.复制文件

将编译后的 Dll 复制到 Web 应用程序目录下的 bin 目录下。Web 应用程序的文件夹位置类似以下路径：

```
C:\Inetpub\wwwroot\wss\VirtualDirectories\80\bin 。
```

4.修改 web.config

WebPart 使用前，需要修改 Web 应用程序的配置文件。

(1)增加 SafeControls 中增加一行，类似下面的写法。

```
<SafeControl Assembly="HelloWebPart" Namespace="HelloWebPart" TypeName="*"
Safe="True" AllowRemoteDesigner="True" />
```

(2)修改信任级别。

将`<trust level="WSS_Minimal" originUrl="" />`改为`<trust level="WSS_Medium" originUrl="" />` 也可以将 WSS_Medium 改 Full。

5.在网站中增加 webpart(网站操作--网站设置--web 部件--新建)

六: 调试

调试也没什么好说的, 附加 `w3wp.exe` 进程, 然后设置断点就可以了