**[asp.net web.config的学习笔记](http://www.cnblogs.com/Bulid-For-NET/archive/2013/01/11/2856632.html)**

http://www.cnblogs.com/Bulid-For-NET/archive/2013/01/11/2856632.html

一直都对web.config不太清楚。这几天趁着项目不紧赶紧再恶补下,发现确实是有很多原来不明白的地方。特意记录下来,希望能与各位看官共同进步。

小弟自学笔记，有不对的地方还请大神指出来。

学习之前,我是抱着一个完全不知道的态度开始的。我想这个方法适用于很多地方,不是有个故事吗:杯子里的水倒掉才能装进去更多的水。所以有很多地方记录的都是挺基础的知识，这也能照顾照顾新手。

**1,什么是web.config?**

MSDN上说:"ASP.NET 配置数据存储在 XML 文本文件中（每个文件都名为 Web.config）。 Web.config 文件可以出现在 ASP.NET 应用程序的多个目录中。"。

在园子里有为大神的说话我觉得也计较容易理解“web.config是asp.net中保存配置信息（比如数据库连接字符串等）的重要文件。它是基于xml的文本文件方式放在Web应用程序的任何目录中，并且默认不随源文件编译到Dll中，而运行环境随时监视着它是否有改变，一但有变动，系统会自动重新加载里面的最新内容。”。

说到底就是网站的配置文件了。

**2,web.config的加载顺序**

前边既然提到了web.config可以在一个网站里出现多个,那在网站运行时候服务器是怎样去执行他们的呢?这就是要了解到web.config的加载顺序了。

相信很多新手特别是没有好好接触过web.config知识的童鞋一直都对一个asp.net网站有很多个配置文件感到棘手,即新奇又感到迷茫,反正我以前是这样的。但是现在知道了下面这些知识就不会了。

IIS在Asp.net网站启动时，会加载配置文件中的配置信息，然后缓存这些信息，不会每次要用都去读取配置文件，只是IIS会随时监视着这些文件的变化，一量有变化，它会重新去读取并缓存配置信息。

Asp.net网站运行时会按照以下方式加载配置文件中的节点信息：

1) 如果在当前运行页面所在的目录下有web.config文件，则查找是否存在所需要的节点，如果存在则返回结果，并停止下一步地查找。

2) 如里所在目录不存在web.config配置或者配置文件里没有所需要的节点，则查找它所在的上一级目录的配置文件中的节点，直到网站根目录。

3) 如果网站根目录中都不存在web.config或者所需要的配置节点，转而到“windows目录\Microsoft.NET\Framework\对应.net版本\config\web.config”中去查找。

4) 如果第3条中还没找到，继续到“windows目录\Microsoft.NET\Framework\对应.net版本\config\machine.config”中去查找。

5) 如果还没找到，那就报错吧。

可能对于第 3 步, 第 4 步有好多童鞋该不明白了,那个是 iis服务器的默认配置目录。可以这样简单的理解:我们让网站去找一个配置信息,那他肯定就会一层一层的去找配置文件里的节点名称,当网站目录里没有找到的时候,服务器会找自己的配置文件。当这些都没找到,报错吧。

这样理解了,就对一个asp.net网站拥有多个配置文件不足为奇了吧。其实对于上边说的这个问题,我觉得是在项目开发过程中养成良好的编程习惯就可以完全避免了。在配置文件的表示字段上,尽量不要有重复的。

**3,web.config节点介绍。**

这个才是正文吖。因为web.config节点有很多需要知道的地方,我们这里只对那些常用的节点做介绍。

前边说了,Web.config文件是一个xml文本文件。所以它的配置信息都是以节点形式存在的。

根节点为<configuration>，常见的子节点有：<configSections> ,<appSettings> <connectionStrings>,<location>,<system.web>等等。

**<configSections />**

这是什么玩意?

我们平时在网站里用到的一些对象或者数据,是配置在web.config里的,但是配在哪呢?就是配置在这个configSections节点下。

web.config是网站运行在iis服务器上的配置文件,他怎么可能允许我们随随便便的配置我们自己的节点呢,不行你自己试试, 在任何位置添加任何没有申明的节点, 系统都不会让你通过, 更不会让你去读它了, 当然, 你打算在别的xml文件里添加节点, 然后读出来, 创建对象, 这个没问题。不过这样一样也就失去了配置文件的意义了。所以,为了系统能有组织的管理用户的在配置文件里的自定义信息, 就要使用configSections了。

configSections 元素指定了配置节和处理程序声明。由于 ASP.NET 不对如何处理配置文件内的设置作任何假设，因此这非常必要。但 ASP.NET 会将配置数据的处理委托给配置节处理程序。配置结构信息如下：

<configSections>

<!--定义配置节处理程序与配置元素之间的关联。-->

<section />

<!--定义配置节处理程序与配置节之间的关联。-->

<sectionGroup />

<!--移除对继承的节和节组的引用。-->

<remove />

<!--移除对继承的节和节组的所有引用，只允许由当前 section 和 sectionGroup 元素添加的节和节组。-->

<clear/>

</configSections>

每个 section 元素标识一个配置节或元素以及对该配置节或元素进行处理的关联 ConfigurationSection 派生类。可以在 sectionGroup 元素中对 section 元素进行逻辑分组，以对 section 元素进行组织并避免命名冲突。section 和 sectionGroup 元素包含在 configSections 元素中。

如果配置文件中包含 configSections 元素，则 configSections 元素必须是 configuration 元素的第一个子元素。

下面我们来示例写一个自定义配置信息，并完成它的SectionHandler.

1,首先我们在<configuration>节点下添加configSections。

<configSections>  
    <sectionGroup name="WebSiteInfo">  
        <section name="basicInfo" type="ConfigurationSectionTest.WebSiteInfoHandler"/>  
        <section name="fileUpload" type="ConfigurationSectionTest.WebSiteInfoHandler"/>  
    </sectionGroup>  
</configSections>

<WebSiteInfo>  
    <basicInfo>  
        <add key="name" value="imecc studio"/>  
        <add key="version" value="1.0"/>  
    </basicInfo>  
    <fileUpload>  
        <add key="fileUploadPath" value="E:\\MyHomePage\\Web\\Upload\\"/>  
        <add key="fileUploadSizeMax" value="2M"/>  
    </fileUpload>  
</WebSiteInfo>

说明: 在configSections下添加一个section/sectionGroup子节点, 写好标示名和类型, 然后在紧接这下面实现这个标示名的细节内容,就是我们当初想添加的自己的自定义信息了.

以上我们在WebSiteInfo节点下定义了两个节点basicInfo和fileUpload，并定义了节点处理程序类ConfigurationSectionTest.WebSiteInfoHandler，并且随后运用了我们定义的节点。

2,我们来看看节点处理程序ConfigurationSectionTest.WebSiteInfoHandler。

任意建立一个项目，新建一个类，或者直接在App\_Code里新建一个类，如下:

|  |
| --- |
| namespace ConfigurationSectionTest  {      /// <summary>      ///WebSiteInfoHandler 的摘要说明      /// </summary>      public class WebSiteInfoHandler : IConfigurationSectionHandler      {          public WebSiteInfoHandler()          {              //              //TODO: 在此处添加构造函数逻辑              //            }            #region IConfigurationSectionHandler 成员            public object Create(object parent, object configContext, System.Xml.XmlNode section)          {             //这里我们首先返回个hello，并且在此处设置一个断点。看看程序什么时候执行到这。              return "hello";          }            #endregion      }    } |

可以看到这个处理程序实现了一个叫做IConfigurationSectionHandler的接口,并且里边有个唯一的方法 public object Create(object parent, object configContext, XmlNode section), 这里的最后一个参数就是自定义信息那个节点了,  通过处理你配置的这堆信息, 返回你要的对象.

3,要使用这个返回的对象,通过System.Configuration.ConfigurationManager.GetSection()方法.

  我们在Default.aspx的Page\_Load事件处理程序中去访问我们自定义的节点，用ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo");注意参数的格式。

这里是 section Group Name/section ,如果是单独的section,就只传这个section的name就好了。

protected void Page\_Load(object sender, EventArgs e)  
{  
    Object o = ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo");  
}

在Page\_Load的代码段里打个断点我们启动调试，看到程序首先执行到ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo");这句。

然后执行到ConfigurationSectionTest.WebSiteInfoHandler中的Create函数。

我们再看看这时处理函数中参数的值：

parent为null

configContext 为配置上下文对象。

section 的InnerXml为<add key="name" value="huchen's homepage" /><add key="version" value="1.0" />

按F11继续执行return "hello", 继续执行...

在执行到Object o = ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo")后面的“}“，我们发现o的值为”hello”。

相信您已经明白的差不多了，当读取自定义节点的内容时，程序去执行我们定义的节点处理程序，并把节点中的内容传给Create函数中的参数。然后我们在Create中自己处理节点下的内容，并返回我们格式化后的节点内容给调用者，也就是ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo")。

注意:程序第一次运行的时候可以调试到Create这个方法,第二次运行的时候就调试不到了,这是因为配置文件时缓存了起来.

只有改变了配置文件或者继承自IConfigurationSectionHandler的类才会重新调用.

好，知道这些以后，我们就来完善我们的代码，我们在Create中处理传进来的节点内容。

为了简单起见，我们引入两个类NameValueCollection，NameValueSectionHandler。

NameValueCollection：表示可通过键或索引访问的关联 String 键和 String 值的集合。

NameValueSectionHandler：提供配置节中的名称/值对配置信息。NameValueSectionHandler 这个类也继承IConfigurationSectionHandler  
反编译可以看出NameValueSectionHandler 的Create方法把参数section的结果转化成了一个集合，就是NameValueCollection。

那么我们可以在节点处理程序中的Create函数中写如下代码：

NameValueCollection configs;  
NameValueSectionHandler baseHandler = new NameValueSectionHandler();  
configs =(NameValueCollection)baseHandler.Create(parent,configContext,section);  
Return configs;

这样我们就可以这样访问我们的节点了：

string myWebSiteName = ((NameValueCollection)ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo"))["name"];

在Default.aspx的Page\_Load事件处理程序中添加如下代码：

string myWebSiteName = ((NameValueCollection)ConfigurationSettings.GetConfig("WebSiteInfo/basicInfo"))["name"];  
Response.Write(myWebSiteName);

Ctrl+F5运行，可以看到页面输出了imecc studio

**<appSettings />**

该节点下主要用来存储asp.net应用程序的一些配置信息，也可以把数据库连接字符串也放在这里，不过.net2.0提供了connectionStrings节点，所以数据库连接字符串还是不建议放在这里，下面为一个图片类型的实例。

<appSettings>

<!--图片类型扩展名-->

<add key="ImgType" value=".bmp;.jpg;.gif;.png"/>

</appSettings>

调用方法为:

string ImgType = ConfigurationManager.AppSettings["ImgType"];

**<connectionStrings />**

connectionStrings和appSettings类似，不过用于保存配置数据库连接信息，下面给一个实例。

<connectionStrings>

<add name="SqlserverConnStr" connectionString="Data Source=Aron1;Initial Catalog= pubs;UserId=sa;Password=asdasd;"/>

<add name="OrcleConnStr" connectionString="Provider=msdaora;Data Source= MyOracleDB;UserId=UserName;Password=asdasd;"/>

</connectionStrings>

调用方式为：

string connStr = ConfigurationManager.ConnectionStrings["SqlserverConnStr"].ConnectionString;

**<system.web />**

<system.web>为.net应用程序的行为方式配置节点，该节点包含很多子节点，很多子节点已经由.net配置好了，这里我们只来看看一些重要的配置节点。

**<compilation />节点**  
<compilation>节点配置 ASP.NET 使用的所有编译设置。

默认的debug属性为“true”，即允许调试，在这种情况下会影响网站的性能，所以在程序编译完成交付使用之后应将其设为“false”。

**<customErrors />**

该节点用于定义一些自定义错误信息。

此节点有Mode和defaultRedirect两个属性，其中defaultRedirect属性是一个可选属性，表示应用程序发生错误时重定向到的默认URL，如果没有指定该属性则显示一般性错误。Mode属性是一个必选属性，它有三个可能值，它们所代表的意义分别如下：  
Mode 说明  
On 表示在本地和远程用户都会看到自定义错误信息。  
Off 禁用自定义错误信息，本地和远程用户都会看到详细的错误信息。  
RemoteOnly 表示本地用户将看到详细错误信息，而远程用户将会看到自定义错误信息。  
这里有必要说明一下本地用户和远程用户的概念。当我们访问asp.net应用程时所使用的机器和发布asp.net应用程序所使用的机器为同一台机器时成为本地用户，反之则称之为远程用户。在开发调试阶段为了便于查找错误Mode属性建议设置为Off，而在部署阶段应将Mode属性设置为On或者RemoteOnly，以避免这些详细的错误信息暴露了程序代码细节从而引来黑客的入侵。

值得一提的是:如果说配置的跳转页面也发生错误,那么网站就没办法处理异常了。

**<authentication />**

该节点为配置 ASP.NET 身份验证方案，该方案用于识别查看 ASP.NET 应用程序的用户。Mode属性包含四种身份验证模式：

1. Windows(默认)

将 Windows 验证指定为默认的身份验证模式。将它与以下任意形式的 Microsoft Internet 信息服务 (IIS) 身份验证结合起来使用：基本、摘要、集成 Windows 身份验证 (NTLM/Kerberos) 或证书。在这种情况下，您的应用程序将身份验证责任委托给基础 IIS。

2. Forms

将 ASP.NET 基于窗体的身份验证指定为默认身份验证模式。

3. Passport

将 Microsoft Passport Network 身份验证指定为默认身份验证模式。

4. None

不指定任何身份验证。您的应用程序仅期待匿名用户，否则它将提供自己的身份验证。

下面的代码示例演示如何为基于窗体的身份验证配置站点、指定传输来自客户端的登录信息的 Cookie 的名称以及指定当初始身份验证失败时使用的登录页的名称。必须将 authorization 节包含在内才能要求对所有用户进行 Forms 身份验证，并拒绝匿名用户访问站点。

<configuration>

<system.web>

<authentication mode="Forms">

<forms name="401kApp" loginUrl="/login.aspx"/>

</authentication>

<authorization>

<deny users="?"/>

</authorization>

</system.web>

</configuration>

Login.aspx中登陆通过：

FormsAuthentication.RedirectFromLoginPage(this.TextBox1.Text, true);

**<httpHandlers />**

HttpHandlers可用于根据请求中指定的 URL 和 HTTP 谓词将传入的请求映射到相应的处理程序。也就是说对于网站的某一个请求,我们可以自定义他的一般处理程序。

相信有很多新手在刚刚入职的时候,会发现很多在学校学的知识(很多都是托控件)到了公司基本上就没有用的地方,打开aspx对应的aspx.cs后置文件,里边一句自己写的代码都找不到。

很是郁闷。其实很多都是用到了这个东西。

看下边这个例子:

<system.web>  
<httpHandlers>  
<add path="httpHandlerTest.ashx" verb="\*" type="aspDoNetTest.Handler.httpHandlerTest\_Handler"/>  
</httpHandlers>  
</system.web>

我们配置了一个Test的handler的一般处理程序,用于实现自定义的用户登录。

传统的我们在处理的时候,会在页面上托两个textbox，再来个Button,然后再Button的click event里写登录的逻辑。

但是这样配置的一个,当我们请求httpHandlerTest.ashx的时候,就会走我们的处理程序。

|  |
| --- |
| namespace aspDoNetTest.Handler  {      public class httpHandlerTest\_Handler:IHttpHandler      {          public void ProcessRequest(HttpContext context)          {              string action =context.Request.QueryString["action"];                switch (action)              {                  case "Login":                      Login(context);                      break;                  case "1":                      break;                  case "2":                      break;                  default:                      break;              }          }            private void Login(HttpContext context)          {              string UserName = context.Request.QueryString["UserName"];          }            public bool IsReusable          {              get              {                  return true;              }          }      }  } |

　　看到*httpHandlerTest\_Handler*类实现了*IHttpHandler*接口。关于这个接口的知识,可以在网上找到他的专一学习知识。

**<httpModules />**

   大家对于Global.asax应该挺熟悉的吧。对于那里边的HttpApplication时间,有些时候可能并不是我们想要的。这时候httpModules就用上了。

对于httpModules,我们可以理解为建立一个通用的可用被插入任何Web应用程序的HttpApplication事件钩子。

使用HttpModule,是可复用的,不需要特定语应用程序代码的,只需要web.config中的一个条目。

举个例子:实现在每个页面上添加了一句话：Add BeginRequest by MyHttpModule!

<system.web>

<httpModules>  
<add name="MyHttpModule" type="aspDoNetTest.Handler.HttpModule"/>  
</httpModules>

</system.web>

|  |
| --- |
| namespace aspDoNetTest.Handler  {      public class HttpModule:IHttpModule      {          public void Dispose()          {            }            public void Init(HttpApplication context)          {              context.BeginRequest += new EventHandler(context\_BeginRequest);          }          void context\_BeginRequest(object sender, EventArgs e)          {              HttpApplication application = (HttpApplication)sender;              HttpContext context = application.Context;              context.Response.Write("Add BeginRequest by MyHttpModule!");          }      }  } |

　　再运行网站,不论请求任何一个页面,都会在网站一开始请求的时候,再页面上输出 *Add BeginRequest by MyHttpModule!*

***<location />***

Location节点是用来指定子配置的资源。如果在asp.net应用程序中想对某个目录做特殊处理，则可以用该节点来实现.举个例子：

下面的代码示例演示如何仅将指定页的上载文件大小限制设置为 128 KB。

<configuration>

<location path="UploadPage.aspx">

<httpRuntime maxRequestLength="128"/>

</location>

</configuration>

**ps:以上所有代码示例我会在后边上传上来供同学们学习。**

**感谢!**

分类: [.NET\_C#](http://www.cnblogs.com/Bulid-For-NET/category/343032.html),[学习笔记](http://www.cnblogs.com/Bulid-For-NET/category/346595.html)

标签: [.NET](http://www.cnblogs.com/Bulid-For-NET/tag/.NET/), [web.config](http://www.cnblogs.com/Bulid-For-NET/tag/web.config/)