**ASP.NET WebForm**

**Web Form的构成**

**当我们新建一个WebForm文件的时候，其文件构成和WinForm项目类似，分为了以下三个部分**

1. 布局页：\*.aspx

　　-> \*.aspx为WebForm项目中的一个布局页面，在该布局页中所编写的代码是多样化的，我们可以编写标准的HTML5 规范的代码、使用ASP.NET服务端控件和通过[ASP.NET内联表达式](ASP.NET内联表达式.docx)来嵌入C#代码

-> 在布局页上所编写的代码并不是最终呈现在用户浏览器上的HTML源代码，在用户每次访问一个我们所建立并已经部 署至服务器上的aspx网页的时候，\*.aspx布局页会根据目前文件上的代码重新进行编译并生成一个临时的、继承于 \*.aspx.cs的类【aspx网页文件名\_aspx】，它实现了IHttpHandler接口，在一个网页请求到达至管道的时候，该类中所 实现的ProcessRequest就是当前WebForm整个页面事件生命周期的主入口，这里有更多关于【[ASP.NET ProcessModel](../ASP.NET基础/ASP.NET Process Model.docx)】 和【[ASP.NET WebForm页面生命周期](ASP.NET WebForm 页面生命周期.docx)】的描述，该篇文档不做过多的描述

1. 代码后置文件：\*.aspx.cs

　　-> 每一个aspx网页中的 \*.aspx 都有一个它所对应的代码后置文件 \*.aspx.cs，它们之间所持有关系的声明是通过一段 [ASP.NET文本模板指令](ASP.NET内联表达式.docx)来确认的

-> 该文件是一个存C#语言所编写的文件，我们除了能够使用C#进行一些相关的功能性处理外，还能在该类中为 WebForm事件的生命周期施以控制

1. ASP.NET控件设计文件：\*.aspx.designer.cs

　　-> 该文件为 \*.aspx.cs 的一个partrial类，我们所使用的ASP.NET服务端控件的实例都会在这个类中进行声明和实例化， 其形式类同WinForm的designer文件，以下不作过多的阐述

**关于Web Form的几个知识点**

**在使用WebForm进行项目开发的时候，有几个关键性的知识点需要我们进行记忆和理解**

1. ViewState

**-> ViewState是什么？**

1. WebForm为了解决HTTP协议的无状态性，其提供了一种页面状态保持机制【ViewState】，每一次的aspx页面的访问，当流程走到【SaveAllState函数】，会为当前所初始化好的 **ASP.NET页面控件树** 进行base64编码，并把编码后的内容生成一段名为【\_\_VIEWSTATE】的Input隐藏域，并把内容加入至【ASP.NET Form】标签当中，当用户在当前页面进行操作并且在基于【ASP.NET Form】标签中提交数据至服务器的时候，【\_\_VIEWSTATE】隐藏域中我们上一次所转码 的内容也会一并提交至服务器，当流程走到【LoadAllState函数】的时候，会把所提交上来的【\_\_VIEWSTATE】隐藏域的内容进行base64解码，然会通过【ProcessPostData函数】把解码后的ASP.NET页面控件树内容逐一填充至相应的控件当中，通过这一机制来保证用户当前页面的会话状态的保持

微信截图_20190602184214

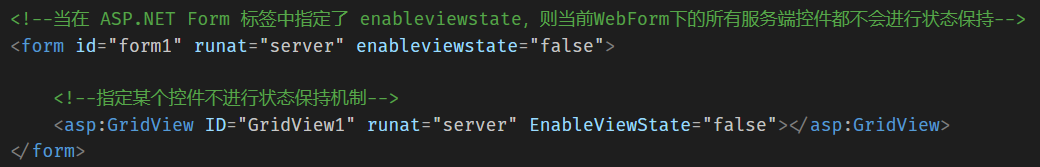
**-> ViewState所带来的一些问题**

1. 当我们在WebForm中使用的控件内容显示过多，那么每一次为ASP.NET页面控件树的内容进行base64编码的时候，字符串的长度都会显示的过长，这时候内容在网络传输的过程中就浪费了很多时间，且数据返回给浏览器客户端进行解析的时候也要耗费很长的时间



**-> 如何禁用【\_\_VIEWSTATE】**

1. 通过ASP.NET控件的属性【enableviewstate】来控制当前所指定的控件是否需要进行状态保持，需要注意的，是当我们在【ASP.NET Form】标签当中指定了这个属性，则其作用范围引用至当前aspx网页下的所有控件，而如果具体的指定某个控件的这个属性，则作用范围只是我们所具体指定的某个控件
2. 彻底根除掉ASP.NET服务端控件，也就是Page页上不能有一个标签带有【runat="server"】，这样就可以把【\_\_VIEWSTATE】彻底的在前端的HTML代码跟除掉，但相反的，如果我们想要这么操作的话，我们也就不能够使用【ViewState】这个对象进行存储一些额外的信息了



**-> 使用我们自定义的ViewState进行当前页面的状态保持**

1. 在 \*.aspx.cs 的最终父类【Control】类当中，有一个【ViewState】的属性能提供我们实现这一机制，该类型所存储的方式是基于键值对集合的方式进行存储
2. 使用【ViewState】属性的时候需要注意的一个问题，这种状态保持机制只能是针对当前aspx页面的，并且当浏览器关闭的时候状态保存机制也会随之失效

微信截图_20190602190111

1. IsPostBack

**-> IsPostBack是什么？**

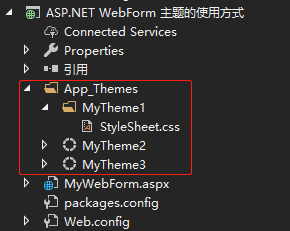
1. 该属性属于 \*.aspx.cs 的父类【Page】类当中的属性，我们可以通过这一属性来判断当前页面的请求是否为第一次请求
2. 他的存在是基于【ViewState】隐藏域的，也就是说当我们在一个aspx页面关闭了【ViewState】隐藏域的机制的话，该属性也会随之失效

**-> IsPostBack是如何实现的**

1. 每一次访问一个aspx页面的时候，当流程走到【DeterminPostBackMode函数】的时候，会判断当前所提交上来的数据是否包含【\_\_VIEWSTATE】数据，如果存在的话就证明当前不是第一次访问页面，【IsPostBack】属性则为true，反之则为false
2. Thems主题

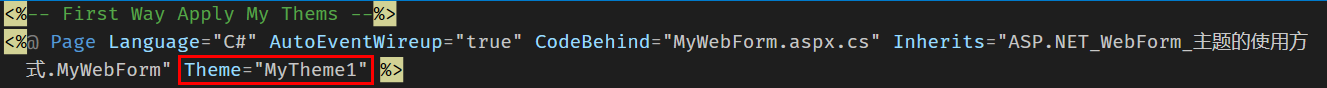
**-> Thems的介绍**

1. 在WebForm中，我们可以使用所指定的主题样式去应用于我们的Page页，主题样式说白了也就是一个CSS文件，但是我们可以把他们都统一放在一个【App\_Themes】文件夹下，当我们为当前WebForm设置了相应的主题，生命周期流程走到【Page\_PreInit】这一步之后，调用【[InitializeThemes](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://System.Web:4.0.0.0:b03f5f7f11d50a3a/System.Web.UI.Page/InitializeThemes()" \o )】去动态的设置我们所的主题

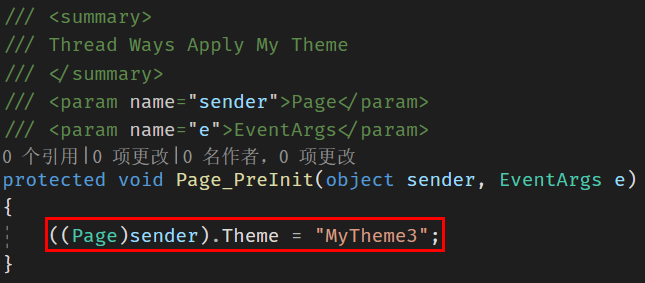


**-> 设置Thems的三种方式**

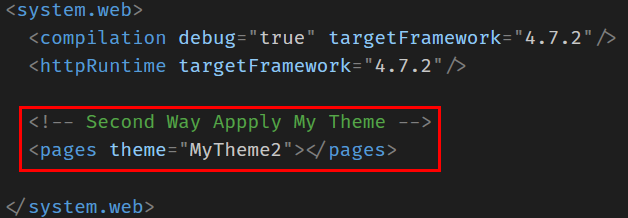
1. 通过文本模板指令的【Theme】属性



1. Page的【Page\_PreInit】Event Handler中通过sender(Page对象)的Theme属性

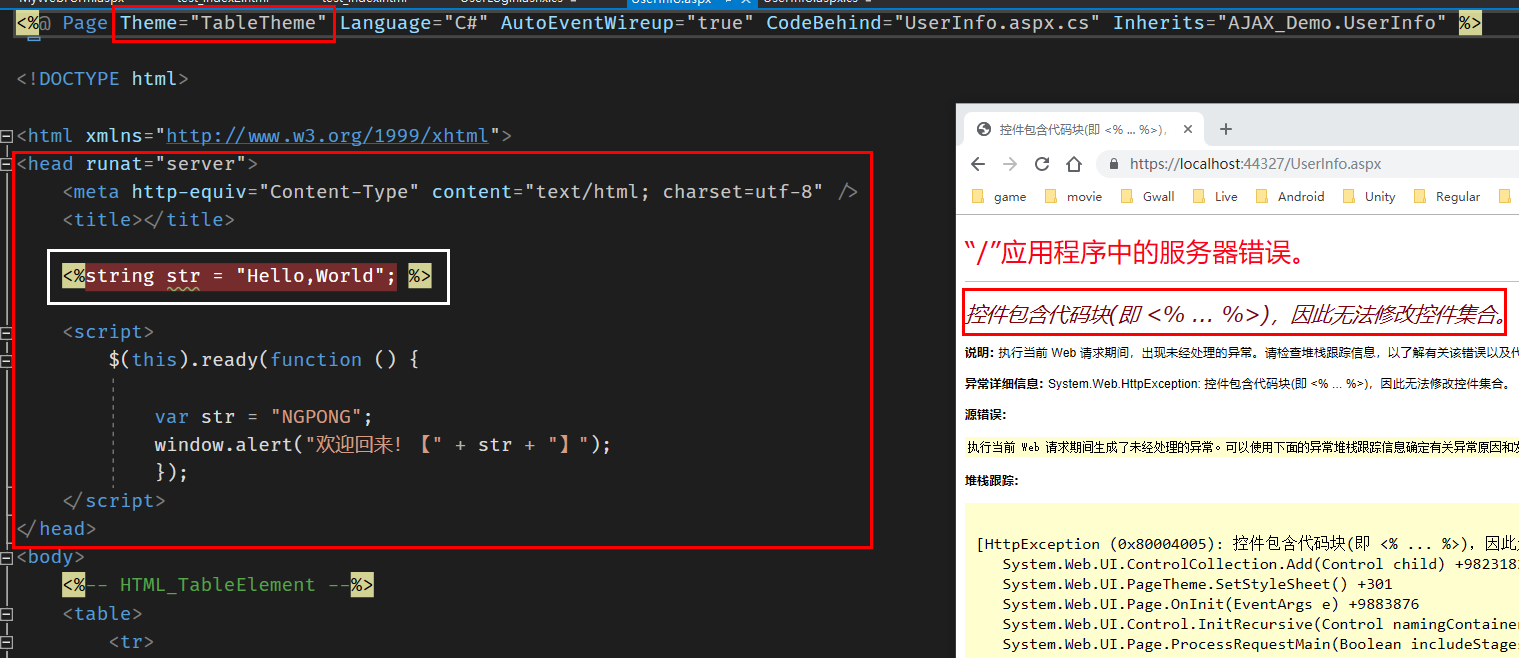


1. 通过配置文件【Web.config】中的节点指定



**-> 使用Thems时需要注意的问题**

1. 如果使用了【Themes主题】，我们则不能在<head></head>标签内使用ASP.NET内联表达式进行C#代码的编写，否则会造成报错



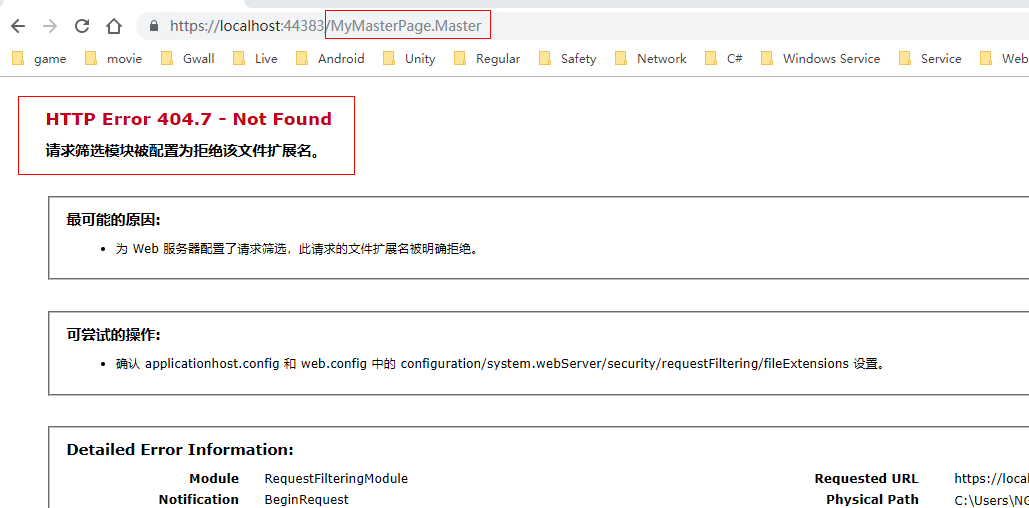
1. Master Page母版页

**-> Master Page的介绍**

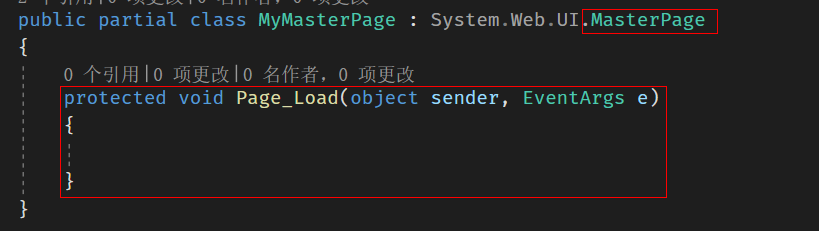
1. 在WebForm开发中，如果诸多WebForm中需要使用到一些相同的布局，例如网站的版权信息、网站登记信息、…，在这种情况之下我们可以新建一个母版页专门用于显示这些信息，并且让需要应用到这些布局的WebForm界面都套用这个母版页以此来实现这种效果

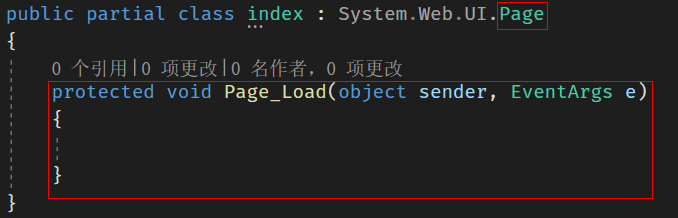
**-> 使用Master Page的注意事项**

1. 母版页文件是一个 \*.Master 文件，但是这个文件我们是不能直接通过浏览器来访问的，因为IIS默认情况下不会为母版页的后缀名注册相应的ISAPI映射功能模块

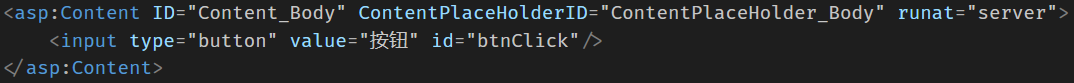


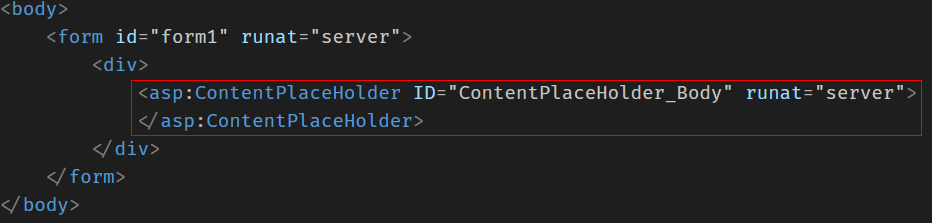
1. 母版页它本质上也算是一个Page页，他也有属于自己的页面生命周期事件用于声明，然而区别于WebForm的是，母版页所继承的是【MasterPage】的这个类，而WebForm继承的则为【Page】这个类，此外还有一点，如果母版页和嵌套了此母版页的WebForm都声明了相同的页面生命周期Event Handler，那么我们在访问这个WebForm的时候，其执行顺序是先执行WebForm中的Event Handler，然后再去执行母版页中的Event Handler





1. WebForm能够嵌套模板页中的布局的实现核心在于两个ASP.NET服务端控件，其一就是声明在母版页中的【ContentPlaceHolder】，该控件的实现效果就是在母版页中的指定放置一个容器，其二就是声明在嵌套母版页的WebForm中的【Content】，该控件在声明的过程当中需要指定相应的【ContentPlaceHolder】的ID属性，其实现效果就是把【Content】中的内容填充到在所指定的【ContentPlaceHolder】容器里面去，需要注意的是，这两个控件的使用必须都要声明【runat="server"】的属性标记



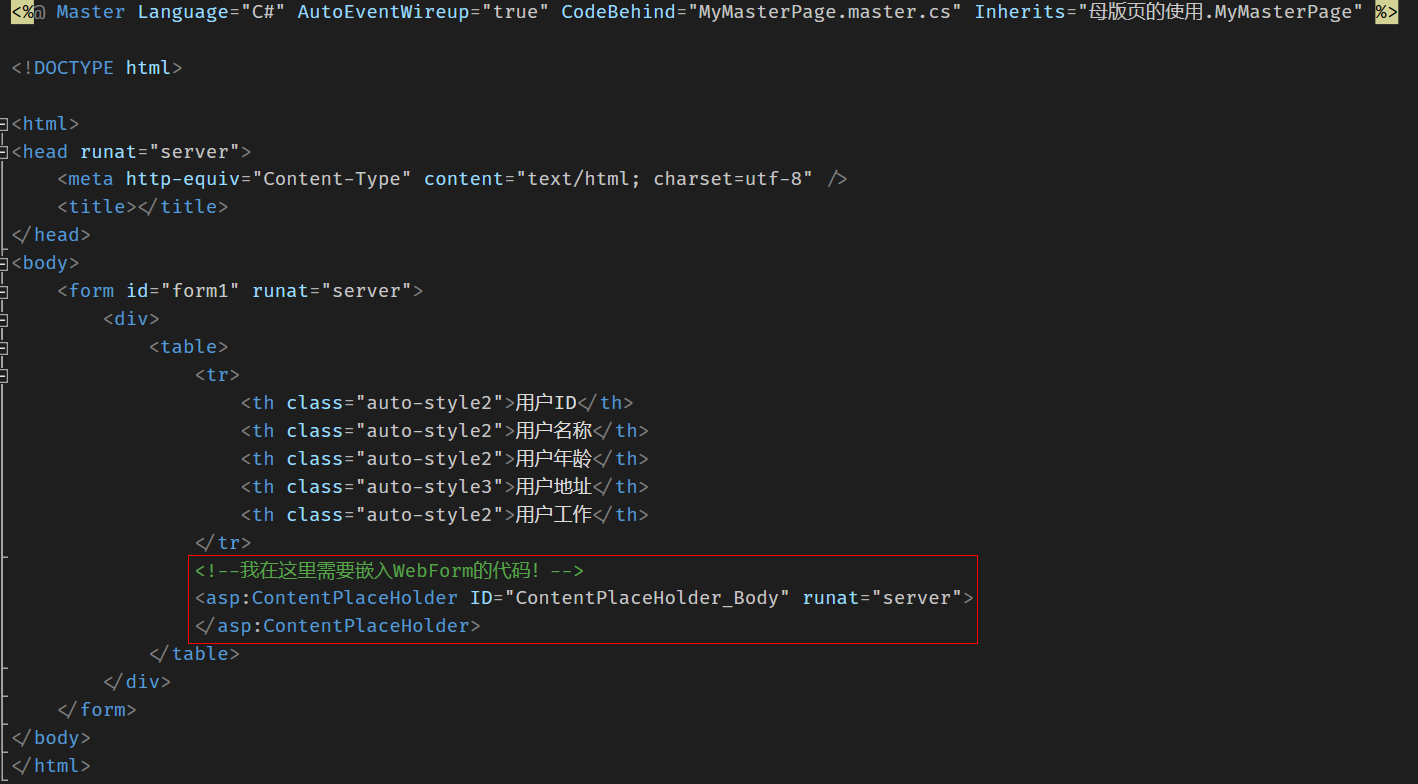


1. 当我们在一个WebForm中使用了母版页后，WebForm中就不需要声明书写html、body、head标签了，因为这些标签在母版页中都已经声明好了，我们在WebForm中的工作职责只是需要通过【Content】控件来填充在母版页中所声明的【ContentPlaceHolder】容器的内容即可，否则的话会报黄页

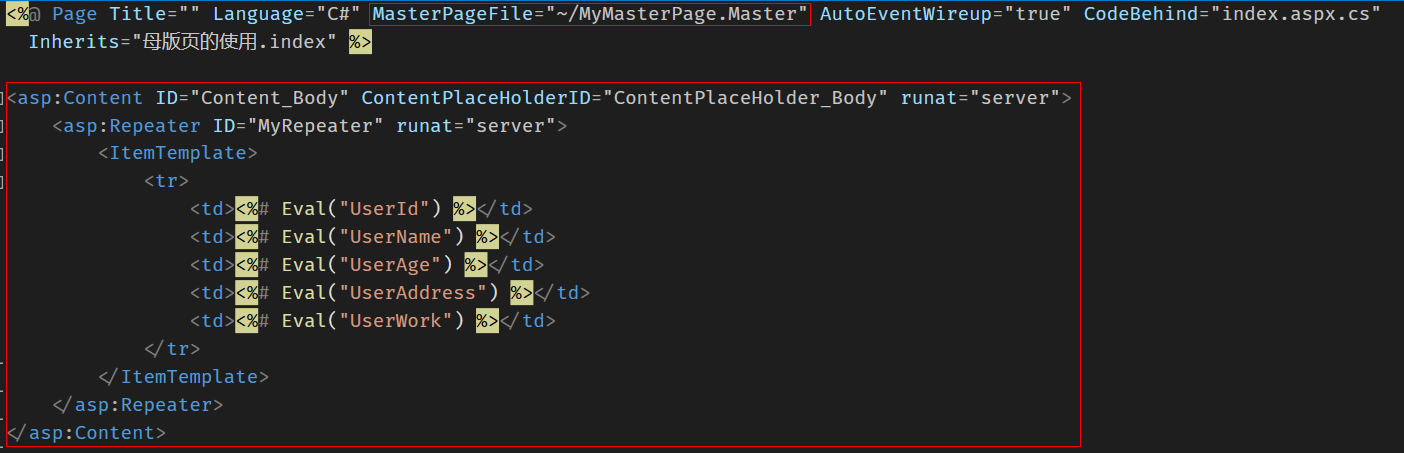


**-> 关于Master Page的使用**

1. 我们需要先声明一个母版页，在母版页中书写一些需要进行统一性的布局HTML代码，并且在需要嵌套WebForm代码的位置中使用【ContentPlaceHolder】声明一个容器



1. 在创建完成母版页之后，我们就需要创建一个需要套用这个母版页的WebForm了，首先我们需要为这个Webform的文本模板指令中的【MasterPageFile】声明为我们在上一步所创建的母版页的位置，其次，我们需要声明一个服务端控件【Content】并指定其属性【ContentPlaceHolderID】为我们在母版页中所创建的【ContentPlaceHolder】的【ID】属性的值，通过这种绑定操作，【Content】标签内部所编写的内容会自动地填充到【ContentPlaceHolder】容器内部当中，自此母版页的使用结束



**ASP.NET WebForm服务端控件**

**关于WebForm的控件其实不推荐使用，其一是因为这一技术不适合目前互联网应用的发展，其二就是我们上一节所讲到的问题【ViewState】隐藏域过长所带来的隐性灾害，然而也有几个需要我们关注的一些重点，所以以下也简单的介绍一下各服务端控件的一些使用，和需要关注的一些重点**

1. ASP.NET服务端控件是什么？

-> 服务端控件是微软所提供给我们可以直接使用到aspx页的一些控件，它和WinForm中的控件是一个意义，当我们在 一个 \*.aspx 页使用了服务端控件的时候，其语法页是遵循标签的形式，但是却不是HTML5的语法，在ASP.NET 页面生 命周期执行最后的步骤控件渲染的时候，会把我们所使用的ASP.NET服务端控件重新改造成相应的一些HTML元素标签， 最后返回给浏览器

1. ASP.NET服务端控件使用的过程当中需要注意的问题

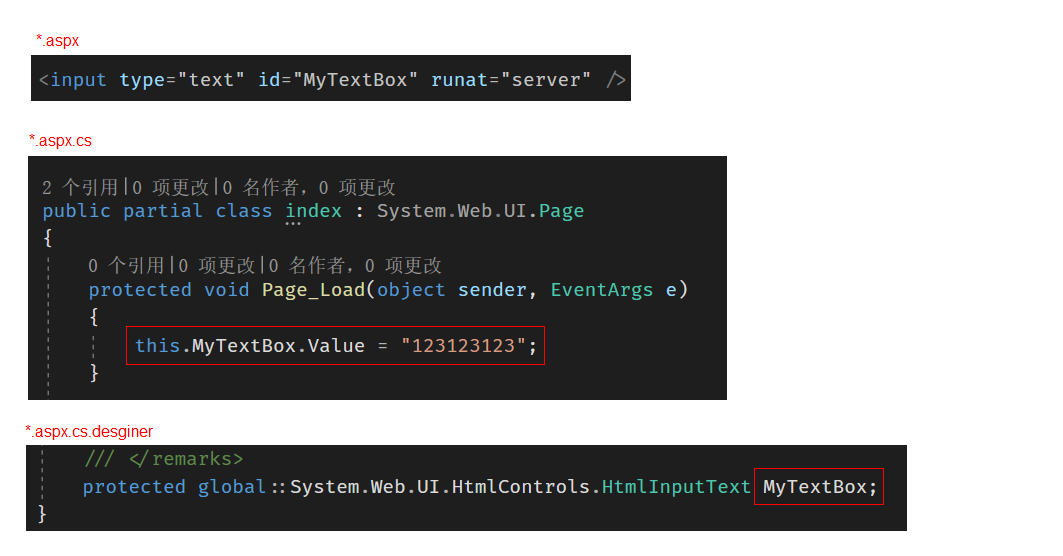
-> 服务端控件在声明的过程当中都带有其相应的前缀，像ASP.NET默认提供给我们的控件的前缀则为asp，当然我们也 可以自定义自己的控件并指定该控件的前缀



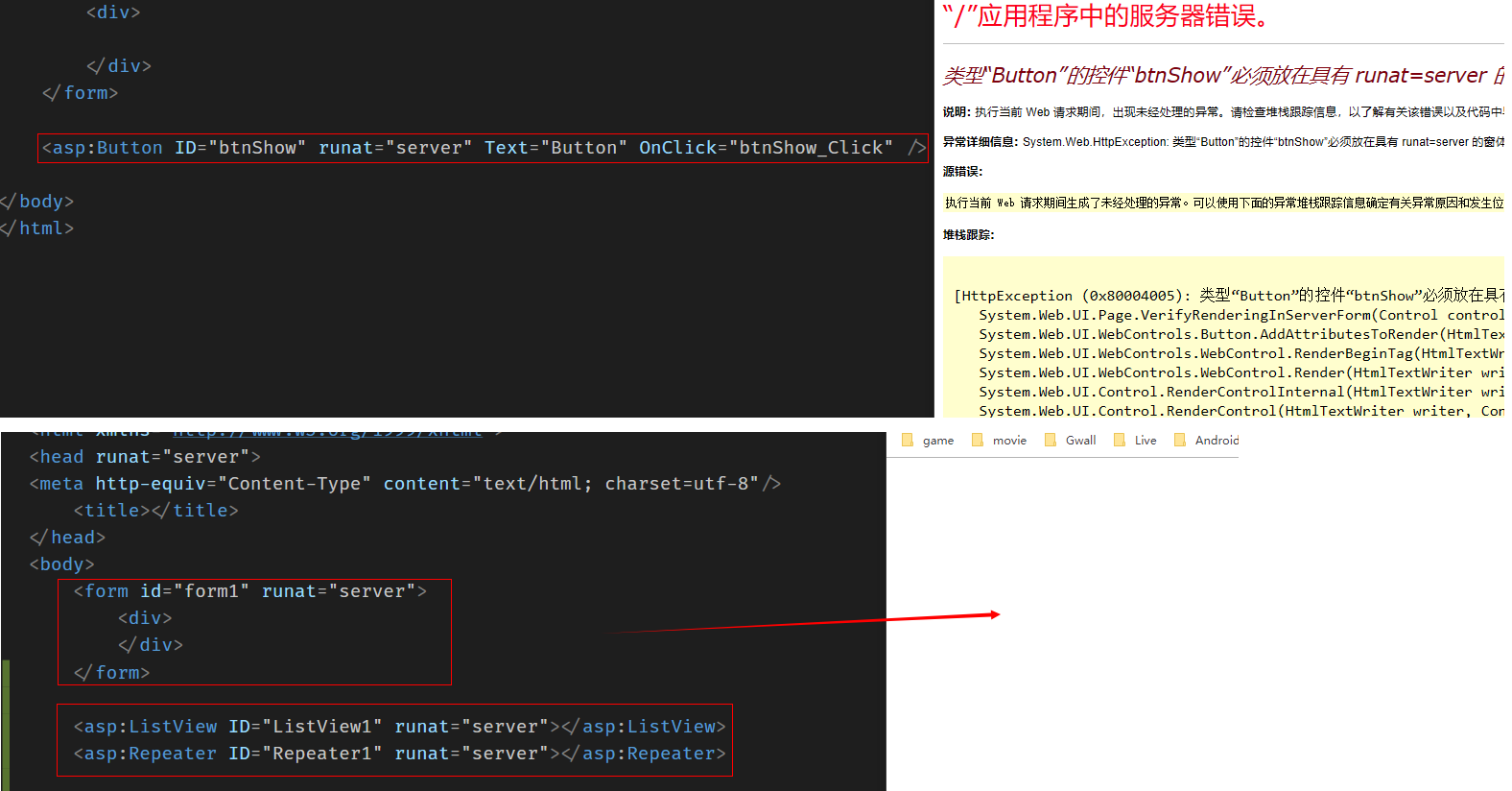
-> 服务端控件必须带有【runat="server"】属性的标记，否则它无法构成一个ASP.NET服务端控件

截图20190625225750

-> 我们在一些HTML标签上加上了【runat="server"】属性的标记后，它也可以称之为服务端控件，在.Net Runtime 在进行aspx页面上控件解析的工作的时候，该标签也会被生成相对应的一些类（HtmlAnchor、HtmlForm、HtmlHead、 HtmlInputButton、HtmlInputCheckBox 、HtmlInputText、HtmlMeta、HtmlTable、HtmlTableRow、HtmlTableCell、 HtmlTitle、…），但是对比ASP.NET自带的一些服务端控件，ASP.NET服务端控件的功能是更为全面和强大一些

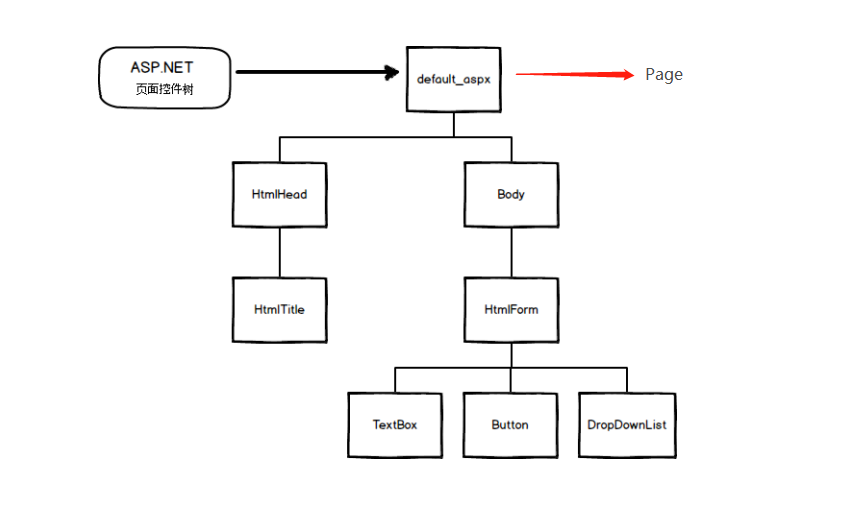


-> 大部分服务端控件都必须声明在带有特殊标记属性【runat="server"】的【form】标签之中，否则在浏览该页的时候 会报黄页，当然某些控件的话是不需要遵循这一规则的，比如说Repeater



1. ASP.NET页面控件树

　　-> 每一个服务端控件都是一个类型，更专业来说它们都是不同的对象，每当我们在aspx页中使用一个控件的时候，在 \*.aspx.desginer.cs 文件下都会为 我们所使用的控件构造一个实例，这些实例都是基于Page类下所构造的（\*.aspx继承 于Page），在这里，我们可以把Page类看作是一个大的控件容器，它是一个模板控件，而所构造控件都是Page类下的 子控件，至此则构造了一个页面控件树



1. ASP.NET服务端控件中的一些特殊属性

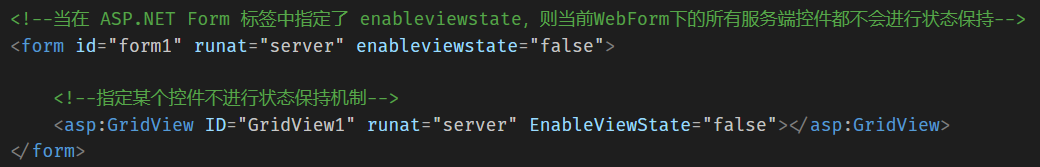
-> runat

1. 标记当前标签的属性，只有当标记为【runat="server"】的控件才被看作为ASP.NET服务端控件

微信截图_20190602194829

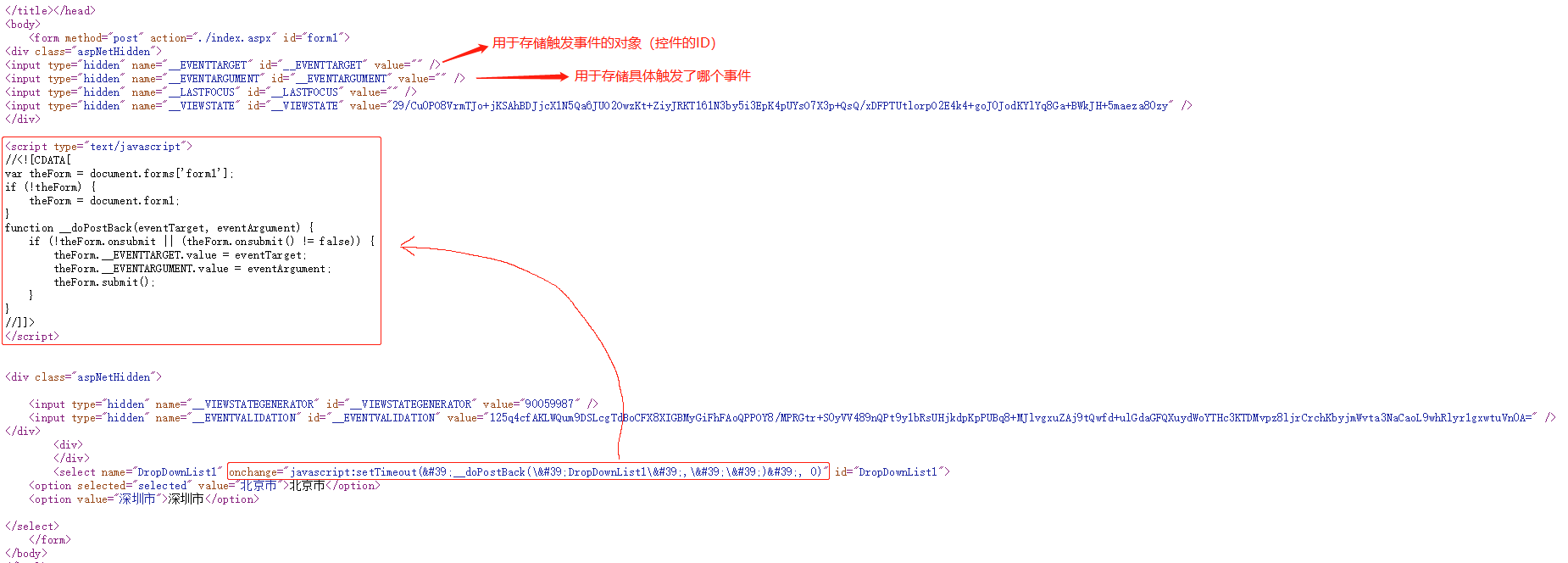
-> enableviewstate

1. 控制当前所指定的控件是否需要进行状态保持，需要注意的，是当我 们在【ASP.NET Form】标签当中指定了这个属性， 则其作用范围引用至当前aspx网页下的所有控件，而如果具体的指定某个控件的这个属性，则作用范围只是我们所 具体 指定的某个控件



-> AutoPostBack

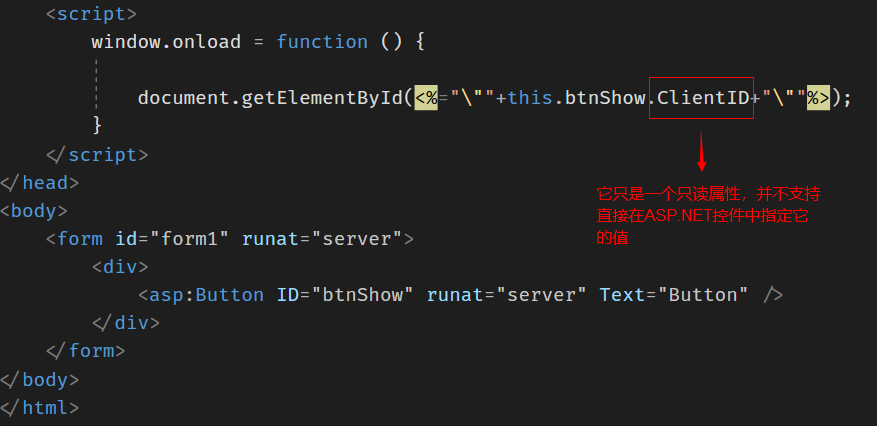
1. 设置当前控件是否启用回发事件，该属性并不适用于所有的控件（如button控件，因为其在最后会渲染成submit控 件），只有当我们设置了该属性的值为true的时候，我们所指定的控件的事件才会被触发
2. 当我们为一个控件设置了该属性后，在最后执行【RenderControl】函数的时候，会为指定了该属性的控件生成一段 【JavaScript】代码，其主要作用的是通过这段JS代码帮我们把标签的内容提交至服务器当中，需要注意的是，除了生成 一段提交数据的JS代码，还会多为我们生成两个隐藏域：【\_\_EVENTTARGET】用于存储所触发事件的控件ID、 【\_\_EVENTARGUMENT】用于存储具体触发了哪个事件信息



1. 当我们改变了HTML标签的值后，调用JS把所改变的数据提交至服务器，在提交的过程当中，上一节说到的两个隐藏 域的内容也会一并提交至服务器，当底层实现流程走到【ProcessPostData】函数函数的时候，会把根据两个隐藏域(【\_\_ EVENTTARGET】【\_\_EVENTARGUMENT】)的内容并结合【AutoPostBack】属性的值为true的控件，把它们放在【\_c hangedPostDataConsumers】的集合当中，当流程继续往下走，走到【[RaiseChangedEvents](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://System.Web:4.0.0.0:b03f5f7f11d50a3a/System.Web.UI.Page/RaiseChangedEvents()" \o )】函数的时候，会遍历【\_ changedPostDataConsumers】，并且判断每个元素是否实现过【IPostBackDataHandler】接口，如果已实现则调用所 注册相应的Event Handler

-> ClientID

1. 该属性为一个只读属性，用于读取该控件在生成HTML代码后的ID属性的值，ASP.NET服务端控件都是不同的对象，当我们使用一个服务端控件后它有一个ID属性，但这个ID属性和ClientID却不一样，我们可以把前者看成是这个控件对象的类名，并不就是该控件最后生成为HTML代码后的ID属性，而ClientID属性则为该控件最后被编译成HTML代码后所指定的ID属性
2. 在ASP.NET中，一般情况下服务端控件的ID属性也是最后所生成的HTML代码的ID属性，但是有些情况下，比如说母版页，我们在母版页中定义了一个控件于使用母版页的aspx页中的某一个控件的ID属性重复了，在这种情况之下，.NET Runtime进行最后编译成HTML代码的步骤的时候，会把重复的ID属性更换成其他的属性，一般情况下都是aspx页上重复的ID属性给更换，现在有这么一个情况，比如说我们在aspx页写了一段javascirpt根据一个元素的ID去获取一个元素，而这个ID的值我们参考的却是服务端控件上的ID属性，这时候就出问题了，因为javascript脚本是在客户端上执行的，也就是浏览器，但是ASP.NET这时候把aspx页上重复的ID属性给编译成其他的值了，这时候就可能导致我们获取不到



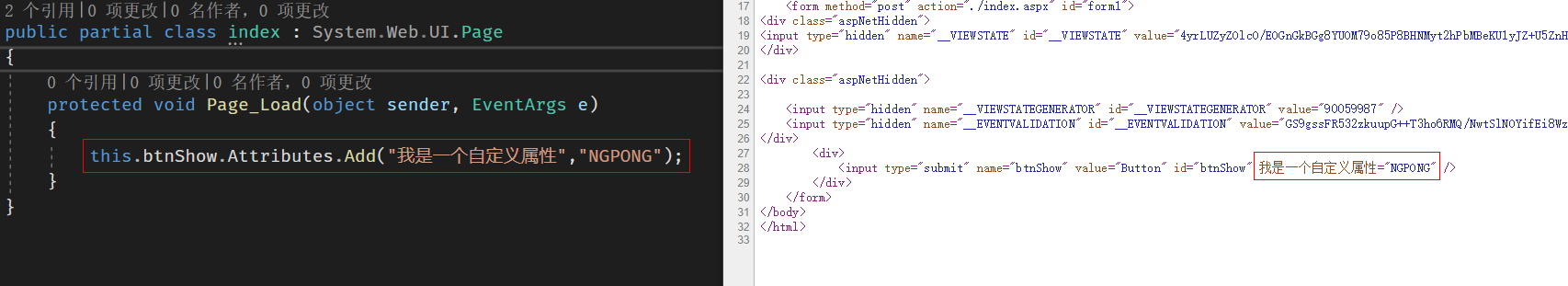
-> Visiable

1. 该属性用于设置ASP.NET控件是否要隐藏，更专业来讲的话，设置当前控件是否需要给编译成HTML代码

截图20190625213228

-> Attributes

1. 我们知道，每一个ASP.NET控件都是一个对象，并且不同的控件在最后都会编译成相应的HTML元素返回给浏览器， 但是有些HTML元素的属性而ASP.NET控件对象却没有声明，也就是我们无法通过控件的实例去访问这个属性(因为都没 有声明)，这时候我们就可以通过Attributes属性去指定这些额外的一些属性的值



1. 关于一些常用的控件

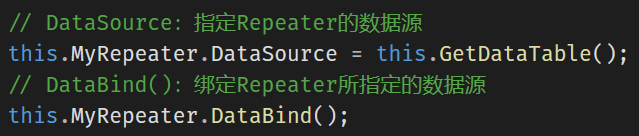
Repeater

**-> 关于Repeater的介绍**

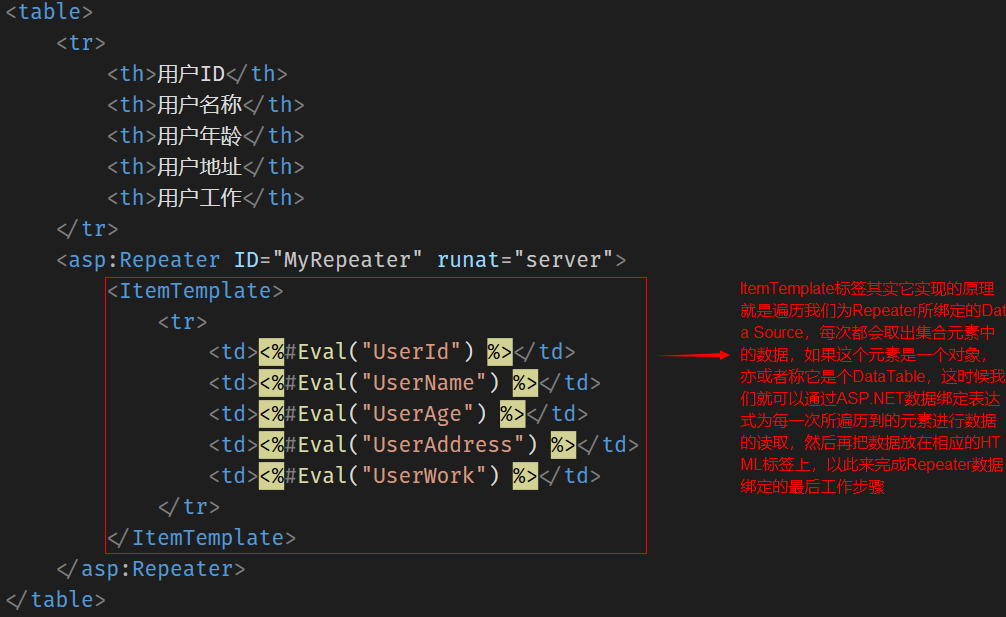
1. 区别于GridView、DataList、…等一些非常臃肿的ASP.NET服务端控件，Repeater控件是一款非常轻量级的用于数据展示的控件
2. Repeater控件不需要像其他的ASP.NET服务端空间一样必须放在带有特殊标记【runat="server"】的Form标签当中，实际上它是可以放在一个aspx页上的任何位置

**-> 关于Repeater绑定数据和显示数据的方式**

1. 和其他数据展示的控件一样，我们也是通过其【DataSource】属性指定该Repeater所对应的数据源，并通过【DataBind】函数来让所指定的数据源与Repeater进行绑定工作，需要注意的是，绑定Repeater数据源的代码最好写在【Page\_LoadComplete】事件下，因为当我们在Repeater的内部使用了一些额外的ASP.NET控件后，当我们触发这些额外的ASP.NET控件的事件的时候，数据会进行回发，但是却会报黄页，因为在【Page\_LoadComplete】会为所触发事件的控件做数据回发的工作，但是我们在【Page\_Load】的时候已经为其重新绑定了数据源，这时候在处理回发工作的时候可能就会找不到所回发事件的控件了，这时候就会报错



1. 是需要注意的是，我们为一个Repeater进行了数据源的赋值和绑定的工作后，Repeater并不会自动的把所绑定的数据编译成相应的HTML代码交给浏览器渲染出来，而是还要通过我们一些写一些代码进行进一步的数据绑定操作，下面就要开始介绍该控件所带有的标签【ItemTemplate】了，它是进行数据绑定最后工作的一个核心部位，ASP.NET会在【ItemTemplate】标签内遍历我们在上一步所绑定Repeater DataSource的集合，每一次遍历ASP.NET都会把集合中的元素取出来，如果这个元素是一个对象（对应表的字段结构），那我们这时候就需要通过ASP.NET数据绑定表达式来把当前循环到的元素（对象）中的某一个属性的值取出来，再配合HTML标签以此来完成数据的最后绑定工作流程



**-> 关于Repeater中的ItemCommand事件**

1. 当我们在Repeater控件的内部使用了另外的一些ASP.NET服务端控件，比如说：button，这时候当我们点击这个button的时候，在触发完这个Button所注册的事件后，会触发ItemCommand事件，如果我们不想注册相应的Button事件，而是通过ItemCommand事件来处理相关的逻辑也是可以实现的，但是我们需要为这个Button另外指定两个属性【CommandName】和【CommandArgument】，【CommandName】属性其实相当于是一个类别的意思，可以指定具体是哪个类别的控件进行触发的，我们可以通过ItemCommand事件中的参数【RepeaterCommandEventArgs】中的【CommandName】属性来获取当前触发这个事件的ASP.NET控件是属于哪个类别下的，而【CommandArgument】相当于是一个控件ID的概念，我们一样可以通过ItemCommand事件中的参数【RepeaterCommandEventArgs】中的【CommandArgument】属性来获取当前具体是哪个控件所触发的事件





**ASP.NET内联表达式**

**内联表达式式适用于WebForm项目，通过它我们能把C#的代码以脚本的形式前入至HTML代码中**

1. 文本模板指令

　　-> @ Page：\*.aspx文件首部所使用的指令，用于声明Page的一些属性设置

1. Language：当前 \*.aspx 所使用的嵌入代码块的语言
2. AutoEventWireup：是否启用页面生命周期
3. CodeBehind：当前 \*.aspx 的代码后置文件名（\*.aspx.cs）
4. Inherits：当前 \*.aspx 的父类（\*.aspx.cs）
5. Theme：当前主题样式
6. EnableViewState：当前Page页是否启用【ViewState】状态保持机制

微信截图_20190602233150

　　-> @ Import：为 \*.aspx 文件引入指定的命名空间

1. NameSpace：需要引入的命名空间

微信截图_20190607194014

-> @ Register：为 \*.aspx 文件引入自定义的控件声明

1. Src：自定义控件文件名
2. TagPrefix：自定义控件前缀
3. TagName：自定义控件的标签名

截图20190625225549

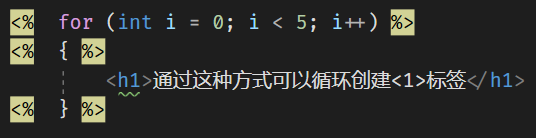
-> @ OutputCache：为 \*.aspx 页面的输出内容进行缓存

1. Duration：缓存内容所保存的绝对过期时间，单位为秒
2. VaryByParam：当前页面的缓存内容按照什么样的参数格式进行保存，关于该属性的详解，请查看[此文档](../ASP.NET基础/ASP.NET Cache.docx)

截图20190630204213

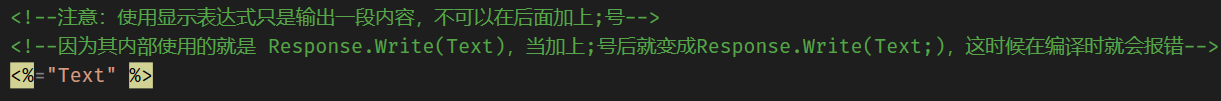
1. 嵌入式代码块

　　-> <% %>：嵌入式代码块能让我们在 \*.aspx 文件下使用C#语言结合HTML语言进行一些逻辑上的处理



1. 显示表达式

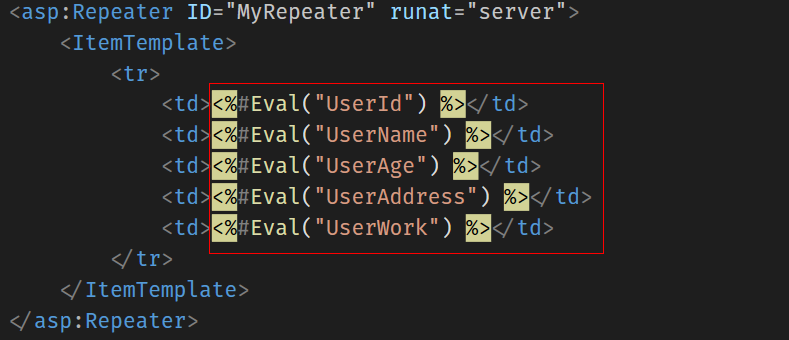
　　-> <%=Text %>：使用显式表达式在最终编译后就会变为Response.Write(Text)，属于一种简写形式



1. 数据绑定表达式

-> <%#Eval(string ItemName) %>：数据绑定表达式通常是配合着Repeater/ListView控件获取数据的使用

-> Eval(string ItemName)：通过ItemName获取【ItemTemplate】标签中所循环到的元素对象的指定属性的值



1. 更多的内容

　　-> [https://docs.microsoft.com/zh-cn/previous-versions/dotnet/netframework-4.0/fy30at8h(v%3dvs.100)](https://docs.microsoft.com/zh-cn/previous-versions/dotnet/netframework-4.0/fy30at8h(v=vs.100))

　　-> [https://support.microsoft.com/en-us/help/976112/introduction-to-asp-net-inline-expressions-in-the-net-fr amework](https://support.microsoft.com/en-us/help/976112/introduction-to-asp-net-inline-expressions-in-the-net-framework)