ASP.NET دورة حياة

ASP.NET life cycle specifies, how:

ASP.NET processes pages to produce dynamic output
The application and its pages are instantiated and processed
ASP.NET compiles the pages dynamically
The ASP.NET life cycle could be divided into two groups:

Application Life Cycle
Page Life Cycle
ASP.NET Application Life Cycle
The application life cycle has the following stages:

User makes a request for accessing application resource, a page. Browser sends this request to the web server.

A unified pipeline receives the first request and the following events take place:

An object of the class ApplicationManager is created.

An object of the class HostingEnvironment is created to provide information regarding the resources.

Top level items in the application are compiled.

Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.

An instance of the HttpApplication object is created and assigned to the request.

The request is processed by the HttpApplication class. Different events are raised by this class for processing the request.

ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

The page life cycle phases are:

Initialization
Instantiation of the controls on the page
Restoration and maintenance of the state
Execution of the event handler codes
Page rendering

Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle. It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Following are the different stages of an ASP.NET page:

Page request - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.

Starting of page life cycle - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.

Page initialization - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.

Page load - At this stage, control properties are set using the view state and control state values.

Validation - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.

Postback event handling - If the request is a postback (old request), the related event handler is invoked.

Page rendering - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.

Unload - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

PreInit - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.

Init - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.

InitComplete - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.

LoadViewState - LoadViewState event allows loading view state information into the controls.

LoadPostData - During this phase, the contents of all the input fields are defined with the <form> tag are processed.

PreLoad - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.

Load - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.

LoadComplete - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler

PreRender - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.

PreRenderComplete - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.

SaveStateComplete - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.

UnLoad - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

WEB.CONFIG مهام

Overview
This tutorial shows you how to automate the process of changing a Web.config file when you deploy it in different destination environments. Most apps have settings in a Web.config file that should be different when you publish the app. Automating the process of making these changes prevents you from having to do them manually every time you publish, which may be boring and prone to error.

Reminder: If you receive an error message or something doesn't work while browsing the tutorial, be sure to check the troubleshooting page.

Web.config conversions vs. web publishing parameters
There are two ways to automate the process of changing the web.config file settings: web.config file transitions and web-publishing parameters.
The Web.config conversion file contains an XML tag that determines how to change the Web.config file when it is published. You can select different changes to specific build configurations and specific deployment profiles. Default build configurations are patch and release, and you can create custom build configurations. The publishing profile is usually compatible with the destination environment. (You'll learn more about posting profiles in the Deployment to IIS as a Test Environment tutorial.)

Web publishing parameters can be used to identify many different settings that must be configured during deployment, including settings in Web.config files. When used to determine Web.config file changes, web

publishing parameters are more complex in terms of setup, but they are useful when they do not know the value to set until you publish. For example, in an enterprise environment, you can create a deployment package and give it to someone in the IT department to install it in production, and that person must be able to enter contact chains or passwords that you do not know.

For the scenario addressed by this series of tutorials, you know in advance everything to do for the Web.config file, so you don't need to use web publishing parameters. You will configure some conversions that vary depending on the configuration of the user build, others vary depending on the user's deployment profile.

Select Web.config settings in Azure
If the Web.config file settings you want to change are in the <connectionStrings> or <appSettings> item, and if you're posting to web apps in Azure App Service, you have another option to automate changes during.

Later, you will create three other conversion files, one each for testing, phased production, and dissemination of profiles. One typical example of a setting that you can handle in a publishing profile conversion file because it depends on the destination environment is the WCF endpoint that is different for the test from the production. You will create profile conversions for deployment in subsequent tutorials after you create the publishing profiles they use.

GLOBAL.ASAX

Global.asax profile definition:

It is an optional file that includes preparations and general events for the entire application, and can also include events related to other httpModules, as in the events start,End of SessionModule, has a name which is Global.asax, and must also be kept in the highest level of interference of application files, i.e. it can not be placed in a branch folder within the site.and comes in two forms, either comes in one code file (Markup +) , or two separate files (Markup) + (Code-behind). We also mention that it's called

the Global Application Class if you try to add it from the Add New Item window.

This file, as we mentioned is Class inherits httpApplication, and when we mentioned in the Role of The Life of The Request, that instance is created from httpApplication, when this file exists, this instance is created from this class which is actually called Global . And since it's a file.

“RUNAT=SERVER” شرح خاصية

Normally, when we write a form page, there will be two buttons below: "send" and "return". As the name suggests, everyone knows their jobs, but in general we will add some validation to the form content, so there is a problem. Since the two buttons are server controls (with runat = "Server"), they will be validated after clicking the button (whether you are using jQuery.validate front-end validation or background control validation for ASP.NET), will be verified first). For the "send" button, that's already what we want, but for the "back" button, we don't want this to happen, but we want it to go back to the previous page without checking.

For this case, my previous solution was to put

```
<asp:Button ID="button_back" runat="server" Text="back"
OnClick="button_back_Click" />
```

It was replaced by

```
<input type="button" value="back" onclick="BackToPage();" />
```

This form. The BackToPage method applies page jumps.

“ISPOSTBACK” شرح خاصية

Get a value indicating whether the page is being displayed for the first time or whether it is being downloaded in response to republishing.

[System.ComponentModel.Browsable(false)]

public bool IsPostBack { get; }

Property value

Logical value

Trueif the page is loaded in response to the republication of the customer;

Features

A browsable feature

Examples

The following example shows how to test the value of the IsPostBack property when the page is loaded to determine whether the page is being displayed for the first time or responding to the republishing process. If the page is first submitted, the code calls page.Validate.

Page encoding (not displayed) contains RequiredFieldValidator controls that display astral tags if no entry is entered into a required input field. Call page: Validation of tags immediately displays asttags when the page is displayed, rather than waiting for the user to click the send button. After republishing, you don't have to call Page.Validate, because this method is called as part of the page's lifecycle.

```
private void Page_Load()
{
  if (! IsPostBack)
  {
    Validate initially to force asterisks
    to appear before the first roundtrip.
    Validate();
  }
}
```

المشرف:علاء الخوّام.

الطالب:محمد يزن احمد سبسبي.

فئة ثانية.