Agenda:

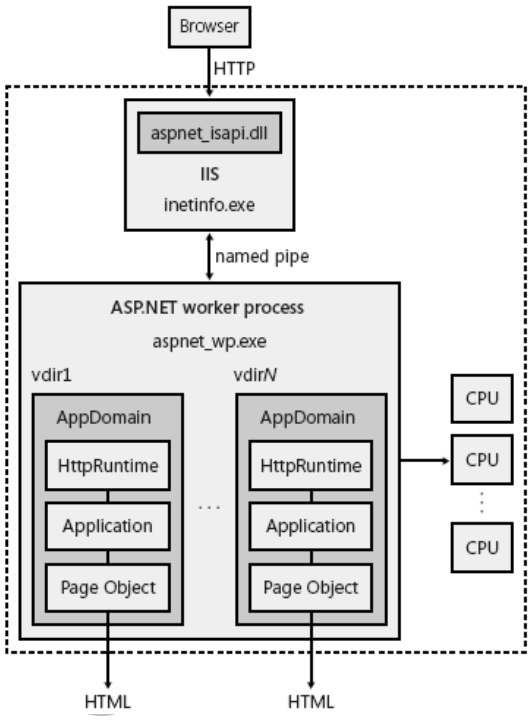
- What is AppDomain?
- Life Cycle of ASP.NET Page
- How Control Manages its State
- What is EnableViewState
- How control raises events
- Event handling in Web Forms
- Writing Custom Classes in WebApplication

What is AppDomain?

An AppDomain is a logical collection of one or more thread with in a process. If a thread of one App domain wants to access the thread of another app domain, it has to use inter-process communication (IPC). A process can have more than one AppDomain and each AppDomain is independent of other AppDomains. If a thread in one AppDomain crashes, only other threads of that AppDomain are affected and threads in other AppDomains would continue to run.

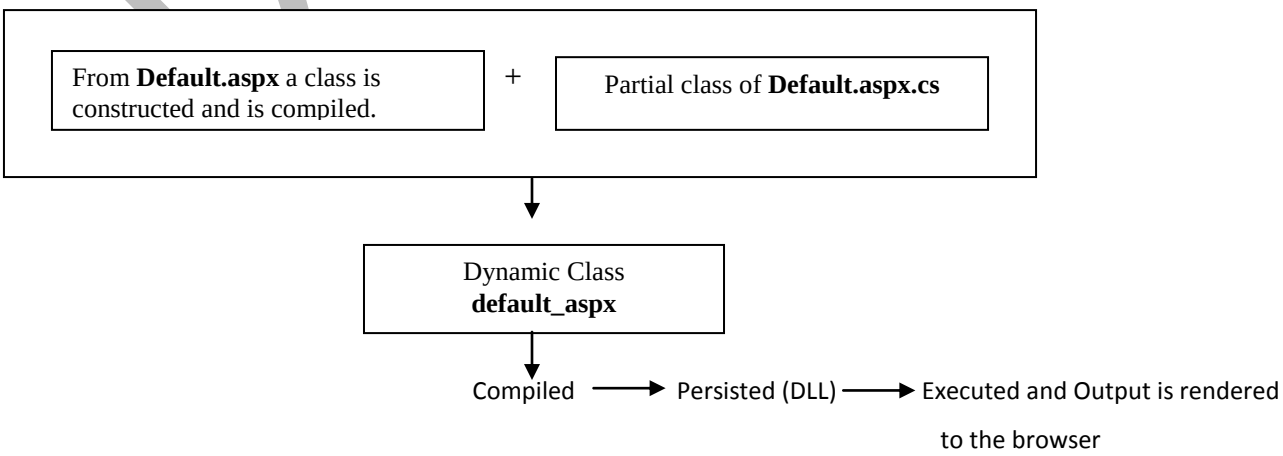
Life Cycle of a Web form

When the Web Server receives the request for the file with the extension .aspx, it forwards it to **aspnet_isapi.dll** which is an ISAPI application executing within the web server process. This ISAPI application is then responsible for launching the **aspnet_wp.exe** worker process. (WebDev.WebServer.Exe for ASP.NET Development server)



In the Worker process a new **AppDomain** is created for every web application.

For every Request a new thread is created in the AppDomain irrespective of client requesting and web form requested.



Note: The Code file name is not dependent on aspx filename. It can any filename but that must be mentioned the “CodeFile” attribute of the Page directive in the requested aspx page.

The ASP.NET Framework at runtime constructs the class inherited from the two partial classes. It's in this class the framework adds additional code it's required to execute.

From second request onwards the framework directly uses the persisted form of the class for executing the request.

ASP.NET now

1. Creates an instance of the Web form class i.e. the dynamic class inherited by framework from the class in code file.

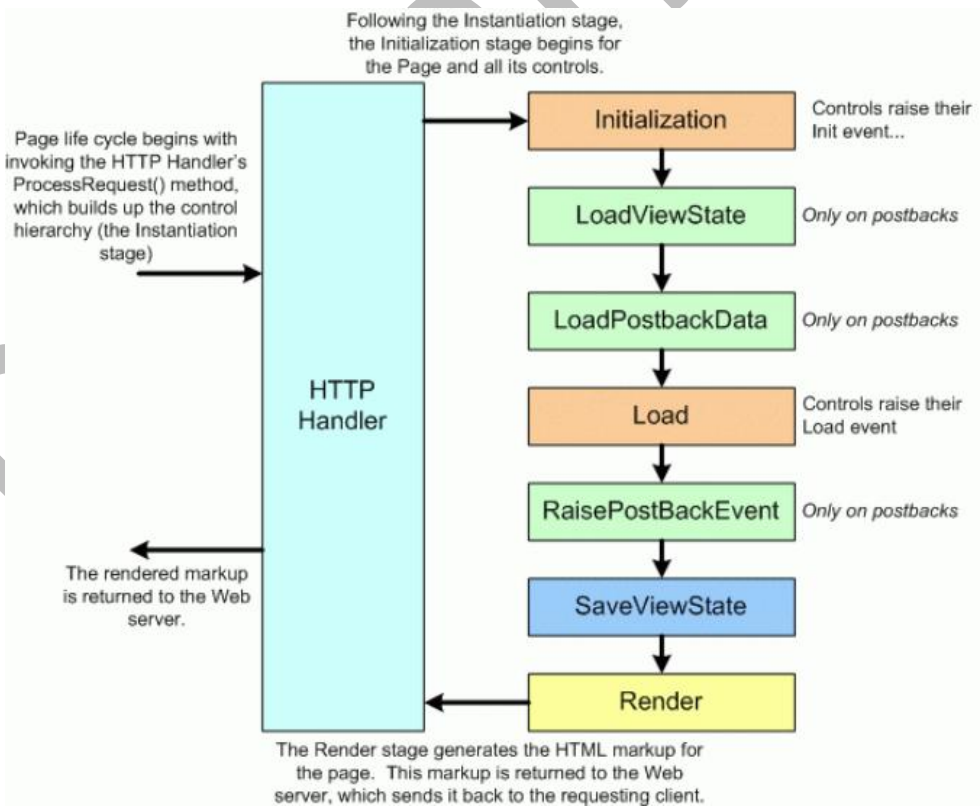
Note: For every trip made to the server i.e. for first request or even if the form is submitted (Postback) the **new instance** the page class is created.

Note: When the web form class is instantiated, the controls of the class are not yet created or instantiated hence the constructor of web form cannot be used for setting or getting control properties.

2. Based on the server side tags (**runat**="**server**") the controls are instanced and based on tag attributes the control properties are set.
3. Based on the state of the server object for every control, the control renders its part to the overall output of the web form rendered to the browser.

Note: All the objects created (for controls and page) are ready for garbage collection once the response of the page is rendered to the web browser i.e. for every request new set of objects are created.

How a Control Manages its State?



Sequence of Steps in Execute of Page Request

Page class object is created
Constructor of page class is executed but cannot be used for initialiazing controls of the page

Page.Init event is raised thus executing Page_Init

(this is not the right place for initializing control properties)

If the form is Posted -

1. The name and value of the control submitted is used for initializing its property

(for example = TextBox1.Text = value submitted for input element by name TextBox1)

2. The hidden field (ViewState) is used for initializing all the properties of the control

(for example = TextBox1.ForeColor = Red)

Page.Load event is raised this executing Page_Load

All Events of all controls on the page are raised and corresponding event handlers are executed.

Page.PreRender event == Page_PreRender

(All modified properties of the control are added to the viewstate (hidden field) rendered to the browser)

Page and thus all controls of the page are Rendered to the browser

Page.Unload ==> Page_Unload

When the page is rendered, every control adds its modified state to the hidden element "**__VIEWSTATE**".

1. For a given response / page there is only one VIEWSTATE rendered to the browser and all controls add their modified state to it. Eg: The TextBox will add its "ForeColor" and all other properties which are changed programmatically to the "__VIEWSTATE".
2. When the form is submitted from the webbrowser, the control initializes any one of its properties from the name-value pair submitted on behalf of the HTML representation of the control. (For Example: The Text property of the TextBox is set based on name-value pair for the TextBox submitted when the form is submitted).
3. Also, after **Init** and before **Load** event, the control uses the data in the hidden element "__VIEWSTATE" and restores its rest of the state. Eg: All properties of the TextBox other than "Text" property are set by retrieving their respective values from __VIEWSTATE which is also submitted along with rest of the form elements being a hidden element in the form.
4. Any type of initialization of the control must be done in **Load** event handler (and preferably in "**if (!IsPostBack)")**. It is not recommended to set or get control properties in **Init** event handler of Page because only after Init event of page the control is said to be completely initialized.

Note: For the explanation of the above feature take an example of ForeColor Property of TextBox.

What is EnableViewState

On a round trip if we are not rendering the output of the same page / web form, the **EnableViewState** property of the controls on the page can be set to **false**. But if done so any changes in state of the control (dynamically changed properties) during the page life cycle are not added to the hidden element __VIEWSTATE and thus if round trip is made to the same page the old state of the control is not restored.

Example: ForeColor of the TextBox is lost if EnableViewState=False.

Note: The Text property of TextBox retains is value even if EnableViewState=False. This is because Text property is not dependent on VIEWSTATE, instead it is dependent on name-value pair of the textbox submitted when the form is submitted to server.

How control raises events

When the form is posted the control compares its name value pair submitted with the old state in __VIEWSTATE, if it's different the control raises an appropriate event on the server. This is done between Init and Load event of Page but event will be raised only after "Load" event of the Page.

Note: Don't set EnableViewState="False" for any control whose events are handled on server.

Note: In an ASP.NET 2.0 WebApplication every Web form can be programmed in different programming language. This is because the code related to this web form is compiled using an appropriate compiler when the first request for that web form is received by the web application and it does so based on the extension of the class file and "Language" attribute in Page directive.

Event Handling of Page Class

<%@ Page Language="CS" **AutoEventWireup**="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

If **AutoEventWireup** is set to "True" the events of page class are mapped to the event handlers based on the name of the event handler method i.e. (**Page_ <eventname**>(object sender, EventArgs e)

Note: Page_<eventname> is case insensitive and parameters are optional.

1. In C#, by default **AutoEventWireup** is set to "true".
2. In C# for event handlers of page class we can **override** the appropriate method of the parent class i.e. instead of writing **Page_Load** method as event handler we can also **override OnLoad** method in the Page class.
3. If both, Page_Load method is written and OnLoad method is overridden then when the Page is loaded the overridden method OnLoad will execute and not the method Page_Load.
4. AutoEventWireup is not used for anything other than Page Events.

Writing Custom Classes in WebApplication

A class added to the code file of a web form can be used only in that codefile. Other web forms cannot use that class, this is because every web form codefile is independently compiled when the corresponding Web form is requested from the web browser.

All the custom user defined classes must be placed under the folder **App_Code** (this folder **must** be created in root directory of the web application).

1. Right Click on Project → Add ASP.NET Folder → **App_Code**

2. Right Click on App_Code folder → Add New Item → Class (Demo.VB) - /AppName/**App_Code**/Demo.vb

Note: The classes under the folder App_Code are all dynamically compiled at run time and can be used in any Web form of the Web application (irrespective of the language it is coded in).

To use more than one programming language in same WebApplication:

/App_Code**/CSFiles**/Demo.cs

/App_Code**/VBFiles**/Demo.vb

To make the compiler know that VB code and C# code has to compiled using appropriate compilers write the following in web.config:

```
<system.web>
    <compilation> - Will be already present
        <codeSubDirectories>
            <add directoryName="VBFiles"/>
            <add directoryName="CSFiles"/>
        </codeSubDirectories>
    </compilation>
<system.web>
```