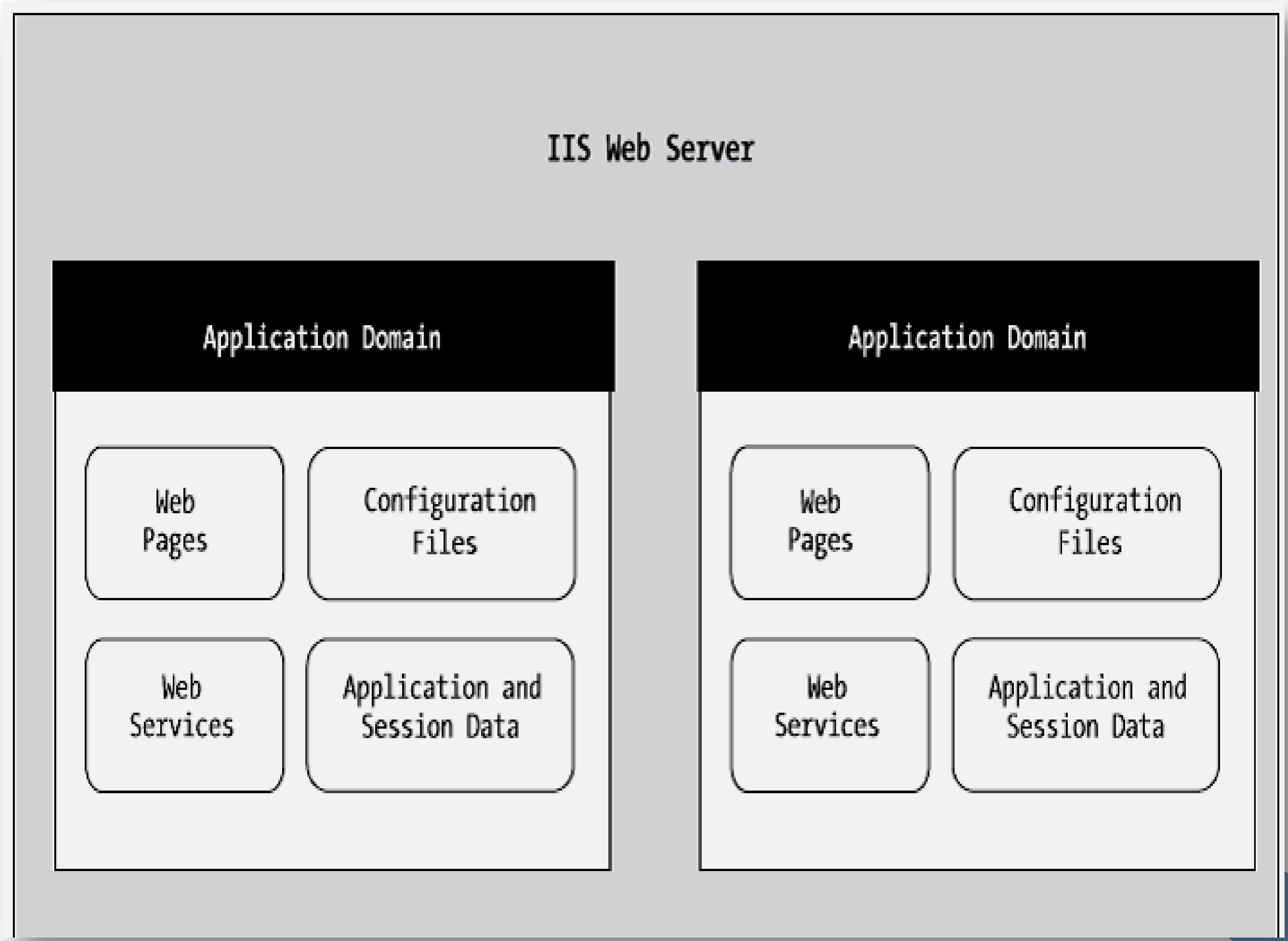
Anatomy of ASP.NET Application

Introduction

- Every ASP.NET application shares a common set of resources and configuration settings.
- Each ASP.NET application is executed inside a separate *application domain* without affecting other web applications.
- Each web application is maintained separately and has its own set of cached, application, and session data.

Introduction

- ASP.NET application describes it as a combination of files, pages, handlers, modules, and executable code that can be invoked from a virtual directory on a web server.


IIS Web Server
Application Domain
Web Pages
Configuration Files
Web Services
Application and Session Data
Application Domain
Web Pages
Configuration Files
Web Services
Application and Session Data

ASP.NET File Types

- .aspx
  - ASP.NET Web Page
- .ascx
  - ASP.NET User Control
- .asmx
  - ASP.NET Web Service
- web.config
  - XML-based configuration file for your ASP.NET application.

ASP.NET File Types

- Global.asax
  - Global Application File. One can use this file to define global variables and react to global events.
- .cs
  - C# source code files.
- .vb
  - VB.NET source code file.

ASP.NET File Types

- ASP.NET Application can hold image files, HTML files, or CSS files also.

- One can create a legitimate ASP.NET application with a single web page (.aspx file) or web service (.asmx file).

ASP.NET Application Directories

- Every web application should have a well-planned directory structure.

- Apart from our own directories, ASP.NET uses a few specialized subdirectories.

- Bin

    - Contains all the compiled .NET components (DLLs) that the ASP.NET web application uses.

ASP.NET Application Directories

- App_Code
  - Contains source code files that are dynamically compiled for use in the application.
- App_Data
  - This directory is reserved for data storage.
  - One can store data files to other directories also.
- App_Themes
  - Stores the themes that are used by your web application.

ASP.NET Application Directories

- App_WebReferences
  - Stores references to web services that the web application uses.
- App_Browsers
  - Contains browser definitions stored in XML files. These XML files define the capabilities of client-side browsers for different rendering actions.
  - ASP.NET defines different browsers and their capabilities in a computer wide configuration file, this directory allows you to distinguish browsers according to different rules for a single application.

ASP.NET Application Directories

- App_GlobalResources
  - Stores global resources that are accessible to every page in the web application.
- App_LocalResources
  - Same as App_GlobalResources but resources are accessible to a specific page only.

Server Controls

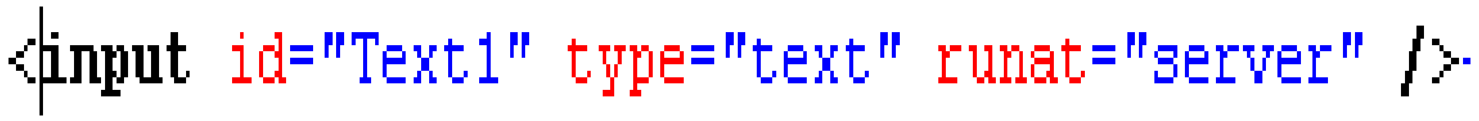
- Server controls are controls that can be programmed at server side.
- Two flavors
  - HTML Server Controls
    - Server-based equivalents for standard HTML elements.
  - Web Server Controls
    - Same as HTML Server control with a a richer object model with a variety of properties for style and formatting details.

HTML Server Controls

- *System.Web.UI.HtmlControls* is the namespace of HTML server controls.
- HTML server controls are classes that represent a standard HTML tag.
- Instances of HTML server controls are automatically created by the ASP.NET runtime if the HTML tag is marked with *runat="server"* attribute.

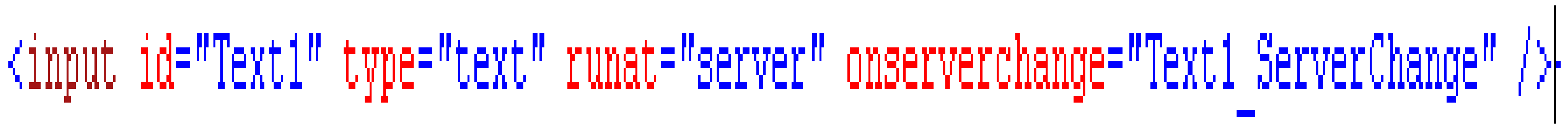
HTML Server Controls

- Do not cover all possible HTML tags.
- Tags such as <*iframe*>, <*frameset*>, <*body*>, and <*hn*> use a more generic programming interface—the *HtmlGenericControl* class.



```
<input id="Text1" type="text" runat="server" />
```

HTML Server Controls



```
<input id="Text1" type="text" runat="server" onserverchange="Text1_ServerChange" />
```

- Web forms are event-driven, which means every piece of code acts in response to a specific event.
- The event-handlers are associated to events using *onserverxxx* attributes.

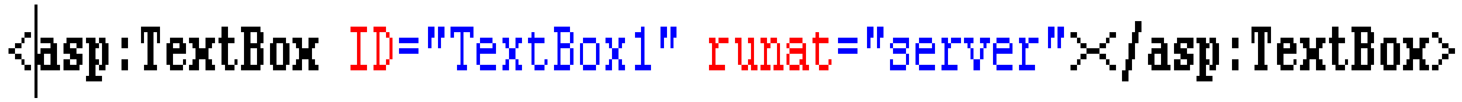
Web Server Controls

- Web Server Controls include input control and some rich-controls such as a calendar, an ad rotator, a drop-down list, a tree view, and a data grid.

- A web control is programmed as an object but doesn’t necessarily correspond to a single element in the final HTML page.

Web Server Controls

- ASP.NET Tags begin with the prefix asp: followed by the class name.
- If there is no closing tag, the tag must end with />.
- Each attribute in the tag corresponds to a control property except *runat*.



```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

ViewState

- ASP.NET page supply the *ViewState* property to manages the information that ASP.NET pages and controls want to persist across successive posts of the same page instance.

- The view state is maintained as a hidden field added to the page and travels back and forth with requests.

- The view state for the page is a cumulative property that results from the contents of the *ViewState* property of the page plus the view state of all the controls hosted in the page.

VIEWSTATE

```
ViewState["count"]=0;
```

- The *StateBag* class is the class behind the view state.
- One should write to the view state only after the *Init* event fires & read upto *PreRender* fires.
- Each item in the *StateBag* class is represented by a *StateItem* object.

Hashed and Base64 encoded.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE“
value="/wEPDwUJMjgzMDgzOTgzZGRhTvi8Ymxr52ux8Uq5KEfe/ZiBxA==" />
```

ViewState

```
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack)
{
ViewState["PostBackCounter"] = 0;
count = (int)ViewState["PostBackCounter"];
Label1.Text = count.ToString();
}
}
```

```
protected void Button1_Click(object sender, EventArgs e)
{
count = (int)ViewState["PostBackCounter"];
count += 1;
Label1.Text = count.ToString();
ViewState["PostBackCounter"] = count;
}
```

ViewState Properties

- Count
  - Gets the number of elements stored in the object.
- Items
  - Property Indexer. It gets or sets the value of an item stored in the class.
- Keys
  - Gets a collection object containing the keys defined in the object.
- Values
  - Gets a collection object containing all the values stored in object.

ViewState Methods

- Add
  - Adds a new *StateItem* object to the collection. If the item already exists, it gets updated.
- Clear
  - Removes all items from the current view state.
- GetEnumerator
  - Returns an object that scrolls over all the elements in the *StateBag*.
- Remove
  - Removes the specified object from the *StateBag* object.

ControlState

- Any information you store in the ControlState is always available to the control.
- Even though view state is disabled, it won't affect control state.
- ControlState does not have a property bag in which you can store ad-hoc values.

Client-Side Scripts

- ASP.NET uses the new *Page.ClientScript* property to register and place JavaScript functions on ASP.NET Pages.

- ClientScript has three methods:

  - RegisterClientScriptBlock

  - RegisterStartupScript

  - RegisterClientScriptInclude

RegisterClientScriptBlock

- *RegisterClientScriptBlock* method adds a JavaScript script after the page's opening server-side <form> tag.

```
protected void Page_Load(object sender, EventArgs e)
{
        string myScript = @”function AlertHello() { alert(‘Hello ASP.NET’); }”;
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
                “MyScript”, myScript, true);
}
```

RegisterClientScriptBlock

- Syntax

  RegisterClientScriptBlock (*type*, *key*, *script*)

  RegisterClientScriptBlock (*type*, *key*, *script,persist*)

- type → The type object specified with *GetType()* method.

- key→ A string literal.

- script→Script block to include.

- persist→ Boolean value specifying the inclusion of <script> tags automatically.

RegisterStartupScript

- *RegisterStartupScript* adds a JavaScript script before the page's closing server-side <form> tag.

- If one wants JavaScript function to be fired after the text box was even placed on the page, use *RegisterStartupScript*.

- Syntax is same like *RegisterClientScriptBlock*.

RegisterStartupScript

- *Example of Working of the Client Script*

```
protected void Page_Load(object sender, EventArgs e)
{
    string myScript = @"alert(document.forms[0]['Text1'].value);";
    Page.ClientScript.RegisterStartupScript(this.GetType(),
        "MyScript", myScript, true);
}
```

RegisterClientScriptInclude

- Javascript code is sometimes separated in *.js* files.
- *RegisterClientScriptInclude* registers the .js files with the page.
- Syntax includes only key & filename.

Client-Side Callback

- Client callback feature enables you to retrieve page values and populate them to an already-generated page without regenerating the page.
- No page-flicker and page reposition will occur.
- An event causes the event to be posted to a script event handler that sends off an asynchronous request to the Web server for processing.
- After the information is loaded, the result is returned to the script callback object. The script code then pushes this data into the Web page using JavaScript's capabilities to do this without refreshing the page.

Client-side Callback

- The Page/Control class supporting client callback of the Web page implements the *System.Web.UI.ICallbackEventHandler* interface & and call the *ClientScript.GetCallbackEventReference* from client-side JavaScript.
- The data passed from client to server and back again is always just a string.

Client-side Callback

- *RaiseCallbackEvent* method will be called first with the string parameter passed by the client callback method.

- *GetCallbackResult* returns the string of data for client-side script to process.

Demo

```
<script type="text/javascript">

       function GetRndFromServer(arg,context)
       {
       document.form1.TextBox1.value=arg;
       }

   </script>
….
<input id="Button1" type="button" value="button“
               onclick="UseCallBack('','')" /></div>
```

Add the script to source view of aspx page.

Add the event handler for HTML element

Demo

```
protected void Page_Load(object sender, EventArgs e)
  {
      string cbref = ClientScript.GetCallbackEventReference(this, "arg",
                                         "GetRndFromServer", "context");
      string cbscr="function UseCallBack(arg,context){"+cbref+";}";
      ClientScript.RegisterClientScriptBlock(this.GetType(),
                                         "UseCallBack",cbscr,true);
  }
```

```
public string GetCallbackResult()
    {
        return cbres;
    }

    public void RaiseCallbackEvent(string eventArgument)
    {
        Random rnd=new Random();
        cbres=rnd.Next().ToString();
    }
```

Implement ICallbackEventHandler in page class _Default