CS108: Advanced Database - ASP.NET Programming

Lecture 02: Controls

Basic Response and Request Processing in ASP.Net

Request and Response

- Web servers and web clients communicate through HTTP messages (protocols)
  - Request message: client -> server
  - Response message: server -> client
- Most HTTP communications consist of a cycle of request and response
  - Request: getting the resource, with user data submitted to server. The server will process user data.
  - Response: generating content and send it back to browsers.

ASP.Net Response Processing

- Most of the HTTP response processing functionalities are provided through the System.Web.HttpResponse class
  - The methods and properties of the HttpResponse class are exposed through the Response property of the Page class (ASP.Net web forms).
  - “Response” is a built-in object and can be directly used in .aspx pages and .aspx.cs code-behind pages.
- Basic methods in ASP.Net to generate dynamic content
  - Response.Write()
  - Expression code block: <%= %>
  - Server controls

Response.Write

- The method writes information to an HTTP response output stream.

```
String title = “Hello”;
Response.Write(“<h1>" + title + "</h1>");
```

- A shortcut to print out a value or an expression
  - <%= [expression]%>
- An expression can be a variable, formula, object property, string concatenation, or anything that returns a value
  - <% String title = “Hello”; %>
  - <%= Title%>
- Expression code block is equivalent to Response.Write(), and are mixed with static content

Mix Dynamic Content and Static Content

- String concatenation: + operator

```
String title = “Hello”;
Response.Write(“<h1>" + title + "</h1>");
```

- Expression

```
<h1><%=title %></h1>
```

- String.Format()

```
Response.Write(String.Format(“<h1>,0-</h1>“, title));
```

- StringBuilder.Append()

```
String title = “Hello”;
StringBuilder html = new StringBuilder();
html.Append(“<h1>”); html.Append(title); html.Append(“<h1>”);
Response.Write(html.ToString());
```

ASP.Net Request Processing

- Most of the HTTP request processing functionalities are provided through the System.Web.HttpRequest class
  - The methods and properties of the HttpRequest class are exposed through the Request property of the Page class (ASP.Net web forms).
  - “Request” is a built-in object and can be directly used in .aspx pages and .aspx.cs code-behind pages.
- Major request method type: Get/Post
- Two basic ways to accept user input
  - URL parameter (Get)
  - HTML Form (Get or Post)

URL Parameter

- URL parameter

https://www.google.com/search?q=ASP.NET

- Request.QueryString: QueryStringProperty wraps URL parameters to collections of parameter names and values
- Example:

URL: …/page.aspx?para1=ibm&para2=computer

```
String p1 = Request.QueryString["para1"];
Or
String p1 = Request.QueryString[0];
```

Use name

Use index

Parameter values are always treated as strings

HTML Form Structure

<form method="post" action="processing.aspx">

…

HTTP method: get or post

Action points to the target which processes form data. It can be another page/resource (even in another site) or the same page.

…

…

(Form elements and other normal HTML tags can be place here.)

<input type="submit" value="Submit" />

</form>

A “submit” button triggers the submission action.

HTML Form Elements

- Form element types
  - Textbox: <input type="text">
  - Password:<input type="password">
  - Text area: <textarea>
  - Combo box: <select>
  - List: <select multiple="multiple">
  - Checkbox: <input type="text">
  - Radio button: <input type="text">
  - Hidden: <input type="hidden">
  - Buttons : Submit, Reset
- More details: http://www.tizag.com/htmlT/forms.php

HTML Form Processing

- Each form element has “name” attribute and “value” attribute
  - “name” attribute -> parameter name
  - “value” attribute -> parameter value
- Form data can be sent either using Get or Post
  - Get
    - Form data are encoded as URL parameters
    - Using Request.QueryString to process form data
  - Post
    - Using Request.Form

Request.Form

- Used when the HTML form method is set to “post”

```
<form id="form2" method="post" action="Default.aspx">
    <input type="text" name="textbox1" value=""/>
    <input type="submit" value="Submit" />
</form>
```

```
String p1 = Request.Form["para1"];
Or
String p1 = Request.Form[0];
```

Use name

Use index

Default.aspx

Parameter values are always treated as strings

ASP.Net
Server Controls and Events

Outline

- ***ASP .NET Control***
- Event Model
- Using JavaScript in ASP .NET

Server Controls

- ASP.NET provides a server-side object-oriented and event programming model for the web through server controls
- Server controls are objects placed in the ASP.Net pages that are processed by ASP.Net runtime
  - Each controls has properties, methods, and events
  - Developers design pages through manipulating these objects and their properties, methods, and events
- Server controls are a higher level abstraction that hide lower level routine development work (such as generating HTML code).
  - It’s more powerful, but more complicated.

Features of Server Controls

- Object-oriented programming style
- Desktop application user interface development style for Web pages
- Output is automatically generated, and customized based on the capabilities of the browser
- Ability to react to events
- Automatic state management

Server Control Types

- HTML server controls
  - They are HTML elements that include a runat=“server” attribute.
  - They map *one to one* with their corresponding HTML tags (with the same attributes), plus *automatic state management* and server-side events.
  - The runat="server" attribute turns them into server controls that can be referenced on the server side.
- Web server controls
  - Web server controls have a standardized set of properties and events.
  - They are not directly mapped to HTML elements on a one-to-one basis.

The runat Attribute

- The runat attribute makes a server control
  - This is true for both HTML and Web controls
- All tags without the runat attribute are copied verbatim to the output stream (HTTP response)

```
<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
```

HTML Server Control Example

HTML server controls are have the same HTML output. They map one to one with their corresponding HTML tags (with the same attributes).

- In the .aspx page:

```
<input id="Value1" type="Text" value="100" size="10"
maxlength="4" runat="server"/>
```

This HTML element becomes a server control.

- In the code-behind page:

The HTML element's **"id"** attribute is used to programmatically reference the control.

```
protected void Page_Load(object sender, EventArgse)
{
   Response.Write(this.Value1.Value);
   this.Value1.Value = "1000";
}
```

This changes the value of the textbox to "1000".

"this" refers to the page itself.

Web Server Control Example

A web server control tag usually tarts with “asp:”

A web server control always has runat=“server” attribute.

- In the .aspx page:

```
<asp:Label id=”TitleLabel" runat="server" Text=”hello" />
```

The text attribute is the content of the label.

- In the code-behind page:

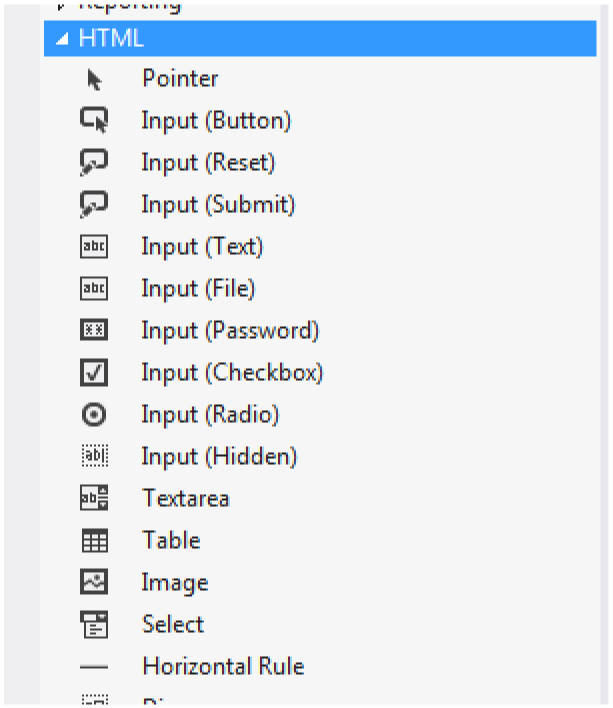
The “id” attribute value is used to programmatically reference the control.

```
protected void Page_Load(object sender, EventArgse)
{
   Response.Write(this.TitleLabel.Text);
   TitleLabel.Text = "hello";
   TitleLabel.BackColor = System.Drawing.Color.LightGreen;
}
```

We can manipulate the object by assigning values to its attributes. ASP.Net will generate corresponding HTML or CSS style code.

HTML Controls

- Always map directly to HTML tags
- All attributes are strictly compatible (map 1:1) with HTML
  - table.bgcolor = "red“;
- They allow us to perform ‘some’ server side processing
- Derived from
- `System.Web.UI.HtmlControls.HtmlControl`
- Supported controls have custom class, others derive from `HtmlGenericControl`



HTML Controls

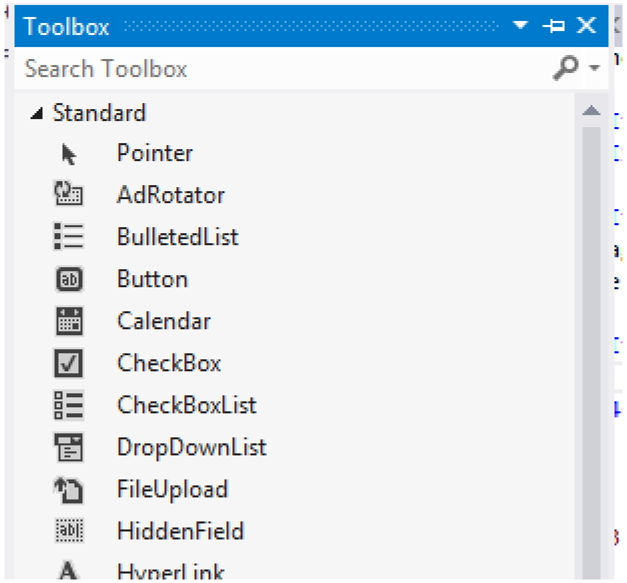
- Supported controls
  - `<a>`
  - `<img>`
  - `<form>`
  - `<table>`
  - `<tr>`
  - `<td>`
  - `<th>`
  - `<select>`
  - `<textarea>`
  - `<button>`
  - `<input type=text>`
  - `<input type=file>`
  - `<input type=submit>`
  - `<input type=button>`
  - `<input type=reset>`
  - `<input type=hidden>`

Using HTML Controls

- Can use controls two ways:
  - Handle everything in action events (e.g. button click)
    - Event code will read the values of other controls (e.g. text, check boxes, radio buttons, select lists)
  - Handle change events as well as action events
- The properties are simple
  - **`Attributes`** returns a collection of attribute / key value pairs
  - **`Style`** gets a CSS collection of applied CSS properties
  - **`Disabled`** indicates whether the control is disabled

Web Server Controls

- Are implemented by the ASP server as .NET Framework classes having a common .NET programming interface
- Web Server controls often have a richer programming interface



Web Server Controls

- Consistent object model

```
Label1.BackColor = Color.Red;
Table1.BackColor = Color.Blue;
```

- Richer functionality
  - E.g. AutoPostBack, additional methods
- Strongly-typed; no generic control
  - Enables better compiler type checking

Web Server Controls

- Web controls appear in HTML markup as namespaced tags
- Web controls have an asp: prefix

```
<asp:button onclick = "button1_click‟ runat=‟server”>
<asp:textbox onchanged = "text1_changed‟ runat=‟server”>
```

- Defined in the System.Web.UI.WebControls namespace
- This namespace is automatically mapped to the asp: prefix

Web Server Control Category

- Basic web controls
  - Basic Web controls provide the similar functionality as HTML server control, with additional methods, events, and properties.
- List controls
  - List controls are special Web server controls that support binding to collections.
- Rich controls
  - Rich controls are built with multiple HTML elements and contain rich functionality.
  - Such as login, calendar, sitemap, etc.
- Web server control complete reference

  http://msdn.microsoft.com/en-us/library/zfzfkea6(v=VS.90).aspx

Basic Web Controls

- Correspond to HTML controls
- Supported controls
  - `<asp:button>`
  - `<asp:imagebutton>`
  - `<asp:linkbutton>`
  - `<asp:hyperlink>`
  - `<asp:textbox>`
  - `<asp:checkbox>`
  - `<asp:radiobutton>`
  - `<asp:image>`
  - `<asp:label>`
  - `<asp:panel>`
  - `<asp:table>`
- `TextBox`, `ListControl`, `CheckBox` don’t automatically do a postback when their controls are changed
- Specify `AutoPostBack = true` to make change events cause a postback

Basic Web Controls

- Form

```
<form id="form1" runat="server"> … </form>
```

- Label

```
<asp:Label id="Label1" runat="server" Text="hello" />
```

- Textbox

```
<asp:TextBox id="TextBox1" runat="server" Text="hello" />
```

- Button

```
<asp:Button id="Button1" runat="server" Text="Go" />
```

- Checkbox

```
<asp:CheckBox id="CheckBox1" runat="server" />
```

List Controls

- Controls that handle repetition
- Supported controls
  - `<asp:dropdownlist>`
  - `<asp:listbox>`
  - `<asp:radiobuttonlist>`
  - `<asp:checkboxlist>`
  - `<asp:repeater>`
  - `<asp:datalist>`
  - `<asp:datagrid>`

Basic List Controls

- Checkbox list

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server">
<asp:ListItemText="" Value=""></asp:ListItem>
</asp:CheckBoxList>
```

- Radio button list

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server">
<asp:ListItemText="" Value=""></asp:ListItem>
</asp:RadioButtonList>
```

- Dropdown list

```
<asp:DropDownList ID="DropDownList1" runat="server">
<asp:ListItemText="" Value="" />
</asp:DropDownList>
```

Basic List Controls

- Listbox list

```
<asp:CheckBoxList id="Check1" runat="server">
  <asp:ListItem>Item 1</asp:ListItem>
  <asp:ListItem>Item 2</asp:ListItem>
  <asp:ListItem>Item 3</asp:ListItem>
  <asp:ListItem>Item 4</asp:ListItem>
  <asp:ListItem>Item 5</asp:ListItem>
</asp:CheckBoxList>
```

- Provides a collection of check box or radio button controls
- Can be populated via data binding

ASP.NET Server Controls (Properties)

- Web Controls provide extensive properties to control display and format, e.g.
  - `Font`
  - `BackColor`, `ForeColor`
  - `BorderColor`, `BorderStyle`, `BorderWidth`
  - `Style`, `CssClass`
  - `Height`, `Width`
  - `Visible`, `Enabled`

ASP.NET Server Controls (Properties)

- **`ID`** - Name that will be used to reference the control instance programmatically
- **`Page`** - Page object on which the control resides
- **`Parent`** - Parent control instance, use for container controls
- **`Visible`** - Make the control instance visible or invisible
- **`EnableViewState`** - Defines whether contents are persisted through view state
-

ASP.NET Server Controls (Properties)

- The **`Style`** property contains references to a collection of

  CSS style

```
MyControl.Style["border-color"] = blue;
```

- The **`CssClass`** property contains the name of a defined

  CSS class

```
txtStyleDemo1.CssClass = "TextBox"
```

Outline

- ASP .NET Control
- ***Event Model***
- Using JavaScript in ASP .NET

Server Control Event Model

- Server controls have events and event handling system to manipulate their properties, status, and behaviors.
  - Events are actions or behaviors defined to happen under a certain condition
- Basic event types
  - Life cycle events: each control has several stages from initiation to release.
  - Interaction events: usually raised by user actions, such as button click, selection or value change, etc.

Event Handling

- Event handlers (methods) are the defined actions when an event is raised.
- Normally event handlers need to be registered (bound) to events

```
override protected void OnInit(EventArgse)
{
   base.OnInit(e);
   this.Load += new System.EventHandler(this.LoadHandler);
}

protected void LoadHandler(object sender, System.EventArgse)
{ … }
```

Binding and event handler to the event.

AutoEventWireUp

- ASP.Net provides an automatic way to bind a default event handler to an event, using the attribute “AutoEventWireUp”
- In the .aspx page

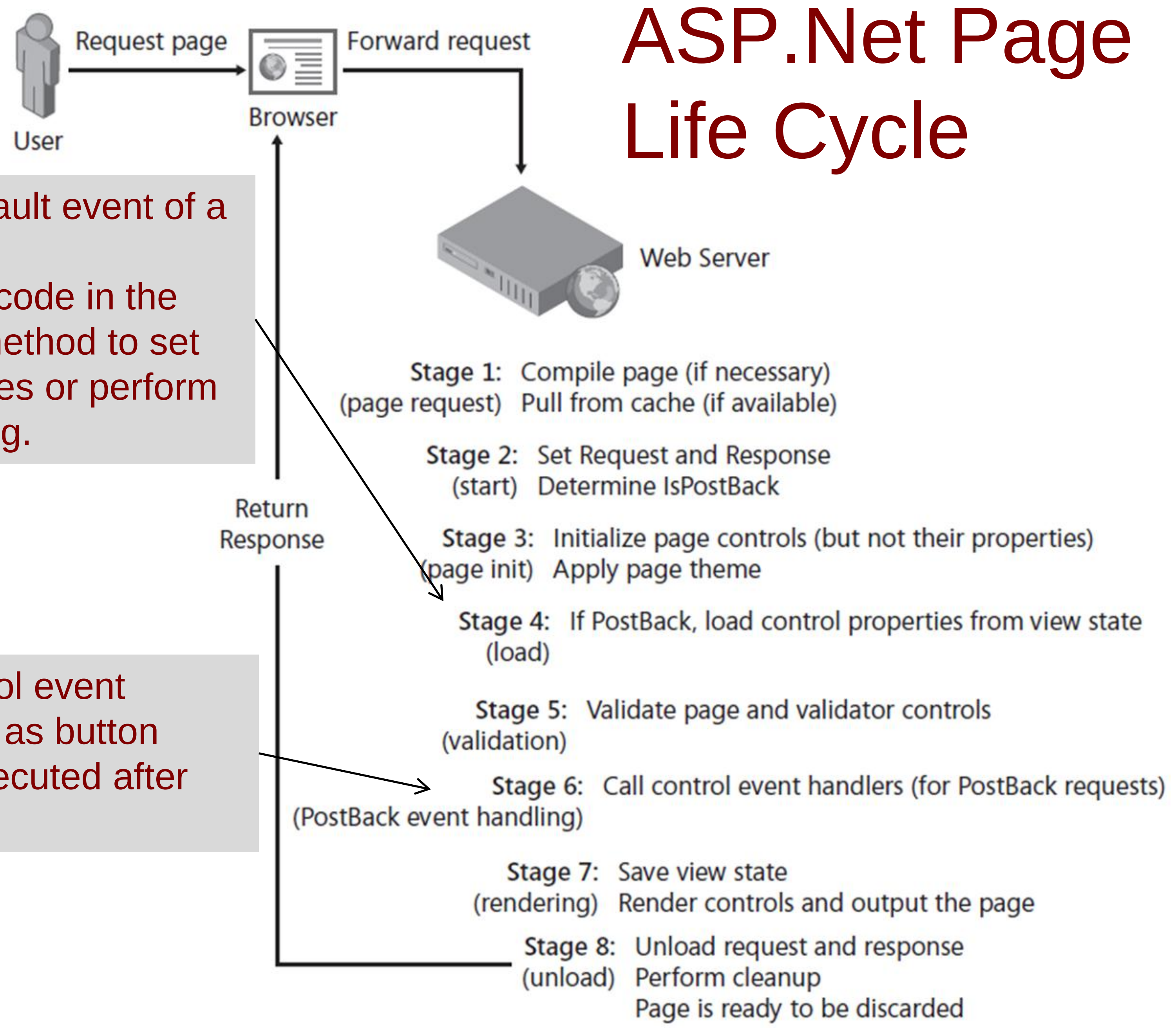
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="…"
Inherits="..." %>
```

- Event handler

```
protected void Page_Load(object sender, EventArgse)
{
   …
}
```

The default event handler needs to be in a predefined naming format.

ASP.Net Page Life Cycle



Stage 1: (page request) Compile page (if necessary) Pull from cache (if available)

Stage 2: (start) Set Request and Response Determine IsPostBack

Stage 3: (page init) Initialize page controls (but not their properties) Apply page theme

Stage 4: (load) If PostBack, load control properties from view state

Stage 5: (validation) Validate page and validator controls

Stage 6: (PostBack event handling) Call control event handlers (for PostBack requests)

Stage 7: (rendering) Save view state Render controls and output the page

Stage 8: (unload) Unload request and response Perform cleanup Page is ready to be discarded

Load is the default event of a Page object.
We often write code in the Page_Load() method to set control properties or perform other processing.

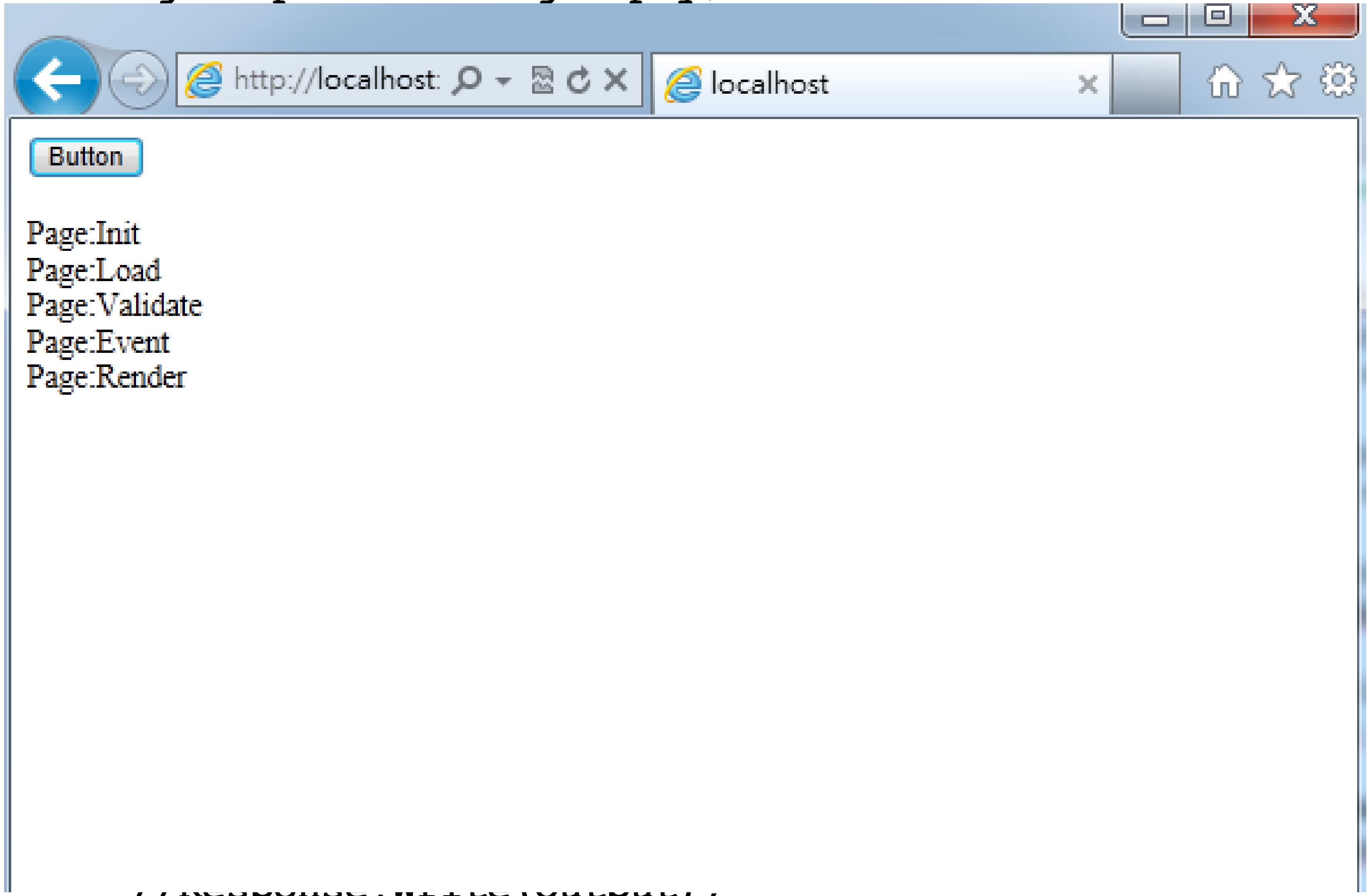
Note that control event handlers (such as button clicked) are executed after loading.

General Page Life-cycle Stages

Stage	Description
Page request	The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled or whether a cached version of the page can be sent in response without running the page.
Start	In the start step, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. Additionally, during the start step, the page's UICulture property is set.
Page initialization	During page initialization, controls on the page are available and each control's UniqueID property is set. Any themes are also applied to the page. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.
Load	During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.
Validation	During validation, the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page.
Postback event handling	If the request is a postback, any event handlers are called.
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering phase, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream of the page's Response property.
Unload	Unload is called after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and any cleanup is performed.

```
public partial class Default : System.Web.UI.Page
{
    string Output = String.Empty;
    protected void Page_Init(object sender, EventArgs e) {
        Output += "Page:Init <br />";
    }
    protected void Page_Load(object sender, EventArgs e) {
        Output += "Page:Load <br />";
    }
    public override void Validate() {
        Output += "Page:Validate <br />";
    }
    protected void Button1_Click(object sender, EventArgs e) {
        Output += "Page:Event <br />";
    }
    protected override void Render(HtmlTextWriter output) {
        Output += "Page:Render <br />";
        base.Render(output);
        Response.Write(Output);
    }
    protected void Page_Unload(object sender, EventArgs e)
    {
        Output += "Page:UnLoad <br />";
        //Response.Write(Output);
    }
}
```

```
public partial class Default : System.Web.UI.Page
{
    string Output = String.Empty;
```



```
Page:Init
Page:Load
Page:Validate
Page:Event
Page:Render
```

```
                                                              {
          //Response.Write(Output);
    }
}
```

Page_Load() Method

- Page_Load() is the default event handling method (auto wired) when a ASP.Netpage has loaded.
- In the .aspx page

```
<h2>Now: <asp:LabelID="CurrectTime" runat="server" Text="" /></h2>
```

- In code-behind page, or <script runat=“server”> code block

```
protected void Page_Load(object sender, EventArgse)
{
    this.CurrectTime.Text = DateTime.Now.ToString();
}
```

Dynamically set the value to the current time in Page_Load(), every time when the page loads, post-back or not.

Event Handler Example

- Button click

```
protected void Button1_Click(object sender, EventArgs e)
```

- Textbox text changed

```
protected void TextBox1_TextChanged(object sender, EventArgs e)
```

- Checkbox, radio button

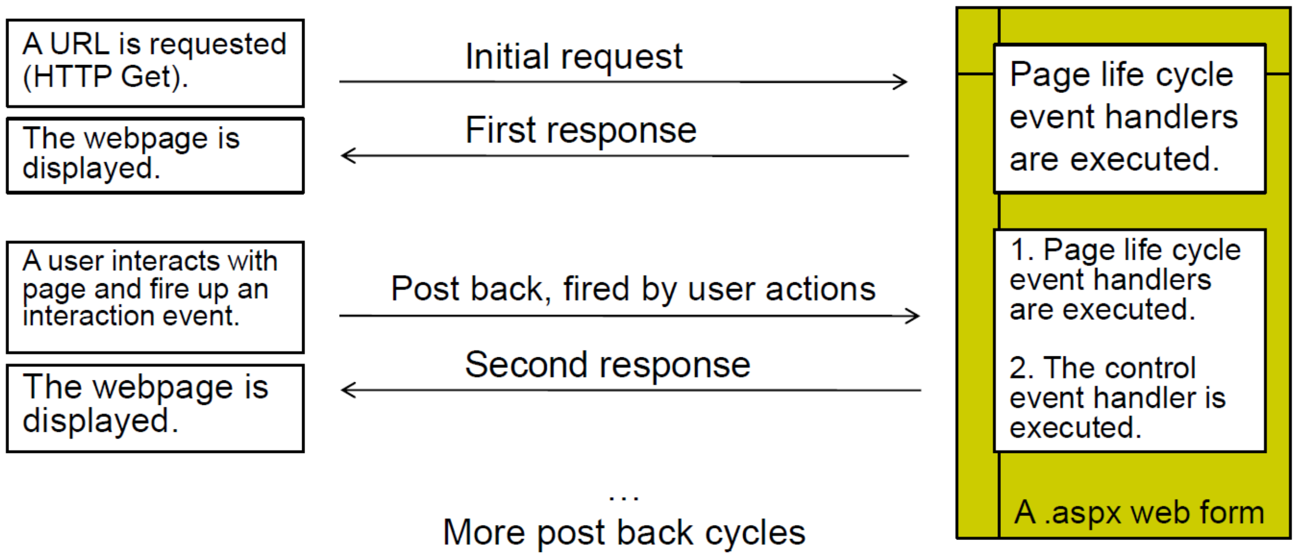
```
protected void CheckBox1_CheckedChanged(object sender,
EventArgs e)
```

- Dropdown list, checkbox list, radio button list

```
protected void DropDownList1_SelectedIndexChanged(object sender,
EventArgs e)
```

Post Back

- Post back refers to a request when a web form posts data to itself, caused by user *interactions* with controls, such as button clicks.
- In a post back, control *interaction event handlers* are called.


A URL is requested (HTTP Get).
The webpage is displayed.
A user interacts with page and fire up an interaction event.
The webpage is displayed.
Initial request
First response
Post back, fired by user actions
Second response
...
More post back cycles
Page life cycle event handlers are executed.
1. Page life cycle event handlers are executed.
2. The control event handler is executed.
A .aspx web form

Post Back Processing

- **`IsPostBack`** - Use the "IsPostBack" property to determine if the page

Change the text only when the page is posted back.

```
if (IsPostBack == true)
   this.CurrectTime.Text = DateTime.Now.ToString();
```

- Auto post back
  - By default, only clicking a button or hitting the enter key in a textbox will cause post back.
  - Set the "AutoPostBack" property to "True" to allow more controls to cause post back when the default event is fired

Post Back Processing

- Auto post back
  - Set the “AutoPostBack” property to “True” to allow more controls to cause post back when the default event is fired
    - Textbox: text is changed and textbox losses focus.
    - Checkbox or radio button: check status is changed.
    - Dropdown list, checkbox list, radio button list: selected item is changed.

```
<asp:CheckBoxID="CheckBox1" runat="server" AutoPostBack="true"
/>
```

View State

- View state is a mechanism used by ASP.NET to store server controls' status between page post-backs.
  - The view state information is stored as an HTML hidden variable in forms and sent in the page's response back to the user.
  - When the user makes the next request, the view state is returned with his or her request.
  - When the page processes, ASP.NET pulls the view state from the page and uses it to reset property values of the page and its controls.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEWAwKOqqrhDwKTjKGwCgKM54rGBpzsu/JHJTw+hNa4kX7r9bgCnes
r" />
```

Outline

- ASP .NET Control
- Event Model
- ***Using JavaScript in ASP .NET***

Using JavaScript

- To get client-side JavaScript into our ASP.NET pages and call that code from the client:
  - Include client-side script into script blocks
  - Create dynamically with
    - **`RegisterClientScriptBlock`**
    - **`RegisterStartupScript`**
    - **`RegisterClientScriptInclude`**

RegisterClientScriptBlock

- Code places the JavaScript directly *after* the opening <form> element in the page.
  - The code executes “**before**” the page has completely loaded
  - Good to register functions but bad for referencing controls

```
public void RegisterClientScriptBlock(
    Type   type,  //The type of the client script to register.
    string key,   //The key of the client script to register.
    string script,//The client script literal to register.
    bool   addScriptTags //A Boolean value indicating whether
                         //to add script tags.
)
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <script type="text/javascript" language="javascript">
       function SayHi() { alert('hi');  }
   </script>
</head>
<body>
    <form id="form1" runat="server">
           Some control here
    </form>
</body>
</html>
```

```
protected void Page_Load(object sender, EventArgs e)
{
    ClientScript.RegisterClientScriptBlock(
        this.GetType(),
        "Alert",
        "SayHi();",
        true );
}
```

RegisterStartupScript

- Code is placed at the **end** of the page
  - Use when you want to reference other page controls
  - Remember that HTML pages rendered and the DOM is processed sequentially

```
public void RegisterStartupScript(
    Type   type,  //The type of the startup script to register.
    string key,   //The key of the startup script to register.
    string script,//The startup script literal to register.
    bool   addScriptTags //A Boolean value indicating whether
                         //to add script tags.
)
```

RegisterClientScriptInclude

- Registers the client script include with the Page object

  using a type, a key, and a URL.
  - the code appears at the **beginning** of the page
  - Use to grab a bunch of JavaScript from a file

```
public void RegisterClientScriptInclude(
   Type   type, //The type of the client script include to
                 register.
   string key,  //The key of the client script include to
                 register.
   string url   //The URL of the client script include to
                 register.
)
```

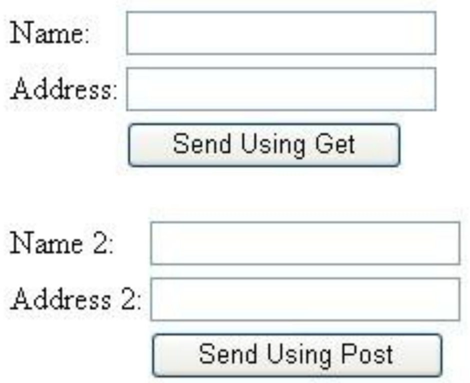
```
public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
       Page.ClientScript.RegisterClientScriptBlock(
           this.GetType(),
           "ScriptBlock",
           "alert('RegisterClientScriptBlock OK!! -- 1')", true);

       Page.ClientScript.RegisterStartupScript(
           this.GetType(),
           "StartupScript",
           "alert('RegisterStartupScript OK!! -- 2')", true);

       Page.ClientScript.RegisterClientScriptInclude(
           this.GetType(),
           "ScriptInclude",
           "http://ajax.aspnetcdn.com/ajax/jQuery/" +
           "jquery-1.11.3.min.js");

       Response.Write(
            "<script>alert('Response.Write OK!! -- 3')</script>");
    }
}
```

```
<script>alert('Response.Write OK!! -- 3')</script>
...
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title></title></head>
<body>
   <form method="post" action="Default.aspx" id="form1">
      <script type="text/javascript">
        //<![CDATA[
          alert('RegisterClientScriptBlock OK!! -- 1')//]]>
      </script>
      <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
1.11.3.min.js" type="text/javascript"></script>
      <div class="aspNetHidden">
         <input type="hidden" name="__VIEWSTATEGENERATOR"
                id="__VIEWSTATEGENERATOR" value="0564CA07" />
      </div>
      <script type="text/javascript">
         //<![CDATA[
           alert('RegisterStartupScript OK!! -- 2')//]]>
      </script>
</form>
</body>
</html>
```

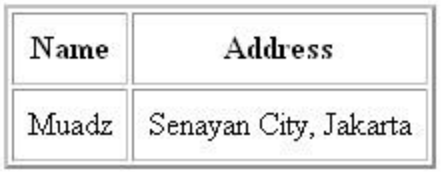
Exercise

- If data passed with Get method we need the following code to retrieve the data:
  - Page.Request.QueryString[<param>];
- If data passed with Post method we need the following code to retrieve the data:
  - Page.Request.Form[<param>];

- Create a page, read.aspx, read the data from the user

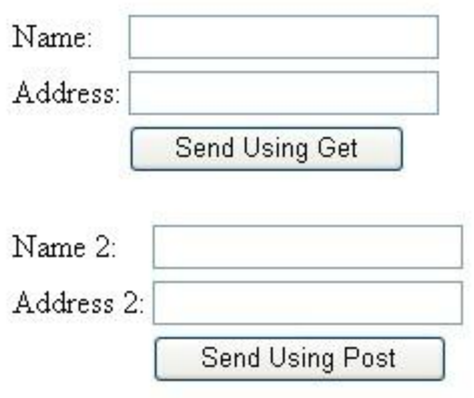


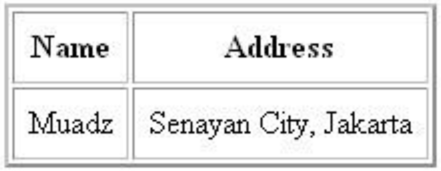
- Create two pages, one is get_display.aspx, the other is post_display.aspx, and then display the data



Name	Address
Muadz	Senayan City, Jakarta

- Combine get_display.aspx, and post_display.aspx into a page, and handle post and get in the same page
- Note: HttpRequest.HttpMethod will give you the HTTP data transfer method (such as GET, POST, or HEAD) used by the client.





Name	Address
Muadz	Senayan City, Jakarta