JOHN ABBOTT
CEGEP/COLLEGE

(420-PS4-AB)
Web Application Life Cycle

Aref Mourtada

Fall 2017

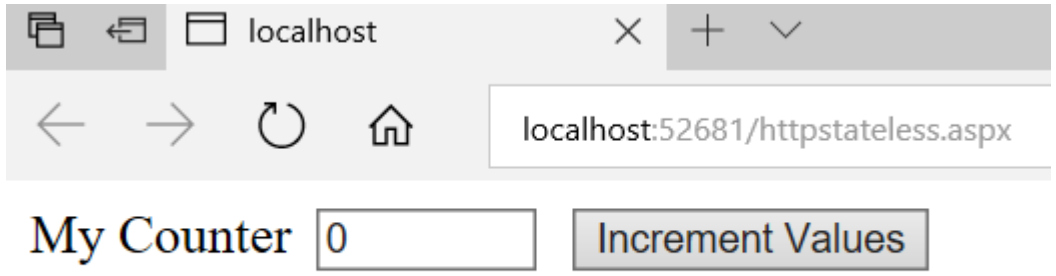
Outline

- HTTP Nature
- State Variables
- Session Variables
- Application Variables
- Web Application Events
- Navigation in a Web Application
- Strongly Typed References

HTTP Protocol

- Web Applications use HTTP protocol.
- HTTP protocol is a stateless protocol
  - It does not retain state between user requests.
  - A web form lives for barely a moment (at the server)
- When a request is received at the server:
  - An Instance of the requested web form is created
  - Events are processed
  - HTML is generated and posted back to the client
  - The web form is immediately disposed (at the server)

Demo: HTTP is Stateless




```
public partial class httpstateless : System.Web.UI.Page
{
    //define a value
    int numberOfClicks = 0;

    protected void Page_Load(object sender, EventArgs e)
    {
        if(!IsPostBack)
        {
            tbNumberofClicks.Text = "0";
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        numberOfClicks++;
        tbNumberofClicks.Text = numberOfClicks.ToString();
    }
}
```

Adding a State Variable

- A state variable travels with every request and respond between client and web browser.

- A state variable can be accessed/assigned value using

```
ViewState["variableName"] = value;
```

- A state variable can be accessed in the code behind.
- The default value for a variable before assignment is `null`.

Demo: ViewState

```
public partial class WebForm1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            tbNumberofClicks.Text = "0";
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        if(ViewState["Clicks"] == null)
        {
            ViewState["Clicks"] = 1;
        }
        tbNumberofClicks.Text = ViewState["Clicks"].ToString();
        ViewState["Clicks"] = (int)ViewState["Clicks"] + 1;
    }
}
```

Web Controls Maintain Properties

- As web forms have short lifetimes, ASP .NET takes special steps to preserve the data entered in the web controls.
- Data entered in controls is sent with each request and restored to controls in `Page_Init`.
- The data & properties of the controls are available in the `Page_Load()`, `Button_Click()` and all other events.

Demo: Web Control Maintaining Value

- Upon clicking the button, the value gets incremented correctly as expected.
- This is possible because the text box is an ASP .NET server control.
- It uses ViewState internally to preserve data across postbacks.

```
public partial class ControlValue : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            tbNumberofClicks.Text = "0";
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        int numberOfClicks = Int32.Parse(tbNumberofClicks.Text) + 1;
        tbNumberofClicks.Text = numberOfClicks.ToString();

    }
}
```

Preserving Data across the Web Application

- ViewState variables are used to preserve data within page post back. By default, a ViewState in a single web form is not available in another web form.

- Techniques to send data from one web form to another:
  1. Session State
  2. Application State
  3. Query Strings
  4. Cookies

Sessions & Session Variables

- What is a *Session* in a web application?
  - A session is a unique instance of the browser.
  - A single user can have multiple sessions by visiting the web application with multiple instances of the browser running with a different session-IDs.

How to get a new session-id? How to force the Session_Start() event to execute?

- Open a different browser. (or private mode)
- Use Cookie-less Sessions (from web.config)

```
<sessionState mode="InProc" cookieless="true"/>
```

ViewState vs Session

ViewState

- ViewState of a web form is available only with in that web form.
- ViewState is stored on the page in a hidden field called ViewState.
- ViewState is used by all ASP .NET controls to retain their state across post backs.

Session:

- Session variables are available across all pages, but only for a given single session.
- Session variables are like singe-user global data.
- Session state variables are stored on the web server.
- Session state variables are cleared, when the user session times out. The default is 20 minutes.
- This is configurable in web.config

```
<sessionState mode="InProc"
timeout="1"/>
```

Demo: Session Variables

```
protected void btnIncerment_Click(object sender, EventArgs e)
{

    if (Session["click"] == null)
    {
        Session["click"] = 0;
    }
    Session["click"] = (int)Session["click"] + 1;
    tbCounter.Text = Session["click"].ToString();

}
```

Application State

- Application State variables are available across all pages and across all sessions.
- Application State variables are like multi-user global data variables.
- Application State variables are stored on the web server.
- Application State variables are cleared, when the process hosting the application is restarted.

Session State Variables vs Application State Variables

- **Session State** variables are available across all pages but only for a given single session.
- Session variables are like single-user global data
  - Only the current session has access to its Session State variables.

- **Application State** variables are available across all pages and across all sessions.
- Application State variables are like multi-user global data.
  - All sessions can read and write Application State variables.

Demo: Application Variables

```
protected void btnIncerment_Click(object sender, EventArgs e)
{
    if (Application["click"] == null)
    {
        Application["click"] = 0;
    }
    Application["click"] = (int)Application["click"] + 1;
    tbCounter.Text = Application["click"].ToString();
}
```

Web Application Events

- In a web application, events can occur or can be triggered at 3 levels:
  - Application Level
    - Example: Application Start, Application End
  - Page Level
    - Example: Page Load
  - Control Level
    - Example: Button Click

Application Level Events

- In an ASP.NET web application, Global.asax file contains the application level events.
- In general, Application events are used to initialize data that needs to be available to all the sessions of the application.
- Session events are used to in initialize data that needs to be available only for a given individual session, but not between multiple ones.

Global.asax Event Methods

```
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
}
void Application_End(object sender, EventArgs e)
{
    //  Code that runs on application shutdown
}
void Application_Error(object sender, EventArgs e)
{
    // Code that runs when an unhandled error occurs
}
void Session_Start(object sender, EventArgs e)
{
    // Code that runs when a new session is started
}
void Session_End(object sender, EventArgs e)
{
    // Code that runs when a session ends.
    // Note: The Session_End event is raised only when the sessionstate mode
    // is set to InProc in the Web.config file. If session mode is set to StateServer
    // or SQLServer, the event is not raised.
}
```

Demo: Using Application Variables with Events Global.asax

JOHN ABBOTT CEGEP/COLLEGE

```
protected void Application_Start(object sender, EventArgs e)
{
    //Create application state variables
    Application["NumberofApplications"] = 0;
    Application["NumberofSessions"] = 0;

    Application["NumberofApplications"] = (int)Application["NumberofApplications"] + 1;
}

protected void Session_Start(object sender, EventArgs e)
{
    Application["NumberofSessions"] = (int)Application["NumberofSessions"] + 1;
}

protected void Session_End(object sender, EventArgs e)
{
    Application["NumberofSessions"] = (int)Application["NumberofSessions"] - 1;
}
```

Demo: Accessing Application Variables in a Form

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Number of applications = " + Application["NumberofApplications"]);
    Response.Write("<br/>");
    Response.Write("Number of online users = " + Application["NumberofSessions"]);

}
```

Navigation in a Web Application

- What are the different page navigation techniques in ASP .NET?
- How would you link pages in an application?
- How do you move from one web form to another web form in asp.net?
- This is a very common interview question in ASP .NET.

Navigation Techniques

- Hyperlink (HTML \ ASP .NET)
- Response.Redirect
- Server.Transfer
- Server.Execute
- Cross-Page postback
- Window.Open (using scripts)

HTML Hyperlink

- Used to navigate across web pages.
- Using hyperlink, you can navigate to another page with in the same application or to an external website.
- The hyperlink uses anchor tag <a>

ASP Hyperlink Control

- The hyperlink control is rendered as an HTML anchor tag `<a>`.
- NavigateURL property is used to navigate to a specific page or form.
  - Can be changed from the server side.
- Maintains History.
- For local pages use "`~/pageName.aspx`"
- Can be used for external links.

Demo: ASP .NET Hyperlink

- On Server

```
<asp:HyperLink ID="HyperLink1" NavigateUrl="~/WebForm2.aspx" Text="Go to webform2" runat="server"></asp:Hype
```

- On Client

```
<div style="font-family: Arial">
    <a id="HyperLink1" href="WebForm2.aspx">Go to webform2</a>
</div>
</form>
```

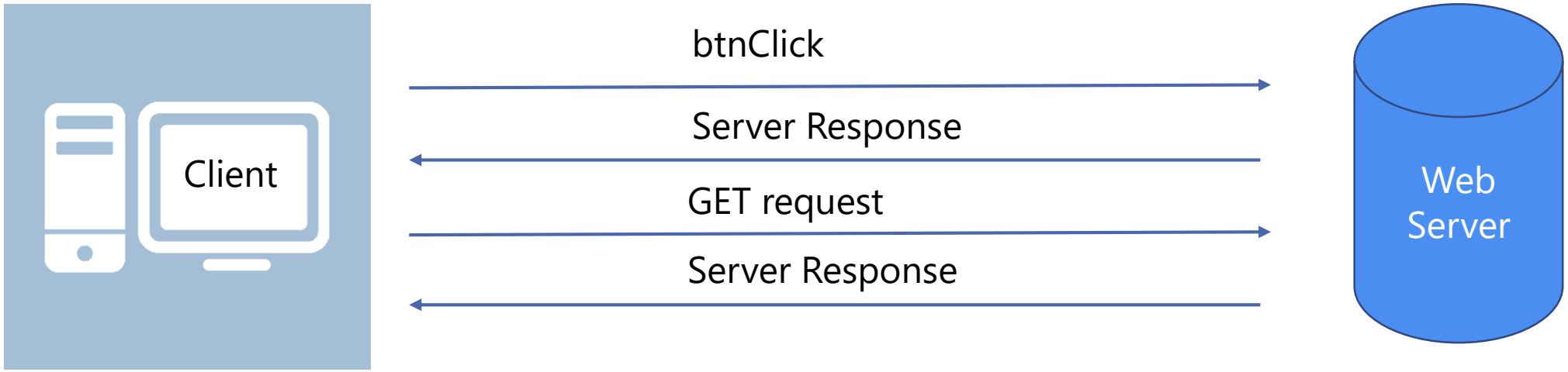
Response.Redirect

- Response.Redirect is similar to clicking on a hyperlink.
  - Does not expose any server side events.
- Maintains history.
- Can be used to navigate internal and external links.

Response.Redirect Cycle

- Upon firing an web control event (button click, drop down list index change.. Etc.) :
  - Fired control event → code calls Response-Redirect() method.
  - Web server receives a request for redirection.
  - Web server then sends a response header to the client.
  - Browser then automatically issues a new GET request to the requested web server.
  - Web server will then provide the new page.
- In short, Response.Redirect causes **2** request/response cycles.

Response.Redirect Cycle



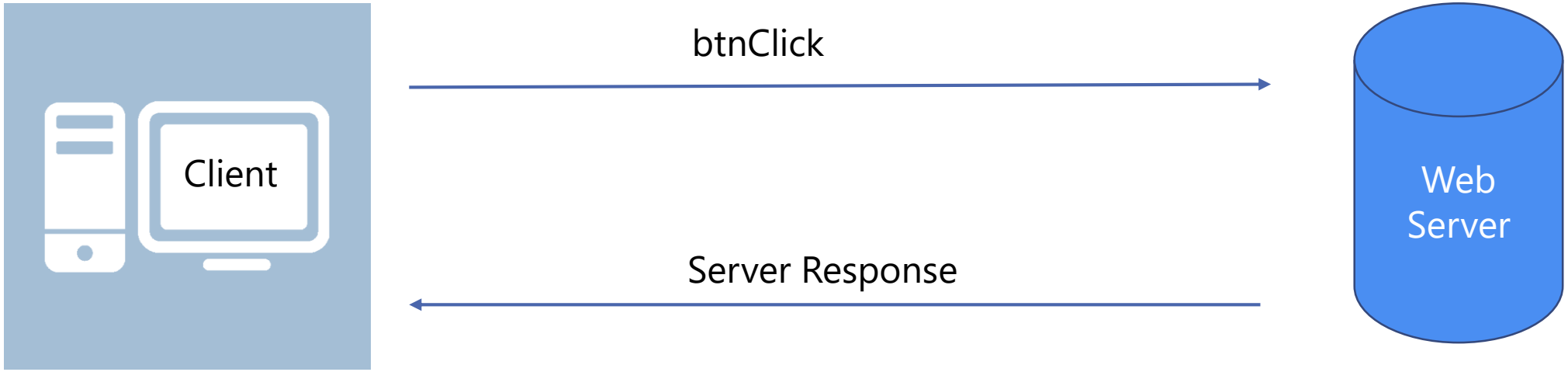
Demo: Response.Redirect

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Redirect("~/WebForm2.aspx");
}
```

Server.Transfer

- Cannot be used to navigate to sites/pages on different webservers.
- Does not change the URL in the address bar.
- Much faster than Response.Redirect as the redirection happens in one Request/Response cycle.
- Form variables from the original request are preserved and accessed through
  - Request.Form (can be disabled)
  - Page.PreviousPage

Server.Transfer Cycle



Demo: Server.Transfer

On the first page

```
protected void btnTransfer_Click(object sender, EventArgs e)
{
    Server.Transfer("~/WebForm2.aspx");
}
```

```
protected void btnTransfer_Click(object sender, EventArgs e)
{
    Server.Transfer("~/WebForm2.aspx", false);
}
```

Demo: Get the Previous Page Values

```
protected void Page_Load(object sender, EventArgs e)
{
    System.Collections.Specialized.NameValueCollection previousFormcollection = Request.Form;
    lblName.Text = previousFormcollection["txtName"];
    lblEmail.Text = previousFormcollection["txtEmail"];
}
```

```
Page previousPage = Page.PreviousPage;
if (previousPage != null)
{
    lblName.Text = ((TextBox)previousPage.FindControl("txtName")).Text;
    lblEmail.Text = ((TextBox)previousPage.FindControl("txtEmail")).Text;
}
```

Server.Execute

- Very similar to Server.Transfer
  - Cannot be used to navigate to sites/pages on different webservers.
  - Does not change the URL in the address bar.
  - Preserves the Form Variables.
- Difference
  - Server.Transfer terminates the execution of the current page and starts the execution of the new page.
  - Server.Execute processes the second web form without leaving the first one. After completing the execution of the first form, the control returns the second web form all in one form.

Demo: Server.Execute

```
protected void btnExecute_Click(object sender, EventArgs e)
{
    Server.Execute("~/WebForm2.aspx");
}
```

Cross Page Posting

- By default, upon a button click → page will be posted back to itself.
- To post back to a different web form upon a button click → use the `PostBackURL` property.
- The target form can check if it was accessed through a regular hyperlink(or any other type) or a post back property using `Page.IsCrossPageBack`.
- This will allow you to know if a web form was accessed through proper path or not.

Demo: PostBackURL

```
Page previousPage = Page.PreviousPage;
if (previousPage != null && previousPage.IsCrossPagePostBack)
{
    lblName.Text = ((TextBox)previousPage.FindControl("txtName")).Text;
    lblEmail.Text = ((TextBox)previousPage.FindControl("txtEmail")).Text;
}
else
{
    lblStatus.Text = "You landed on this page using a technique other than cross page post back";
}
```

Strongly Typed References

- FindControl() method will generate an exception if the control ID name was misspelled.
- Solution: Having strongly typed references.
- How?
  - Create public properties.
  - TypeCasting using the previous form type.

Demo: Create Public Properties

- On the main form we create public properties that we want to exposed to target form.
- Get provides read-only functionality.

```
public string name
{
    get
    {
        return tbName.Text;
    }
}

public string email
{
    get
    {
        return tbEmail.Text;
    }
}
```

Demo: Type Casting (on target form)

```
protected void Page_Load(object sender, EventArgs e)
{
    StronglyTypedReferencesMain previosPage = (StronglyTypedReferencesMain)Page.PreviousPage;
    if (previosPage != null)
    {
        lbName.Text = previosPage.name;
        lbEmail.Text = previosPage.email;
    }
}
```

Q & A