State Management

Introduction

- All real-world applications need to maintain their own state to serve users' requests.
- Web Applications require special system-level tools to manage state.
- The reason is stateless nature of HTTP, protocol for web application.

Why State Management?

- Web applications use a highly efficient disconnected access pattern.
- In a web request, the client connects to the web server and requests a page. When the page is delivered, the connection is severed, and the web server abandons any information it has about the client.
- As client needs to be connected for few seconds, a web server can handle thousands of requests without a performance hit.
- To retain information of user actions, state management is required.

State Management

- We are working in a connectionless environment
- Which means we can’t use normal variables to persist data between server round trips, and pages.
- Instead we have a range of techniques provided by Asp.Net and HTTP that we can use:

Types of State Management

Server-Side State Management	Client-Side State Management
Application state ● Information is available to all users of a Web application	Cookies ● Text file stores information to maintain state
Session state ● Information is available only to a user of a specific session	The ViewState property ● Retains values between multiple requests for the same page
Database ● In some cases, use database support to maintain state on your Web site	Query strings ● Information appended to the end of a URL

ViewState

- ASP.Net mechanism to persist data **through server round trips** on a single page.

- All controls (including the Page object) have *EnableViewState* property that is defaulted to true

- You can add your own data to view state using :

```
ViewState["Price"]==price;
decimal price=(decimal)ViewState["Price"];
```

- ViewState materializes as a hidden field in the HTML output (_VIEWSTATE).

- Tamper-proof; optionally encryptable.

QUERYSTRING

- A query string is information appended to the end of a page's URL. A typical example might look like the following:

QueryString

$\text{http://localhost/test.aspx}\boxed{\text{?}\underbrace{\text{category=basic}}\text{\&}\underbrace{\text{price=100}}}$

- Querystring can pass data from one page to another, even across web sites and servers.

QUERYSTRING

- Read querystring values through *Request.QueryString*

```
MyValue = Request.QueryString[“category"];
```

Cookies

- Cookies store data in the browser
  - Either memory resident only (because no Expiry Date is given) and so only available throughout a session.
  - Or if an Expiry Date is given then Cookies will persist between sessions for a user (with a consistent profile, login or machine)

```
HttpCookie ck = new HttpCookie("MyVariable",MyValue);
```

Cookies

- Create a cookie using *HttpCookie* and then add it to *Response.Cookies* collection to send it to the browser.

```
ck.Expires = DateTime.Now.AddDays(30);
Response.Cookies.Add(ck);
```

Cookies

- Read a cookie by retrieving it from the *Request.Cookies* collection:

```
HttpCookie ck=Request.Cookies[“MyVariable”];
```

- Delete a cookie by setting it's expiry date to yesterday (or earlier) and adding to the Response.Cookies collection

```
ck.Expires = DateTime.Now.AddDays(-1);
Response.Cookies.Add(ck);
```

HttpCookie Properties

Name	*Description*
Name	Cookie name (e.g., "UserName=Jeffpro")
Value	Cookie value (e.g., "UserName=Jeffpro")
Values	Collection of cookie values (multivalue cookies only)
HasKeys	True if cookie contains multiple values
Domain	Domain to transmit cookie to
Expires	Cookie's expiration date and time
Secure	True if cookie should only be transmitted over HTTPS
Path	Path to transmit cookie to

Client-Side State Management

- Less Secure
- Less Reliable
- Limits amount of information.

Cross-Page PostBack

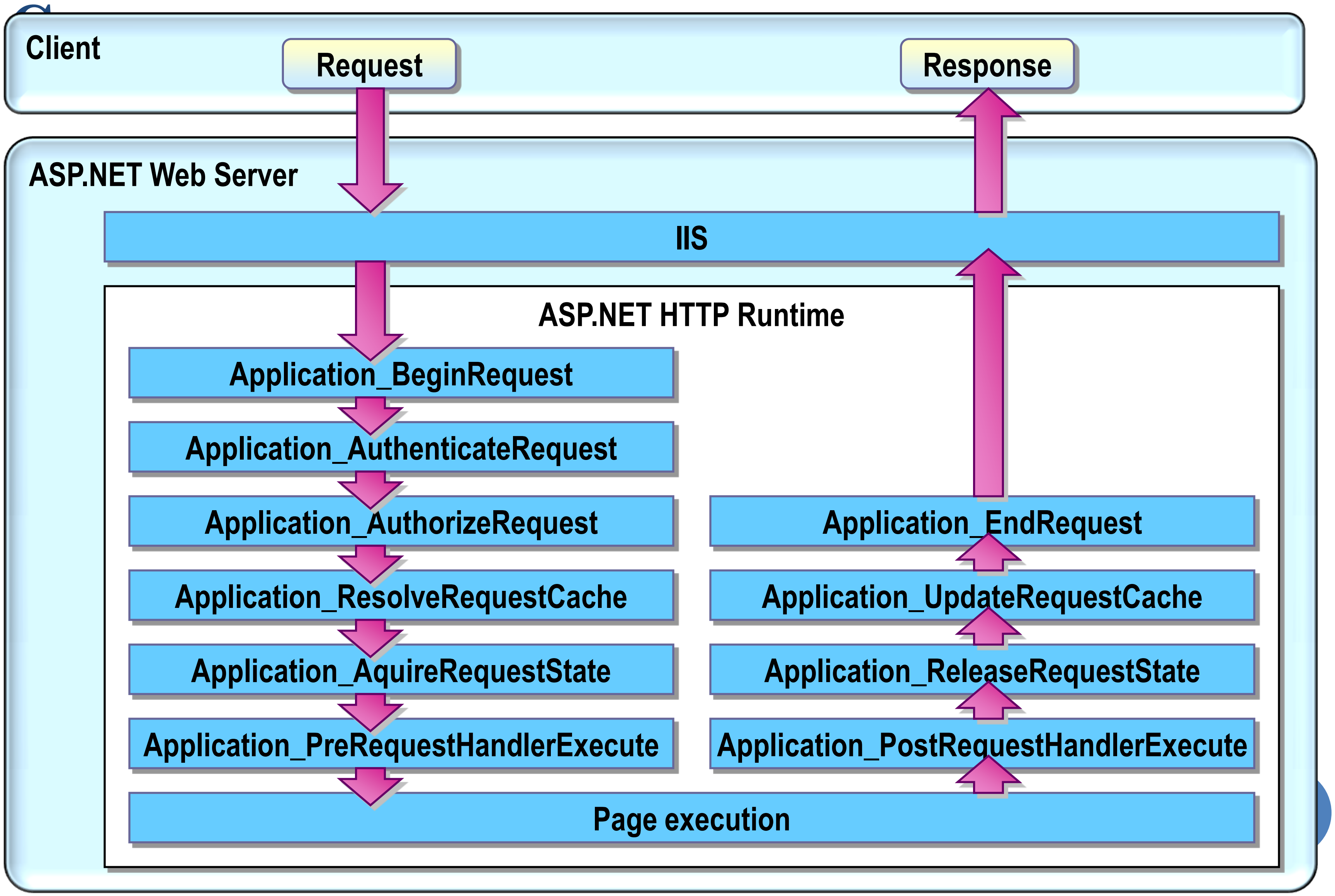
- When a cross-page request occurs, the *PreviousPage* property of the current Page class holds a reference to the page that caused the postback.
- To get a control reference from the *PreviousPage*, use the *Controls* property or use the *FindControl* method.
- To avoid casting, one can use <%@ PreviousPageType …%> directive.

```
TextBox text = PreviousPage.FindControl(“TextBox1”) as TextBox;
```

```
<%@ PreviousPageType VirtualPath=“somepage.aspx” %>
```

Global.asax file

- Only one Global.asax file per Web application.
- Used to handle application and session events.
- The file is optional.


Client
Request
Response
ASP.NET Web Server
IIS
ASP.NET HTTP Runtime
Application_BeginRequest
Application_AuthenticateRequest
Application_AuthorizeRequest
Application_ResolveRequestCache
Application_AquireRequestState
Application_PreRequestHandlerExecute
Page execution
Application_PostRequestHandlerExecute
Application_ReleaseRequestState
Application_UpdateRequestCache
Application_EndRequest

Session State

- Read/write per-user data store
- Accessed through Session property
  - Page.Session – ASPX
  - HttpApplication.Session - Global.asax
- Provider-based for flexible data storage
  - In-process (default)
  - State server process
  - SQL Server
- Cookied or cookieless

Session State

- Sessions timeout after a period of inactivity (default 20mins. Can configured in web.config)
- Global.asax has events for the Session that one can utilize (e.g. Session_Start)
- By default session uses a memory resident cookie to relate the user to their set of variables.
- ASP.NET tracks each session using a unique 120-bit identifier, referred as *SessionID*.
- Session state is available *after* the *AcquireRequestState* event fires.

Session State

- *Session_OnEnd* event is supported only in In-Process mode (default mode).
- Disadvantage of in process storage:
  - Not Scalable
- ASP.NET provides out of process storage of session state
  - State can be stored in a SQL Server database or a state server
- Advantages of out of process storage:
  - Scalable

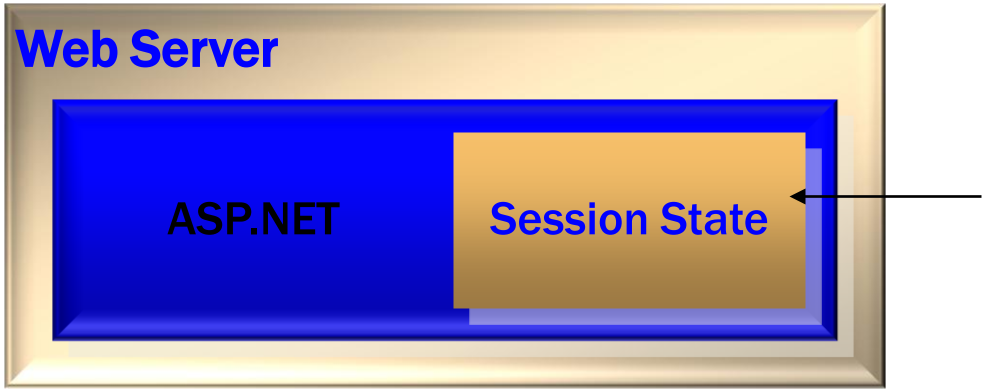
Using Session State

```
// Write a ShoppingCart object to session state
ShoppingCart cart = new ShoppingCart ();
Session["Cart"] = cart;
  .
  .
  .
// Read this user's ShoppingCart from session state
ShoppingCart cart = (ShoppingCart) Session["Cart"];
  .
  .
  .
// Remove this user's ShoppingCart from session state
Session.Remove ("Cart");
```

In-Process Session

```
<!-- Web.config -->
<configuration>
  <system.web>
    <sessionState mode="InProc" />
      ...
  </system.web>
</configuration>
```

Types stored in session state using in process model do NOT have to be serializable.

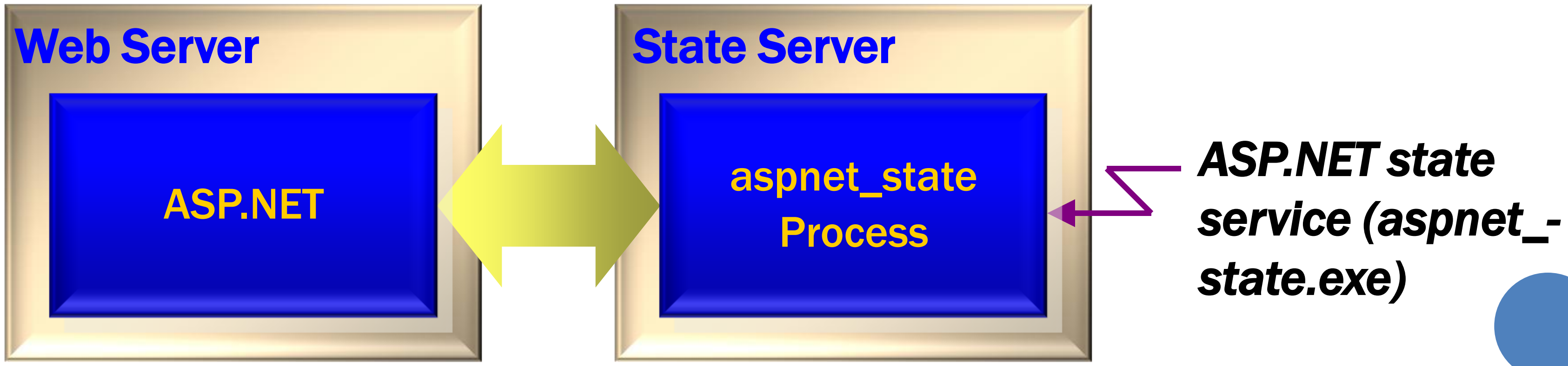


Session state stored inside ASP.NET's worker process

State Server Session State

```
<!-- Web.config -->
<configuration>
  <system.web>
    <sessionState mode="StateServer"
      stateConnectionString="tcpip=10.1.46.46:42424" />
        ...
  </system.web>
</configuration>
```

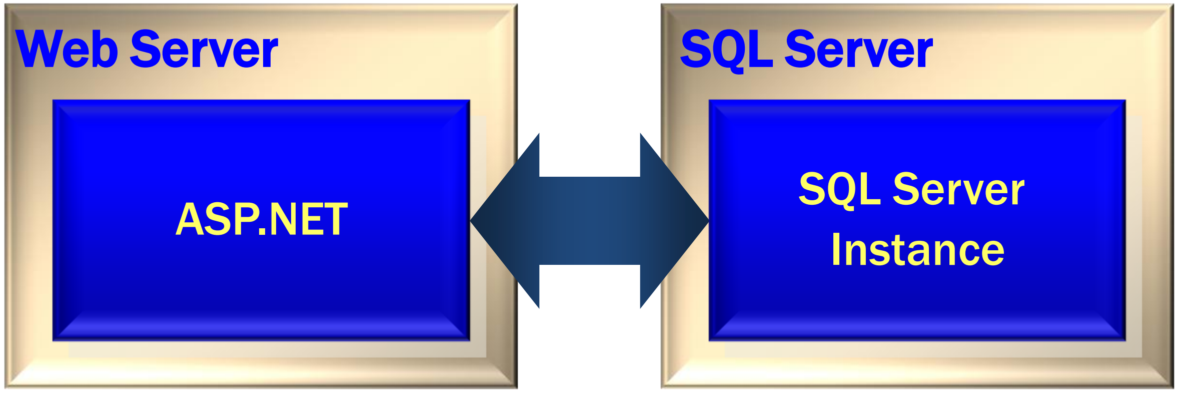
Types stored in session state using this model must be serializable.



SQL Server Session State

```
<!-- Web.config -->
<configuration>
  <system.web>
    <sessionState mode="SQLServer"
      sqlConnectionString="server=.\SQLEXPRESS;integrated security=true" />
        ...
  </system.web>
</configuration>
```

Types stored in session state using this model must be serializable.



Out-of Process Session State

- StateServer, SQLServer are two out-of process session state configuration.
- The ASPState database must be created before SQL Server session state can be used.
- ASP.NET comes with two SQL scripts for creating the database. InstallSqlState.sql creates a database that stores data in temp tables (memory), meaning that if the database server crashes, session state is lost.
- InstallPersistSqlState, which was introduced in ASP.NET 1.1, creates a database that stores data in disk-based tables and thus provides a measure of robustness.

EnableSessionState Attribute

- Attribute of <%@ Page…%> directive.
- *EnableSessionState="True":* The page requires read and write access to the Session. The Session with that SessionID will be locked during each request.
- *EnableSessionState="False":* The page does not require access to the Session.
- *EnableSessionState="ReadOnly":* The page requires read-only access to the Session.

Application State

- Application state allows you to store global objects that can be accessed by any client.
- Items in application state never time out.
- As any client can update application store, simultaneous read/write leads to confusion.
- Use *Lock* and *Unlock* methods, to allow only one client to access application state collection.

Application State

```
protected void Application_Start(Object sender,EventArgs e)
{
  Application["NumberofVisitors"] = 0;
}
```

```
Application.Lock();
Application["NumberOfVisitors"] =
                        (int)Application["NumberOfVisitors"]  + 1;
Application.UnLock();
```

Other State Management Techniques

- Hidden Fields
- Server.Transfer & Context.Items (Short term Storage)
- Rarely used.