ASP.NET State Management

Lecture Overview

- Client state management options
- Server state management options

Introduction to State Management

- Remember that ASP.NET (and the Web) is stateless
  - The Web server does not keep track of past client requests
- Different technologies handle the issue of statement management differently
  - ASP.NET is somewhat unique in this regard

Types of State Management

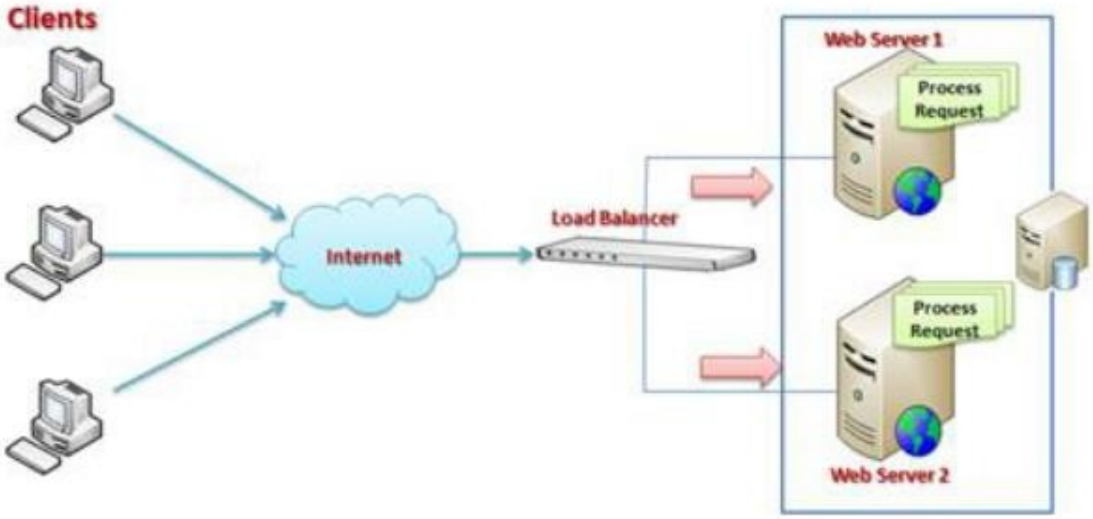
- ASP.NET offers two categories of state management
  - Pure client-side statement management
  - Server-side state management

State Management (Issues)

- Client state management consumes bandwidth and introduces security risks because sensitive data is passed back and forth with each page postback
- Preserving state on a server can overburden servers
  - We also must consider Web farms and Web gardens

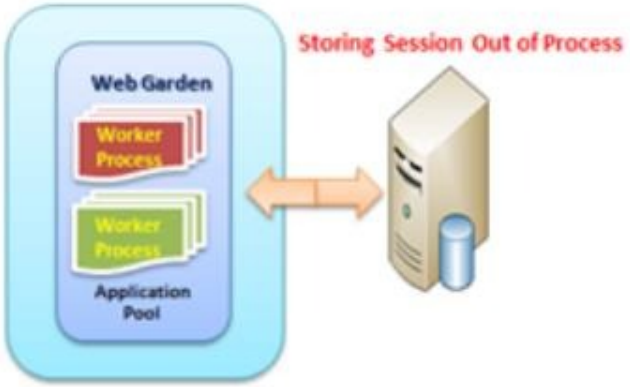
Web Farm

- A single ASP.NET application is hosted on multiple web servers



Web Garden

- A single application pool contains multiple worker processes run across multiple CPUs



Client State Management (Introduction)

- View state
- Control state
- Hidden fields
- Cookies
- Query strings

State Management (`ViewState`)

- `ViewState` works in a couple of different ways
  - It's managed for you automatically via the ASP.NET infrastructure
  - Or you can take control yourself via the `ViewState` object
- `ViewState` only provides state for a single page
  - It Won't work with cross page postbacks or other page transfers

State Management (`ViewState`) (auto)

- Just enable `ViewState` and the corresponding control retains is value from one postback to the next
  - This happens by default
- While simple, it does add overhead in terms of page size
- `MaxPageStateFieldLength` controls max size
- You can disable `ViewState` at the page level
  - `<%@ Page EnableViewState="false" %>`

State Management (`ViewState`) (manual)

- `ViewState` can be set on the server up to the `Page_PreRenderComplete` event
- You can save any serializable object to `ViewState`
  - See example ViewState.aspx

State Management (`ControlState`)

- The `ControlState` property allows you to persist information as serialized data
  - It's used with custom controls (UserControl)
- The wiring is not automatic
- You must program the persisted data each round trip
  - The `ControlState` data is stored in hidden fields
- More later when we create a user control

State Management (Hidden Fields)

- Use the `HiddenField` control to store persisted data
- The data is stored in the `Value` property

- It's simple and requires almost no server resources
- It's available from the ToolBox and works much like a text box control

State Management (Cookies)

- Use to store small amounts of frequently changed data
  - Data is stored on the client's hard disk
  - A cookie is associated with a particular URL
- All data is maintained as client cookies
- Most frequently, cookies store personalization information

Cookie Limitations

- While simple, cookies have disadvantages
- A cookie can only be 4096 bytes in size
- Most browsers restrict the total number of cookies per site
- Users can refuse to accept cookies so don't try to use them to store critical information

Cookies Members

- Use the `Response.Cookies` collection to reference a cookie
  - `Value` contains the cookie's value
  - `Expires` contains the cookie's expiration date
- Cookies can also store multiple values using subkeys
  - Similar to a QueryString
  - It's possible to restrict cookie scope by setting the `Path` property to a folder
- See example cookies.aspx

State Management (Query Strings)

- As you know, query strings are just strings appended to a URL
- All browsers support them and no server resources are required

State Management (Query Strings)

- `HttpContext.Current.Request.RawURL` gets a URL posted to the server
  - Or just use `Request.QueryString`
- `HttpUtility.ParseQueryString` breaks a query string into key / value pairs
  - It returns a `NameValueCollection`

Server State Management Options

- Application state
- Session state
- Profile properties
- Database support

Server State Management

- `HttpApplicationState` applies to your entire application
  - `Application` object
- `HttpSessionState` applies to the interaction between a user's browser session and your application
  - `Session` object
- Page caching also relates to state management

The Application's State (Introduction)

- An instance of the `HttpApplicationState` object is created the first time a client requests a page from a virtual directory
  - Application state does not work in a Web farm or Web garden scenario
    - Unless we push data to a database
  - It's volatile – If the server crashes, the data is gone
- It's really just a collection of key/value pairs

The `HttpApplicationState` Class (Members 1)

- The `AllKeys` property returns an array of strings containing all of the keys
- `Count` gets the number of objects in the collection
- `Item` provides read/write access to the collection

The `HttpApplicationState` Class (Members 2)

- `StaticObjects` returns a reference to objects declared in the global.asax file
  - `<object>` tags with the scope set to `Application`
- The `Add` method adds a new value to the collection
- The `Remove` method removes the value associated with a key

The `HttpApplicationState` Class (Members 3)

- `Lock` and `Unlock` lock the collection, respectively
- `Get` returns the value of a collection item
  - The item can be referenced by string key or ordinal index
- `Set` store a value corresponding to the specified key
  - The method is thread safe

Application State (Best Practices)

- Memory to store application state information is permanently allocated
  - You must write explicit code to release that memory
  - So don't try to persist too much application state information
- The `Cache` object, discussed later, provides alternatives to storing application state information

Profile Properties

- It's a way to permanently store user-specific data
- Data is saved to a programmer-defined data store
- It takes quite a bit of programming
- The whole thing is unique to windows

Profiles (Enabling)

- Web.config must be modified to enable profiles

```
<profile>
  <properties>
    <add name="PostalCode" />
  </properties>
</profile>
```

- You can then reference with code as follows

```
Profile.PostalCode = txtPostalCode.Text;
```

Session State (Introduction)

- It is used to preserve state for a user's 'session'
- Session state works in Web farm or Web garden scenarios
  - Since the data is persisted in the page
- Session data can be stored in databases such as SQL Server or Oracle making it persistent
- It's very powerful and the features have been significantly enhanced in ASP 2.0

Session State (Configuration)

- The `<web.config>` file contains a `<sessionState>` section
  - The entries in this section configure the state client manager

Web.Config <sessionState>

```
<sessionState mode="[Off|InProc|StateServer|SQLServer|Custom]"
   timeout="number of minutes" cookieName="session identifier
   cookie name" cookieless=
   "[true|false|AutoDetect|UseCookies|UseUri|UseDeviceProfile]
   " regenerateExpiredSessionId="[True|False]"
   sqlConnectionString="sql connection string"
   sqlCommandTimeout="number of seconds"
   allowCustomSqlDatabase="[True|False]"
   useHostingIdentity="[True|False]"
   stateConnectionString="tcpip=server:port"
   stateNetworkTimeout="number of seconds"
   customProvider="custom provider name">
   <providers>...</providers> </sessionState>
```

Session State Providers (1)

- `Custom` – State information is saved in a custom data store
- `InProc` – State information is preserved in the ASP.NET worker process named (aspnet_wp.exe or w3wp.exe)
  - This is the default option
- `Off` – Session state is disabled

Session State Providers (2)

- `SQLServer` – State data is serialized and stored in an SQL server instance
  - This might be a local or remote SQL Server instance
- `StateServer` – State information is stored in a separate state server process (aspnet_state.exe)
  - This process can be run locally or on another machine

Session State Providers (Best Practices)

- Use an SQL server provider to ensure that session state is preserved
  - Note that there is a performance hit here because the session information is not stored in the page
- The InProc provider is not perfect
  - The ASP worker process might restart thereby affecting session state

Session State Identifiers

- Browser sessions are identified with a unique identifier stored in the `SessionID` property
- The `SessionID` is transmitted between the browser and server via
  - A cookie if cookies are enabled
  - The URL if cookies are disabled
    - Set the `cookieless` attribute to true in the `sessionState` section of the Web.config file

HttpSessionState (Members 1)

- `CookieMode` describes the application's configuration for cookieless sessions
  - `IsCookieless` is used to depict whether cookies are used to persist state
- `IsNewSession` denotes whether the session was created with this request
- `SessionID` gets the unique ID associated with this session
- `Mode` denotes the state client manager being used

`HttpSessionState` (Members 2)

- `Timeout` contains the number of minutes to preserve state between two requests
- `Abandon` sets an internal flag to cancel the current session
- `Add` and `Remove` add or remove an item to the session state
- `Item` reads/writes a session state value
  - You can use a string key or ordinal index value