State Management

Today You Will Learn

- Session State
- Session State Configuration
- Application State
- An Overview of State Management Choices

Session State

- An application might need to store and access complex information such as custom data objects, which can't be easily persisted to a cookie or sent through a query string.
- Or the application might have stringent security requirements that prevent it from storing information about a client in view state or in a custom cookie.
- In these situations, you can use ASP.NET's built-in **session state** facility.

Session State

- It allows you to **store any type of data in memory on the server**.
- The information is **protected**, because it is never transmitted to the client, and it's <u>uniquely bound to a specific session</u>.
- Every client that accesses the application has **a different session** and a distinct collection of information.

Session State

Session Tracking

- **ASP.NET** tracks each session using a unique **120-bit identifier**.
- ASP.NET uses a **proprietary algorithm** to generate this value.
- This ID is the only piece of session-related information that is **transmitted between the web server and the client**.
- When the client presents the **session ID**, ASP.NET looks up the corresponding session, **retrieves the objects** you stored previously, and **places them into a special collection** so they can be accessed in your code.
- This process takes place automatically.

Session State

- For this system to work, the client must present the **appropriate session ID** with each request.
- You can accomplish this in two ways:
  - ***Using cookies*:** In this case, the **session ID** is transmitted in a special **cookie** (named **ASP.NET_SessionId**), which ASP.NET creates automatically when the session collection is used.

  This is the default.

  - ***Using modified URLs*:** In this case, the **session ID** is transmitted in a specially modified (or *munged*) **URL**.

  This allows you to create applications that use session state with clients that don't support cookies.

Session State

- Though Session State solves many of the problems associated with other forms of state management, it forces the **server to store additional information in memory**.

- This extra memory requirement, even if it is small, can quickly grow to **performance-destroying** levels as hundreds or thousands of clients access the site.

Session State

Using Session State

- You can interact with session state using the **System.Web.SessionState.HttpSessionState** class, which is provided in an ASP.NET web page as the built-in Session object.
- The syntax for adding items to the collection and retrieving them is basically the same as for adding items to a page's view state.
- For example, you might store a DataSet in session memory like this:

```
Session["InfoDataSet“] = dsInfo;
```

- You can then retrieve it with an appropriate conversion operation:

```
dsInfo = (DataSet)Session["InfoDataSet“];
```

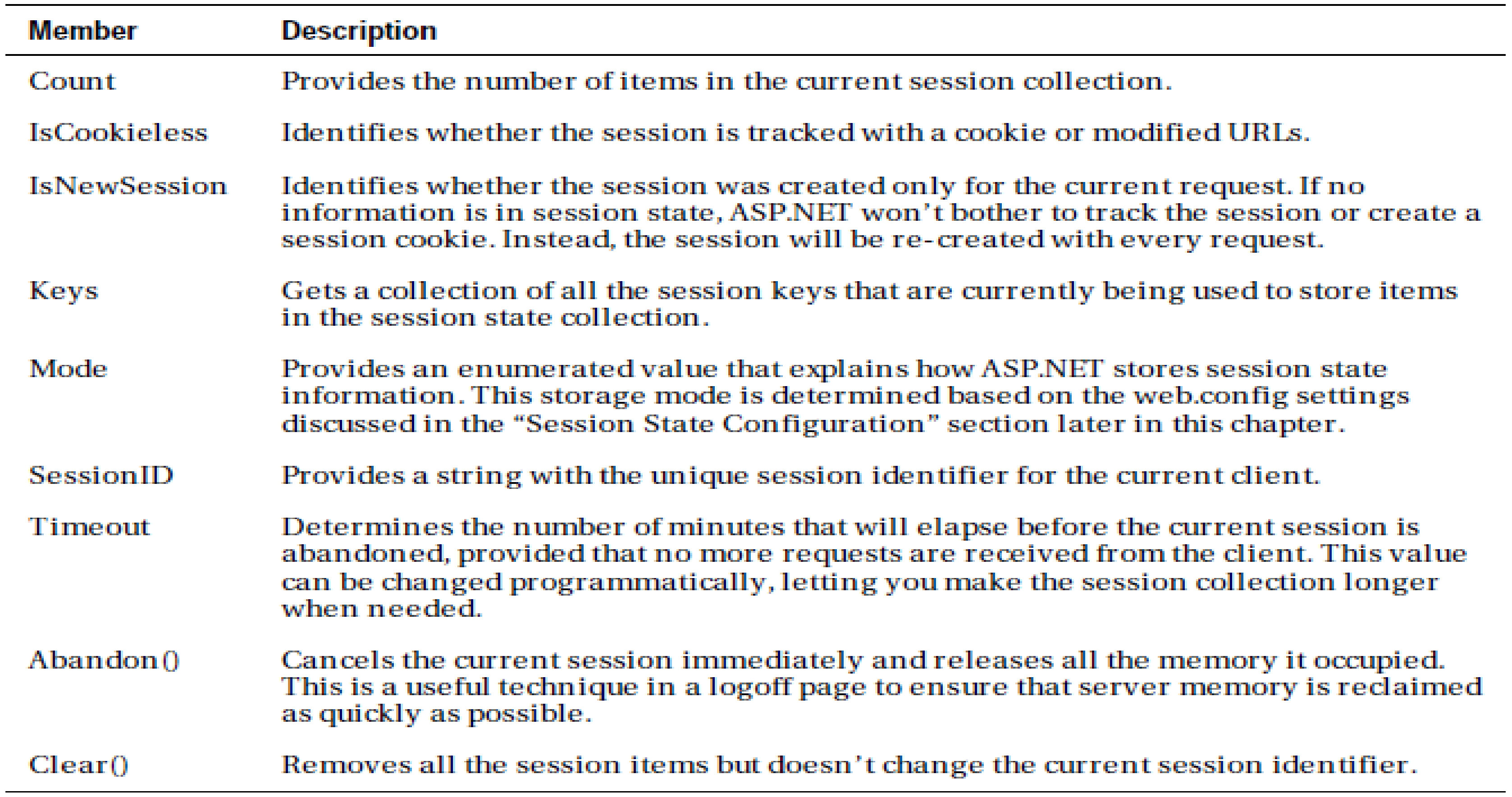
- Of course, before you attempt to use the **dsInfo** object, you'll need to check that it actually exists—in other words, that it isn't a null reference.

Session State

- **Session state** is **global** to your **entire application for the current user**.
- **However, session state can be lost in several ways**:
  - If the user closes and restarts the browser.
  - If the user accesses the same page through a **different browser** window, although the session will still exist if a web page is accessed through the original browser window.
  - If the session **times out** due to inactivity.
  - If your web page code ends the session by calling the **Session.Abandon()** method.
- In the **first two cases**, the session actually remains in memory on the web server, because ASP.NET has no idea that the client has closed the browser or changed windows. The session will linger in memory, remaining inaccessible, until it eventually expires.

Session State

HttpSessionState Members



Member	Description
Count	Provides the number of items in the current session collection.
IsCookieless	Identifies whether the session is tracked with a cookie or modified URLs.
IsNewSession	Identifies whether the session was created only for the current request. If no information is in session state, ASP.NET won't bother to track the session or create a session cookie. Instead, the session will be re-created with every request.
Keys	Gets a collection of all the session keys that are currently being used to store items in the session state collection.
Mode	Provides an enumerated value that explains how ASP.NET stores session state information. This storage mode is determined based on the web.config settings discussed in the "Session State Configuration" section later in this chapter.
SessionID	Provides a string with the unique session identifier for the current client.
Timeout	Determines the number of minutes that will elapse before the current session is abandoned, provided that no more requests are received from the client. This value can be changed programmatically, letting you make the session collection longer when needed.
Abandon()	Cancels the current session immediately and releases all the memory it occupied. This is a useful technique in a logoff page to ensure that server memory is reclaimed as quickly as possible.
Clear()	Removes all the session items but doesn't change the current session identifier.

Session State

A Session State Example

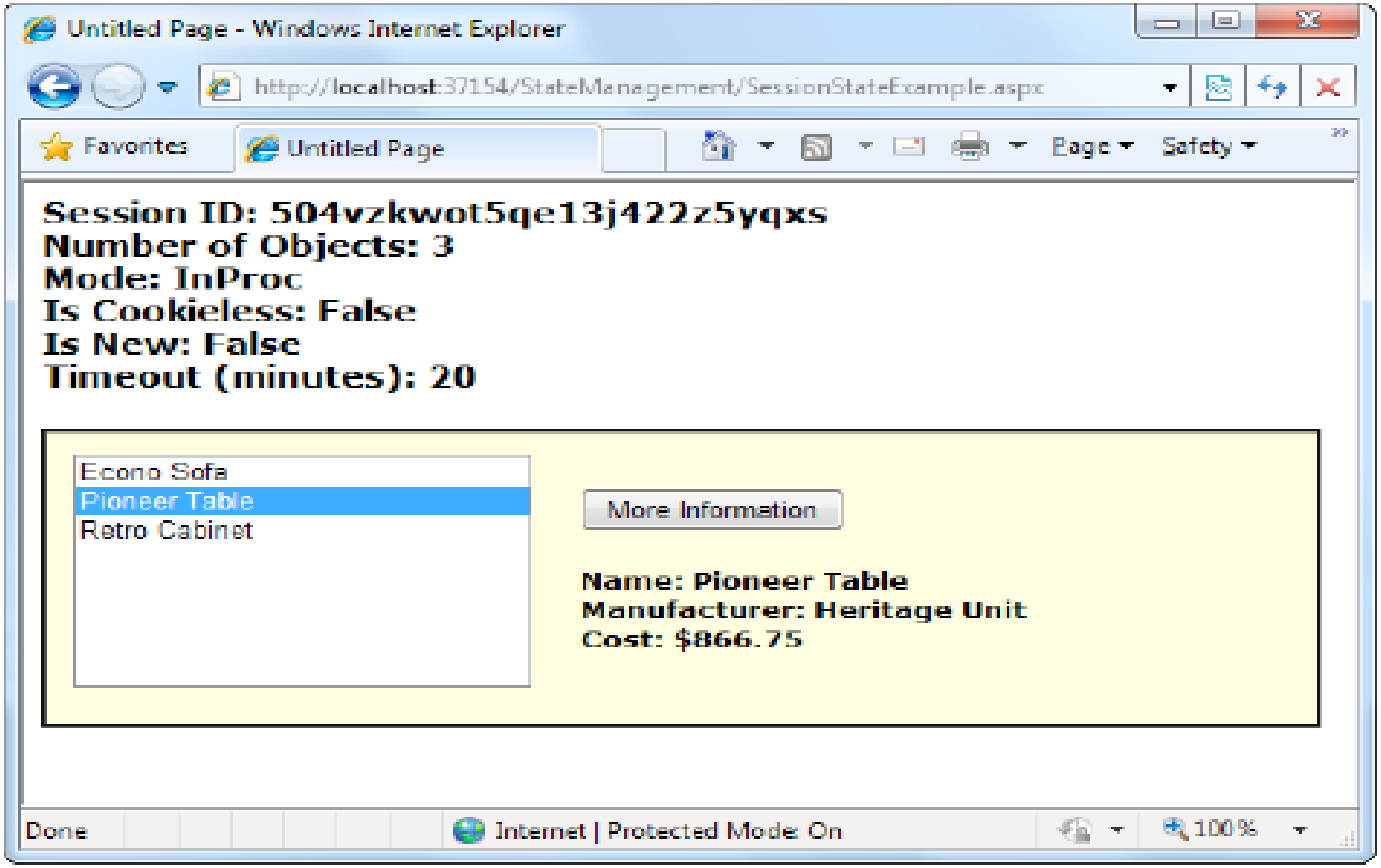
- The next example uses **session state** to store several **Furniture data objects**. The data object combines a few related variables and uses a special constructor so it can be created and initialized in one easy line.

```
public class Furniture
{
    public string Name;
    public string Description;
    public decimal Cost;
    public Furniture(string name, string description, decimal cost)
    {
        Name = name;
        Description = description;
        Cost = cost;
    }
}
```

Session State

- Three Furniture objects are created the first time the page is loaded, and they're stored in session state.
- The user can then choose from a list of furniture piece names.
- When a selection is made, the corresponding object will be retrieved, and its information will be displayed, as shown in next Figure.

Session State


Untitled Page - Windows Internet Explorer
http://localhost:37154/StateManagement/SessionStateExample.aspx
Favorites
Untitled Page
Page
Safety
More Information
Done
Internet | Protected Mode: On
100%

Session ID: 504vzkwot5qe13j422z5yqxs
Number of Objects: 3
Mode: InProc
Is Cookieless: False
Is New: False
Timeout (minutes): 20

Econo Sofa
Pioneer Table
Retro Cabinet

Name: Pioneer Table
Manufacturer: Heritage Unit
Cost: $866.75

Session State Configuration

- You configure **session state** through the **web.config** file for your **current application .**
- The configuration file allows you to set advanced options such as the **timeout and the session state mode**.
- The following listing shows the most important options that you can set for the **<sessionState>** element.
- **Note:** Keep in mind that you won't use all of these details at the same time. Some settings apply only to certain **session state *modes***.

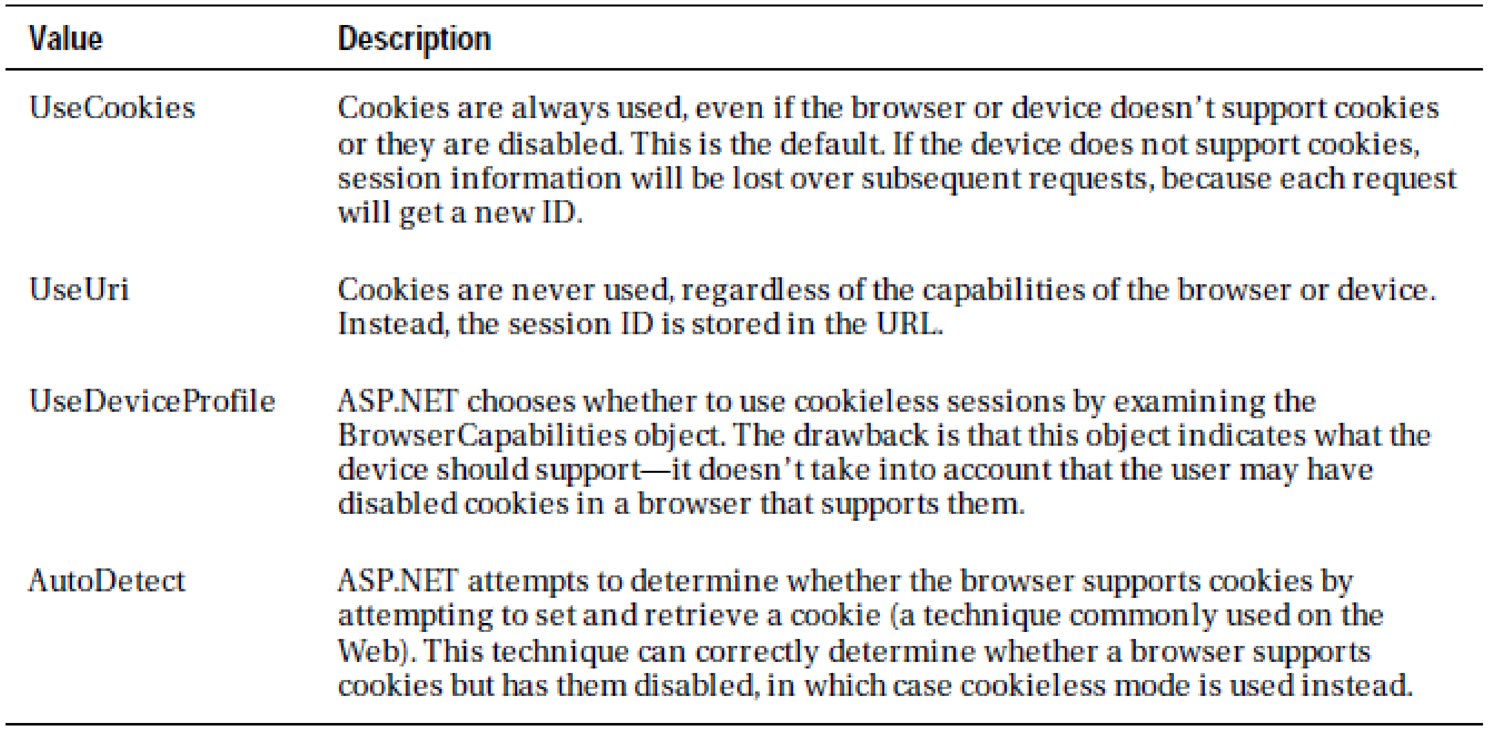
Session State Configuration

```
<?xml version="1.0" ?>
  <configuration>
    <system.web>
      <!-- Other settings omitted. -->
      <sessionState
          cookieless="UseCookies"
          cookieName="ASP.NET_SessionId"
          regenerateExpiredSessionId="false"
          timeout="20"
          mode="InProc"
          stateConnectionString="tcpip=127.0.0.1:42424"
          stateNetworkTimeout="10"
          sqlConnectionString="data source=127.0.0.1;Integrated Security=SSPI"
          sqlCommandTimeout="30"
          allowCustomSqlDatabase="false"
          customProvider=""
          compressionEnabled="false" />
    </system.web>
  </configuration>
```

Session State Configuration

Session State Configuration – Cookieless

- You can set the cookieless setting to one of the values defined by the HttpCookieMode enumeration, as described in Table.



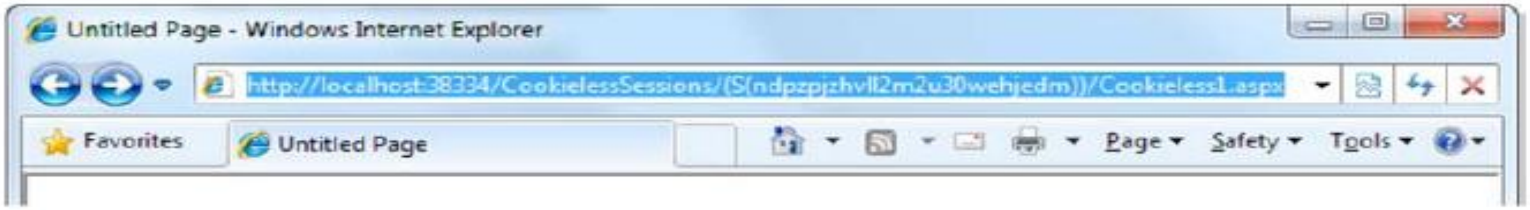
Value	Description
UseCookies	Cookies are always used, even if the browser or device doesn't support cookies or they are disabled. This is the default. If the device does not support cookies, session information will be lost over subsequent requests, because each request will get a new ID.
UseUri	Cookies are never used, regardless of the capabilities of the browser or device. Instead, the session ID is stored in the URL.
UseDeviceProfile	ASP.NET chooses whether to use cookieless sessions by examining the BrowserCapabilities object. The drawback is that this object indicates what the device should support—it doesn't take into account that the user may have disabled cookies in a browser that supports them.
AutoDetect	ASP.NET attempts to determine whether the browser supports cookies by attempting to set and retrieve a cookie (a technique commonly used on the Web). This technique can correctly determine whether a browser supports cookies but has them disabled, in which case cookieless mode is used instead.

Session State Configuration

- Here's an example that forces **cookieless** mode (which is useful for testing):

  ```
  <sessionState cookieless="UseUri" ... />
  ```

- In **cookieless** mode, the <u>session ID will automatically be inserted into the URL</u>.
- When ASP.NET receives a request, it will remove the ID, retrieve the session collection, and forward the request to the appropriate directory. Foll. Figure shows a munged URL.



Untitled Page - Windows Internet Explorer

http://localhost:38334/CookielessSessions/(S(ndpzpjzhvll2m2u30wehjedm))/Cookieless1.aspx

Session State Configuration

- Because the session ID is inserted in the current URL, relative links also automatically gain the session ID.
- In other words, if the user is currently stationed on **Page1.aspx** and clicks a **relative link to Page2.aspx**, the relative link includes the **current session ID** as part of the URL.
- The same is true if you call Response.Redirect() with a relative URL, as shown here:

  ```
  Response.Redirect("Page2.aspx");
  ```

Session State Configuration

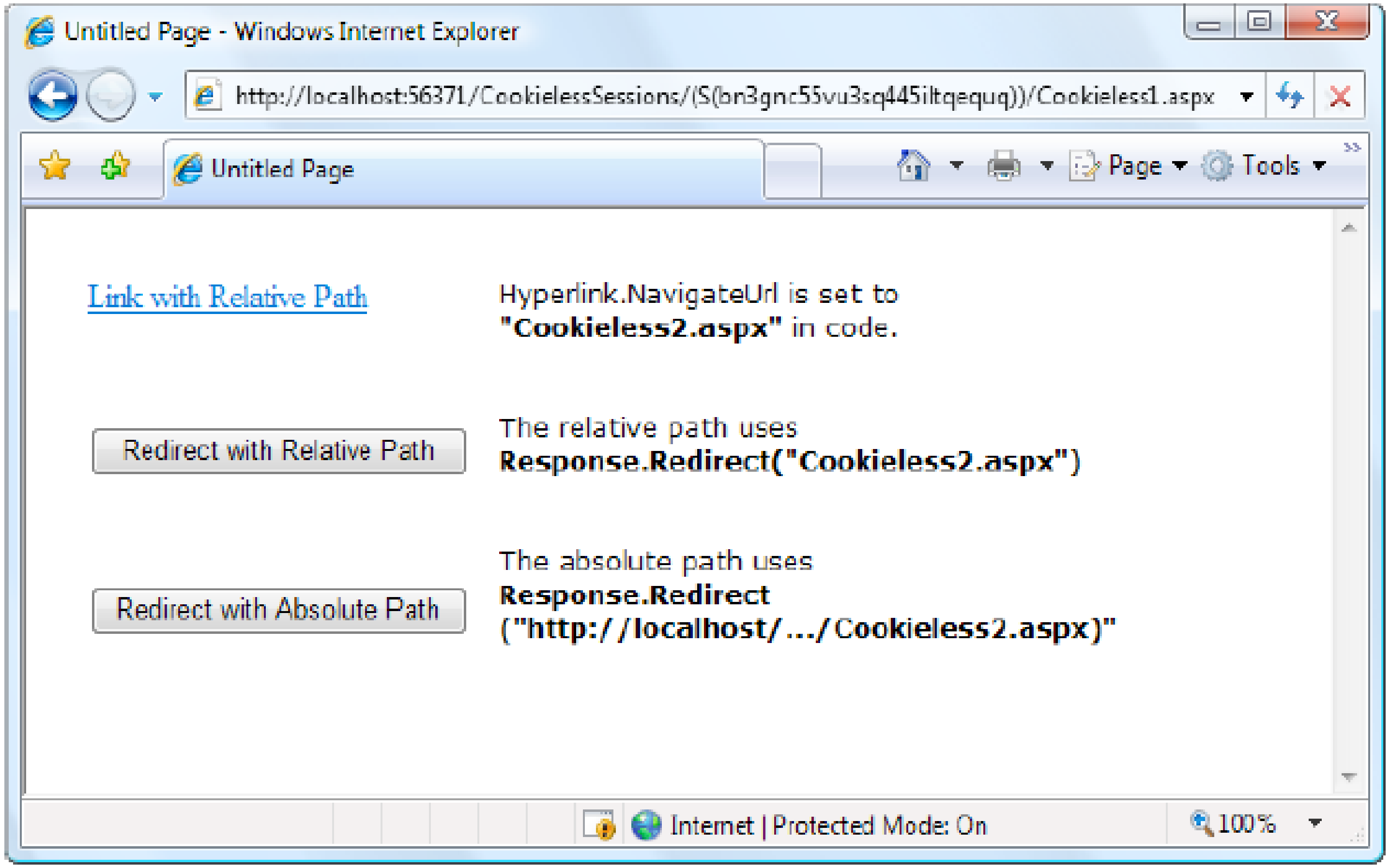
- Figure in next page shows a sample website that tests cookieless sessions.
- It contains **two pages** and uses **cookieless mode**.

- The first page (Cookieless1.aspx) contains a **HyperLink control** and **two buttons**, all of which take you to a second page (Cookieless2.aspx).

- The trick is that these controls have different ways of performing their navigation.
- Only **two** of them work with cookieless session—the third loses the current session.

Session State Configuration

Three tests of cookieless sessions


Untitled Page - Windows Internet Explorer
http://localhost:56371/CookielessSessions/(S(bn3gnc55vu3sq445iltqequq))/Cookieless1.aspx
Untitled Page
Page
Tools
Link with Relative Path
Redirect with Relative Path
Redirect with Absolute Path
Internet | Protected Mode: On
100%

Hyperlink.NavigateUrl is set to **"Cookieless2.aspx"** in code.

The relative path uses **Response.Redirect("Cookieless2.aspx")**

The absolute path uses **Response.Redirect ("http://localhost/.../Cookieless2.aspx)"**

Session State Configuration

- The **HyperLink control** navigates to the **page** specified in its **NavigateUrl property**, which is set to the relative path **Cookieless2.aspx**.

- If you click this link, the session ID is retained in the URL, and the new page can retrieve the session information. This demonstrates that **cookieless sessions** work with **relative links**.

- The **two buttons** on this page use **programmatic redirection** by calling the **Response.Redirect()** method.

Session State Configuration

- The first button uses the **relative path Cookieless2.aspx**, much like the HyperLink control.
- This approach works with cookieless session state and preserves the munged URL with no extra steps required.

```
protected void cmdLink_Click(Object sender, EventArgs e)
{
    Response.Redirect("Cookieless2.aspx");
}
```

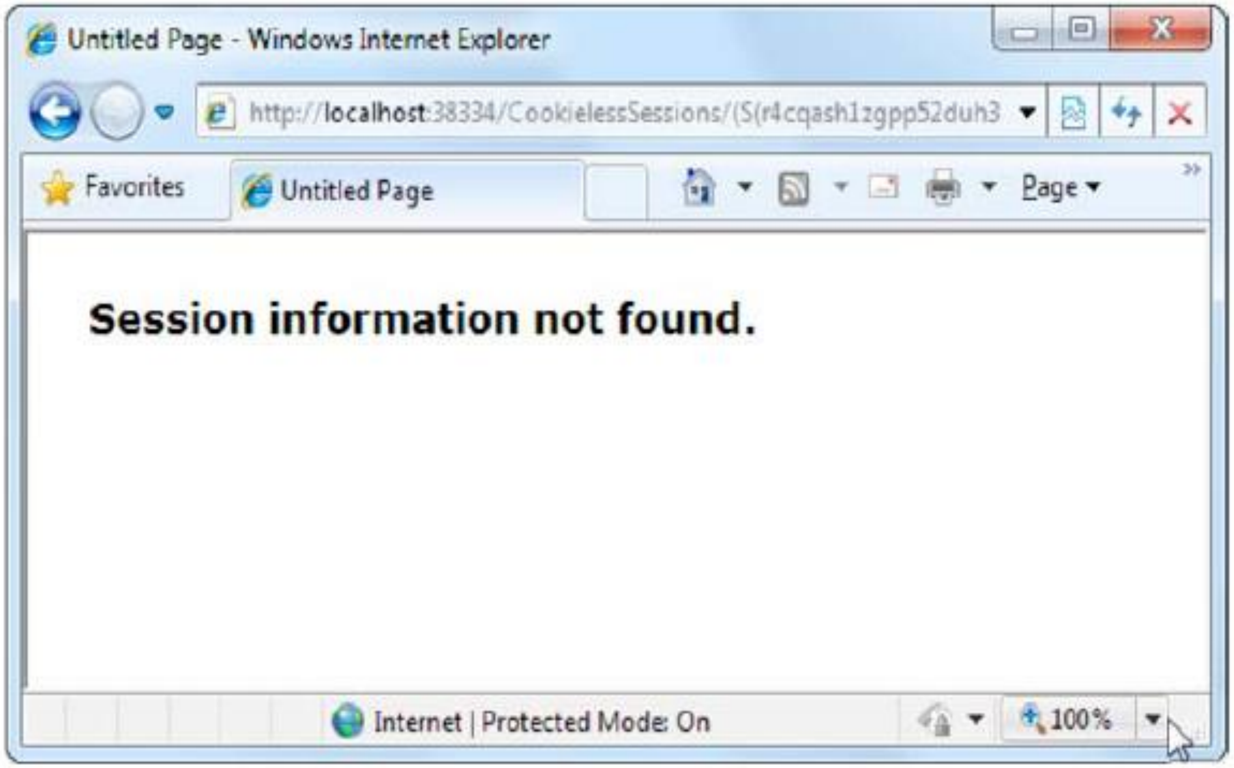
Session State Configuration

- The only real **limitation** of **cookieless state** is that you **cannot use absolute links** (links that include the full URL, starting with http://).
- The **second button** uses an absolute link to demonstrate this problem. Because ASP.NET cannot insert the session ID into the URL, the session is lost.

```
protected void cmdLinkAbsolute_Click(Object sender, EventArgs e)
{
    Response.Redirect("http://localhost:56371/CookielessSessions/Cookieless2.aspx");
}
```

Session State Configuration

- Now the target page (as shown in fig) checks for the session information but can't find it.


Untitled Page - Windows Internet Explorer
http://localhost:38334/CookielessSessions/(S(r4cqash1zgpp52duh3
Favorites
Untitled Page
Page
Internet | Protected Mode: On
100%

Session information not found.

Session State Configuration

Session State Configuration-Timeout

- This specifies the number of minutes that ASP.NET will wait, without receiving a request, before it abandons the session.

```
<sessionState timeout="20" ... />
```

- You can also programmatically change the session timeout in code.

Session State Configuration

- For example, if you know a session contains an unusually large amount of information, you may need to limit the amount of time the session can be stored.
- You would then warn the user and change the **Timeout** property.
- Here's a sample line of code that changes the timeout to 10 minutes:

```
Session.Timeout = 10
```

Session State Configuration

Session State Configuration – MODE

- The remaining session state settings allow you to configure ASP.NET to use different session state services, depending on the mode that you choose

Application State

- **Application state** allows you to store **global objects** that can be accessed by any client.
- Application state is based on the System.Web.HttpApplicationState class, which is provided in all web pages through the built-in Application object.
- **Application state** supports the same type of objects, retains information on the server, and uses the same dictionary-based syntax.

Application State

- A common example with application state is a global counter that tracks how many times an operation has been performed by all the web application’s clients.
- For example, you could create a **global.asax** event handler that tracks how many sessions have been created or how many requests have been received into the application.
- Or you can use similar logic in the **Page.Load** event handler to track how many times a given page has been requested by various clients.

Application State

```
protected void Page_Load(Object sender, EventArgs e)
{
        // Retrieve the current counter value.
        int count = 0;
        if (Application["HitCounterForOrderPage"] != null)
        {
                count = (int)Application["HitCounterForOrderPage"];
        }
        // Increment the counter.
        count++;
        // Store the current counter value.
        Application["HitCounterForOrderPage"] = count;
        lblCounter.Text = count.ToString();
}
```

Application State

- Once again, application state items are stored as objects, so you need to **cast** them when you retrieve them from the collection.
- Items in application state never time out.
- They last until the application or server is **restarted** or the application domain **refreshes** itself .
- Application state isn't often used, because it's generally inefficient. In the previous example, the counter would probably not keep an accurate count, particularly in times of heavy traffic.

Application State

- For example, if two clients requested the page at the same time, you could have a sequence of events like this:

  1. User A retrieves the current count (432).
  2. User B retrieves the current count (432).
  3. User A sets the current count to 433.
  4. User B sets the current count to 433.

- In other words, one request isn't counted because <u>two clients access the counter at the same time</u>.

Application State

- To prevent this problem, you need to use the **Lock()** and **Unlock()** methods, which explicitly <u>allow only one client to access the Application state collection at a time</u>.

```
protected void Page_Load(Object sender, EventArgs e)
{
    // Acquire exclusive access.
    Application.Lock();
    int count = 0;
    if (Application["HitCounterForOrderPage"] != null)
    {
        count = (int)Application["HitCounterForOrderPage"];
    }
    count++;
    Application["HitCounterForOrderPage"] = count;
    // Release exclusive access.
    Application.Unlock();
    lblCounter.Text = count.ToString();
}
```

Application State

- Unfortunately, all other clients requesting <u>the page will be stalled until the Application collection is released</u>.
- This can <u>drastically reduce performance</u>.
- Application state is rarely used in the .NET world because its two most common uses have been replaced by easier, more efficient methods:
  - In the past, application state was used to store **application-wide constants**, such as a database connection string. This type of constant can be stored in the **web.config** file, which is generally more flexible because you can change it easily without needing to hunt through web page code or recompile your application.

Application State

- ○ **Application state** can also be used to store frequently used information that is time-consuming to create.

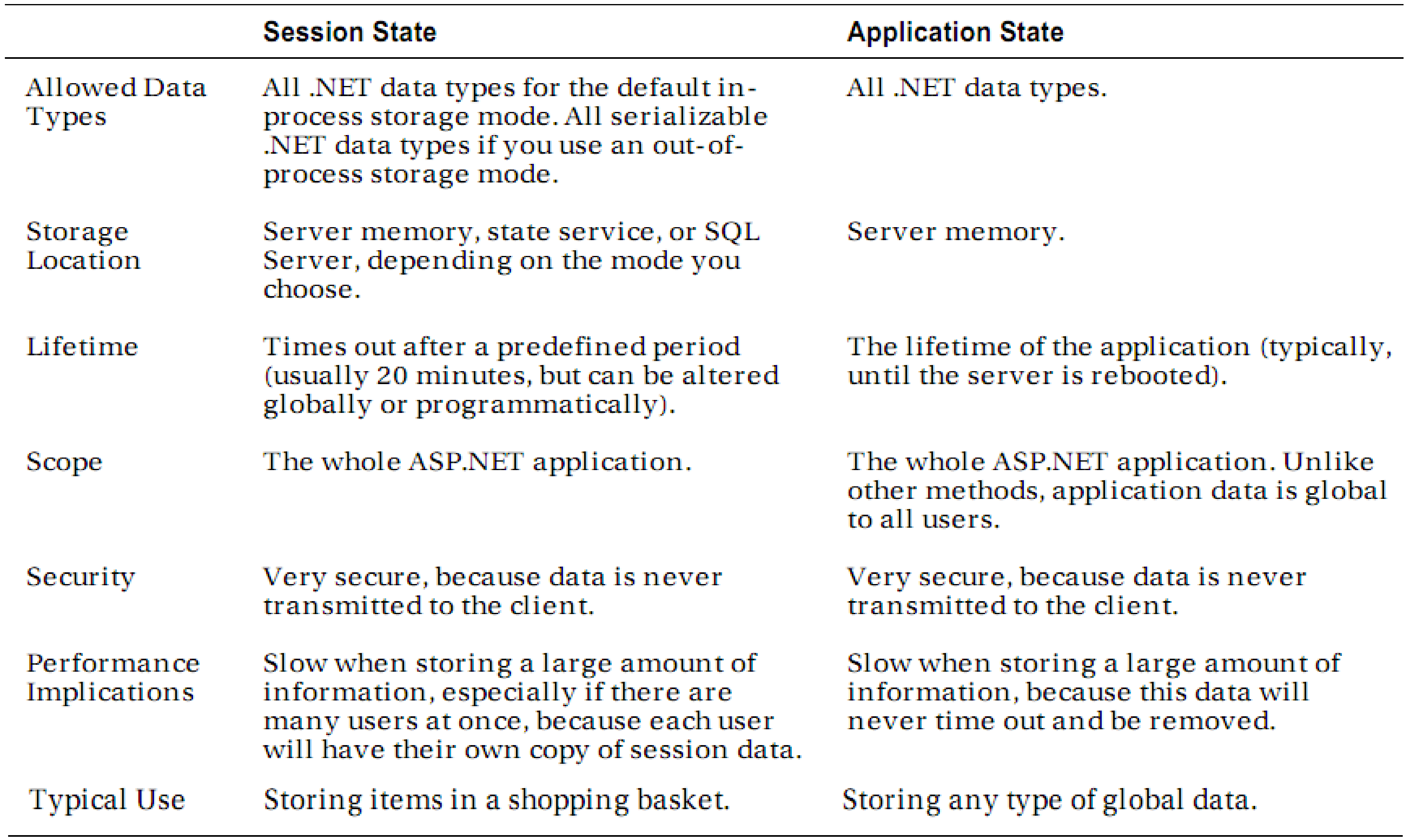
- However, using application state to store this kind of information raises all sorts of problems about how to check whether the data is valid and how to replace it when needed.

- Many uses of application state can be replaced more efficiently with **caching**.

An Overview of State Management Choices

	View State	Query String	Custom Cookies
Allowed Data Types	All serializable .NET data types.	A limited amount of string data.	String data.
Storage Location	A hidden field in the current web page.	The browser's URL string.	The client's computer (in memory or a small text file, depending on its lifetime settings).
Lifetime	Retained permanently for postbacks to a single page.	Lost when the user enters a new URL or closes the browser. However, this can be stored in a bookmark.	Set by the programmer. Can be used in multiple pages and can persist between visits.
Scope	Limited to the current page.	Limited to the target page.	The whole ASP.NET application.
Security	Tamperproof by default but easy to read. You can enforce encryption by using the ViewStateEncryptionMode property of the Page directive.	Clearly visible and easy for the user to modify.	Insecure, and can be modified by the user.
Performance Implications	Slow if a large amount of information is stored, but will not affect server performance.	None, because the amount of data is trivial.	None, because the amount of data is trivial.
Typical Use	Page-specific settings.	Sending a product ID from a catalog page to a details page.	Personalization preferences for a website.

An Overview of State Management Choices



	Session State	Application State
Allowed Data Types	All .NET data types for the default in-process storage mode. All serializable .NET data types if you use an out-of-process storage mode.	All .NET data types.
Storage Location	Server memory, state service, or SQL Server, depending on the mode you choose.	Server memory.
Lifetime	Times out after a predefined period (usually 20 minutes, but can be altered globally or programmatically).	The lifetime of the application (typically, until the server is rebooted).
Scope	The whole ASP.NET application.	The whole ASP.NET application. Unlike other methods, application data is global to all users.
Security	Very secure, because data is never transmitted to the client.	Very secure, because data is never transmitted to the client.
Performance Implications	Slow when storing a large amount of information, especially if there are many users at once, because each user will have their own copy of session data.	Slow when storing a large amount of information, because this data will never time out and be removed.
Typical Use	Storing items in a shopping basket.	Storing any type of global data.

END OF LECTURE