ASP.NET Session State Overview

12/04/2014 • 9 minutes to read

In this article

Use ASP.NET session state to store and retrieve values for a user.

This topic contains:

- Background
- Code Examples
- Class Reference

A Visual Studio project with source code is available to accompany this topic: Download .

Background

ASP.NET session state enables you to store and retrieve values for a user as the user navigates ASP.NET pages in a Web application. HTTP is a stateless protocol. This means that a Web server treats each HTTP request for a page as an independent request. The server retains no knowledge of variable values that were used during previous requests. ASP.NET session state identifies requests from the same browser during a limited time window as a session, and provides a way to persist variable values for the duration of that session. By default, ASP.NET session state is enabled for all ASP.NET applications.

Alternatives to session state include the following:

- Application state, which stores variables that can be accessed by all users of an ASP.NET application.
- Profile properties, which persists user values in a data store without expiring them.
- ASP.NET caching, which stores values in memory that is available to all ASP.NET applications.
- View state, which persists values in a page.
- Cookies.
- The query string and fields on an HTML form that are available from an HTTP request.

For a comparison of different state-management options, see ASP.NET State Management Recommendations.

Session Variables

Session variables are stored in a SessionStateItemCollection object that is exposed through the HttpContext.Session property. In an ASP.NET page, the current session variables are exposed through the Session property of the Page object.

The collection of session variables is indexed by the name of the variable or by an integer index. Session variables are created by referring to the session variable by name. You do not have to declare a session variable or explicitly add it to the collection. The following example shows how to create session variables in an ASP.NET page for the first and last name of a user, and set them to values retrieved from TextBox controls.

VB

Copy

```vb
Session("FirstName") = FirstNameTextBox.Text
Session("LastName") = LastNameTextBox.Text
```

Session variables can be any valid .NET Framework type. The following example stores an ArrayList object in a session variable named StockPicks. The value returned by the StockPicks session variable must be cast to the appropriate type when you retrieve it from the SessionStateItemCollection.

VB

Copy

```vb
' When retrieving an object from session state, cast it to
' the appropriate type.
Dim stockPicks As ArrayList = CType(Session("StockPicks"), ArrayList)

' Write the modified stock picks list back to session state.
Session("StockPicks") = stockPicks
```

Note

When you use a session-state mode other than **InProc**, the session-variable type must be either a primitive .NET type or serializable. This is because the session-variable value is stored in an external data store. For more information, see **Session-State Modes**.

Session Identifiers

Sessions are identified by a unique identifier that can be read by using the SessionID property. When session state is enabled for an ASP.NET application, each request for a page in the application is examined for a SessionID value sent from the browser. If no SessionID value is supplied, ASP.NET starts a new session and the SessionID value for that session is sent to the browser with the response.

By default, SessionID values are stored in a cookie. However, you can also configure the application to store SessionID values in the URL for a "cookieless" session.

A session is considered active as long as requests continue to be made with the same SessionID value. If the time between requests for a particular session exceeds the specified time-out value in minutes, the session is considered expired. Requests made with an expired SessionID value result in a new session.

Security Note

SessionID values are sent in clear text, whether as a cookie or as part of the URL. A malicious user could get access to the session of another user by obtaining the SessionID value and including it in requests to the server. If you are storing sensitive information in session state, it is recommended that you use SSL to encrypt any communication between the browser and server that includes the SessionID value.

Cookieless SessionIDs

By default, the SessionID value is stored in a non-expiring session cookie in the browser. However, you can specify that session identifiers should not be stored in a cookie by setting the cookieless attribute to true in the sessionState section of the Web.config file.

The following example shows a Web.config file that configures an ASP.NET application to use cookieless session identifiers.

Copy

```
<configuration>
  <system.web>
    <sessionState cookieless="true"
      regenerateExpiredSessionId="true" />
  </system.web>
</configuration>
```

ASP.NET maintains cookieless session state by automatically inserting a unique session ID into the page's URL. For example, the following URL has been modified by ASP.NET to include the unique session ID lit3py55t21z5v55vlm25s55:

Copy

```
http://www.example.com/(S(lit3py55t21z5v55vlm25s55))/orderform.aspx
```

When ASP.NET sends a page to the browser, it modifies any links in the page that use an application-relative path by embedding a session ID value in the links. (Links with absolute paths are not modified.) Session state is maintained as long as the user clicks links that have been modified in this manner. However, if the client rewrites a URL that is supplied by the application, ASP.NET may not be able to resolve the session ID and associate the request with an existing session. In that case, a new session is started for the request.

The session ID is embedded in the URL after the slash that follows the application name and before any remaining file or virtual directory identifier. This enables ASP.NET to resolve the application name before involving the SessionStateModule in the request.

Note

To improve the security of your application, you should allow users to log out of your application, at which point the application should call the **Abandon** method. This reduces the potential for a malicious user to get the unique identifier in the URL and use it to retrieve private user data stored in the session.

Regenerating Expired Session Identifiers

By default, the session ID values that are used in cookieless sessions are recycled. That is, if a request is made with a session ID that has expired, a new session is started by using the SessionID value that is supplied with the request. This can result in a session unintentionally being shared when a link that contains a cookieless SessionID value is used by multiple browsers. (This can occur if the link is passed through a search engine, through an e-mail message, or through another program.) You can reduce the chance of session data being shared by configuring the application not to recycle session identifiers. To do this, set the regenerateExpiredSessionId attribute of the sessionState configuration element to true. This generates a new session ID when a cookieless session request is made with an expired session ID.

Note

If the request that is made with the expired session ID is made by using the HTTP POST method, any posted data will be lost when regenerateExpiredSessionId is true. This is because ASP.NET performs a redirect to make sure that the browser has the new session identifier in the URL.

Custom Session Identifiers

You can implement a custom class to supply and validate SessionID values. To do so, create a class that inherits the SessionIDManager class and override the CreateSessionID and Validate methods with your own implementations. For an example, see the example provided for the CreateSessionID method.

You can replace the SessionIDManager class by creating a class that implements the ISessionIDManager interface. For example, you might have a Web application that associates a unique identifier with non-ASP.NET pages (such as HTML pages or images) by using an ISAPI filter. You can implement a custom SessionIDManager class to use this unique identifier with ASP.NET session state. If your custom class supports cookieless session identifiers, you must implement a solution for sending and retrieving session identifiers in the URL.

Session Modes

ASP.NET session state supports several storage options for session variables. Each option is identified as a session-state Mode type. The default behavior is to store session variables in the memory space of the ASP.NET worker process. However, you can also specify that session state should be stored in a separate process, in a SQL Server database, or in a custom data source. If you do not want session state enabled for your application, you can set the session mode to Off.

For more information, see Session-State Modes.

Session Events

ASP.NET provides two events that help you manage user sessions. The **Session_OnStart** event is raised when a new session starts, and the **Session_OnEnd** event is raised when a session is abandoned or expires. Session events are specified in the Global.asax file for an ASP.NET application.

The **Session_OnEnd** event is not supported if the session Mode property is set to a value other than InProc, which is the default mode.

ⓘ **Note**

If the Global.asax file or Web.config file for an ASP.NET application is modified, the application will be restarted and any values stored in application state or session state will be lost. Be aware that some anti-virus software can update the last-modified date and time of the Global.asax or Web.config file for an application.

For more information, see Session-State Events.

Configuring Session State

Session state is configured by using the sessionState element of the system.web configuration section. You can also configure session state by using the EnableSessionState value in the @ Page directive.

The sessionState element enables you to specify the following options:

- The mode in which the session will store data.
- The way in which session identifier values are sent between the client and the server.
- The session Timeout value.
- Supporting values that are based on the session Mode setting.

The following example shows a sessionState element that configures an application for SQLServer session mode. It sets the Timeout value to 30 minutes, and specifies that session identifiers are stored in the URL.

Copy

```
<sessionState mode="SQLServer"
  cookieless="true "
  regenerateExpiredSessionId="true "
  timeout="30"
  sqlConnectionString="Data Source=MySqlServer;Integrated Security=SSPI;"
  stateNetworkTimeout="30"/>
```

You can disable session state for an application by setting the session-state mode to Off . If you want to disable session state for only a particular page of an application, you can set the EnableSessionState value in the @ Page directive to false. The EnableSessionState value can also be set to ReadOnly to provide read-only access to session variables.

Concurrent Requests and Session State

Access to ASP.NET session state is exclusive per session, which means that if two different users make concurrent requests, access to each separate session is granted concurrently. However, if two concurrent requests are made for the same session (by using the same SessionID value), the first request gets exclusive access to the session information. The second request executes only after the first request is finished. (The second session can also get access if the exclusive lock on the information is freed because the first request exceeds the lock time-out.) If the EnableSessionState value in the @ Page directive is set to ReadOnly, a request for the read-only session information does not result in an exclusive lock on the session data. However, read-only requests for session data might still have to wait for a lock set by a read-write request for session data to clear.

Back to top

Code Examples

How to: Save Values in Session State

How to: Read Values from Session State

Implementing a Session-State Store Provider

Back to top

Class Reference

The following table lists key classes that relate to session state are in the System.Web.SessionState namespace.

Member	Description
SessionIDManager	Manages unique identifiers for ASP.NET session state.
SessionStateItemCollection	Used to store session state variables.

Back to top

See Also

Concepts

Implementing a Session-State Store Provider