ASP.NET State Management Overview

A new instance of the Web page class is created each time the page is posted to the server. In traditional Web programming, this would typically mean that all information associated with the page and the controls on the page would be lost with each round trip. For example, if a user enters information into a text box, that information would be lost in the round trip from the browser or client device to the server.

To overcome this inherent limitation of traditional Web programming, ASP.NET includes several options that help you preserve data on both a per-page basis and an application-wide basis. These features are as follows:

- View state
- Control state
- Hidden fields
- Cookies
- Query strings
- Application state
- Session state
- Profile Properties

View state, control state, hidden fields, cookies, and query strings all involve storing data on the client in various ways. However, application state, session state, and profile properties all store data in memory on the server. Each option has distinct advantages and disadvantages, depending on the scenario.

Client-Based State Management Options

The following sections describe options for state management that involve storing information either in the page or on the client computer. For these options, no information is maintained on the server between round trips.

View State

The ViewState property provides a dictionary object for retaining values between multiple requests for the same page. This is the default method that the page uses to preserve page and control property values between round trips.

When the page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field, or multiple hidden fields if the amount of data stored in the ViewState property exceeds the specified value in the MaxPageStateFieldLength property. When the page is posted back to the server, the page parses the view-state string at page initialization and restores property information in the page.

You can store values in view state as well. For more information on using View State, see ASP.NET View State Overview. For recommendations about when you should use view state, see ASP.NET State Management Recommendations.

Control State

Sometimes you need to store control-state data in order for a control to work properly. For example, if you have written a custom control that has different tabs that show different information, in order for that control to work as expected, the control needs to know which tab is selected between round trips. The ViewState property can be used for this purpose, but view state can be turned off at a page level by developers, effectively breaking your control. To solve this, the ASP.NET page framework exposes a feature in ASP.NET called control state.

The ControlState property allows you to persist property information that is specific to a control and cannot be turned off like the ViewState property.

Hidden Fields

ASP.NET allows you to store information in a HiddenField control, which renders as a standard HTML hidden field. A hidden field does not render visibly in the browser, but you can set its properties just as you can with a standard control. When a page is submitted to the server, the content of a hidden field is sent in the HTTP form collection along with the values of other controls. A hidden field acts as a repository for any page-specific information that you want to store directly in the page.

> **Security Note**
>
> It is easy for a malicious user to see and modify the contents of a hidden field. Do not store any information in a hidden field that is sensitive or that your application relies on to work properly. For more information, see ASP.NET State Management Recommendations.

A HiddenField control stores a single variable in its Value property and must be explicitly added to the page. For more information, see HiddenField Web Server Control Overview.

In order for hidden-field values to be available during page processing, you must submit the page using an HTTP POST command. If you use hidden fields and a page is processed in response to a link or an HTTP GET command, the hidden fields will not be available. For usage recommendations, see ASP.NET State Management Recommendations.

Cookies

A cookie is a small amount of data that is stored either in a text file on the client file system or in-memory in the client browser session. It contains site-specific information that the server sends to the client along with page output. Cookies can be temporary (with specific expiration times and dates) or persistent.

You can use cookies to store information about a particular client, session, or application. The cookies are saved on the client device, and when the browser requests a page, the client sends the information in the cookie along with the request information. The server can read the cookie and extract its value. A typical use is to store a token (perhaps encrypted) indicating that the user has already been authenticated in your application.

> **Security Note**
>
> The browser can only send the data back to the server that originally created the cookie. However, malicious users have ways to access cookies and read their contents. It is recommended that you do not store sensitive information, such as a user name or password, in a cookie. Instead, store a token in the cookie that identifies the user, and then use the token to look up the sensitive information on the server.

For more information about using cookies, see Cookies and ASP.NET State Management Recommendations.

Query Strings

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

```
http://www.contoso.com/listwidgets.aspx?category=basic&price=100
```

In the URL path above, the query string starts with a question mark (?) and includes two attribute/value pairs, one called "category" and the other called "price."

Query strings provide a simple but limited way to maintain state information. For example, they are an easy way to pass information from one page to another, such as passing a product number from one page to another page where it will be processed. However, some browsers and client devices impose a 2083-character limit on the length of the URL.

Security Note

Information that is passed in a query string can be tampered with by a malicious user. Do not rely on query strings to convey important or sensitive data. Additionally, a user can bookmark the URL or send the URL to other users, thereby passing that information along with it. For more information, see ASP.NET State Management Recommendations and How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings.

In order for query string values to be available during page processing, you must submit the page using an HTTP GET command. That is, you cannot take advantage of a query string if a page is processed in response to an HTTP POST command. For usage recommendations, see ASP.NET State Management Recommendations.

Server-Based State Management Options

ASP.NET offers you a variety of ways to maintain state information on the server, rather than persisting information on the client. With server-based state management, you can decrease the amount of information sent to the client in order to preserve state, however it can use costly resources on the server. The following sections describe three server-based state management features: application state, session state, and profile properties.

Application State

ASP.NET allows you to save values using application state — which is an instance of the HttpApplicationState class — for each active Web application. Application state is a global storage mechanism that is accessible from all pages in the Web application. Thus, application state is useful for storing information that needs to be maintained between server round trips and between requests for pages. For more information, see ASP.NET Application State Overview.

Application state is stored in a key/value dictionary that is created during each request to a specific URL. You can add your application-specific information to this structure to store it between page requests.

Once you add your application-specific information to application state, the server manages it. For usage recommendations, see ASP.NET State Management Recommendations.

Session State

ASP.NET allows you to save values by using session state — which is an instance of the HttpSessionState class — for each active Web-application session. For an overview, see ASP.NET Session State Overview.

Session state is similar to application state, except that it is scoped to the current browser session. If different users are using your application, each user session will have a different session state. In addition, if a user leaves your application and then returns later, the second user session will have a different session state from the first.

Session state is structured as a key/value dictionary for storing session-specific information that needs to be maintained between server round trips and between requests for pages. For more information, see ASP.NET Session State Overview.

You can use session state to accomplish the following tasks:

- Uniquely identify browser or client-device requests and map them to an individual session instance on the server.
- Store session-specific data on the server for use across multiple browser or client-device requests within the same session.
- Raise appropriate session management events. In addition, you can write application code leveraging these events.

Once you add your application-specific information to session state, the server manages this object. Depending on which options you specify, session information can be stored in cookies, on an out-of-process server, or on a computer running Microsoft SQL Server. For usage recommendations, see ASP.NET State Management Recommendations.

Profile Properties

ASP.NET provides a feature called profile properties, which allows you to store user-specific data. This feature is similar to session state, except that the profile data is not lost when a user's session expires. The profile-properties feature uses an ASP.NET profile, which is stored in a persistent format and associated with an individual user. The ASP.NET profile allows you to easily manage user information without requiring you to create and maintain your own database. In addition, the profile makes the user information available using a strongly typed API that you can access from anywhere in your application. You can store objects of any type in the profile. The ASP.NET profile feature provides a generic storage system that allows you to define and maintain almost any kind of data while still making the data available in a type-safe manner.

To use profile properties, you must configure a profile provider. ASP.NET includes a SqlProfileProvider class that allows you to store profile data in a SQL database, but you can also create your own profile provider class that stores profile data in a custom format and to a custom storage mechanism such as an XML file, or even to a web service.

Because data that is placed in profile properties is not stored in application memory, it is preserved through Internet Information Services (IIS) restarts and worker-process restarts without losing data. Additionally, profile properties can be persisted across multiple processes such as in a Web farm or a Web garden. For more information, see ASP.NET Profile Properties Overview.

See Also

Concepts

ASP.NET State Management Recommendations
ASP.NET Cookies Overview
ASP.NET View State Overview
ASP.NET Session State Overview
ASP.NET Application State Overview
ASP.NET Profile Properties Overview