December 2001

Five Browser Independent Techniques for Retaining Application State in Active Server Page Applications

Kyle Lutes
Purdue University

FIVE BROWSER INDEPENDENT TECHNIQUES FOR RETAINING APPLICATION STATE IN ACTIVE SERVER PAGE APPLICATIONS

Kyle D. Lutes
Computer (Information Systems) Technology Department
Purdue University
kdlutes@tech.purdue.edu

Abstract

One major problem presented to web application developers is how to retain data between interactions with the user. This data, called state information, is not kept in memory because web applications use the HTTP protocol, which is stateless. So, developers of web-based applications typically employ one of several coding techniques to maintain state information. This paper describes five techniques that are commonly used to retain state in Active Server Pages (ASP) applications, along with each technique's relative advantages and disadvantages. Client techniques store state information on the actual client computers, and server techniques store state information on the web servers. Three client techniques that are discussed include using Cookies, hidden HTML form fields, and URL Query Strings. The two server techniques that are presented include using ASP Sessions, and building your own custom solution.

Keywords: ASP, state management

Introduction

State information, sometimes called context information, can be defined as information an application must remember between interactions with the user. Maintaining state information in a typical two-tier client/server application is easily accomplished by storing the data in global- and/or module- scope variables. Data stored in this fashion will exist in memory as long the application is running (global scope) or the object is loaded in memory (module scope).

One of the problems that exists when developing web-based applications is that the HTTP protocol is connectionless, and so stateless. Web servers treat each client request independently from all other requests. This means the web server has no way of associating a request from a client with any previous requests from the same client. While this allows web servers to execute efficiently and probably seemed like a good idea when the HTTP protocol was being developed to serve up static HTML documents, it creates significant problems for developers of web-based applications. So with the HTTP protocol, there is no simple way of storing state information. Instead, developers of web-based applications typically employ one of several coding techniques to maintain state information. These techniques can be classified as either client-side, or server-side.

Client Techniques

Client techniques store state information on the actual client computers, and are generally considered more scalable because the server is not burdened with the task of storing all this data for each user. Client techniques also work in web server load-balanced environments.

Cookies

Netscape introduced cookies with the first version of its browser. Since then, the World Wide Web Consortium has endorsed a cookie standard. Cookies are small files that a web-based application can ask the browser to save information into. This information can then be retrieved at a later time by the same, or different web application.

Using cookies to maintain state information in ASP applications is relatively easy because the ASP technology includes the intrinsic Response and Request objects that have methods to read and write cookie information. Saving state information in a cookie is as simple as this:

```
Response.Cookies("UserName") = "Bob"
```

Which can then later be retrieved by:

```
UserName = Request.Cookies("UserName")
```

Most web browsers have the ability to use cookies, but many users consider them an invasion of privacy, in that they are being used to track the sites the individual visits, or that they might expose personal information. Indeed, flaws in both Netscape Navigator (Netscape 2000) and Microsoft Internet Explorer (Festa 2000) have been discovered that potentially allow malicious web sites to exploit cookie information. Because cookies are not always popular, most web browsers have user configuration options that allow the user to disable cookies on his or her computer. And if cookies are disabled, no state information can be saved. For an e-commerce site that is trying to reach as many customers as possible, requiring the user to enable cookies could mean the loss of a sale.

Hidden HTML Form Fields

Another technique for saving state information on the client is to hide the data in hidden HTML form fields. When a web page has data that needs to be remembered, the application dynamically alters the HTML page being sent to include the state information as fields in an HTML form on that page. The state information fields are marked “hidden” so that the user does not see them. When the user submits a page containing hidden form fields, the web application can then retrieve the values from the hidden fields.

This technique has an advantage over cookies in that users can’t disable it. However, it has a disadvantage in that the coding for maintaining state information is much more complex. Retrieving state information from a hidden form field is easy, but placing data into hidden form fields requires that all HTML pages within the application be dynamically altered by the application to include this state information. Additionally, application logic is now tightly coupled with the user interface HTML implementation and all pages must include HTML forms even if they normally wouldn’t.

URL Query Strings

Query Strings are the extra data that follow a question mark on a URL. This example,

```
http://www.delmarit.com?user=Bob
```

contains the query string “user=Bob” which contains a *variable=value* pair. QueryStrings can contain multiple values. This example,

```
http://www.delmarit.com?user=Bob;pw=red
```

contains the query string “user=Bob;pw=red”, which might indicate the user’s name is “Bob” and his password is “red.” Retrieving the contents of a query string is easily accomplished using the ASP Request object’s QueryString collection. For example,

```
UserName = Request.QueryString("user")
Password = Request.QueryString("pw")
```

The query string technique shares much of the same advantages and disadvantages as the hidden HTML form fields technique, including complicated application logic. However, query strings are limited to a much smaller amount of data (about 2,000 bytes), and their values are always visible in the address bar of the browser. This makes tampering with their values, either on purpose or by accident, very easy to do.

Problems

Relying on the client computer to store state information has several problems. First, state information is limited to string data types. Second, since the state information must be continually transmitted back and forth between the client and server, it results in increased network traffic and thus slightly slower response times. But arguably the biggest drawback of storing state information on the client is that the user can alter it.

One common scenario of this type of tampering involves so-called e-commerce shopping cart applications. Suppose a shopping cart application stores a customer's order information in hidden HTML form fields. Such information might include customer billing information, product ID, product name, quantity, and price. A malicious customer could use his or her client computer to save the HTML file with the hidden fields, modify the price values for the products being ordered, open the altered HTML in a web browser, and then submit the page back to the server. If the shopping cart software running on the server uses the prices from the hidden form fields, the customer's credit card could be billed for an incorrect amount. In February 2000, Internet Security Systems (2000) published a report listing eleven shopping cart applications that were vulnerable to exactly this kind of price changing using form tampering.

Server Techniques

Server techniques store state information on the web servers, and are considered more secure since data (possibly private data) is not stored on the client computer and so it can't be tampered with. Server techniques may also be more flexible because they do not have to rely on features of the web browsers running on the client computer.

ASP Sessions

The most common technique for maintaining state information in ASP applications is by using ASP sessions. A new session is started when a user first requests a page from an ASP application, and ends after the user leaves. Each user of the application is given a unique session. Developers can store state information using the ASP Session object like this:

```
Session.Contents("UserName") = "Bob"
```

And later retrieve it by:

```
UserName = Session.Contents("UserName")
```

The contents of a session are unique for each user, and will be retained until the web server decides the session isn't being used any more and then destroys it. The default is 20 minutes of inactivity, but this value can be changed using a web server configuration option, or by changing the Session object's Timeout property within ASP code. ASP code can also explicitly destroy session information by calling the Session object's Abandon method.

Sessions are extremely easy to use. They don't have to be declared, initialized, or explicitly destroyed. In addition to being easy to use, another advantage of Sessions is that they are not limited to storing simple string data types. In fact, they can be used to store complex COM objects such as ActiveX Data Object (ADO) database connections. However, storing COM objects in session variables is generally discouraged due to increased demand of server resources.

The disadvantage of using sessions is how they work. Session values are stored in the memory of the web server computer that handles a request from a client. As the number of users of an ASP application increases, so must the memory on the web server. In addition, if the application runs on web servers in a load-balanced environment, sessions will not work. In load-balanced environments, subsequent requests from the same client computer may get routed to different web server computers based on which web server computer is least busy at the time. If the first request from a client gets routed to Server A, and session information is stored in Server A's memory, and a second request from the same client gets routed to Server B, the code running on Server B will not be able to retrieve the previously saved state information on Server A. Workarounds do exist, but they solve the problem by forcing all requests from an individual client to always be routed to the same web server computer, which partially defeats the purpose of a using load-balancing environments in the first place.

A bigger problem with ASP sessions is that they require the client browser to support cookies. When a new session begins, ASP generates a unique identifier and requests the client browser to store the identifier in a cookie. Whenever a request comes from a browser, ASP attempts to retrieve the unique session identifier from the cookie in order to know which session values to use. Since session values are stored on the server, malicious users cannot tamper with them. However, if a paranoid user disables cookies on his or her browser, sessions won’t work.

Build Your Own Custom Solution

Because all of the techniques mentioned previously have obvious disadvantages, developers of enterprise and e-commerce web applications are wise to develop their own custom framework for state information retention. The disadvantage of “rolling your own” of course is the initial programming effort involved, but this is easily outweighed by overcoming the problems with the previous techniques, and by the long-term flexibility that is gained.

A custom technique begins by first choosing a storage location for state information. Popular choices for storing state information are as records in a database table or as simple binary or text files in a web server’s file system. Note that storing state information in this manner will also require a process to be run that deletes “expired” data. Second, a unique identifier must be generated for each user. This generated identifier could be an identity column from a record in a database table, a user name, a timestamp, a random number, a GUID, or any combination of the previous items. Finally, a mechanism must be chosen to allow the application to retrieve the identifier from the client each time the client issues a request. Obvious choices are by storing it in a cookie, hiding it in a hidden HTML form field, or by including it in the URL as a query string. Because of the problems associated with cookies and hidden HTML form fields, including it in the URL as a query string is likely the best choice.

If the ASP application is being developed largely using a programming language that supports encapsulation such as Visual Basic (Lutes 2000), developers are also encouraged to encapsulate all state information into an “application state” object. Such an object can be specific for each application, or a generalized state management object could be created. This technique works particularly well because when the object is created, it can automatically attempt to restore any state information previously saved. And when it is destroyed, it can automatically save its contents to the chosen storage media. Also, encapsulating the handling of state information allows the actual location of the state information values to be changed without affecting the rest of the application.

Conclusions and Recommendations

One of the problems that exists when developing web-based applications is that the HTTP protocol is connectionless, and so stateless. Web servers treat each client request independently from all other requests. This means the web server has no way of associating a request from a client with any previous requests from the same client. Five browser independent techniques for maintaining state information have been described. The relative advantages and disadvantages of the five techniques are summarized in Table 1. For small ASP applications that need to be quickly built, and when the developer can confidently require the user to enable cookies, using ASP sessions to store state information is probably the best choice. However, because of the security problems with client-side techniques, and because ASP Sessions require cookies, developers of enterprise and e-commerce applications will be best served by spending a little extra time developing their own framework for state information retention.

Table 1. Comparison of State Retention Techniques

	Client Techniques			**Server Techniques**	
	Cookies	**Hidden HTML Form Fields**	**Query Strings**	**ASP Sessions**	**Custom Solution**
Advantages					
Relatively easy to code	x			x	x
Can store complex data types (i.e. COM objects)				x	
Works well in load balancing environments	x	x	x		x
Disadvantages					
User can tamper with state information	x	x	x		
Requires Cookies	x			x	
Relatively complicated to code		x	x		
Can only store primitive data types (i.e. strings and numbers)	x	x	x	x	
Can store only small amount of data (~2,000 bytes)			x		
Increased network traffic and slightly slower response times	x	x	x		
Requires server resources to save state information				x	x
Application logic tightly coupled to the user interface implementation		x			

References

Exploit Enables Reading of Bookmark Links and Some Attributes of HTML Files. (2000, May 2). Netscape. Retrieved March 10, 2001, from the World Wide Web: http://home.netscape.com/security/jscookie.html.

Festa, Paul, (2000, May 11). IE Hole Exposes Web Surfers' Private Data. C|Net News.com. Retrieved March 10, 2001, from the World Wide Web: http://news.cnet.com/news/0-1005-200-1857707.html.

Form Tampering Vulnerabilities in Several Web-Based Shopping Cart Applications. (2000, February 1). Internet Security Systems. Retrieved March 10, 2001, from the World Wide Web: http://xforce.iss.net/alerts/advise42.php.

Lutes, Kyle D., (2000, October). Develop Scriptless Web Apps Using Visual Basic. InformIT. Retrieved October 31, 2000, from the World Wide Web: http://www.informit.com.