Maintaining State

Overview

1. The HTTP protocol is stateless: once the server is finished processing a page, the Page object is destroyed (including of course any instance variables). There are many situations where we need to remember data across postbacks. This is called *maintaining state*. For instance, when a user logs into a system, we need to *remember* that they are logged in, otherwise someone could just bookmark a page deeper in the site and come back to it later without logging in. ASP.NET provides a number of ways of maintaining state[1] [2] [3] [4]. We will consider two techniques in detail, and then briefly overview other techniques.

Overview of State Management

2. State management can be broken down into client-based options and server-based options

   Client-Based Options:

   1. ViewState – generally used for structured data
   2. Cookies
   3. Hidden Form Fields – generally used for a few variables
   4. QueryString – generally used for a few variables

   Server-Based Options:

   5. Application
   6. Session – generally used for structured data
   7. Profile Properties
   8. Database Support

Using Session Memory

3. When you access a page in a web application for the first time, a *session* is created. It is an object in memory on the server. We can store practically anything in this session object. The technique is simple. In server side code, we may define a variable, *value*:

```
MyClass value = …       // The object we want to persist
```

   To place *value* in session memory we give it a string *key* and write a statement like this:

```
Session[“key”] = value;       // Store the object
```

   On a subsequent postback, we can retrieve *value* from session memory with a statement like this:

```
MyClass value = (MyClass)Session[“key”];  // Retrieve a value
```

[1] http://msdn.microsoft.com/en-us/library/75x4ha6s.aspx
[2] http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx
[3] https://www.c-sharpcorner.com/UploadFile/2072a9/state-management-in-Asp-Net/
[4] https://www.codeproject.com/Articles/492397/State-Management-in-ASP-NET-Introduction

4. The key to using Session is that anytime your code modifies a value you immediately save the new value in Session. A very common problem is that somewhere in your code you forget to do this, or do it place where the code doesn't get executed.

5. See *CreateOrder.aspx* for an example that uses Session to store an instance of a custom class to maintain state across multiple pages (*AddProducts.aspx*, *ReviewOrder.aspx*)

QueryString

6. QueryString – With this technique, you put the information you want to maintain in key/value pairs in the URL. The sender writes code like this:

```
Response.Redirect( "newPage.aspx?key1=value1&key2=value2" );
```

   The receiving page writes code like this to retrieve the values:

```
value1 = Request.QueryString["key1"];
value2 = Request.QueryString["key2"];
```

7. You can check to see if a particular key is present in a query string with a statement like this:

```
if (Request.QueryString.AllKeys.Contains("key1")) {
```

8.

Advantages:	Disadvantages:
• No server resources are needed • Simple to code	• Security risks – the data is passed in plain text. • Limited size. Generally recommended to use less than 2000 characters.

ViewState

9. ViewState – We discussed earlier that ViewState is used to persist the properties of controls as part of the page lifecycle. You can also explicitly store practically any information you want in ViewState and it will travel to the browser and be returned when the next postback occurs where our code can read it back into memory. You can store multiple data values separately or it can be more convenient to store just one custom object created from a class to hold the data items. Here is how we will do this in ASP.NET:

```
MyClass value = …                    // The object we want to persist
ViewState[“key”] = value;            // Store the object
```

On a subsequent postback:

```
MyClass value = (MyClass)ViewState[“key”];      // Retrieve a value
```

10.

Advantages:

- No server resources are needed
- Simple to code

Disadvantages:

- Requires more bandwidth, could take a longer time to transmit
- Requires more memory on the client side, mobile devices may not have this capacity
- Security risks – the data is hashed, encoded, and compressed, but not encrypted!

Hidden Form Fields

11. Hidden Form Fields – We can use the HTML element:

```
<input id="Hidden1" type="hidden"/>
```

Which can then be addressed by JavaScript. The value is not displayed, but is hidden (but visible in clear text) in the HTML. We can address it on the server by using the *runat* attribute:

```
<input id="Hidden1" type="hidden" runat="server"/>
```

Then, it will be visible just as any variable is.

Cookies

12. We will not consider cookies in class. An "A" version of an assignment will have you use them. Find out more[5].

Page Properties

OMIT for now, Spring 2020

13. We can define a property in the page class to store a value for us, using the *get* to retrieve the value from Session and the *set* to store the value in Session. This makes the event-handler coding simpler. We no longer have to save a value to session immediately after it is modified.

14. Page Property Example 1

    a. Suppose we want to save a *Sum* variable across postbacks. We can define the *Sum* property:

```
public int Sum {
    get { return (int)Session["Sum"]; }
    set { Session["Sum"] = value; }
}
```

    b. Then, in an event-handler we might write code like this:

```
// Calls the setter
Sum += Convert.ToInt32(txtNum.Text);
```

    or:

```
// Calls the getter
txtMessage.Text += "Sum=" + Sum + "\n";
```

    c. A concern is how does *Sum* get its initial value? In the first time we call the getter, *Sum* may not even be in session in which case the page would crash. A solution in this case might be to detect if it is the first time on the page and then write a statement like this:

```
protected void Page_Load(object sender, EventArgs e) {
       // Calls the setter for the first time, initializing the value to 0
```

[5] http://msdn.microsoft.com/en-us/library/ms178194.ASPX

```
    if (!Page.IsPostBack) {
            Sum = 0;
    }
}
```

d. However, suppose we are trying to maintain *Sum* in session across multiple pages. For example, we leave a page where *Sum* was placed in session, the return later, expecting the previous value of *Sum* to be remembered. The code above would simply reset it to 0. A solution in this case would be to not only check if it is the first time on the page, but also to check if it is actually in session:

```
protected void Page_Load(object sender, EventArgs e) {
    if (!Page.IsPostBack) {
            if(Session["Sum"]==null) {
                    Sum = 0;
            }
    }
}
```

Thus, each situation needs careful thought as to what can go wrong.

15. Page Property Example 2

a. In this example we want to persist a List of Product objects. First we define the *Products* property (Product class is not shown):

```
public List<Product> Products {

    get { return (List<Product>)Session["products"]; }

    set { Session["products"] = value; }
}
```

b. Then, in an event-handler we might write code like this:

```
// Get price from user.
double price = Convert.ToDouble(txtPrice.Text);

// Create product object with price.
Product p = new Product(price);

// Add Product to List (which stores in Session)
Products.Add(p);
```

c. In Page_Load we could create the List.

```
if (!Page.IsPostBack) {
    // Create an empty List of Products
    Products = new List<Product>();
}
```

16.

Advantages:	Disadvantages:
• Simple to code • Extensible – can customize this to persist session values to XML, a database, or a web service	• Requires more memory on server.