State Management In ASP.NET MVC

As we all know, HTTP is a stateless protocol, i.e., each HTTP request does not know about the previous request. If you are redirecting from one page to another page, then you have to maintain or persist your data so that you can access it further. To do this, there were many techniques available in ASP.NET like ViewState, SessionState, ApplicationState etc.

ASP.NET MVC also provides state management techniques that can help us to maintain the data when redirecting from one page to other page or in the same page after reloading. There are several ways to do this in ASP.NET MVC -

1. Hidden Field
2. Cookies
3. Query String
4. ViewData
5. ViewBag
6. TempData

The above objects help us to persist our data on the same page or when moving from “Controller” to “View” or “Controller” to Controller”. Let us start practical implementation of these so that we can understand in a better way.

**Hidden Field**

It is not new, we all know it from HTML programming. Hidden field is used to hide your data on client side. It is not directly visible to the user on UI but we can see the value in the page source. So, this is not recommended if you want to store a sensitive data. It’s only used to store a small amount of data which is frequently changed.

The following code is storing the Id of employee and its value is 1001.

1. @Html.HiddenFor(x=>x.Id)

If you open the page source code for that page in the browser, you will find the following line of code, which is nothing but the HTML version of the above code with value. Just focus on last three attributes.

1. <input data-val="true" data-val-number="The field Id must be a number." data-val-required="The Id field is required." id="Id" name="Id" type="hidden" value="1001" />
2. <input type=”hidden” value=”333” />

**Cookies**

Cookies are used for storing the data but that data should be small. It is like a small text file where we can store our data. The good thing is that a cookie is created on client side memory in the browser. Most of the times, we use a cookie for storing the user information after login with some expiry time. Basically, a cookie is created by the server and sent to the browser in response. The browser saves it in client-side memory.

We can also use cookies in ASP.NET MVC for maintaining the data on request and respond. It can be like the below code.

1. HttpCookie cookie = **new** HttpCookie("TestCookie");
2. cookie.Value = "This is test cookie";
3. **this**.ControllerContext.HttpContext.Response.Cookies.Add(cookie);

Here, we have created one cookie and named it as “TestCookie”. We can create multiple cookies and add with Response. For getting the value from an existing cookie, we first need to check that the cookie is available or not; then, we can access the value of the cookie.

1. **if** (**this**.ControllerContext.HttpContext.Request.Cookies.AllKeys.Contains("TestCookie"))
2. {
3. HttpCookie cookie = **this**.ControllerContext.HttpContext.Request.Cookies["TestCookie"];
5. ViewBag.CookieMessage = cookie.Value;
6. }

Cookies are all depended on expiry, you can create a cookie on one action method in a controller and it will save on the client side and can be accessed in another action method easily.

**Query String**

In ASP.NET, we generally use a query string to pass the value from one page to the next page. Same we can do in ASP.NET MVC as well.

*http://localhost:49985/home/editemployee?name=TestValue*

I am making one request with the above URL. You can see that in this, I am passing name’s value as a query string and that will be accessible on “EditEmployee” action method in “Home” controller as the following image shows.  
  


**ViewData**

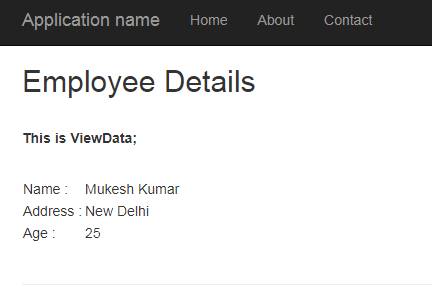
It helps us to maintain your data when sending the data from Controller to View. It is a dictionary object and derived from ViewDataDictionary. As it is a dictionary object, it takes the data in a key-value pair.

Once you create ViewData object, pass the value, and make redirection; the value will become null. The data of ViewData is not valid for next subsequent request. Take care of two things when using ViewData, first, check for null and second, check for typecasting for complex data types.

1. **public** ActionResult Index()
2. {
3. Employee emp = **new** Employee()
4. {
5. Id = 1001,
6. Name = "Mukesh Kumar",
7. Address = "New Delhi",
8. Age = 25
9. };
11. ViewData["Message"] = "This is ViewData";
12. ViewData["Emp"] = emp;
14. **return** View();
15. }

The above code contains two ViewData dictionary objects - ViewData["Message"] and ViewData["Emp"]. The first one is a simple string value but the next one contains complex employee data. When the View is going to render, we need to first check the ViewData for null and if it is not, then we can get the value.

1. @{
2. ViewBag.Title = "Home Page";
3. }
5. <div **class**="row">
6. <div **class**="col-md-4">
7. <h2>Employee Details</h2>
8. <br />
9. <p>
10. @**if**(ViewData["Message"] !=**null**)
11. {
12. <b>@ViewData["Message"].ToString();</b>
13. }
14. </p>
15. <br />
16. @**if** (ViewData["Emp"] != **null**)
17. {
19. **var** emp = (MVCStateManagement.Models.Employee)ViewData["Emp"];
20. <table>
21. <tr>
22. <td>
23. Name :
24. </td>
25. <td>
26. @emp.Name
27. </td>
28. </tr>
29. <tr>
30. <td>
31. Address :
32. </td>
33. <td>
34. @emp.Address
35. </td>
36. </tr>
37. <tr>
38. <td>
39. Age :
40. </td>
41. <td>
42. @emp.Age
43. </td>
44. </tr>
45. </table>
46. }
47. </div>
48. </div>



**ViewBag**

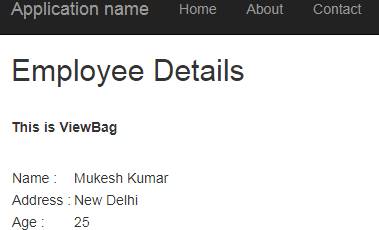
The ViewBag’s task is same as that of ViewData. It is also used to transfer the data from Controller to View. However, the only difference is that ViewBag is an object of Dynamic property introduced in C# 4.a. It is a wrapper around ViewData. If you use ViewBag rather than ViewData, you will not have to do typecasting with the complex objects and do not need to check for null.

If we consider the same above code with ViewBag, the output will be same.

1. **public** ActionResult Index()
2. {
3. Employee emp = **new** Employee()
4. {
5. Id = 1001,
6. Name = "Mukesh Kumar",
7. Address = "New Delhi",
8. Age = 25
9. };
11. ViewBag.Message = "This is ViewBag";
12. ViewBag.Emp = emp;
14. **return** View();
15. }

On View, you have to change ViewData with ViewBag.

1. @{
2. ViewBag.Title = "Home Page";
3. }
5. <div **class**="row">
6. <div **class**="col-md-4">
7. <h2>Employee Details</h2>
8. <br />
9. <p>
11. <b>@ViewBag.Message</b>
13. </p>
14. <br />
15. @{
16. **var** emp = ViewBag.Emp;
17. <table>
18. <tr>
19. <td>
20. Name :
21. </td>
22. <td>
23. @emp.Name
24. </td>
25. </tr>
26. <tr>
27. <td>
28. Address :
29. </td>
30. <td>
31. @emp.Address
32. </td>
33. </tr>
34. <tr>
35. <td>
36. Age :
37. </td>
38. <td>
39. @emp.Age
40. </td>
41. </tr>
42. </table>
43. }
44. </div>
45. </div>



**Note**

1. If you are using ViewData that is not defined on Controller, then it will throw an error; but with ViewBag, it will not.
2. Do not use ViewBag and ViewData with the same name, otherwise, only one message will display. See the following code in the controller is using both ViewData and ViewBag with same name “Message”.
   1. **public** ActionResult Index()
   2. {
   3. ViewData["Message"] = "This is ViewData";
   4. ViewBag.Message = "This is ViewBag";
   6. **return** View();
   7. }
   8. On view defined both as following.
   9. <b>@ViewBag.Message</b>
   10. @**if**(ViewData["Message"]!=**null**)
   11. {
   12. ViewData["Message"].ToString();
   13. }

The output will show only one message and that will be the last one [in this case, message will be “This is ViewBag”].

**TempData**

TempData is also a dictionary object as ViewData and stores value in key/value pair. It is derived from TempDataDictionary. It is mainly used to transfer the data from one request to another request or we can say subsequent request. If the data for TempData has been read, then it will get cleaned. To persist the data, there are different ways. It all depends on how you read the data.

**No Read Data**

If you haven’t read the data in redirection process, then your data is available with the next subsequent request. You can see that in the following code, we have set up a TempData[“Emp” but neither read it in any action method nor in view.

So, once the “About” page renders and if we move to “Contact” page, the TempData[“Emp”] will be available.

**Note**  
Do not read data on View.

1. **public** ActionResult Index()
2. {
3. Employee emp = **new** Employee() { Id = 1001, Name = "Mukesh Kumar", Address = "New Delhi", Age = 25 };
5. //Setting the TempData
6. TempData["Emp"] = emp;
7. **return** RedirectToAction("Index1");
8. }
9. **public** ActionResult Index1()
10. {
11. //Not reading TempData
12. **return** RedirectToAction("Index2");
13. }
14. **public** ActionResult Index2()
15. {
16. //Not reading TempData
17. **return** RedirectToAction("About");
18. }
19. **public** ActionResult About()
20. {
21. //Not reading TempData
22. **return** View();
23. }
25. **public** ActionResult Contact()
26. {
27. //Data will available here because we have not read data yet
28. **var** tempEmpData = TempData["Emp"];
29. **return** View();
30. }

**Normal Data Read**

If you read the data on “About” page when it will render and try to access the value on “Contact” page, it will not be available.

1. @{
2. ViewBag.Title = "About";
3. }
4. <h2>About Page</h2>
5. <br />
6. @{
7. **var** data = (MVCStateManagement.Models.Employee)TempData["Emp"];
8. }

TempData will not be available with Contact page because we have already read that data on “About” page. TempData is only available with subsequent request if you have not read yet.

1. **public** ActionResult Contact()
2. {
3. //Data will not available here because already read on About page
4. **var** tempEmpData = TempData["Emp"];
5. **return** View();
6. }

**Keep TempData**

If you still want to persist your data with the next request after reading it on “About” page that you can use “Keep()” method after reading data on “About” page. The Keep method will persist your data for next subsequent request.

1. @{
2. **var** data = (MVCStateManagement.Models.Employee)TempData["Emp"];
3. TempData.Keep();
4. }
5. **public** ActionResult Contact()
6. {
7. //TempData will available here because we have keep on about page
8. **var** tempEmpData = TempData["Emp"];
9. **return** View();
10. }

**Peek TempData**

Using Peek() method, we can directly access the TempData value and keep it for next subsequent request.

1. @{
2. **var** data = (MVCStateManagement.Models.Employee)TempData.Peek("Emp");
3. }

When we move to “Contact” page, the TempData will be available.

1. **public** ActionResult Contact()
2. {
3. //TempData will available because we have already keep data using Peek() method.
4. **var** tempEmpData = TempData["Emp"];
5. **return** View();
6. }