Session: 9

State Management and Optimization

Objectives

- Define and describe state management
- Explain and describe how to use cookies and application and session state
- Explain how to use caches to improve performance of an application
- Explain the process of bundling and minification

State Management

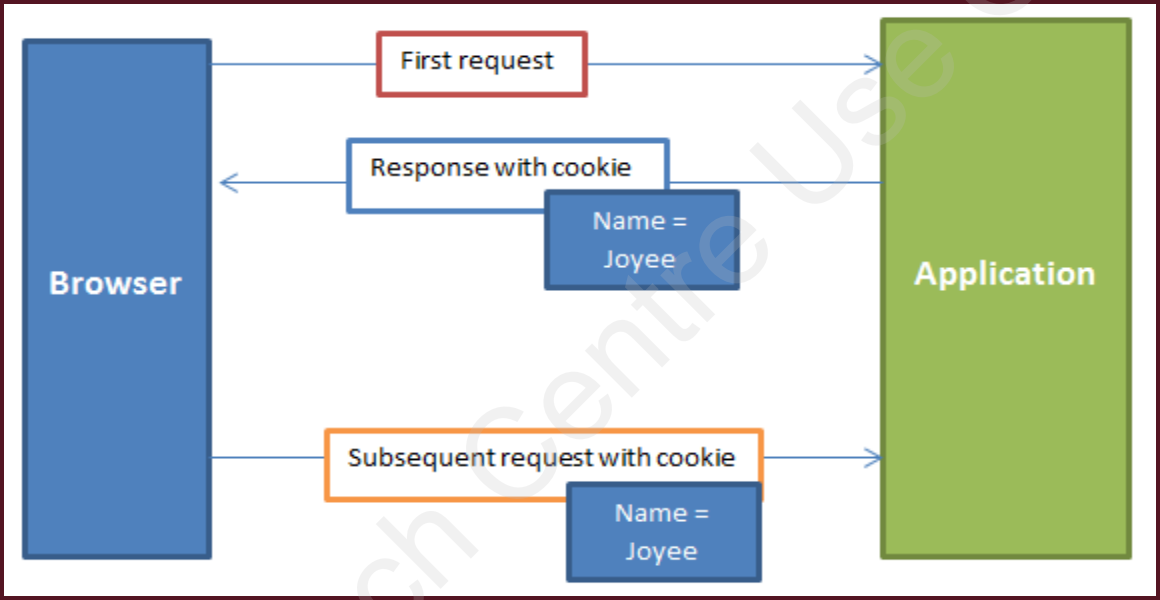
- Web pages use HTTP protocol to communicate between a Web browser and a Webserver.
- HTTP is a stateless protocol and cannot automatically indicate whether the sequential requests are coming from the same or different clients.
- You can implement state management in an ASP.NET MVC application using one of the following options:
  - Cookies
  - TempData
  - Application State
  - Session State

Cookies 1-5

- Cookies are used to store small pieces of information related to a user's computer, such as its IP address, browser type, operating system, and Web pages last visited.
- The purpose of storing this information is to offer a personalized experience to the user.
- Cookies are sent to a client computer along with the page output.
- These cookies are stored on the client's computer.
- When a browser requests the same page the next time, it sends the cookie along with the request information.
- The Web server reads the cookie and extracts its value.
- It then, process the Web page according to the information contained in the cookie and renders it on the Web browser.

Cookies 2-5

- Following figure shows how cookies are transmitted between browser and application:



- There are two types of cookies:
  - The first one is the session cookies that are stored in the browser's memory that are transmitted through the header during every request.
  - The other type of cookie is the persistent cookies that are stored in text files on a user's computer. This type of cookie is useful when you need to store information for a longtime.

Cookies 3-5

- Following code snippet shows how to create persistent cookies:

Snippet

```
Response.Cookies ["UserName"].Value = "John";
Response.Cookies ["UserName"].Expires = DateTime.Now.AddDays(2);
Response.Cookies ["LastVisited"].Value = DateTime.Now.ToString();
Response.Cookies ["LastVisited"].Expires = DateTime.Now.AddDays(2);
```

- This code creates the UserName and LastVisited cookies. The UserName cookie stores the name of a user and the LastVisited cookie stores the date and time when the user last visited the page. The expiry period for both the cookies is set as 2 days.
- Once you have a cookie, you can access the value of the cookie by using the built-in Request object.
- On the other hand to modify a cookie, you need to use the built-in Response object.

Cookies 4-5

- Following code snippet shows how to access a cookie that stores a single value:

Snippet

```
if (Request.Cookies["UserName"].Value != null)
{
string Name = Request.Cookies["UserName"].Value;
}
```

- This code specifies that if the value of the UserName cookie is not null, then, the value is assigned to the Name string.

Cookies 5-5

- You can also access the values from a cookie that store multiple values.
- Following code snippet shows how to access a cookie that stores multiple values:

Snippet

```
if (Request.Cookies["UserInfo"] != null)
{
string Name = Request.Cookies["UserInfo"]["UserName"];
stringVisitedOn = Request.Cookies["UserInfo"]["LastVisited"];
}
```

- In this code:
  - The UserInfo cookie contains two sub-keys named, UserName and LastVisited.
  - If the value of the UserInfo cookie is not null, the value of the two sub-keys are assigned to the Name and VisitedOn strings respectively.

TempData 1-3

- TempData:
  - Is a dictionary that stores temporary data in the form of key-value pairs.
  - Allows store values between requests.
- You can add values to TempData by adding them to TempData collection.
- The data stored in TempData is available during the current and subsequent requests.
- TempData is useful in situations when you need to pass data to a view during a page redirect.

TempData 2-3

- Following code snippet shows the content of a controller class named HomeController:

Snippet

```
public class HomeController : Controller
    {
publicActionResult Index()
        {
TempData["tempText"] = "Welcome to MVC";
returnRedirectToAction("Browse");
        }

publicActionResultBrowse()
        {
return View();
        }
    }
```

- In this code, when you send request for the Index() action method, the value, Welcome to MVC, is stored in the TempData key, named tempText. Hence, you are automatically redirected to the Browse() action method that will render a view.

TempData 3-3

- Following code snippet shows the content of the Browse() action method:

Snippet

```
@{varmsg = TempData["tempText"] as String;}
@msg
```

- The code renders a view that access the data stored for it in TempData and then display it to the user.

Application State 1-5

- Application state enables storing application-specific information as key-value pairs.
- When a user accesses any URL, application state is created for the first time.
- After that it stores the application-specific information.
- This information can be shared with all the pages and user sessions of the application.
- For this, you need to use the HttpApplicationState class.
- This class is accessed by using the Application property of the HttpContext object.

Application State 2-5

- Whenever any application events occur, the application state is initialized and manipulated.
- These application events include the following:
  - Application.Start: Event is raised when an application receives a request for a page for the first time, and when the server or the application is restarted. Event handler of this event contains the code for initializing the application variables.
  - Application.End: Event is raised when the server or the application is stopped or restarted. Event handler of this event contains the code to clear the resources that the application has already used.
  - Application.Error: Event is raised when an unhandled error occurs.
- The handlers for these three types of events are defined in the Global.asax file.
- You can write the code to manipulate the application state in one of these event handlers.

Application State 3-5

- To store application-specific information in application state, you need to create variables and objects.
- Then, you can add these variables and objects to the application state.
- These variables and objects are accessible for any components of an ASP.NET MVC application.
- You need to write the code for initializing these application variables and objects inside the Application_Start() function available in the Global.asax file.
- Following code snippet shows creating a variable named TestVariable and then, stores it in the application state:

Snippet

```
HttpContext.Application ["TestVariable"] = "Welcome to MVC";
```

- This code creates a variable named, TestVariable. All the pages and the user sessions of the application can access the value stored in this variable.

Application State 4-5

- When you run the application that includes the newly created variable, any pages of this application can retrieve the value of this variable.
- Following code snippet shows how to access the value of TestVariable:

Snippet

```
stringval = (string)HttpContext.Application["TestVariable"];
```

- This code access the value included in the TestVariable and then, stores it in a local variable named val.
- Once you have created and added a variable or object to an application state, you can also remove it from the application state using the Remove() method.

Application State 5-5

- Following code snippet shows how to remove an application variable named, TestVariable, from the application state:

Snippet

```
HttpContext.Application.Remove("TestVariable");
```

- This code uses the Remove() method that will remove the variable named, TestVariable from the application state.
- You can also use the RemoveAll() method to remove all the available variables present in the application state.
- Following code snippet shows using the RemoveAll() method:

Snippet

```
HttpContext.Application.RemoveAll();
```

- This code will remove all the existing variables or objects from the application state.

Session State 1-7

- A session state stores session-specific information for an ASP.NET MVC application.
- However, the scope of session state is limited to the current browser session.
- When many users access an application simultaneously, then, each of these users will have a different session state.
- Similar to application state, a session state stores application-specific data in key-value pairs.
- These session-specific information should be maintained between server round trips and requests for pages.

Session State 2-7

- An active session is identified and tracked by a unique session ID string containing American Standard Code for Information Interchange (ASCII) characters.
- The ASP.NET MVC Framework provides the SessionStateModule class that enables you to generate and obtain the session ID strings.
- Like application state, you can also store objects and variables in a session state.
- However, a session variable remains active for 20 minutes without any user interaction.
- You can use the same methods as application state, to add and access variable or objects from the session state.

- Following code snippet shows adding a variable named, TestVariable in a session state:

Snippet

```
Session["TestVariable"]="Welcome to MVC Application";
```

- This code creates a variable named, TestVariable that contains 'Welcome to MVC Application' as its value.
- Following code snippet shows how to access the value of the newly created variable:

Snippet

```
stringval = (string) Session["TestVariable"];
```

- This code will access the value of TestVariable and stores it in the variable named, val.

Session State 4-7

- While using session state, you need to consider the following issues:
  - A variable or an object that is added to a session state remains until a user closes the browser window. By default, a variable or object is automatically removed from the session state after 20 minutes if a user does not request a page.
  - A variable or an object added to the session state is related to a particular user.
- Similar to application state, a session state can be globally accessed by the current user.
- A session state can be lost in case of the following situations:
  - User closes or restarts the browser.
  - User accesses the same Web page through a different browser window.
  - The session times out because of inactivity.
  - The Session.Abandon() method is called within the page code.

- To remove an object or a variable that has been added to a session state, you can use the Remove() or RemoveAll() methods.
- While using session states in an application, you need to configure it in the Web.config file of the application.
- The Web.config file enables you to define advanced options, such as the timeout and the session state mode.
- Following code snippet shows the options that you can configure inside the <sessionState> element:

Snippet

```
<sessionState
cookieless="UseCookies" cookieName="Test_SessionID"
timeout="20"
mode="InProc" />
```

- This code uses the cookieless, timeout, and mode session state attributes.

- You can use the cookieless attribute to specify whether a cookie is to be used or not.
- Following table lists the values that you can specify for the cookieless attribute:

Value	Description
UseCookies	Allows you to specify that cookies are always used.
UseUri	Allows you to specify that cookies are never used.
UseDeviceProfile	Allows you to specify that the application should check the Browser Capabilities object to identify whether to use cookies.
AutoDetect	Allows you to specify that cookies should be used if the browser supports them.

Session State 7-7

- You can use the timeout attribute of session state to specify a time period in minutes.
- You can use the mode attribute to specify where the values of the session state is to be stored.
- You can use one of the following values of the mode attribute:
  - Custom: Allows you to specify that a custom data store should be used to store the session-state data.
  - InProc: Allows you to specify that the data is to be stored in the same process as the application. The Session.End event is raised only in this mode.
  - Off: Allows you to specify that the session state is disabled.
  - SQLServer: Allows you to specify that the session state is to be stored by using SQL Server database to store the state data.
  - StateServer: Allows you to specify that the session state is to be stored in a service running on the server or a dedicated server.

Optimizing Web Application Performance

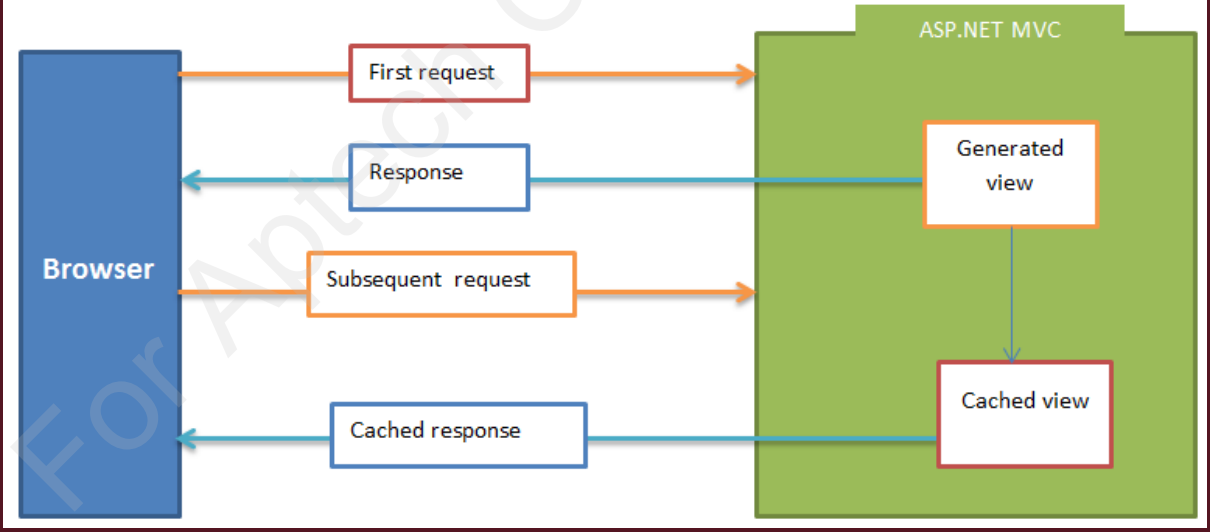
- You can use a Web application for different purposes, such as discussion forums, online shopping, and social networking.
- To improve the performance of an application, you can reduce the number of interaction between the server and the database.
- The ASP.NET MVC Framework provides different techniques to improve the performance of an application.
- Following are two techniques that you can use to improve performance of an application:
  - First one is caching that you can use to optimize the performance of an application. Caching enables reducing the number of interactions between the server and database.
  - Second one is bundling and minification, which allows you to reduce the number of requests and the size of the requests between a client and a server.

Output Caching 1-6

- Output caching allows you to cache the content returned by an action method.
- As a result, the same content does not need to be generated each time the action method is invoked.
- You need to consider the following factors, before implementing output caching:
  - Check whether output caching should be used in the application. If the content of the application changes frequently, output caching may not be useful.
  - Check the time duration for which the output has to be cached. This would depend on how frequently data associated with the output changes.
  - Check the location where you need to cache the output. You can cache the output on different locations, such as client machines and server.

Output Caching 2-6

- In an ASP.NET MVC application, you can use output caching to cache the pages of the application for a specific time period.
- When a user request arrives for a same page within that time period, the application renders the cached page to the user instead of re-rendering the page.
- Following figure shows how ASP.NET MVC implements output caching:



Output Caching 3-6

- In an application, you can use caching at an action method.
- You can use the output cache attribute to an action method or the whole controller that needs to be cached. Thus, you can cache the result returned by the action method.
- Following code snippet shows how to add an output cache attribute to an action method:

Snippet

```
public class HomeController : Controller
  {
     [OutputCache]
publicActionResult Index()
     {
      //Code to perform some operations
     }
 }
```

- In this code, the [OutputCache] attribute is added to the Index() action method of the Home controller.

Output Caching 4-6

- You can also use caching to cache the output for a specified duration.
- Caching the output for a small period of time is known as micro caching that you can use when the traffic on an application is high.
- Following code snipept shows specifying the duration and location of a cache attribute:

Snippet

```
public class HomeController : Controller
  {
   [OutputCache(Duration=5)]
publicActionResult Index()
  {
ViewBag.Message="This page is cashed for " + DateTime.Now;
return View();
   }
  }
```

- In this code, the output is cached for 5 seconds. This will refresh the page in every five seconds.

Output Caching 5-6

- You can also define the location where you want the data to be cached.
- By convention, the data is cached in three locations when you use the [OutputCache] attribute.
- These locations can be the server, a proxy server, and the Web browser.
- However, you can specify a location of the output to be cached.
- For this, you need to modify the Location property of the [OutputCache] attribute.

Output Caching 6-6

- You can use any one of the following values for the Location property:
  - Any: Specifies that the output cache should be stored on the browser, a proxy server, or the server where the request was processed.
  - Client: Specifies that the output cache should be stored on the browser where the request originated.
  - Downstream: Specifies that the output cache should be stored in any HTTP cache-capable device other than the original server.
  - Server: Specifies that the output cache should be stored on the server where the request was processed.
  - None: Specifies that the output cache is disabled for the requested page.

Data Caching 1-2

- In recent times, most of the Web applications contain dynamic data.
- These data needs to be retrieved from a database.
- So executing a database query for different users every time can reduce the performance of the application.
- To overcome such problems, you can use the data caching technique.
- To perform data caching in an ASP.NET MVC application, you can use the MemoryCache class.
- Following code snippet shows how to use the MemoryCache class to cache data:

Snippet

```
Customer dtUser = System.Runtime.Caching.MemoryCache.Default.
AddOrGetExisting("UserData", getUser(),
System.DateTime.Now.AddHours(1));
```

Data Caching 2-2

- In the preceding code:
  - The AddOrGetExisting() method enables the application to refresh and access data from the cache. This method accesses the data from the cache if it contains the relevant data.
  - If there is no relevant data in the cache, then, the AddOrGetExisting() method allows adding the data to the cache by invoking the getUser() method.
  - The first parameter of the AddOrGetExisting() method specifies the unique identifier of the object that needs to be stored in the memory cache.
  - The second parameter specifies the object that needs to be stored in the cache, and the third parameter specifies the time when the cache should expire.
- You can use a HTTP caching technique in the browser and proxy cache.
- Storing data in the browser cache allows you to reduce the process of downloading content from the server repeatedly.

Bundling and Minification

- The ASP.NET MVC Framework provides two techniques known as Bundling and Minification.
- These techniques allow you to improve the load time of a Web page in an application.
- This is done by reducing the number of user requests to the server and the size of the requested resources.

Using Bundling 1-3

- Bundling is a technique that allows you to reduce the user requests in your application by combining several individual scripts into a single request.
- When there are multiple files that need to be downloaded, at times it became very time consuming.
- As a solution, you can use the bundling technique that allows you to combine multiple files and then, it can be downloaded as a single entity.
- To use the bundling technique in an application, you need to define the files that you want to combine together.
- You can achieve this using the BundleConfig class in the App_Start folder of your application.
- You can create a bundle using the RegisterBundles() method of the BundleConfig class.

Using Bundling 2-3

- Following code snippet shows using the RegisterBundles() method of the BundleConfig class:

Snippet

```
public static void RegisterBundles(BundleCollection bundles) {
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                        "~/Scripts/jquery.unobtrusive*",
                        "~/Scripts/jquery.validate*"));
bundles.Add(new tyleBundle("~/Content/css").Include("~/Content/Site.css",
"~/Content/styles.css"));
}
```

- In this code:
  - The RegisterBundles() method creates, registers, and configures bundles.
  - A new instance of the ScriptBundle class is created to create a bundle of scripts.
  - The path, ~/bundles/jqueryval, has been assigned to the bundle.
  - Then, the Include() method on the bundle object enables you to specify the files that should be included in the bundle.
  - Finally, the * symbol is used to indicate that the bundle should include all the script files whose name begins with jquery.unobtrusive or jquery.validate.

Using Bundling 3-3

- You can include these bundled files in a view.
- For that, you need to use the @Styles.Render and @Scripts.Render methods in your view.
- After that, you need to specify a path that you have configured in the bundle configuration as a parameter to these methods.
- Following code snippet shows using the @Styles.Render and @Scripts.Render methods in a view:

Snippet

```
@Scripts.Render("~/bundles/jqueryval")
@Styles.Render("~/Content/css")
```

Using Minification

- Minification allows you to reduce the size of a file by removing unnecessary whitespaces and comments, and shortening the variable names.
- You can use such code while using interpreted languages, such as JavaScript.
- By using minification, you can reduce the size of a file and thus reduce the time required to access it from the server and load it into the browser.

Summary

- The ASP.NET MVC Framework provides state management features that automatically indicate whether the sequential requests are coming from the same or different clients.
- Cookies are used to store small pieces of information related to a user's computer, such as its IP address, browser type, and operating system.
- TempData is a dictionary that stores temporary data in the form of key-value pairs
- Application state enables storing application-specific information as key-value pairs.
- A session state stores session-specific information for an ASP.NET MVC application.
- Output caching allows you to cache the content returned by an action method, as a result, the same content does not need to be generated each time the action method is invoked.
- The ASP.NET MVC Framework provides two techniques known as Bundling and Minification that allows you to improve the load time of a Web page in an application.