Caching Application And Data

UNIT - 6

Caching

- Caching is a technique of storing frequently used data/information in memory, so that, when the same data/information is needed next time, it could be directly retrieved from the memory instead of being generated by the application.
- Caching is extremely important for performance boosting in ASP.Net, as the pages and controls are dynamically generated here. It is especially important for data related transactions, as these are expensive in terms of response time.
- Caching places frequently used data in quickly accessed media like the random access memory of the computer. The ASP.Net runtime includes a key-value map of CLR objects called cache. This lives with the application and is available via the HttpContext and System.Web.UI.Page.

Caching

- In some respect, caching is similar to storing the state objects. However, the storing information in state objects is deterministic, i.e., you can count on the data being stored there, and caching of data is nondeterministic.
- The data will not be available if its lifetime expires, or the application releases its memory, or caching does not take place for some reason.
- You can access items in the cache using an indexer and may control the lifetime of objects in the cache and set up links between the cached objects and their physical sources.

Page Output Caching

- Page Output Caching **caches an entire page**. It provides you to cache the entire rendered contents of a page in memory. The next time that any user requests the same page, the page is retrieved from the cache.
- **Rendering a page may involve some complex processes like, database access, rendering complex controls etc. Output caching allows bypassing** the round trips to server by caching data in memory. Even the whole page could be cached.
- The **Output Cache** directive is responsible of output caching. It enables output caching and provides certain control over its behavior.

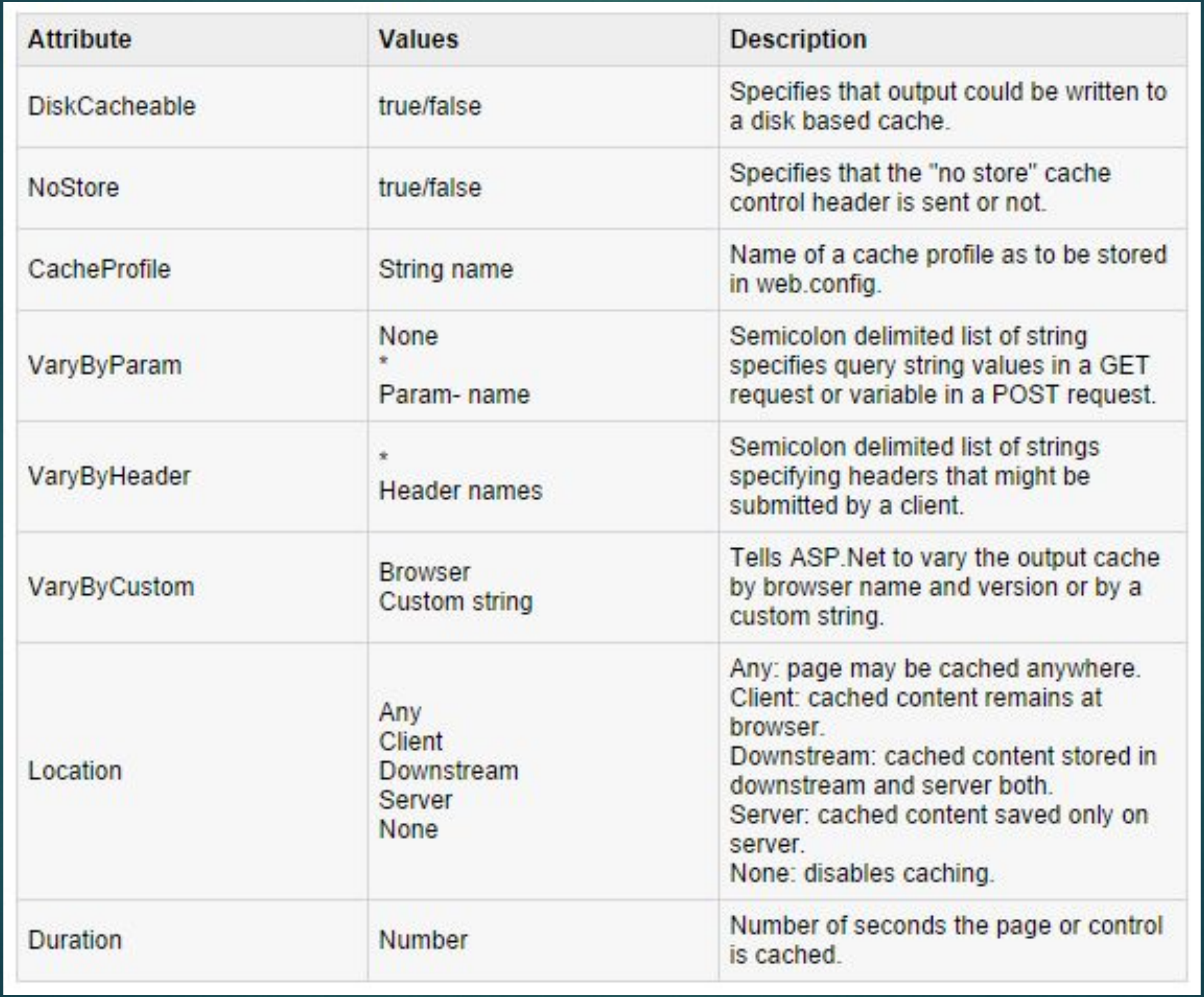
Page Output Caching

- Syntax for OutputCache directive:

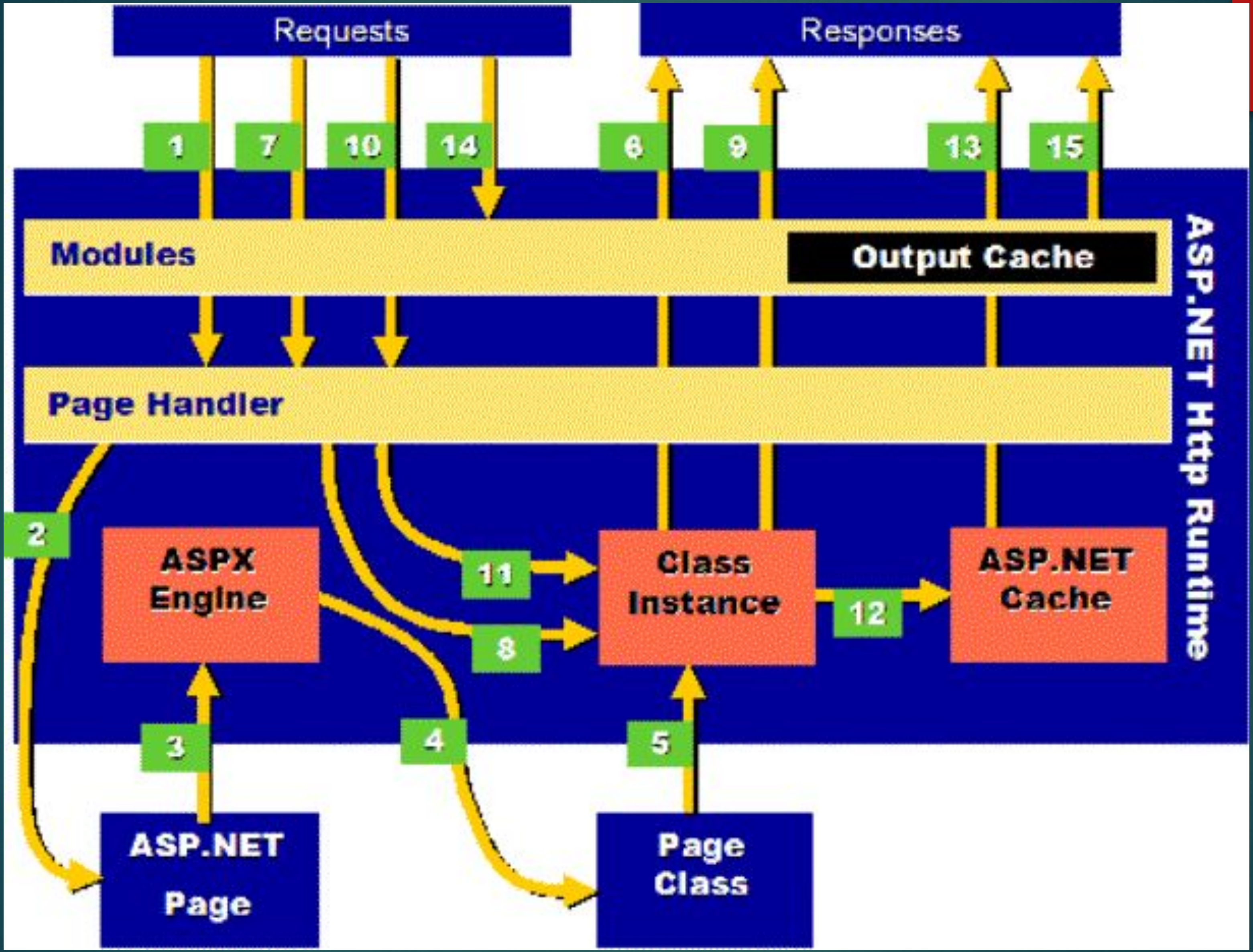
  <%@ OutputCache Duration="15" VaryByParam="None" %>
- Put this directive under the page directive . This tells the environment to cache the page for 15 seconds.
- The following event handler for page load would help in testing that the page was really cached:
- protected void Page_Load(object sender, EventArgs e)

  ```
   {
    Response.Write("This page was generated and cache at:" +
  DateTime.Now.ToString());
  }
  ```

Page Output Caching



Attribute	Values	Description
DiskCacheable	true/false	Specifies that output could be written to a disk based cache.
NoStore	true/false	Specifies that the "no store" cache control header is sent or not.
CacheProfile	String name	Name of a cache profile as to be stored in web.config.
VaryByParam	None * Param- name	Semicolon delimited list of string specifies query string values in a GET request or variable in a POST request.
VaryByHeader	* Header names	Semicolon delimited list of strings specifying headers that might be submitted by a client.
VaryByCustom	Browser Custom string	Tells ASP.Net to vary the output cache by browser name and version or by a custom string.
Location	Any Client Downstream Server None	Any: page may be cached anywhere. Client: cached content remains at browser. Downstream: cached content stored in downstream and server both. Server: cached content saved only on server. None: disables caching.
Duration	Number	Number of seconds the page or control is cached.


Requests
Responses
1
7
10
14
6
9
13
15
Modules
Output Cache
ASP.NET Http Runtime
Page Handler
2
ASPX Engine
11
8
Class Instance
12
ASP.NET Cache
3
4
5
ASP.NET Page
Page Class

Partial Page Caching

- Partial-Page Output Caching, or page fragment caching, allows **specific regions of pages to be cached**.
- ASP.NET provides a way to take advantage of this powerful technique, requiring that the part(s) of the page you wish to have cached **appear in a User Control**.
- One way to specify that the contents of a User Control should be cached is to supply an **Output Cache directive at the top of the User Control.** That's it! The content inside the User Control will now be cached for the specified period, while the ASP.NET Web page that contains the User Control will continue to serve dynamic content.
- (Note that for this you should not place an **Output Cache directive in the ASP.NET Web page** that contains the User Control - just inside of the User Control.)

Data Caching

- The main aspect of data caching is **caching the data source controls**. We have already discussed that the data source controls represent data in a data source, like a database or an XML file.
- These controls derive from the abstract class Data Source Control and have the following inherited properties for implementing caching:

Data Caching

- **CacheDuration** - It sets the number of seconds for which the data source will cache data.
- **CacheExpirationPolicy** - It defines the cache behavior when the data in cache has expired.
- **CacheKeyDependency** - It identifies a key for the controls that auto-expires the content of its cache when removed.
- **EnableCaching** - It specifies whether or not to cache the data.

Example

- To demonstrate data caching, create a new website and add a new web form on it. Add a Sql Data Source control with the database connection already used in the data access tutorials.

For this example, add a label to the page, which would show the response time for the page.

```
<asp:Label ID="lbltime" runat="server"></asp:Label>
```

- Apart from the label, the content page is same as in the data access tutorial. Add an event handler for the page load event:

```
protected void Page_Load(object sender, EventArgs e)
{
lbltime.Text = String.Format("Page posted at: {0}",
DateTime.Now.ToLongTimeString());
}
```

- When you execute the page for the first time, nothing different happens, the label shows that, each time you refresh the page, the page is reloaded and the time shown on the label changes.
- Next, set the EnableCaching attribute of the data source control to be 'true' and set the Cacheduration attribute to '60'. It will implement caching and the cache will expire every 60 seconds.
- The timestamp changes with every refresh, but if you change the data in the table within these 60 seconds, it is not shown before the cache expires.

Object Caching

- Object caching provides more flexibility than other cache techniques. You can use object caching to place any object in the cache. The object can be of any type - a data type, a web control, a class, a dataset object, etc.
- The item is added to the cache simply by assigning a new key name, shown as follows Like:
- Cache["key"] = item
- ASP.NET also provides the Insert() method for inserting an object to the cache. This method has four overloaded versions. Let us see them:

Creating Caching

- The .NET data caching API is comprised of the two classes in the System.Web.Caching namespace.
- The first class, Cache, is the class we'll be using to add and remove items from the data cache.
- The second class, CacheDependency, is used when assigning a cache dependency to an item in the data cache
- value = Cache("key") - or - value = Cache.Get("key")

Object caching

- While the Output Cache directive is easy to use and work with, it allows you only to cache the entire page, or an entire User Control.
- While this is just fine for some situations, there will be times where caching an object will be much smarter. While the following example might seem a bit silly, it does show how the Cache object can be used to store for instance an Array List with custom objects in it.
- Create a new project, or use an existing page and change the Code Behind Page_Load method to something like this (this example doesn't require you to modify the markup part):

Example – object caching

```
protected void Page_Load(object sender, EventArgs e)
{
    ArrayList datestamps;
    if(Cache["datestamps"] == null)
    {
        datestamps = new ArrayList();
        datestamps.Add(DateTime.Now);
        datestamps.Add(DateTime.Now);
        datestamps.Add(DateTime.Now);

   Cache.Add("datestamps", datestamps, null, System.Web.Caching.Cache.NoAbsoluteExpiration, new TimeSpan(0, 0, 60),
System.Web.Caching.CacheItemPriority.Default, null);
   }
    else
        datestamps = (ArrayList)Cache["datestamps"];

    foreach(DateTime dt in datestamps)
        Response.Write(dt.ToString() + "<br />");
}
```

Absolute Cache Expiration

- Absolute expiration means It will **expire cache after some time period** set at the time of activating cache.
- This will be absolute expiration whether cache will be used or not It will expire the cache.
- This type of expiration used to cache data which are not frequently changing.
- string cache Data = "The data to be cached";

  //Absolute Expiration

```
Cache.Insert("AbsoluteCacheKey", cacheData,
  null, DateTime.Now.AddMinutes(1),
  System.Web.Caching.Cache.NoSlidingExpiration);
```

Sliding Cache Expiration

- Sliding expiration means It will **expire cache after time period at the time of activating cache** if any request is not made during this time period.
- This type of expiration is useful when there are so many data to cache.
- So It will put those items in the cache which are frequently used in the application.
- So it will not going to use unnecessary memory.
- string cacheData = "The data to be cached";

```
//Sliding Expiration
Cache.Insert("SlidingExpiration", cacheData, null,
System.Web.Caching.Cache.NoAbsoluteExpiration, TimeSpan.FromMinutes(1));
```