ASP.NET - DATA CACHING

What is 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 such as the random access memory of the computer. The ASP.NET runtime includes a key-value map of CLR objects called cache. This resides with the application and is available via the HttpContext and System.Web.UI.Page.

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 in the following cases:

- If its lifetime expires,

- If the application releases its memory,

- If 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.

Caching in ASP.Net

ASP.NET provides the following different types of caching:

- **Output Caching** : Output cache stores a copy of the finally rendered HTML pages or part of pages sent to the client. When the next client requests for this page, instead of regenerating the page, a cached copy of the page is sent, thus saving time.

- **Data Caching** : Data caching means caching data from a data source. As long as the cache is not expired, a request for the data will be fulfilled from the cache. When the cache is expired, fresh data is obtained by the data source and the cache is refilled.

- **Object Caching** : Object caching is caching the objects on a page, such as data-bound controls. The cached data is stored in server memory.

- **Class Caching** : Web pages or web services are compiled into a page class in the assembly, when run for the first time. Then the assembly is cached in the server. Next time when a request is made for the page or service, the cached assembly is referred to. When the source code is changed, the CLR recompiles the assembly.

- **Configuration Caching** : Application wide configuration information is stored in a configuration file. Configuration caching stores the configuration information in the server memory.

In this tutorial, we will consider output caching, data caching, and object caching.

Output Caching

Rendering a page may involve some complex processes such as, 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 OutputCache directive is responsible of output caching. It enables output caching and provides certain control over its behaviour.

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)
{
    Thread.Sleep(10000);
    Response.Write("This page was generated and cache at:" +
    DateTime.Now.ToString());
}
```

The **Thread.Sleep** method stops the process thread for the specified time. In this example, the thread is stopped for 10 seconds, so when the page is loaded for first time, it takes 10 seconds. However, next time you refresh the page it does not take any time, as the page is retrieved from the cache without being loaded.

The OutputCache directive has the following attributes, which helps in controlling the behaviour of the output cache:

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 specifies 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	Any: page may be cached anywhere.
	Client	Client: cached content remains at browser.
	Downstream	Downstream: cached content stored in downstream and server both.
	Server	
	None	Server: cached content saved only

		on server.
		None: disables caching.
Duration	Number	Number of seconds the page or control is cached.

Let us add a text box and a button to the previous example and add this event handler for the button.

```
protected void btnmagic_Click(object sender, EventArgs e)
{
    Response.Write("<br><br>");
    Response.Write("<h2> Hello, " + this.txtname.Text + "</h2>");
}
```

Change the OutputCache directive:

```
<%@ OutputCache Duration="60" VaryByParam="txtname" %>
```

When the program is executed, ASP.NET caches the page on the basis of the name in the text box.

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 DataSourceControl and have the following inherited properties for implementing 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 SqlDataSource 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());
}
```

The designed page should look as shown:

ID	Title	AuthorLastName	AuthorFirstName	Topic	Publisher
0	abc	abc	abc	abc	abc
1	abc	abc	abc	abc	abc
2	abc	abc	abc	abc	abc
3	abc	abc	abc	abc	abc
4	abc	abc	abc	abc	abc

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.

```
<asp:SqlDataSource ID = "SqlDataSource1" runat = "server"
   ConnectionString = "<%$ ConnectionStrings: ASPDotNetStepByStepConnectionString %>"
   ProviderName = "<%$ ConnectionStrings:
ASPDotNetStepByStepConnectionString.ProviderName %>"
   SelectCommand = "SELECT * FROM [DotNetReferences]"
   EnableCaching = "true" CacheDuration = "60">
</asp:SqlDataSource>
```

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:

Overload	Description
Cache.Insert(*key*, *value*;	Inserts an item into the cache with the key name and value with default priority and expiration.
Cache.Insert*key*, *value*, *dependencies*;	Inserts an item into the cache with key, value, default priority, expiration and a CacheDependency name that links to other files or items so that when these change the cache item remains no longer valid.
Cache.Insert*key*, *value*, *dependencies*, *absoluteExpiration*, *slidingExpiration*;	This indicates an expiration policy along with the above

Cache.Insert *key, value, dependencies, absoluteExpiration, slidingExpiration, priority, onRemoveCallback* ;	This along with the parameters also allows you to set a priority for the cache item and a delegate that, points to a method to be invoked when the item is removed.

Sliding expiration is used to remove an item from the cache when it is not used for the specified time span. The following code snippet stores an item with a sliding expiration of 10 minutes with no dependencies.

```csharp
Cache.Insert("my_item", obj, null, DateTime.MaxValue, TimeSpan.FromMinutes(10));
```

Example

Create a page with just a button and a label. Write the following code in the page load event:

```csharp
protected void Page_Load(object sender, EventArgs e)
{
    if (this.IsPostBack)
    {
        lblinfo.Text += "Page Posted Back.<br/>";
    }
    else
    {
        lblinfo.Text += "page Created.<br/>";
    }

    if (Cache["testitem"] == null)
    {
        lblinfo.Text += "Creating test item.<br/>";
        DateTime testItem = DateTime.Now;
        lblinfo.Text += "Storing test item in cache ";
        lblinfo.Text += "for 30 seconds.<br/>";
        Cache.Insert("testitem", testItem, null,
        DateTime.Now.AddSeconds(30), TimeSpan.Zero);
    }
    else
    {
        lblinfo.Text += "Retrieving test item.<br/>";
        DateTime testItem = (DateTime)Cache["testitem"];
        lblinfo.Text += "Test item is: " + testItem.ToString();
        lblinfo.Text += "<br/>";
    }

    lblinfo.Text += "<br/>";
}
```

When the page is loaded for the first time, it says:

```
Page Created.
Creating test item.
Storing test item in cache for 30 seconds.
```

If you click on the button again within 30 seconds, the page is posted back but the label control gets its information from the cache as shown:

```
Page Posted Back.
Retrieving test item.
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

Loading [MathJax]/jax/output/HTML-CSS/jax.js