CACHING

Introduction

- Caching allows you to store a particular version of ASP.NET page on the server.
- The requests to the same page are served with this cached version.
- No processing is required every time.
- This is referred as Output Caching.
- One can also cache data i.e. variables, objects of our application.
- This is referred as Data Caching.

@OutputCache directive

- OutputCache directive is used to enable caching for a ASP.NET page or even user control.
- The directive tells ASP.NET the duration in seconds for which the page is to be cached.

```
<%@ OutputCache Duration="20" VaryByParam="TextBox1"
VaryByHeader="none" VaryByCustom="none" %>
```

- The VaryByXXXX attributes tells how ASP.NET should cache multiple versions of the page.
- The *Duration* and *VaryByParam* are mandatory.

VaryByParam Attribute

- The VaryByParam attribute causes a new instance of a page to be cached when a different parameter is passed to the page.
- The parameter can be either a query string parameter (GET) or a form parameter (POST).
- It also accepts a semicolon-separated list of strings.

Attributes of OutputCache

- VaryByControl
  - A semicolon-separated list of strings that represent properties of the user control.
  - Each distinct combination of values for the specified properties will originate a distinct copy of the page in the cache.
  - Applies to User Control only.
- VaryByCustom
  - A semicolon-separated list of strings that lets you maintain distinct cached copies of the page based on the browser type or user-defined strings.

Attributes of OutputCache

- VaryByHeader
  - A semicolon-separated list of HTTP headers.
  - Applies to page only.
- Location
  - Specifies a valid location to store the output of a page.
  - Applies to page only.

Attributes of OutputCache

- Shared
  - Indicates whether the user control output can be shared with multiple pages. False by default.
  - Applies to User control only.
- SqlDependency
  - Indicates a dependency on the specified table on a given SQL Server database.
  - Whenever the contents of the table changes, the page output is removed from the cache.
  - Value format → Database:Table

VaryByCustom Attribute

- The value string of *VaryByCustom* is passed to the *GetVaryByCustomString* method, if any, in the *global.asax* file.

- The method takes the string and returns another string that is specific to the request.

```
<%@ OutputCache Duration="20" VaryByParam=“none“
VaryByCustom=“device” %>
```

```
public override string GetVaryByCustomString(HttpContext context,
string custom)
{
if (custom == "device")
return Request.Browser.Type;
return base.GetVaryByCustomString(context, custom);
}
```

HttpCachePolicy (Programmatically)

- A programming interface alternative to using the *@OutputCache* directive.

- It provides direct methods to set cache-related HTTP headers.

- *Response* has a property *Cache* which represents *HttpCachePolicy* object.

Properties of HttpCachePolicy

- VaryByHeaders
  - Gets an object of type *HttpCacheVaryByHeaders*, representing the list of all HTTP headers that will be used to vary cache output.
- VaryByParams
  - Gets an object of type HttpCacheVaryByParams, representing the list of parameters received by a GET or POST request that affect caching.

Methods of HttpCachePolicy

- AppendCacheExtension
  - Appends the specified text to the *Cache-Control* HTTP header.
- SetNoServerCaching
  - Disables server output caching for the current response.
- SetLastModified
  - Sets the *Last-Modified* HTTP header to a particular date and time.
- SetMaxAge
  - Sets the *max-age* attribute on the *Cache-Control* header to the specified value. The sliding period cannot exceed one year.

Methods of HttpCachePolicy

- SetSlidingExpiration
  - Sets cache expiration to sliding. When cache expiration is set to sliding, the *Cache-Control* header is renewed at each response.
- SetExpires
  - Sets the *Expires* header to an absolute date and time.
- SetLastModifiedFromFileDependencies
  - Sets the *Last-Modified* HTTP header to the most recent timestamps of the files upon which the page is dependent.

Methods of HttpCachePolicy

- SetCacheability
  - Sets the *Cache-Control* HTTP header to any of the values taken from the *HttpCacheability* enumeration type.
- SetVaryByCustom
  - Sets the *Vary* HTTP header to the specified text string.

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
Response.Cache.SetCacheability(HttpCacheability.Public);
Response.Cache.VaryByParams["employeeid;lastname"] = true
```

Vary Programmatically

Vary By Header

```
Response.Cache.VaryByHeaders["Accept-Language"] = true;
```

If one wants to programmatically vary the pages in the cache by all HTTP header names, do

```
HttpCacheVaryByHeaders.VaryByUnspecifiedParameters();
```

Vary By Custom

```
Response.Cache.SetVaryByCustom("browser");
```

Caching User Control

- The OutputCache directive can be applied to the user control.
- It is referred as partial caching..
- Both the page and the controls are cached individually.
- Always check cacheable controls are null or not in code, as cache expires & the control may not be available.

Shared Attribute

- Distinct pages don't share the output of the same cacheable user control.
- Each page will maintain its own copy of the user control response instead.
- It may flood the Web server memory with copies and copies of the user control responses.
- To allow distinct pages to share the same output of a common user control, one need to set *Shared* attribute to true

Data Caching

- Caching API lets you store data into a global, system-managed object—the *Cache* object.

- The *Cache* object is a smarter and thread-safe container that can automatically remove unused items, support various forms of dependencies, and optionally provide removal callbacks and priorities.

Cache & Application

- *Cache* is a thread-safe object and does not require you to explicitly lock and unlock before access. *Application* is not thread-safe.
- The *Cache* object lets you associate a duration as well as a priority with an item. Items in *Application* have no duration & retained till we remove them.
- Items in *Cache* can have associated dependencies.
- Application & Cache, Both are application-wide storage.

Properties of Cache

- Count
  - Gets the number of items stored in the cache.
- Item
  - An indexer property that provides access to the cache item identified by the specified key.
- NoAbsoluteExpiration
  - A static constant that indicates a given item will never expire.
- NoSlidingExpiration
  - A static constant that indicates sliding expiration is disabled for a given item.

Methods of Cache

- Insert
  - Inserts the specified item into the cache. It allows you to specify dependencies, expiration and priority policies, and a remove callback. Do not return.
  - Overwrites existing item. Mostly used method
- Remove
  - Removes the specified item from the cache. Returns removed item.
- Add
  - Adds the specified item to the cache. Do not add if item with same key exists. Rarely used method.
  - Returns added item.

Adding Items to Cache

```
Employee emp=new Employee();
emp.Name="Ram";
emp.Age=26
Cache["myemp"]=emp;
```

```
Employee emp=new Employee();
emp.Name="Ram";
emp.Age=26;
Cache.Insert(myemp);
```

Retrieving Item from Cache

```
Employee emp;
if(!(Cache["myemp"]=null)
{
        emp=Cache["myemp"] as Employee;
        Response.Write(emp.Name);
}
Else
{
        Response.Write("The Item does not exist in Cache");
}
```

Replacement for statement using Item property.

```
emp=Cache.Item["myemp"] as Employee;
```

To remove item explicitly.

```
emp=Cache.Remove("myemp");
```

Inserting Items Dependent on File Timestamp

- One can set dependency between the cached item and disk file such that whenever the disk file is modified the cached item is removed.

```
if (Cache["catalog"] == null)
      {
          StreamReader sr = File.OpenText(Server.MapPath("./Catalog.txt"));
          string str = sr.ReadToEnd();
          sr.Close();
          CacheDependency cd =
             new CacheDependency(Server.MapPath("./Catalog.txt"));
          Cache.Insert("catalog", str, cd);
      }
      Label1.Text = Cache["catalog"] as String;
```

Inserting Cache Item Dependent on Another Cached Item

- One can also remove an item from the cache when another cached item changes.

```
if (Cache["mytxt"] == null)
     {
          string[] keys=new string[1];
          keys[0]= "catalog";
          CacheDependency cd = new CacheDependency(null, keys);
          Cache.Insert("mytxt", "Hi", cd);
     }
```

Inserting Items With Expiry

- You can tell Cache object that you want to remove an item on some fixed date.

```
Cache.Insert("mytxt1", "Hello", null, DateTime.Now.AddMinutes(1),
 Cache.NoSlidingExpiration);
```

- You can also specify that you want to remove an item after certain idle time.

```
Cache.Insert("mytxt1", "Hello", null, Cache.NoAbsoluteExpiration,
 new TimeSpan(0,0,1));
```

Inserting Items With Priority

- ASP.NET removes cached items on its own.
- One may hint ASP.NET which items to be removed first than others i.e. One can set priority of cached items.

```
Cache.Insert("mytxt1", "Hello", null, Cache.NoAbsoluteExpiration,
new TimeSpan(12,0,0),CacheItemPriority.High,null);
```

Notification for Removal of Cached Item

- When an item is removed from the cache we can receive a notification by supplying a callback method of type *CacheItemRemovedCallback*.

- The signature of the method has three parameters

  - Key→ string.
  - Value→ Object.
  - Reason→CacheItemRemovedReason.

CacheItemRemovedReason Enum

- DependencyChanged
  - Removed because the associated dependency changed.
- Expired
  - Removed because expired.
- Removed
  - Programmatically removed from the cache using *Remove*.
- UnderUsed
  - Removed by the system to free memory.

Item Removal Callback

```
CacheItemRemovedCallback myCallback =
           new CacheItemRemovedCallback(Notify);
Cache.Insert("mytxt1", "Hello", null, Cache.NoAbsoluteExpiration,
             new TimeSpan(12,0,0),CacheItemPriority.High,myCallback);
```

```
private void Notify(string key, object value, CacheItemRemovedReason reason)
  {
     switch (reason)
     {
        case CacheItemRemovedReason.DependencyChanged:
           ...
           break;
        case CacheItemRemovedReason.Expired:
           ...
           break;
     }
  }
```

Sql Cache Dependency

- aspnet_regsql.exe -S <Server> -U <Username> -P <Password> -ed -d Northwind -et -t Employees

- <caching>

  <sqlCacheDependency enabled = "true" pollTime = "1000" >
  <databases>

   <add name="Northwind"
  connectionStringName="NorthwindConnectionString1"
  pollTime = "1000" />

   </databases>

   </sqlCacheDependency>

   </caching>
- <%@ OutputCache Duration="3600" SqlDependency="Northwind:Employees" VaryByParam="none" %>