Caching in ASP.NET

PRESENTED BY

SHILPA KHURANA

A.P CSE DEPT.

Caching in ASP.NET

- Caching is the most critical factor in creating scalable, high performance web application.
- Caching Locations:

  Web Server, Proxy Server and client Browser.
- Types of Caching:

  -Output Caching

  -Fragment Caching

  -Data Caching

Output Caching

- What is output caching?
- @ OutputCache directive and the cache object
- Output caching attributes:

  -Duration

  -Location

  -VaryByParam

  -VaryByHeader

  -VaryByCustom

What Is Output Caching?

- Pages that use the output cache are executed one time, and the page results are cached
- The pre-executed page is then served to later requests
- Performance and scalability both benefit

  -Server response times reduced

  -CPU load reduced
- Appropriate caching of pages affects site performance dramatically

OutputCache Directive and the Cache Object

- @ OutputCache declaratively controls caching behavior

  For .aspx, .asmx, or .ascx
- The cache object programmatically controls caching behavior

```
<%@ OutputCache Duration="600" Location="Any"
      VaryByParm="none" %>
```

Is equivalent to:

```
[C#]
Response.Cache.SetExpires(DateTime.Now.AddSeconds(600));
Response.Cache.SetCacheability(HttpCacheability.Public);
```

Output Cache Members: Duration and Location

- Duration sets the time to cache the output
  -In seconds Required

```
<%@ OutputCache Duration="600" Location="Any"
      VaryByParam="none" %>
```

- Location sets the location to cache the output
- Server: The output is held in memory on the Web server and is used to satisfy requests
- Downstream: A header is added to the response to indicate to proxy servers to cache the page
- Client: A header is added to the response indicating to browsers to cache the page
- Any: Output cache can be located on any of these locations None: No output caching is turned on for the item

OutputCache Members: VaryByParam and VaryByHeader

- VaryByParam
- The cache stores multiple copies of a page based on specific Querystring or Form parameters and any combinations thereof

```
<%@ OutputCache Duration="10"
    VaryByParam="location;count" %>
```

- VaryByHeader
- The cache stores multiple copies of a page based on HTTP headers

```
<%@ OutputCache Duration="60"
    VaryByHeader="Accept-Language" %>
```

OutputCache Members: VaryByCustom

- VaryByCustom

  -If the value is “Browser,” cache varies by browser type and major version

  -If the value is a custom string, you must override HttpApplication.GetVaryByCustomString in the Global.asax and implement your own caching logic

Fragment Caching

- Just as you can vary the versions of a page that are output cached, you can output cache regions of a page
- Regions are defined based on user controls
- User controls contain their own @OutputCache directive
- Fragment caching supports

  -VaryByParam

  -VaryByControl
- Location not supported because fragments must reside on server to be assembled

Fragment Caching a User Control

```
[*.ascx]
<%@ Language="C#" %>
<%@ OutputCache Duration="10"
  VaryByControl="State;Country"
    VaryByParam="*"%>
<script runat=server>
  public String State {
    get { return state.Value; }
    set { state.Value = State; } }

  public String Country {
    get { return country.Value; }
    set { country.Value = Country; } }
</script>
```

VaryByControl

- VaryByControl

  -The sixth attribute supported by OutputCache

  -Only supported in user control caching

  -Caching is based on user control properties

```
<%@ OutputCache Duration="10"
    VaryByControl="State;Country"
    VaryByParam="*"%>
```

Data Caching

- The data cache holds application data such as strings, datasets, and other objects
- Adding items to the data cache is easy

```
Cache [“counter”] = mycount.text
```

- Although similar to the familiar application variables model, it is much more powerful

```
Application[“counter”] = mycount.text
```

Working with the Cache Object

- Cache object features

  -Dependencies allow logic to invalidate cached items

  -Scavenging (automatic expiration)

  -Callbacks when an item is removed from cache
- To use dependencies or callbacks, use Cache.Insert or Cache.Add
- Code using cached items must be able to both create or insert, and retrieve cached items

```
Public DataSet GetProductData()
{
  if (Cache["ProductData"] = null)
  {
    Cache["ProductData"] = LoadDataSet();
  }
  Return Cache["ProductData"];
}
```

Cache Dependencies

- File-based dependencies
  -Cached item invalidated when files change
- Key-based dependencies
  -Cached item invalided when another cached item changes
- Time-based dependencies
  -Absolute time-based invalidations
  -Sliding time-based invalidations
- SQL dependencies
  -SQL based invalidations