



Lesson Objectives

- In this lesson, you will learn:
- Method to improve the performance of your application by using the application cache
 - Method to avoid unnecessary round-trips to the database
 - Method to manage items in the cache



7.1: Why Caching?



Rationale behind Caching

- Caching is a process of storing frequently used data on the server to fulfill subsequent requests.
- It improves Performance, Scalability, and Availability of a Web Application.

Why Caching?

Performance is the key requirement of any application. One of important technique which helps in the performance of an application is **caching**.

Caching is the process of storing frequently used data on the server to fulfill subsequent requests. Reusing pages from memory is much faster than re-creating pages every time they are requested.

Caching helps us to achieve three important aspects of QoS (Quality Of Service). They are as follows:

- **Performance:** Caching improves application performance by minimizing data retrieval and formatting operations.
- **Scalability:** Since caching minimizes data retrieval and formatting operations, it reduces the load on server resources thus increasing the scalability of the application.
- **Availability:** Since the application uses data from a cache, the application will survive failures in other systems and databases.

Caching is about storing data in memory the first time it is requested and then re-using it for the following requests for a specified period of time.

7.2: Benefits of Caching



Advantages

- Caching provides the following benefits to the user:
 - Faster page rendering
 - Minimization of database hits
 - Minimization of the consumption of server resources

7.3: Types of Caching

Different Methods of Caching

➤ Following are the different types of Caching:

- Output or Page Caching:
 - It involves caching an entire page.
 - It is useful for pages which are frequently accessed.
- Partial Page Caching:
 - It involves caching portions of a Web Page.
 - It is useful for objects and data that are expensive to construct.
- Data Caching:
 - It involves caching data in a Cache Object.
 - The cache object enables you to store everything from simple name / value pairs to more complex objects like datasets and entire aspx pages.
 - It is useful when cache items are to be given an expiration policy.

Types of Caching:

ASP.NET has several types of caching that can be used by Web applications:

- **Output Caching:**

Output caching is useful when the contents of an entire page can be cached.

On a heavily accessed site, caching frequently accessed pages for even a minute at a time can result in substantial **throughput gains**. While a page is cached by the output cache, subsequent requests for that page are served from the output page without executing the code that created it.

- **Partial Page Caching:**

Sometimes it is not practical to cache an entire page - perhaps portions of the page must be created or customized for each request. In this case, it is often worthwhile to identify objects or data that are expensive to construct and are eligible for caching. Once these items are identified, they can be created once and then cached for some period of time.

Additionally, fragment caching can be used to cache regions of a page's output.

7.4: Output (Page) Caching

Concept of Output (Page) Caching

- Dynamically generated webpage content can be kept on the server's memory.
- Output caching can be applied by specifying the Output Directive:
 - `<% OutputCache Duration="60" VaryByParam="None"%>`
- The Duration attribute specifies how long (in seconds), the page will be held in the memory.

Output (Page) Caching:

- Output Caching is a way to keep the dynamically generated page content in the server's memory to fulfill future requests. Output caching is a powerful technique that increases **request/response throughput** by caching the content generated from dynamic pages.
- Output caching is a method to keep dynamically generated page content in the server's memory for later retrieval. On a heavily accessed site, caching frequently accessed pages for even a minute at a time can result in substantial **throughput gains**. While a page is cached by the output cache, subsequent requests for that page are served from the output page without executing the code that created it. To make a response eligible for output caching, it must have a valid **expiration/validation policy** and **public cache visibility**. The output cache can be configured by specifying the Output Directive as follows:
 - `<%@ OutputCache Duration="15" VaryByParam="none" %>`
- When output caching is enabled, an output cache entry is created on the first GET request to the page. Subsequent GET or HEAD requests are served from the output cache entry until the cached request expires. The output cache also supports variations of cached GET or POST name/value pairs.
- The output cache respects the **expiration** and **validation policies** for pages. Suppose a page is in the output cache and has been marked with an expiration policy that indicates that the page expires 60 minutes from the time it is cached. Then the page is removed from the output cache after 60 minutes. If another request is received after that time, the page code is executed and the page can be cached again. This type of

expiration policy is called **absolute expiration** - that is a page is valid until a certain time.

7.5: Attributes of Output Caching

Attributes

➤ VaryByParam:

- The VaryByParam attribute can specify which QueryString parameters cause a new version of the page to be cached.

```
<%@ OutputCache Duration="60" VaryByParam =
"ProductID;CategoryID" %>
```

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

```
<%OutputCache VaryByParam=" txtfirstname" %>
```

Attributes of Output Caching:

Let us discuss the attributes of output caching one by one:

VaryByParam:

- The VaryByParam attribute determines which versions of the page output are cached. One can generate different responses based on whether an HTTP-POST or HTTP-GET response is required. The VaryByParam attribute can specify which QueryString parameters cause a new version of the page to be cached.

```
<%@ OutputCache Duration="60" VaryByParam =
"ProductID;CategoryID" %>
```

- For example:** Suppose you have a page called **products.aspx** that includes products and category information in the QueryString such as **productID** and **CategoryID**. The page directive, shown above caches the products page for every different value of CategoryID and ProductID.
- If you want to cache a new version of the page based on any combination of parameters, you can use VaryByParam = "*", as shown below:

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

- If the cache has to be modified based on firstname value in the query string and the text box name is txtfirstname then you can specify as follows:

```
<%OutputCache VaryByParam=" txtfirstname" %>
```


7.5: Attributes of Output Caching

Attributes

➤ VaryByControl:

- It is used to get good performance from complex user controls that render a lot of HTML that does not change often.

```
<%@ OutputCache Duration="1296000"  
    VaryByControl="drpdownCountries" %>
```

➤ VaryByCustom:

- To make the Cache even more flexible, VaryByCustom provides the ability to cache a page based on a string. It is then controlled from the actual code of the application.

Attributes of Output Caching (contd.):

VaryByControl:

- **VaryByControl** can be an easy way to get good performance from complex user controls that render a lot of HTML that does not change often.
- **For example:** Imagine a User Control that renders a Combo Box showing the names of all the countries in the world. Certainly, the names of countries do not change that often. So you can set cache based on this user control where **duration** attribute can be set to say one fortnight.

```
<%@ OutputCache Duration="1296000"  
    VaryByControl="drpdownCountries" %>
```

7.5: Demo



Demo on Output Caching

➤ Understanding Output Caching



7.6: Partial Page Caching



Concept of Partial Page Caching

- Partial Page caching allows to cache certain blocks of a website.
- It is achieved with the use of user controls.
- Use "shared" attribute in the output cache directive to share user control's output among multiple pages.

Partial Page Caching:

- Partial page caching allows you to cache certain blocks of your website. You can, for example, only cache the center of the page.
- Partial page caching is achieved with the caching of user controls. You can build your ASP.NET pages consisting of numerous user controls and then apply output caching on the user controls you select. This will cache only parts of the page that you want, leaving other parts of the page outside the reach of caching.
- Typically, **UserControls** are placed on multiple pages to maximize reuse. However, when these **UserControls** (ASCX files) are cached with the **@OutputCache** directive, they are cached on a per page basis. That means, even if a User Control outputs identical HTML when placed on *pageA.aspx*, as it does when placed on *pageB.aspx*, its output is cached twice. You can prevent this from happening by adding **Shared = true** in the output cache directive.

```
<%@ OutputCache Duration="500" VaryByParam="*"
Shared="true" %>
```

- By putting the **Shared** attributed, the memory savings can be surprisingly large.

7.7: Post Cache Substitution



Concept of Post Cache Substitution

- Use Post Cache Substitution when you want the benefits of output caching but also want to add some dynamic content on the page.
- Two ways of using Post Cache Substitution are as follows:
 - `Response.WriteSubstitution` Method
 - `<asp:Substitution>` Control

Post Cache Substitution:

- **Output Caching** enables you to cache entire page. However often you want the benefits of output caching, but you also want to add a small bit of dynamic content on the page. ASP.NET 2.0 introduced **Post-Cache substitution** as an opportunity to make changes to about-to-be-rendered page.
- For this to work, a control is added to page that acts as a place holder. It calls a method that you specify after the cached content has been returned. This method returns any string output.
- There are two ways in which you can control the Post-cache substitution:
 - Call the new **Response.WriteSubstitution** method, by passing it a reference to the desired substitution method callback.
 - Add a `<asp:Substitution>` control to the page at the desired location and set its **methodName** attribute to the name of callback method.

7.7: Demo



Demo on Post Cache Substitution

➤ Understanding Post cache Substitution



7.6: Data Caching



Concept of Data Caching

➤ Data Caching allows to store data, which requires extensive server resources to be created.

- Data Caching is done programmatically.
- It can store from simple items to complex objects like dataset.
- Accessing the cache Object is same as accessing the Session Object.
- Examples:

```
Cache["dataset"] = mydataset;
```

```
DataSet ds = new DataSet();  
ds = (DataSet)Cache["dataset"];
```

Data Caching:

- With **Data Caching**, you can store data that requires extensive server resources to be created. By caching this data, you are creating it once and then using it many times without recreating that data again. This can improve your application performance significantly.
- Output Caching is done almost **declaratively**. However, you can perform data caching programmatically, as well. You can use cache object to start caching specific data items for later use on a particular page. The **cache object** enables you to store everything from simple name/value pairs to more complex objects like **datasets** and entire **aspx pages**.
- Using the **data cache** in the simplest and most naïve way supported by ASP.NET is very much like accessing the **Session object**. Accessing the **Session object** involves using an **indexer** (the square brace syntax) and a consistent **index** to store and retrieve data. The data cache works in exactly the same way. Although it has some other features for managing items in the cache.

```
Cache["dataset"] = mydataset;
```

- After an item is in cache, you can retrieve it later as shown below:

```
DataSet ds = new DataSet();  
ds = (DataSet)Cache["dataset"];
```

- Using **cache object** is an outstanding way to cache your pages. The

fragments shown above illustrate a simple use of the Cache object.

7.9: Strategy of Data Caching



Stepwise Segregation

- The strategy for caching a piece of data usually involves the following steps:
1. Look in the cache for the data element.
 2. If it is there, use it (bypassing the expensive database round-trip).
 3. If the data is unavailable in the cache, make a round-trip to the database to fetch it.
 4. Cache the data so it is available next time around.

Strategy for Data Caching:

Example:

```
public static IEnumerable<T> Take<T>(  
    this IEnumerable<T> source,  
    int count);
```

```
public static IEnumerable<T> TakeWhile<T>(  
    this IEnumerable<T> source,  
    Func<T, bool> predicate);  
public static IEnumerable<T> TakeWhile<T>(  
    this IEnumerable<T> source,  
    Func<T, int, bool> predicate);
```


7.10: Managing the Cache



Steps Involved

➤ Following steps are involved while managing the cache:

- Setting up an absolute expiration time
- Setting up a sliding expiration time
- Setting up dependencies between cached items and their backing sources (for example: database, file, directory dependencies, or dependencies upon other cache entries)
- Managing a relative priority of cached items
- Setting up callback functions to be called when items are removed

Managing the Cache:

- The last slides about Data Caching explained how to place items in the cache and to give an index. However, at times you may need a bit more control over the items in the cache.
- **For example:** What if the physical source backing one of the items you cache changes? If getting accurate information out to your users is important, you may want to know about the change so that you can handle it (perhaps by reloading the new information into the cache). As another example, what if you knew that the data in your cache would become invalid after a certain period of time, or on a certain date? You would want to ensure that the data in the cache is invalidated and appropriately refreshed with new data.
- In addition to placing items in the cache using the **indexer**, the Cache object implements a parameterized method named **Insert** that allows you control over many aspects of the **cached item**.
- The above slide shows the various ways in which you may control cache entries.

7.11: Managing the Cache using Insert Method



Different Overloads in Inert Method

➤ Overloads of a Insert Method:

- Insert (String, Object)
- Insert (String, Object, CacheDependency)
- Insert (String, Object, CacheDependency, DateTime, TimeSpan)
- Insert (String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, CacheItemRemovedCallback)

Managing the cache using Insert method:

The Cache's insert method includes four overloads, as shown below:

- **Insert (String, Object)**
It directly corresponds to the indexer version. It blindly places the object in the Cache using the string key in the first parameter.
- **Insert (String, Object, CacheDependency)**
It inserts an object into the Cache and associates it with a dependency.
- **Insert (String, Object, CacheDependency, DateTime, TimeSpan)**
It inserts an object into the Cache, associating it with a dependency and an expiration policy.
- **Insert (String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, CacheItemRemovedCallback)**
It inserts an object into the Cache. It associates a dependency and expiration and priority policies. It also associates the Cache entry with a delegate for a callback to notify the application when the item is removed from the cache.

7.11: Managing the Cache

Demo on Data Caching

➤ Understanding Data Caching



7.12: Cache Dependencies



Use of Cache Dependencies

- Cache dependencies allow the validity of a cache item to be dependent on an external file or on another cache item.
- If a dependency changes, then the cache item is invalidated and removed from the cache.

Cache Dependencies:

- In addition to allowing objects in the cache to expire by duration, you may set up dependencies for the cached items.
- For example, imagine that our program loads some data from a file and places it into the cache. The backing file (that is, the source of the cached information) may change, making the data in the cache invalid. ASP.NET supports setting up a **dependency** between the **cached item** and the **file** so that changing the file invalidates the cached item.
- The conditions under which the cached items may be flushed include the following scenarios:
 - when a file changes,
 - when a directory changes,
 - another cache entry is removed, or
 - data in a table in an SQL Server changes (this is an often requested feature finally available in ASP.NET 2.0!)

Example:

- An application reads financial information from an XML file that is periodically updated. The application processes the data in the file and creates a graph of objects that represent that data in a consumable format.
- The application caches that data and inserts a dependency on the file from which the data was read.
- When the file is updated, the data is removed from the cache and the application can reread it and reinsert the updated copy of the data.

7.13: SQL Server based Cache Dependency

Use of SQL Server based Cache Dependency

- It allows invalidating cache whenever underlying data in the database changes.
- It performs a one-time setup of your SQL Server database.
 - Use aspnet_regsql.exe tool from the command prompt
- Example:

```
aspnet_regsql.exe -S atrgsql -U sa -P admin@123 -d  
Northwind -ed
```

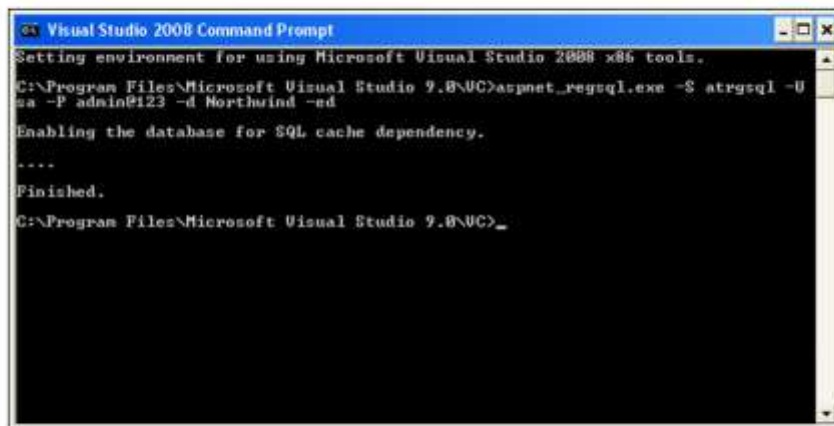
SQL Server based Cache Dependency:

- SQL Server Cache Dependency allows invalidating cache whenever underlying data in the database changes. No stale data is stored in cache.
- To utilize the new SQL Server Cache dependency feature, one must perform a one-time setup of your SQL Server database. We have to use **aspnet_regsql.exe** tool to enable the database for this new **invalidation feature**.
- We need to pass following parameters to aspnet_regsql.exe:

```
aspnet_regsql.exe -S atrgsql -U sa -P admin@123 -d Northwind -ed
```

7.13: SQL Server based Cache Dependency

Use of SQL Server based Cache Dependency



```
Visual Studio 2008 Command Prompt
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
C:\Program Files\Microsoft Visual Studio 9.0\VC>aspnet_regsql.exe -S atgsql -U
sa -P admin@123 -d Northwind -ed
Enabling the database for SQL cache dependency.
....
Finished.
C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

SQL Server based Cache Dependency (contd.):

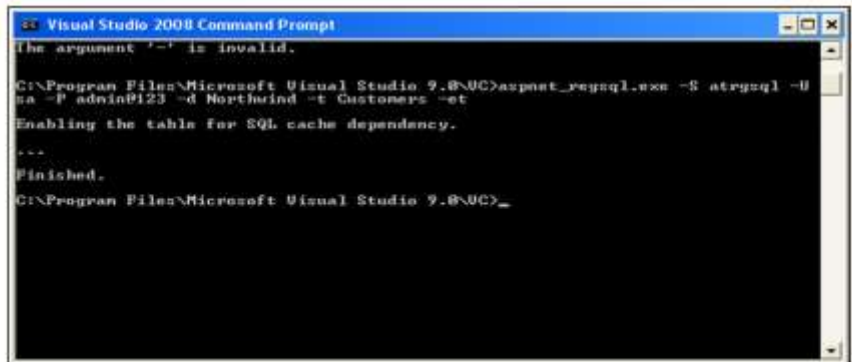
- In the above slide, you can see that we simply enabled the Northwind database for SQL cache invalidation. The name of the SQL machine was passed in with **-S**, the username with **-U**, the password with **-P** the database with **-d**, and most importantly, the command to enable SQL cache invalidation was **-ed**.
- Now that you have enabled the database for SQL cache invalidation, you can enable one or more tables contained within the Northwind database.

7.13: SQL Server based Cache Dependency

Use of SQL Server based Cache Dependency

➤ Enabling Tables:

```
aspnet_regsql.exe -S atrgsql -U sa -P admin@123 -d  
Northwind -t Customers -et
```



```
Visual Studio 2008 Command Prompt  
The argument '-' is invalid.  
C:\Program Files\Microsoft Visual Studio 9.0\VC>aspnet_regsql.exe -S atrgsql -U  
sa -P admin@123 -d Northwind -t Customers -et  
Enabling the table for SQL cache dependency.  
...  
Finished.  
C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

SQL Server based Cache Dependency (contd.):

- You enable one or more tables by using the following command:

```
aspnet_regsql.exe -S atrgsql -U sa -P admin@123 -d  
Northwind -t Customers -et
```

- The above slide shows how to enable **Customers** table from the command prompt. You can see that this command is not much different from the one for enabling database, except for the extra **-t customers** entry and the use of **-et** to enable the table rather than **-ed** to enable a database.
- After the table is enabled, you can begin using the SQL Cache invalidation feature.

7.13: SQL Server based Cache Dependency



Use of SQL Server based Cache Dependency

```
<connectionStrings>
  <add name="NorthwindConnectionString" connectionString="Data
Source=atrgsql;Initial Catalog=Northwind;User ID=sa;Password
= admin@123;" providerName="System.Data.SqlClient" />
</connectionStrings>
<system.web>
  <caching>
    <sqlCacheDependency enabled="true">
      <databases>
        <add name="Northwind" connectionString Name
="NorthwindConnectionString" pollTime="500"/>
      </databases>
    </sqlCacheDependency>
  </caching>
```

SQL Server based Cache Dependency (contd.):

- From the above slide, you can see that connection string information is utilized later in the configuration settings for SQL Server cache invalidation.
- The SQL Server cache invalidation is configured using the new **<caching>** element. This element must be nested within the **<system.web>** elements.
- Since you are working with SQL Server cache dependency, you must use a **<sqlCacheDependency>** child node. You can enable the entire process by using the `enabled=true` attribute.

7.13: SQL Server based Cache Dependency

Demo on SQL Server based Cache Dependency

➤ Understanding SQL Server Cache Dependency



7.14: Application Object and Caching



A Comparison

- The application object is global. Hence it should be used very carefully.
- The caching object is more flexible and helps you control an object's lifetime.
- Caching increases your application's performance, scalability, availability.
- If memory is scarce, then cache can be invalidated

Application Object and Caching:

- The main difference between the **application object** and **cache** is that the cache object has some more powerful features that allows you to control the cached data. With Cached object, each of the data item has its **priority state** and **expiration time**.
- This object has two important issues handling.
 - When your system's memory becomes short, the cache object knows to remove data items with low priority and free its memory. By doing so, the cache ensures that unnecessary items do not consume valuable server resources.
 - One more good advantage (in ASP.NET, of course) is to cache the pages' data. ASP.NET allows you to cache web pages or portions of them.
- Hence for more **complex data manipulation caching**, the recommended way is to use the **cache object** and **API**. However, if you want to use and cache some "dummy" data (or just data that not has to be modified and managed during the application), use the application object.

7.15: Caching: Best Practices and Guidelines



Best Practices

- Use Output caching to cache the entire page for a specified duration, when a static result of a rendered page needs to be cached.
- Use cache API when data needs to be manipulated or to programmatically cache application-wide data that is shared and accessed by multiple users.
 - Avoid using the cache API in the following cases:
 - The data being cached is user-specific. Consider using session state instead.
 - The data is updated in real time.

7.15: Caching: Best Practices and Guidelines



Best Practices

- Do not cache too many items. Avoid caching items that can be easily re-computed or any item that will rarely be used again.
- Do not put items with fast expiration.
 - Caches work best if items can live there for long periods of time.
 - Fast expiring items often cause high churn rate of the cache, providing little benefit and often more work for the cache cleanup code.

7.15: Caching: Best Practices and Guidelines



Best Practices

- Separate dynamic data from static data in pages. Use user controls to partition the page.
 - By partitioning the content, selected portions of the page can be cached to reduce processing and rendering time.
- Cache static data and dynamic data that is expensive to create or retrieve.
 - If the data is used more often than it is updated, it is also a candidate for caching.

7.15: Caching: Best Practices and Guidelines



Best Practices

- Do not cache expensive resources that are shared, such as database connections because it creates contention..
- Do not cache per-user data that spans requests — use session state for that.
- Avoid storing DataReader objects in the cache because these objects keep the underlying connections open.

Summary

➤ In this lesson, you have learnt:

- The methods to improve the performance of your application by using the application cache.
- The methods to avoid unnecessary round-trips to the database.
- The methods to manage items in the cache.



Review Questions

➤ Question 1: ____ is not a caching option.

- Data Caching
- Page Caching
- Fragment Caching
- Web Caching

➤ Question 2: Which of the following parameter of a `OutputCache` Directive can be used with a `UserControl`?

- `VaryByParam`
- `VaryByUser`
- `VaryByControl`
- `VaryByCustom`



Review Questions

➤ Question 3: Arrange the following steps in a sequence to define the strategy adopted by Data Caching:

- Cache the data so it's available next time around
- If it's there, use it (bypassing the expensive database round-trip)
- If the data is unavailable in the cache, make a round-trip to the database to fetch it
- Look in the cache for the data element

