Security Fundamentals

Today you will learn

- Caching
- Caching guidelines and benefits
- Types of Caching

Caching

- Caching is often used to store information that's retrieved from a database
- With caching there can be reduction in time and, lessen the burden imposed on the database to certain extent
- With System using caching, some data requests wont require a database connection and query, those information can be retrieved directly from the server memory, which is much faster proposition
- Storing information in memory in not a good idea as server memory is limited resource

Caching

- Storing too much information will end up in paging some information to disk, potentially slowing down the entire system
- Caching is self-limiting.
- Lifetime of the information is at the discretion of the server
- If cache becomes full or other applications consume a large amount of memory, data will be selectively evicted from the cache, ensuring that the application continues to perform well

Caching Guidelines

Here are two caching guidelines to keep you on the right track:

Cache data (or web pages) that are expensive: In other words, cache information that's time-consuming to create. The results of a database query or contents of a file are good examples. Not only does it take time to open a database connection or a file, but it can also delay or lock out other users who are trying to do the same thing at the same time.

Cache data (or web pages) that are used frequently: There's no point setting aside memory for information that's never going to be needed again. For example, you might choose not to cache product detail pages because there are hundreds of different products, each with its own page. But it makes more sense to cache the list of product categories because that information will be reused to serve many different requests.

Benefits of Caching

Performance is a measure of how quickly a web page works for a single user. Caching improves performance because it bypasses bottlenecks like the database. As a result, web pages are processed and sent back to the client more quickly.

Scalability measures how the performance of your web application degrades as more and more people use it at the same time. Caching improves scalability because it allows you to reuse the same information for requests that happen in quick succession. With caching, more and more people can use your website, but the number of trips to the database won't change very much. Therefore, the overall burden on the system will stay relatively constant, as shown in Figure 23-1.

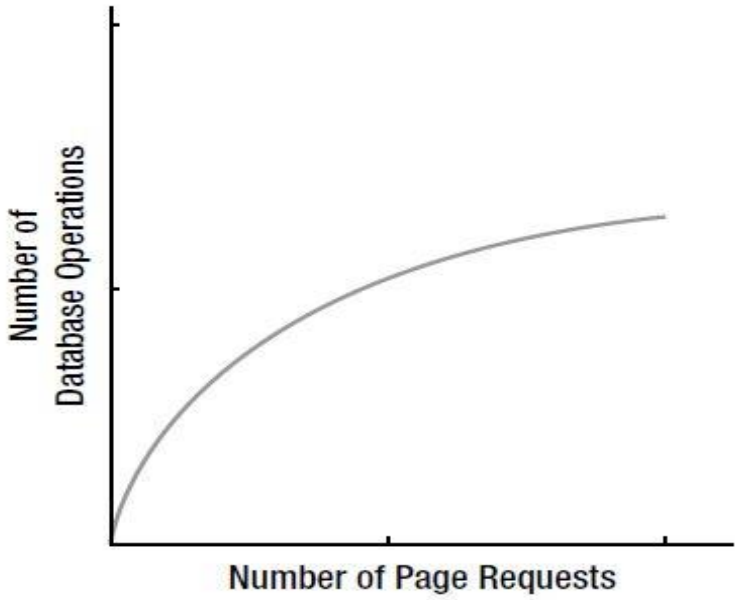

Figure 23-1. *The effect of good caching*

Types of Caching

ASP.NET really has two types of caching. Your applications can and should use both types because they complement each other:

- *Output caching*: This is the simplest type of caching. It stores a copy of the final rendered HTML page that is sent to the client. The next client that submits a request for this page doesn't actually run the page. Instead, the final HTML output is sent automatically. The time that would have been required to run the page and its code is completely reclaimed.
- *Data caching*: This is carried out manually in your code. To use data caching, you store important pieces of information that are time-consuming to reconstruct (such as a DataSet retrieved from a database) in the cache. Other pages can check for the existence of this information and use it, thereby bypassing the steps ordinarily required to retrieve it. Data caching is conceptually the same as using application state, but it's much more server-friendly because items will be removed from the cache automatically when it grows too large and performance could be affected. Items can also be set to expire automatically.

Specialized types of Caching

Also, two specialized types of caching build on these models:

- *Fragment caching*: This is a specialized type of output caching—instead of caching the HTML for the whole page, it allows you to cache the HTML for a portion of it. Fragment caching works by storing the rendered HTML output of a user control on a page. The next time the page is executed, the same page events fire (and so your page code will still run), but the code for the appropriate user control isn't executed.
- *Data source caching*: This is the caching that's built into the data source controls, including the SqlDataSource and ObjectDataSource. Technically, data source caching uses data caching. The difference is that you don't need to handle the process explicitly. Instead, you simply configure the appropriate properties, and the data source control manages the caching storage and retrieval.

Output Caching

- Final rendered HTML of the page is cached
- When the same page is requested again, the page life cycle doesn't start and code is not executed
- Instead, the cached HTML is served.
- Output caching gets the theoretical maximum performance increase because all the overhead of the code is sidestepped

Output Caching

- The most common approach to cache an ASP .NET page is to insert the OutputCache directive at the top of .aspx file, just below the Page directive

<%@OutputCache Duration="20" VaryByName="None" %>

```
public partial class OutputCaching : System.Web.UI.Page
{
    protected void Page_Load(Object sender, EventArgs e)
    {
        lblDate.Text = "The time is now:<br />";
        lblDate.Text += DateTime.Now.ToString();
    }
}
```

Output Caching

<%@OutputCache Duration="20" VaryByName="None" %>

- The Duration attribute instruct ASP .NET to cache the page for 20 seconds

Output Caching

- Current time is displayed when the page is accessed for the first time. If a page is refreshed within a short time the page will not be updated with most recent time

- Instead, ASP .NET will automatically send the cached HTML output, until it expires in 20 seconds

- If ASP .NET receives a request after the cached page has expired, ASP .NET will run the page code again, generate a new cached copy of the HTML output, and use that for next 20 seconds.

Fragment Caching

In some cases, you may find that you can't cache an entire page, but you would still like to cache a portion that is expensive to create and doesn't vary frequently (like a list of categories in a product catalog). One way to implement this sort of scenario is to use data caching to store just the underlying information used for the page. You'll examine this technique in the next section. Another option is to use fragment caching.

To implement fragment caching, you need to create a user control for the portion of the page you want to cache. You can then add the OutputCache directive to the user control. The result is that the page will not be cached, but the user control will.

Fragment caching is conceptually the same as page caching. It has only one catch: if your page retrieves a cached version of a user control, it cannot interact with it in code. For example, if your user control provides properties, your web page code cannot modify or access these properties. When the cached version of the user control is used, a block of HTML is simply inserted into the page. The corresponding user control object is not available. To avoid potential problems in your code, don't attempt to interact with it in your code or check that it's not a null reference before you do.

Data Caching

- Data Caching is the most flexible type of caching
- It forces to take specific additional steps to implement in code
- The basic principle of data caching is to add the items that are expensive to create to a built-in collection object called Cache.
- Cache is a property of the Page class, and returns an instance of the System.Web.Caching.Cache class.
- It's globally available to all request from all clients in the application, but it has three key difference.

Key Difference between Caching and Application Object

- Three key differences

The Cache object is thread-safe: This means you don't need to explicitly lock or unlock the Cache object before adding or removing an item. However, the objects in the Cache object will still need to be thread-safe themselves. For example, if you create a custom business object, more than one client could try to use that object at once, which could lead to invalid data. You can code around this limitation in various ways; one easy approach that you'll see in this chapter is to just make a duplicate copy of the object if you need to work with it in a web page.

Items in the Cache object are removed automatically: ASP.NET will remove an item if it expires, if one of the objects or files it depends on changes, or if the server becomes low on memory. This means you can freely use the cache without worrying about wasting valuable server memory because ASP.NET will remove items as needed. But because items in the cache can be removed, you always need to check whether a cache object exists before you attempt to use it. Otherwise, you could generate a null reference exception.

Key Difference between Caching and Application Object

Items in the cache support dependencies: You can link a cached object to a file, a database table, or another type of resource. If this resource changes, your cached object is automatically deemed invalid and released.

END OF LECTURE