**ASP.NET Caching Overview**

Often an application can increase performance by storing in memory data that is accessed frequently and that requires significant processing time to create. For example, if your application processes large amounts of data using complex logic and then returns the data as a report accessed frequently by users, it is efficient to avoid recreating the report every time a user requests it. Similarly, if your application includes a page that processes complex data but that is updated only infrequently, it is inefficient for the server to recreate that page on every request.

To help you increase application performance in these situations, ASP.NET provides caching using two basic caching mechanisms. The first is application caching, which allows you to cache data you generate, such as a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) or a custom report business object. The second is page output caching, which saves the output of page processing and reuses the output instead of re-processing the page when a user requests the page again.

http://i.msdn.microsoft.com/Global/Images/clear.gif Application Cache

The application cache provides a programmatic way for you to store arbitrary data in memory using key/value pairs. Using the application cache is similar to using application state. However, unlike application state, the data in the application cache is volatile, meaning it is not stored in memory for the life of the application. The advantage of using the application cache is that ASP.NET manages the cache and removes items when they expire or become invalidated, or when memory runs low. You can also configure application caching to notify your application when an item is removed. For more information see [Caching Application Data](http://msdn.microsoft.com/en-us/library/6hbbsfk6.aspx).

The pattern when using the application cache is to determine whether an item exists in the cache any time you access an item, and if it does, to use it. If the item does not exist, you can recreate the item and then place it back in the cache. This pattern ensures that you always have the latest data in the cache.

For more information see [How to: Retrieve Values of Cached Items](http://msdn.microsoft.com/en-us/library/xhy3h9f9.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Page Output Cache

The page output cache stores the contents of a processed ASP.NET page in memory. This allows ASP.NET to send a page response to a client without going through the page processing lifecycle again. Page output caching is especially useful for pages that do not change often but require significant processing to create. For example, if you are creating a high-traffic Web page to display data that is not frequently updated, page output caching can dramatically increase the performance of that page. Page caching can be configured individually for each page, or you can create cache profiles in the Web.config file, which allow you to define caching settings once and then use those settings with multiple pages.

Page output caching provides two models for page caching: full page caching and partial page caching. Full page caching allows the entire contents of a page to be persisted to memory and used to fulfill client requests. Partial page caching allows parts of a page to be cached and other parts to be dynamic. For more information see [Caching ASP.NET Pages](http://msdn.microsoft.com/en-us/library/06bh14hk.aspx).

Partial page caching can work in two ways: control caching and post-cache substitution. Control caching, also sometimes referred to as fragment caching, allows you to cache parts of the page output by including the information in a user control and then marking the user control as cacheable. This allows specific content within a page to be cached, while the overall page is not cached and therefore recreated each time. For example, if you create a page that displays largely dynamic content, such as stock information, but has sections that are static, such as weekly summaries, you can place the static sections in user controls and allow them to be cached.

Post-cache substitution is the opposite. The page as a whole is cached, but fragments within the page are dynamic. For example, if you create a page that is static for set periods of time, you can set the entire page to be cached. If you added a [Label](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.label.aspx) control to the page that displayed the user's name, the [Label](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.label.aspx) would stay the same for each page refresh and each user, showing the name of the user who requested that page before it was cached. However, with post-cache substitution, you can configure the page to be cached, but mark individual sections of the page as not cacheable. In this case, you could add your [Label](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.label.aspx) controls to a non-cacheable section and they would be dynamically created for each user and page request. For more information, see [Caching Portions of an ASP.NET Page](http://msdn.microsoft.com/en-us/library/h30h475z.aspx).

**Caching Pages Based on Request Parameters**

In addition to caching a single version of a page, ASP.NET page output caching provides features to create multiple versions of the page that vary by different request parameters. For more information, see [Caching Multiple Versions of a Page](http://msdn.microsoft.com/en-us/library/xadzbzd6.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Automatic Data Removal

ASP.NET can remove data from the cache for one of these reasons:

* Because memory on the server is low, a process known as scavenging.
* Because the item in the cache has expired.
* Because the item's dependency changes.

To help you manage cached items, ASP.NET can notify your application when items are removed from the cache.

**Scavenging**

Scavenging is the process of deleting items from the cache when memory is scarce. Items are removed when they have not been accessed in some time or when items are marked as low priority when added to the cache. ASP.NET uses the [CacheItemPriority](http://msdn.microsoft.com/en-us/library/system.web.caching.cacheitempriority.aspx) object to determine which items to scavenge first. For more information see [How to: Add Items to the Cache](http://msdn.microsoft.com/en-us/library/18c1wd61.aspx).

**Expiration**

In addition to scavenging, ASP.NET automatically removes items from the cache when they expire. When adding an item to the cache, you can set it to expire as described in the following table.

|  |  |
| --- | --- |
| **Expiration Type** | **Description** |
| Sliding expiration | Specifies how long after an item was last accessed that it expires. For example, you can set an item to expire 20 minutes after it was last accessed in the cache. |
| Absolute expiration | Specifies that an item expires at a set time, regardless of how often it is accessed. For example, you can set an item to expire at 6:00 PM or after four hours. |

**Dependencies**

You can configure an item's lifetime in the cache be dependent on other application elements such as files or databases. When the element that a cache item depends on changes, ASP.NET removes the item from the cache. For example, if your Web site displays a report that the application creates from an XML file, you can place the report in the cache and configure it to have a dependency on the XML file. When the XML file changes, ASP.NET removes the report from the cache. When your code requests the report, the code first determines whether the report is in the cache, and if not, your code can recreate it. Therefore, an up-to-date version of the report is always available.

ASP.NET caching supports the dependencies described in the following table.

|  |  |
| --- | --- |
| **Dependency** | **Description** |
| Key dependency | Items in the application cache are stored in key/value pairs. Key dependency allows an item to be dependent on the key of another item in the application cache. When the original item is removed, the item that has the key dependency is also removed. For example, you could add a cache item named ReportsValid, and then cache several reports that are dependent on the ReportsValid key. When the ReportsValid item is removed, all the dependent cached reports are likewise removed from the cache. |
| File dependency | An item in the cache is dependent on an external file. If the file is modified or deleted, the cached item is removed. |
| SQL dependency | An item in the cache is dependent on changes in a table in a Microsoft SQL Server 2005, SQL Server 2000, or SQL Server 7.0 database. For SQL Server 2005, an item can be dependent on a row in a table. For more information, see [Caching in ASP.NET with the SqlCacheDependency Class](http://msdn.microsoft.com/en-us/library/ms178604.aspx). |
| Aggregate dependency | An item in the cache is dependent on multiple elements through the use of the [AggregateCacheDependency](http://msdn.microsoft.com/en-us/library/system.web.caching.aggregatecachedependency.aspx) class. If any of the dependencies change, the item is removed from the cache. |
| Custom dependency | An item in the cache is configured with a dependency that you create in your own code. For example, you can create a custom Web service cache dependency that removes data from the cache when a call to a Web service results in a particular value. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Application Cache Item Removal Notification

You can be notified when an item is removed from the application cache. For example, if you have an item that takes considerable amount of processing time to create, you can be notified when it is removed from the cache so that you can replace it immediately. As a result, the next time the item is requested, the user does not have to wait for it to be processed. For more information, see [How to: Notify an Application When an Item Is Removed from the Cache](http://msdn.microsoft.com/en-us/library/7kxdx246.aspx).