Configuring IIS

As our web application's hosting environment, IIS has some influence on its overall performance, for example, the smaller the IIS pipeline is, less code will be executed on each request. There are mechanisms in IIS which can be used to increase our application's performance regarding latency and throughput, as well as some mechanisms which, when tuned properly, can improve the overall performance of our application.

Output Caching

We already saw that ASP.NET provides its own mechanism for output caching, so why does IIS need yet another mechanism for output caching? The answer is quite simple: there are other content types we want to cache, not only ASP.NET pages. For example, we may want to cache a set of static image files that are frequently requested, or the output of a custom HTTP handler. For that purpose, we can use the output cache provided by IIS.

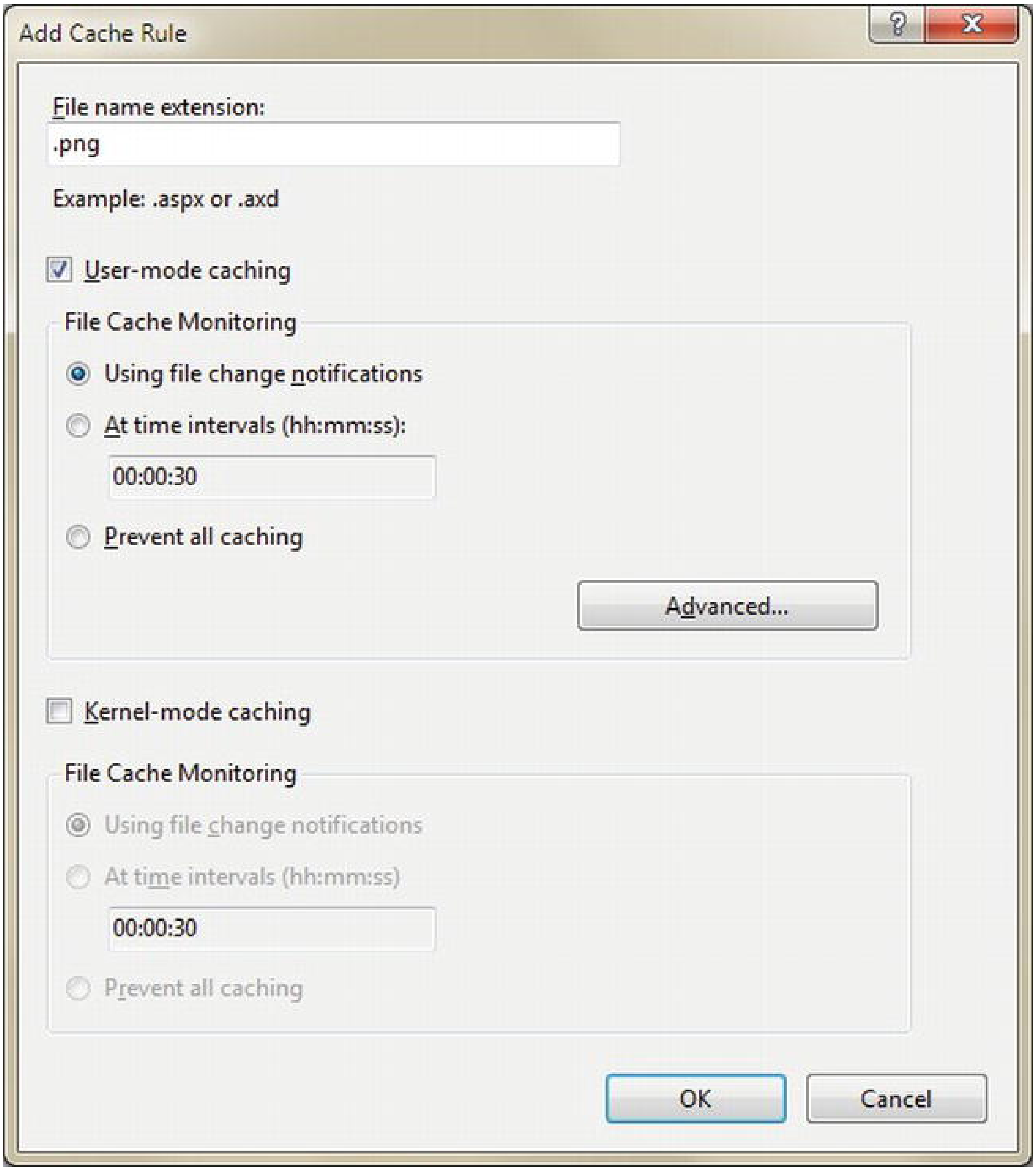
IIS has two types of output cache mechanisms: user-mode cache and kernel-mode cache.

User-Mode Cache

Just like ASP.NET, IIS is capable of caching responses in-memory, so subsequent requests are answered automatically from memory without accessing static files from disk, or invoking server-side code.

To configure the output cache for your web application, open the IIS Manager application, select your web application, open the *Output Caching* feature. Once opened, click the *Add. . .* link from the *Actions* pane to add a new cache rule, or select an existing rule to edit it.

To create a new user-mode cache rule, add a new rule, type the file name extension you wish to cache, and check the *User-mode caching* checkbox in the *Add Cache Rule* dialog, as shown in Figure 11-2.



Add Cache Rule

File name extension: .png

Example: .aspx or .axd

[✓] User-mode caching

File Cache Monitoring

[✓ Using file change notifications]

At time intervals (hh:mm:ss): 00:00:30

Prevent all caching

Advanced...

[ ] Kernel-mode caching

File Cache Monitoring

[✓ Using file change notifications]

At time intervals (hh:mm:ss) 00:00:30

Prevent all caching

OK Cancel

Figure 11-2. The Add Cache Rule dialog

Once you checked the checkbox, you can select when the cached item will be removed from memory, such as after the file has been updated or after some time has passed since the content was first cached. File changes is more suitable for static files, whereas time interval is more suitable for dynamic content. By pressing the *Advanced* button you can also control how the cache will store different versions of the output (options are according to query string or HTTP headers).

Once you add a caching rule, its configuration will be stored in your application's web.config file, under the `system.webServer` ➤ `caching` section. For example, setting the rule to cache .aspx pages, causes them to expire after 30 minutes, and varies the output by the Accept-Language HTTP header will generate the following configuration:

```
<system.webServer>
  <caching>
  <profiles>
  <add extension = ".aspx" policy = "CacheForTimePeriod" kernelCachePolicy = "DontCache"
  duration = "00:00:30" varyByHeaders = "Accept-Language" />
  </profiles>
  </caching>
</system.webServer>
```

Kernel-Mode Cache

Unlike user-mode cache, which stores the cached content in the IIS's worker process memory, kernel-mode caching stored cached content in the HTTP.sys kernel-mode driver. Using kernel-mode caching provides faster response time, however it is not always supported. For example, kernel-mode caching cannot be used when the request contains a query string, or when the request is not an anonymous request.

Setting up a cache rule for kernel-mode is done in a similar manner to user-mode caching. In the rules dialog, check the *Kernel-mode caching* checkbox, and then select the caching monitoring setting you wish to use.

You can use both kernel-mode and user-mode caching in the same rule. When both modes are used, kernel-mode caching is attempted first. If not successful, for example, when the request contains a query string, the user-mode caching is applied.

Tip When using a time interval for monitoring with both kernel-mode and user-mode set, make sure the time interval is identical in both settings, otherwise the interval for kernel-mode will be used for both.

Application Pool Configuration

Application pools control how IIS creates and maintains the worker processes which eventually host our code. When you install IIS and ASP.NET several application pools are created, according to the .NET framework versions installed on the web server, to which you can add new pools as you install more web applications on the server. When an application pool is created, it has some default settings which control its behavior. For example, every application pool, when created, has a default setting for the idle timeout after which the application pool shuts down.

Understanding the meaning of some of these settings can help you configure the way the application pool works, so it will serve your application's needs more precisely.

Recycling

By changing the recycling settings you can control when the application pool restarts the worker process. For example, you can set that the worker process be recycled every couple of hours, or when it exceeds a certain amount of memory. If your web application consumes a lot of memory over time (for example due to stored objects), increasing the number of recycles can help maintain its overall performance. On the other hand, if your web application performs normally, reducing the number of recycles will prevent loss of state information.

Tip You can use the `ASP.NET\Worker Process Restarts` performance counter to check the number of times your application pool recycled itself and the recycling frequency. If you see many recycles with no apparent reason, try to correlate the results with the application's memory consumption and CPU usage to verify it hasn't crossed any thresholds defined in the application pool configuration.

Idle Timeouts

The default setting of an application pool shuts down the pool after 20 minutes of inactivity. If you expect such idle timeframes, for example when all the users go out for lunch, you may want to increase the timeout period, or even cancel it.

Processor Affinity

By default, an application pool is configured to use all the cores available in the server. If you have any special background process running on the server which needs as much CPU time as it can get, you can tweak the affinity of the pool to use only specific cores, freeing the other cores for the background process. Of course this will also require you to set the background process's affinity so it won't compete with the worker process over the same cores.

Web Garden

The default behavior of an application pool is to start one worker process which handles all the requests for the application. If your worker process handles several requests at once, and those requests compete for the same resource by locking on it, you may end up with contention, causing latency in returning responses. For example, if your application is using a proprietary caching mechanism that has locks that prevent concurrent requests from inserting items to the cache, requests will start to synchronize one after the other, causing latencies that will be hard to detect and fix. Although we can sometimes fix the code to use less locking, it is not always possible. Another way of resolving this contention issue is to spin up multiple worker processes, all running the same application, and each handling its own set of requests, thus lowering the contention rate in the application.

Another scenario where running several processes of the same web application is useful, is when you have a 64-bit IIS server running a 32-bit web application. 64-bit servers usually have lots of memory, but 32-bit applications can only use up to 2 GB memory, which often leads up to frequent GC cycles and probably frequent application pool recycles. By spinning two or three worker processes for a 32-bit web application, the application can better utilize the server's available memory and reduce the number of GC cycles and application pool recycles it requires.

In the IIS application pool configuration, you can set the maximum number of worker processes that are permitted to service requests. Increasing the value to more than 1 (which is the default), will spin more worker processes as requests come in, up to the defined maximum. An application pool that has more than one worker process is referred to as a "Web Garden." Each time a connection is made from a client, it is assigned to a worker process which services the requests from that client from now on, allowing the requests from multiple users to be balanced between the processes, hopefully lowering to contention rate.

Note that using web gardens has its disadvantages. Multiple worker processes take up more memory, they prevent the use of the default in-proc session state, and when multiple worker processes are running on the same machine you may find yourself dealing with local resource contention, for example if both worker processes try to use the same log file.