MEMCACHED FOR C# - A WALKTHROUGH

A while ago I wrote about Object caching in .NET 4. I think that the **Object Caching in .NET 4** is a great tool to use for caching and literally takes minutes to get up and running. However, one of the downsides of using the *System.Runtime.Caching* in .NET 4 is that every time your application pool recycles in IIS, you lose the objects in cache. The same happens if you make a change to your web.config or redeploy your application.

This isn't a problem if you are running smaller applications and cached data isn't vital to their performance. However, for high-traffic applications that need to have the cache up all the time, this becomes a problem. Another thing to consider is the possibility of load-balanced servers - using the **Object Caching in .NET 4** isn't distributed. That means it will only store its cache per application pool. If you have two servers it means that you will have two different cache stores.

I have recently been introduced to the world of Memcached. **Memcached** is a free & open source, high-performance, distributed memory object caching system intended for use in speeding up dynamic web applications by alleviating database load. You can think of it as a short-term memory for your applications. The brilliant thing about **Memcached** is that you run it on one server and point all the applications to the **Memcached** server. This way all servers use the

POPULAR

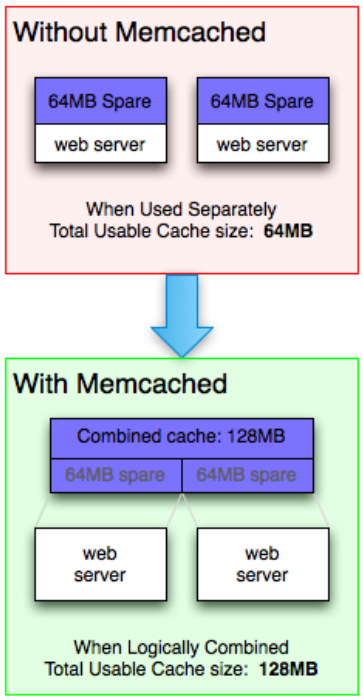
ASP.NET MVC HTML5 TOOLKIT

I'M WRITING A BOOK!

A SIMPLE HTML MINIFIER FOR ASP.NET

MVC AND THE HTML5 APPLICATION CACHE

same cache! Another bonus is that even if you redeploy your app, or recycle the app pool in any way - the cached objects are still there. This is great news for larger applications that need to be online constantly and can't afford any downtime with regards to it's cache.



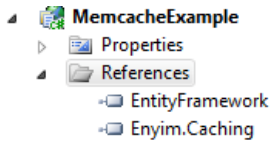
Memcached is no new comer to the web. It is used by major players such as Wikipedia, Flickr, Twitter, Youtube, WordPress.com and Digg. The advantages of using **Memcached** are endless and getting it setup on your machine is really simple. There are two parts to getting Memcached setup - you need the **Memcached** server and then you need to get the client API. There are few client API available for most of the common languages out there (Ruby, C#, PHP, C, Java).

I had a bit of trouble actually finding a decent ready-to-go implementation of **Memcached** for Windows. I got my version 1.4 from this site, but it seems that there are also other versions available here and here. Once you have downloaded the files, you can either run **Memcached** as a console application or install it as a Windows service (This is the recommended option). In order to install it as a service, run the following from the command line:

C:\yourmemcachedpath\memcached -d install

Next, you need to get the client API to start using the cache. There are a few different options with regards to the C# client API, but I chose

Enyim Memcached. Open Visual Studio, create a new project and add a reference to the **Enyim.Caching** library that you downloaded.



You will also need to update your web.config or app.config to include the following lines. In your configSections:

```
<configSections>
  <sectionGroup name="enyim.com">
    <section name="memcached"
             type="Enyim.Caching.Configuration.MemcachedClientSection,
                   Enyim.Caching" />
  </sectionGroup>
</configSections>
```

And then just before the end of the web.config file add the following:

```
  <enyim.com>
    <memcached protocol="Binary">
      <servers>
        <add address="127.0.0.1" port="11211" />
      </servers>
    </memcached>
  </enyim.com>
</configuration>
```

You will notice that the address is "**127.0.0.1**" and port "**11211**". This would normally be the network location that your Memcached server is hosted on, but in this instance I am testing it locally and this is the default.

Using the library is really simple and intuitive. Although it's not a real world example, the following code will store and retrieve a value from the cache.

```
using (MemcachedClient client = new MemcachedClient())
{
    // Store the record
    client.Store(StoreMode.Set, "currentTime", DateTime.Now.ToString());

    // Retrieve the value
    string value = client.Get<string>("currentTime");
}
```

A while ago I wrote about Object caching in .NET 4. I created a little wrapper that used simple syntax and wrapped around the **System.Runtime.Caching** namespace. Using the same class, we can just update it to point to our **Memcached** server. If it makes it easier for you to use the class I wrote and combined with the **Memcached API**, it is available for download here. Also, just in case the Memcached servers

are not available at any of the locations I defined above - you can download version 1.4 here.

I searched the web and found that there is very little documentation with regards to C# out there - I hope this article helped to get you up to speed with Memcached!

DATE: MONDAY, SEPTEMBER 26, 2011
TAGGED AS: CACHE

SHARE
SHARE ON TWITTER | SHARE ON GOOGLE+

COMMENTS

Darren - 9/27/2011
Dean I had a look at this last year for C# and at the time was disappointed with the lack of knowledge and support within the community. This is a useful post and a step in the right direction. Memcached is a very good tool - one that ASP.Net developers would benefit from using more. Thanks

Dean Hume - 9/29/2011
Hi Darren Thanks for the comment! Yeah, I had the same problem when I was researching Memcache. Hope it helped - Dean

friism - 10/6/2011
If you don't want to mess around with setting up memcached on Windows, you can run you app on AppHarbor (https://appharbor.com/). We also maintain an updated memcached sessionstateprovider, there's a guide on how to use it here: http://support.appharbor.com/kb/tips-and-tricks/using-memcached-backed

Gabriel Rodriguez - 10/6/2011
I was recently considering playing with memcached and .NET but just like you said, there wasn't a lot of material around. Gonna probably try it tonight thanks to your article. Thanks Dean.

Shaun Danielz - 10/7/2011
Perhaps your article should cover AppFabric caching services?

Tony - 12/9/2011
How to avoid using Web.config and app.config?