Contents

[References 2](#_Toc109826961)

[Cache 3](#_Toc109826962)

[Policies 3](#_Toc109826963)

[Popular techniques 3](#_Toc109826964)

[Caching in ASP.Net Core 3](#_Toc109826965)

[In-Memory caching 3](#_Toc109826966)

[Distributed caching 3](#_Toc109826967)

[Response caching 4](#_Toc109826968)

[In Memory V/s Distributed Cache 4](#_Toc109826969)

[In-Memory Caching – Implementation 4](#_Toc109826970)

[MemoryCacheEntryOption 6](#_Toc109826971)

[GetOrCreate/ GetOrCreateAsync 6](#_Toc109826972)

[MemoryCache.Compact 7](#_Toc109826973)

[Distributed Cache Implementation 7](#_Toc109826974)

# References

|  |
| --- |
| <https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-6.0> |
|  |
|  |

# Cache

Cache is the mechanism of improving performance by keeping **recent** or **most often** used data items in in memory location that are faster and computationally cheaper to access then memory storages

When the cache is full, the algorithm must choose which items to discard to make the room for new ones.

Caching works well with data that changes infrequently and is expensive (having more logic) to generate.

## Policies

There are lot of categories of policies are available like

* Queue based policy
* Recently based policy
* Frequently based policy

## Popular techniques

And some for the popular techniques used in application are

* LRU – Least Recently Used - Discard the least recently used item first
* LFU – Least Frequently Used – Counts how often an item is needed. Those that are used least often are discarded first

# Caching in ASP.Net Core

## In-Memory caching

* It uses application server memory to store the cache data
* This type of caching is suitable for a single server or multiple server using session affinity
* **Session Affinity / Sticky session** – it means request from the client are ***always routed to the*** **same serve** for processing.

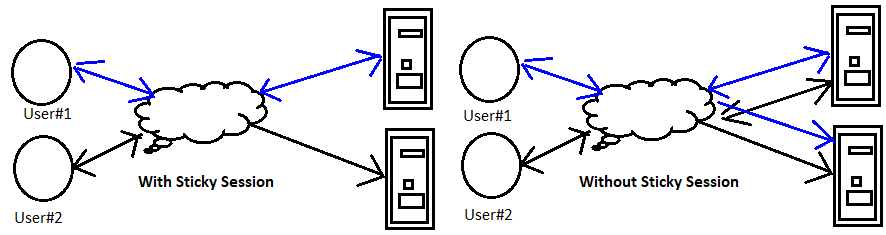
## Distributed caching

* Use a distribution cache to store data in memory when the application is hosted in the **cloud** or **server farms.**
* The cache is **shared across the servers** that process the request.
* A client can submit request that handled by any server in the group if the cache data for the client are available.
* ASP.Net core works with
  + SQL Server
  + Radis
  + NCache

## Response caching

* Response caching middleware enable caching server responses based on HTTP Cache headers.

**Sticky Session**



## In Memory V/s Distributed Cache

|  |  |
| --- | --- |
| In-Memory Cache | Distributed Cache |
| Represents cache stored in the memory of web (application)server | Distributed cache shared by multiple app servers, typically maintained by external app server that access it. |
| Requires sessions to be sticky | Requires sessions to be non-sticky |
| In memory cache can stores any objects | Distributed cache interface is limited to byte[] |
|  |  |
| The In memory and distributed cache stores cache items in the form of key value pairs | |

## In-Memory Caching – Implementation

ASP.net core supports several different caches. The simplest cache is based on IMemberCache

* IMemoryCache represents the cache stored in the memory of web server
* Apps need to ensure the session are sticky when using in memory cache(ensures requests from all the clients go to same server)
* Non Sticky sessions in the web farms require distributed cache.
* In memory cache can store any object
* Microsoft.Extension.Caching.Memory names extends In memory cache
* Limit the cache size using SetSize, Size, SizeLimit and Limit methods.

Note: ASP.net core does not limit cache size based on memory pressure.

**Sample view model**

|  |
| --- |
| public class CacheSampleViewModel  {  public DateTime CurrentDateTime { get; set; }  public DateTime CacheDateTime { get; set; }  } |

1. Sample controller code

|  |
| --- |
| public class CacheDemoController : Controller  {  private readonly IMemoryCache memoryCache;  public CacheDemoController(IMemoryCache memoryCache)  {  this.memoryCache = memoryCache;  }  public IActionResult Index()  {  CacheSampleViewModel cacheSampleViewModel = new CacheSampleViewModel();  OnGet(cacheSampleViewModel);  return View(cacheSampleViewModel);  } |

IMemoryCache - Represents a local in-memory cache whose values are not serialized.

1. Implementing sample in memory cache

|  |
| --- |
| public void OnGet(CacheSampleViewModel cacheSampleViewModel)  {  cacheSampleViewModel.CurrentDateTime = DateTime.Now;  var cacheKey = "CreateDate";  // check if data exits with in cache with given key  if (!memoryCache.TryGetValue(cacheKey, out DateTime cacheValue))  {  // setting cache value for key  cacheValue = cacheSampleViewModel.CurrentDateTime;  // setting cache options  var cacheEntryOption = new MemoryCacheEntryOptions()  .SetSlidingExpiration(TimeSpan.FromSeconds(5));  memoryCache.Set(cacheKey,cacheValue ,cacheEntryOption);  }  cacheSampleViewModel.CacheDateTime = cacheValue;  } |

1. Configure AddMemoryCache service

|  |
| --- |
| builder.Services.AddMemoryCache(); |

Note: From Asp.Net core, we no need to explicitly register IMemoryCache dependency injection. Calling the above method will take care of dependency injection

1. Rendering in view

|  |
| --- |
| @model CacheSampleViewModel  <ul>  <li>Current Time : @Model.CurrentDateTime</li>  <li>Cache Time : @Model.CacheDateTime</li>  </ul> |

SetSlidingExpiration – sets how long the cache entry can be inactive before it will be removed.

In the above sample SetSlidingExpiration is set to 5 seconds, which means each time the cache data is accessed, it remains in memory for 5 seconds. If it not accessed for 5 seconds it get evicted from cache.

### MemoryCacheEntryOption

It provides some of the important extension methods to deals with cache like

* SetPriority - sets the priority to the cache
* RegisterPostEvictionCallback – get called after the entry is evicted from the cache.

|  |
| --- |
| public void OnGet(CacheSampleViewModel cacheSampleViewModel)  {  cacheSampleViewModel.CurrentDateTime = DateTime.Now;  var cacheKey = "CreateDate";  // check if data exits with in cache with given key  if (!memoryCache.TryGetValue(cacheKey, out DateTime cacheValue))  {    // setting cache value for key  cacheValue = cacheSampleViewModel.CurrentDateTime;  // setting cache options  var cacheEntryOption = new MemoryCacheEntryOptions()  .SetPriority(CacheItemPriority.NeverRemove)  .RegisterPostEvictionCallback(PostEvictionCallback, memoryCache)  ;  ;  memoryCache.Set(cacheKey,cacheValue ,cacheEntryOption);  }  cacheSampleViewModel.CacheDateTime = cacheValue;  } |

|  |
| --- |
| private static void PostEvictionCallback(  object cacheKey, object cacheValue, EvictionReason evictionReason, object state)  {  var memoryCache = (IMemoryCache)state;  memoryCache.Set(  cacheKey,  $"Entry {cacheKey} was evicted: {evictionReason}.");  } |

### GetOrCreate/ GetOrCreateAsync

This extension method is responsible for getting the cache, if cache is not found with the provided key, it will create a cache. With the configured sliding expiration.

|  |
| --- |
| public void OnGetCacheGetOrCreate(CacheSampleViewModel cacheSampleViewModel)  {  var cacheKey = "cacheKeyEntry";  var cacheValue = memoryCache.GetOrCreate(cacheKey, cacheEntry =>  {  cacheEntry.SlidingExpiration = TimeSpan.FromSeconds(5);  return DateTime.Now;  });  cacheSampleViewModel.CurrentDateTime = DateTime.Now;  cacheSampleViewModel.CacheDateTime = cacheValue;  } |

### MemoryCache.Compact

It attempts to remove specific percentage of the cache in the following order

* All expired items
* Items by priority – lowest priority items will removed first
* Least recently used objects
* Items with earliest absolute expiration
* Items with earliest sliding expiration

|  |
| --- |
| \_myMemoryCache.Cache.Remove(CacheKeys.Entry);  \_myMemoryCache.Cache.Compact(.25); |

## Distributed Cache Implementation

A distributed cache is the cache shared by multiple app servers, typically maintained as an external server to the app server that accessed it.

It improves the performance and scalability of an ASP.Net core application, especially when it is hosted on the cloud or server farm.

It has several advantages over other caches

* It consistent across requests to multiple servers
* It available (survives) even after app server restarts or deployments
* It will not use local memory

1. Install dotnet-sql-cache from .Net CLI

|  |
| --- |
| dotnet tool install --global dotnet-sql-cache |

1. Executing the below command in .Net CLI ensures to create a table with TestCache in existing SampleDB database.

|  |
| --- |
| dotnet sql-cache create "Data Source=myServer;Initial Catalog=SampleDB;Integrated Security=True;" dbo TestCache |

1. App.settings.js – configuring connection string

|  |
| --- |
| "ConnectionStrings": {  "DistCache\_ConnectionString": "Data Source=DESKTOP-PD9SE9U\\SQLEXPRESS01;Initial Catalog=SampleDB;Integrated Security=True"  }, |

1. Program.cs – add below code

|  |
| --- |
| builder.Services.AddDistributedSqlServerCache(options =>  {  options.ConnectionString = builder.Configuration.GetConnectionString("DistCache\_ConnectionString");  options.SchemaName = "dbo";  options.TableName = "TestCache";  });  app.Lifetime.ApplicationStarted.Register(() =>  {  var currentTime = DateTime.Now.ToString();  byte[] TestCache = System.Text.Encoding.UTF8.GetBytes(currentTime);  var options = new DistributedCacheEntryOptions()  .SetSlidingExpiration(TimeSpan.FromSeconds(5));  app.Services.GetService<IDistributedCache>().Set("cacheCurrentTime", TestCache, options);  }); |

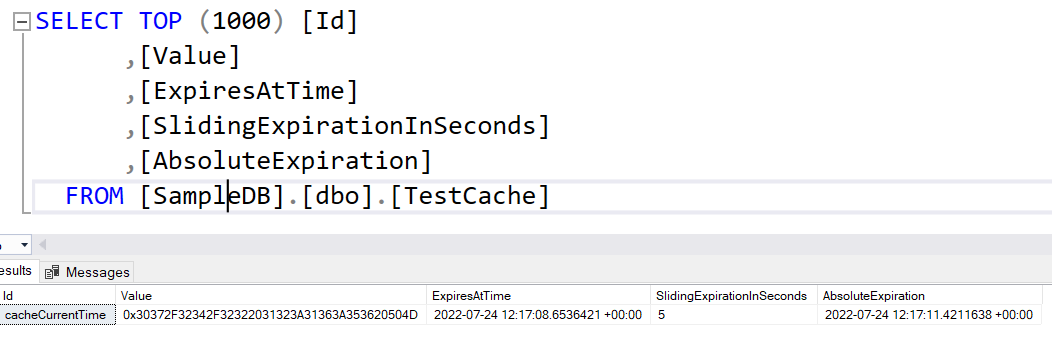
1. Controller – sample code

|  |
| --- |
| public void OnGet(CacheSampleViewModel cacheSampleViewModel)  {  var cacheCurrentTime = DateTime.MinValue.ToString();  // check data exists in cache with key  var encodedCachedTimeUTC = caching.Get("cacheCurrentTime");  // if exists get the cache data  if (encodedCachedTimeUTC != null)  {  cacheCurrentTime = Encoding.UTF8.GetString(encodedCachedTimeUTC);  cacheSampleViewModel.CacheResponse = "Data retrieved from cache with key: cacheCurrentTime";  }  else {  // if not exists, save data in cache  OnPostResetCachedTime();  cacheSampleViewModel.CacheResponse = "Data saved in cache with key: cacheCurrentTime";  }  cacheSampleViewModel.CurrentDateTime = DateTime.Now;  cacheSampleViewModel.CacheDateTime = Convert.ToDateTime(cacheCurrentTime);  }  public void OnPostResetCachedTime()  {  var currentTimeUTC = DateTime.UtcNow.ToString();  byte[] encodedCurrentTimeUTC = Encoding.UTF8.GetBytes(currentTimeUTC);  var options = new DistributedCacheEntryOptions()  .SetSlidingExpiration(TimeSpan.FromSeconds(5))  .SetAbsoluteExpiration(TimeSpan.FromSeconds(15));  caching.Set("cacheCurrentTime", encodedCurrentTimeUTC, options);  }  public IActionResult Index()  {  CacheSampleViewModel cacheSampleViewModel = new CacheSampleViewModel();  OnGet(cacheSampleViewModel);  return View(cacheSampleViewModel);  } |

1. View – Sample Code

|  |
| --- |
| @model CacheSampleViewModel  <ul>  <li>Current Time : @Model.CurrentDateTime</li>  <li>Cache Time : @Model.CacheDateTime</li>  <li>Cache Response: @Model.CacheResponse</li>  </ul> |

1. Backend table and data



1. Output

