2.3. Caching

2.3.1. Introduction

StudioX provides an abstraction for caching. It internally uses this cache abstraction. While default implementation uses MemoryCache, it can be implemented and changable for any other caching provider. StudioX.RedisCache implements cache in Redis for instance (see "Redis Cache Integration" section below).

2.3.2. ICacheManager

Main interface for caching is **ICacheManager**. We can inject it and use it to get a cache. Example:

```
public class TestAppService : ApplicationService
{
    private readonly ICacheManager cacheManager;
    public TestAppService(ICacheManager cacheManager)
    {
        this.cacheManager = cacheManager;
    }

    public Item GetItem(int id)
    {
        return readonly cacheManager.GetCache("MyCache")
                .Get(id.ToString(), () => GetFromDatabase(id)) as Item;
    }

    public Item GetFromDatabase(int id)
    {
        //... retrieve item from database
    }
}
```

In this sample, we're injecting **ICacheManager** and getting a cache named **MyCache**. Cache names are case sensitive, than means "MyCache" and "MYCACHE" are different caches.

WARNING: GetCache Method

Do not use GetCache method in your constructor. This may dispose the Cache if your class is not singleton.

2.3.3. ICache

ICacheManager.**GetCache** method returns an ICache. A cache is singleton (per cache name). It is created first time it's requested, then returns always the same cache object. So, we can share same cache with same name in different classes (clients).

In the sample code, we see simple usage of ICache.Get method. It has two arguments:

- **key**: A string unique key of an item in the cache.
- **factory**: An action which is called if there is no item with the given key. Factory method should create and return the actual item. This is not called if given key has present in the cache.

ICache interface also has methods like GetOrDefault, Set, Remove and Clear. There are also async versions of all methods.

ITypedCache

ICache interface works string as key and object as value. ITypedCache is a wrapper to ICache to provide type safe, generic cache. We can use generic GetCache extension method to get an ITypedCache:

```
ITypedCache<int, Item> myCache = cacheManager.GetCache<int, Item>("MyCache");
```

Also, we can use AsTyped extension method to convert an existing ICache instance to ITypedCache.

2.3.4. Configuration

Default cache expire time is 60 minutes. So, if you don't use an item in the cache for 60 minutes, it's automatically removed from the cache. You can configure it for all caches or for a specific cache.

```
//Configuration for all caches
Configuration.Caching.ConfigureAll(cache =>
{
    cache.DefaultSlidingExpireTime = TimeSpan.FromHours(2);
});

//Configuration for a specific cache
Configuration.Caching.Configure("MyCache", cache =>
{
    cache.DefaultSlidingExpireTime = TimeSpan.FromHours(8);
});
```

This code should be placed PreInitialize method of your module. With such a code, MyCache will have 8 hours expire time while all other caches will have 2 hours.

Your configuration action is called once cache is first created (on first request). Configuration is not restricted to DefaultSlidingExpireTime only, since cache object is an ICache, you can use it's properties and methods freely configure and initialize it.

2.3.5. Entity Caching

While StudioX's cache system is general purpose, there is an EntityCache base class that can help you if you want to cache entities. We can use this base class if we get entities by their Ids and we want to cache them by Id to not query from database frequently. Assume that we have a Person entity like that:

```
public class Person : Entity
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

And assume that we frequently want to get Name of people while we know their Id. First, we should create a class to store cache items:

```
[AutoMapFrom(typeof(Person))]
public class PersonCacheItem
{
    public string Name { get; set; }
}
```

We should not directly store entities in the cache since caching may need to serialize cached objects and entities may not be serialized (especially if they have navigation properties). That's why we defined a simple (DTO like) class to store data in the cache. We added AutoMapFrom attribute since we want to use AutoMapper to convert Person entities to PersonCacheItem objects automatically. If we don't use AutoMapper, we should override MapToCacheItem method of EntityCache class to manually convert/map it.

While it's not required, we may want to define an interface for our cache class:

```
public interface IPersonCache : IEntityCache<PersonCacheItem>{}
```

Finally, we can create the cache class to cache Person entities:

```
public class PersonCache : EntityCache<Person, PersonCacheItem>, IPersonCache,
ItransientDependency
{
    public PersonCache(ICacheManager cacheManager, IRepository<Person> repository)
        : base(cacheManager, repository) { }
}
```

That's all. Our person cache is ready to use. Cache class can be transient (as in this example) or singleton. This does not mean the cached data is transient. It's always cached globally and accessed thread-safe in your application.

Now, whenever we need Name of a person, we can get it from cache by the person's Id. An example class that uses the Person cache:

```
public class MyPersonService : ITransientDependency
{
    private readonly IPersonCache personCache;

    public MyPersonService(IPersonCache personCache)
    {
        this.personCache = personCache;
    }

    public string GetPersonNameById(int id)
    {
        return this.personCache[id].Name; //alternative: this.personCache.Get(id).Name;
    }
}
```

We simply injected IPersonCache, got the cache item and got the Name property.

How EntityCache Works

- It gets entity from repository (from database) in first call. Then gets from cache in subsequent calls.
- It automatically invalidates cached entity if this entity is updated or deleted. Thus, it will be retrieved from database in the next call.

- It uses IObjectMapper to map entity to cache item. IObjectMapper is implemented by AutoMapper module. So, you need to AutoMapper module if you are using it. You can override MapToCacheItem method to manually map entity to cache item.
- It uses cache class's FullName as cache name. You can change it by passing a cache name to the base constructor.
- It's thread-safe.
- If you need more complex caching requirements, you can extend EntityCache or create your own solution.

2.3.6. Redis Cache Integration

Default cache manager uses in-memory caches. So, it can be a problem if you have more than one concurrent web server running the same application. In that case, you may want to a distributed/central cache server. You can use Redis as your cache server easily.

First, you need to install StudioX.RedisCache nuget package to your application (you can install it to your Web project, for example). Then you need to add a DependsOn attribute for StudioXRedisCacheModule and call UseRedis extension method in PreInitialize method of your module, as shown below:

```
using StudioX.Runtime.Caching.Redis;

namespace MyProject.StudioXZeroTemplate.Web
{
    [DependsOn(
        //...other module dependencies
        typeof(StudioXRedisCacheModule))]
    public class MyProjectWebModule : StudioXModule
    {
        public override void PreInitialize()
        {
            //...other configurations
            Configuration.Caching.UseRedis();
        }

        //...other code
    }
}
```

StudioX.RedisCache package uses "**localhost**" as **connection string** as default. You can add connection string to your config file to override it. Example:

```
<add name="StudioX.Redis.Cache" connectionString="localhost"/>
```

Also, you can add setting to appSettings to set database id of Redis. Example:

```
<add key="StudioX.Redis.Cache.DatabaseId" value="2"/>
```

Different database ids are useful to create different key spaces (isolated caches) in same server.

UseRedis method has also an overload that takes an action to directly set option values (overrides values in the config file).

See Redis documentation for more information on Redis and it's configuration.

Note: Redis server should be installed and running to use Redis cache in StudioX.

Radis cached for window:

- https://github.com/MSOpenTech/redis
- https://github.com/rgl/redis/downloads