**内存缓存**

包：Microsoft.Extensions.Caching.Memory

1. 注册内存缓存

services.AddMemoryCache();

1. 基本使用示例

    public class Worker : BackgroundService

    {

        private readonly ILogger<Worker> \_logger;

        private readonly IServiceProvider \_serviceProvider;

        public Worker(ILogger<Worker> logger, IServiceProvider serviceProvider)

        {

            \_logger = logger;

            \_serviceProvider = serviceProvider;

        }

        protected override async Task ExecuteAsync(CancellationToken stoppingToken)

        {

            while (!stoppingToken.IsCancellationRequested)

            {

                Console.WriteLine(Get());

                await Task.Delay(1000, stoppingToken);

            }

        }

        public string Get()

        {

            IMemoryCache cache = \_serviceProvider.GetService<IMemoryCache>();

            // 从缓存中获取键为 key1 的项，如果没有或过期，则调用第二个参数Fun生成

            return cache.GetOrCreate<string>("key1", cacheEntity => {

                // 设置过期时间，过期后会重新调用该函数生成值

                // 设置过期时间

                // 如果5秒内没有访问该值，则过期

                // 否则，过期时间刷新为访问的时间 +5 秒

                cacheEntity.SlidingExpiration = TimeSpan.FromSeconds(5);

                // 设置绝对过期时间

                // 过期时间为15秒后，无论这15秒内是否有访问都会过期

                cacheEntity.AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(15);

                // 设置该项优先级

                // NeverRemove 为不会从缓存中移除，除非过期

                // 优先级越低，内存不足时，将会移除该项

                cacheEntity.SetPriority(CacheItemPriority.NeverRemove);

                // 注册缓存项从缓存移除时的回调函数

                cacheEntity.RegisterPostEvictionCallback(

                    (object key, object value, EvictionReason reason, object state) => { });

                return DateTime.Now.ToString();

            });

        }

        public void Remove()

        {

            IMemoryCache cache = \_serviceProvider.GetService<IMemoryCache>();

            // 从缓存中移除键为key1的缓存项

            cache.Remove("key1");

        }

    }

1. 内存缓存与EFCore

EFCore也使用内存缓存，如果我们使用依赖注入注入IMemoryCache ，则注入的可能是EFCore使用的缓存，所以最好的方法是我们重新注入我们的缓存（单例）

    public class IEMemoryCache : MemoryCache, IIEMemoryCache

    {

        public IEMemoryCache() : base(new MemoryCacheOptions {

            SizeLimit = null

        })

        {

        }

    }

**分布式缓存**

1. 使用示例

public string Get()

{

    IDistributedCache cache = \_serviceProvider.GetService<IDistributedCache>();

    // 获取键为 mykey1 的缓存项

    string value = cache.GetString("mykey1");

    if (string.IsNullOrEmpty(value))

    {

        // 设置键为 mykey1 的缓存项

        cache.SetString("mykey1", "abc");

        return "no value";

    }

    return value;

}

public void Remove()

{

    IDistributedCache cache = \_serviceProvider.GetService<IDistributedCache>();

    // 从缓存中移除键为mykey1的缓存项

    cache.Remove("mykey1");

}

1. 注册分布式缓存

目前有很多种分布式缓存可以注册

1. 内存分布式缓存

内存分布式缓存其使用内存作为缓存，单其却实现了分布式缓存的接口

services.AddDistributedMemoryCache();

1. redis分布式缓存

services.AddStackExchangeRedisCache (options => {

    // 配置 Redis 连接字符串

    // 127.0.0.1:6379 为 Redis 服务地址，123456 为 Redis 连接密码

    options.Configuration = "127.0.0.1:6379,password=123456";

    // 随便起的实例名，我们在 Redis 服务器保存的 key 均以该实例名开头

    options.InstanceName = "SampleInstance";

});