

第20期-分布式缓存模块

2020年8月23日 8:49

安装模块包

```
abp add-package Volo.Abp.Caching  
Install-Package Volo.Abp.Caching
```

```
[DependsOn(typeof(AbpCachingModule))]  
public class MyModule : AbpModule
```

使用 IDistributedCache 接口

ASP.NET Core 定义了 IDistributedCache 接口用于 GET/SET 缓存值，但是会有以下问题：

- 适用于 Byte 数组，需要自己将对象序列化成字节数组。
- 使用单个缓存池，无法隔离不同组合或者业务的缓存项。

ASP.NET Core 中的分布式缓存

使用 IDistributedCache<TCacheItem> 接口

- 内部自动序列化反序列化缓存对象，默认使用 JSON 序列化，但可以替换依赖注入系统中 IDistributedCacheSerializer 服务的实现来覆盖默认的处理。
- 自动添加缓存KEY前缀，默认前缀是类名称，如果你的类名以 CacheItem 结尾，那么 CacheItem 会被忽略，不应用到缓存名称上，你也可以在缓存类上使用 CacheNameAttribute 设置缓存的名称。
- 它自动将当前的租户 ID 添加到缓存键中，以区分不同租户的缓存项 (只有在你的应用程序是多租户的情况下生效)，在缓存类上应用 IgnoreMultiTenancyAttribute, 可以在所有的租户间共享缓存。
- 允许为每个应用程序定义全局缓存键前缀，不同的应用程序可以在共享的分布式缓存中拥有自己的隔离池。

使用方式

```
public class CarCacheItem  
{
```

```

    public Guid Id { get; set; }
    public string Name { get; set; }
    public float Price { get; set; }
}

public class CarService : IRemoteService, ITransientDependency
{
    private readonly IDistributedCache<CarCacheItem> _cache;

    public CarService(IDistributedCache<CarCacheItem> cache)
    {
        _cache = cache;
    }

    public async Task<CarCacheItem> GetAsync(Guid bookId)
    {
        return await _cache.GetOrAddAsync(
            bookId.ToString(),
            async () => await GetBookFromDatabaseAsync(bookId),
            () => new DistributedCacheEntryOptions
            {
                AbsoluteExpiration = DateTimeOffset.Now.AddHours(1)
            }
        );
    }

    private async Task<CarCacheItem> GetBookFromDatabaseAsync(Guid bookId)
    {
        await Task.Delay(TimeSpan.FromSeconds(3));

        return new CarCacheItem { Id = bookId, Name = "Porsche", Price =
88.88F };
    }
}

```

使用 IDistributedCache<TCacheItem, TCacheKey> 接口

IDistributedCache<TCacheItem> 接口默认了键是 string 类型 (如果你的键不是string类型需要进行手动类型转换), IDistributedCache<TCacheItem, TCacheKey> 将键的类型泛型化试图简化手动转换的操作。

IDistributedCache<TCacheItem, TCacheKey> 在内部使用键对象的 ToString() 方法转换类型为string, 如果你的将复杂对象做为键,那么需要重写类的 ToString 方法。

```

public class UserInOrganizationCacheKey
{
    public Guid UserId { get; set; }
}

```

```

        public Guid OrganizationId { get; set; }

        //构建缓存key
        public override string ToString()
        {
            return $"{UserId}_{OrganizationId}";
        }
    }

    public class BookService : ITransientDependency
    {
        private readonly IDistributedCache<UserCacheItem,
        UserInOrganizationCacheKey> _cache;

        public BookService(
            IDistributedCache<UserCacheItem, UserInOrganizationCacheKey> cache)
        {
            _cache = cache;
        }

        ...
    }

```

使用 AbpDistributedCacheOptions 可以配置分布式缓存行为

```

Configure<AbpDistributedCacheOptions>(options =>
{
    options.HideErrors = true;
});

```

分布式缓存 Redis 配置

在 Docker 中运行 Redis 服务

```

docker run -p 6379:6379 -d redis:latest redis-server
docker ps
docker exec -it <id> /bin/sh

```

<https://www.runoob.com/docker/docker-tutorial.html>

安装 Redis 扩展模块

```

abp add-package Volo.Abp.Caching.StackExchangeRedis

```

配置 Redis 连接字符串

```
"Redis": {  
  "Configuration": "127.0.0.1:6379"  
}
```

测试地址

<https://localhost:5001/api/app/car?carid=e87c1777-600a-456b-96a3-151e71550617>

使用 Redis-Cli 查看缓存

查询所有KEY键: keys *

查看键所存的类型: type <key>

查看HASH类型: hgetall <key>

删除所有KEY缓存: flushall 或 flushdb

<https://www.runoob.com/redis/redis-tutorial.html>