МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА

ДОСЛІДЖЕННЯ ПРОЦЕСІВ ОПТИМІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА ДОПОМОГОЮ КЕШУВАННЯ

Виконав
ст. гр. ІПЗ-111м Єременко І.Д.
Керівник
к.т.н., доц. Резніченко Ю.С.

Мета та задачі роботи

Метою є створення концепції модульної інтеграції кешів та розробка на її основі ASP .NET Core (C#) бібліотеки

Задачі:

1. Виконати аналіз патернів об'єктно-орієнтованого та функціонального проектування
2. Створити концепцію модульної інтеграції кешів
3. Розробити бібліотеку Caching у ASP .NET Core (C#) та спроектувати «інтерфейс» для роботи з нею
4. Розробити Unit тести та Benchmark тести для бібліотеки Caching

Інструменти розробки

- .NET + ASP .NET Core + C#
- JetBrains Rider
- Redis
- MemCached
- NuGet
- NUnit, Moq, FluentAssertions, Benchmark .NET

Патерни функціонального програмування

```
/// <inheritdoc />
0+1 usages  mrlldd
public Task<Result> SetAsync<T>(IReadOnlyCachesCollection<T> caches, T value, CancellationToken token = default)
{
    return ExecuteInSequenceAsync(caches, func: (x:IUnknownStoreCache<T> , ct:CancellationToken) => x.SetAsync(value, ct), token);
}

3 usages  mrlldd
private static async Task<Result> ExecuteInSequenceAsync<T>(IReadOnlyCachesCollection<T> caches,
    Func<IUnknownStoreCache<T>, CancellationToken, ValueTask<Result>> func, CancellationToken token)
{
    var count:int = caches.Count;
    var fails = new List<Result>();
    for (var i = 0; i < count; i++)
    {
        var valueTask = func(caches.ElementAt(i), token);
        if (valueTask.IsCompletedSuccessfully)
        {
            if (!valueTask.Result.Successful) fails.Add(valueTask.Result);
            continue;
        }

        var awaited:Result = await valueTask;
        if (!awaited.Successful) fails.Add(awaited);
    }

    return fails.Any()
        ? new SequenceStrategyFailException(fails)
        : Result.Success;
}
```

Функції вищого порядк

```
0+13 usages  mrlldd
public Result Remove(string key, ICacheStoreOperationMetadata metadata)
{
    LogRemoveTry(key, metadata);
    var result = sourceCacheStore.Remove(key, metadata);
    LogRemovingResult(result, key, metadata);
    return result;
}

2 usages  new *
private void LogRemoveTry(string key, ICacheStoreOperationMetadata metadata)
{
    logger.Log(loggingOptions.LogLevel,
        message: "[{Store}] [{CacheStoreOperationId:D5}] Trying to remove entry with key \"{EntryKey}\".",
        params args: storeLogPrefix, metadata.OperationId, key);
}

2 usages  new *
private void LogRemovingResult(Result result, string key, ICa
{
    if (result.Successful)
        logger.Log(
            loggingOptions.LogLevel,
            message: "[{Store}] [{CacheStoreOperationId:D5}] Suc
            params args: storeLogPrefix, metadata.OperationId, ke
    else
        logger.Log(
            loggingOptions.ErrorsLogLevel,
            result,
            message: "[{Store}] [{CacheStoreOperationId:D5}] Fai
            params args: storeLogPrefix, metadata.OperationId, ke
}
```

```
1+16 usages  mrlldd
public ValueTask<Result<T>> GetAsync<T>(string key, ICacheStoreOperationMetadata metadata,
    CancellationToken token = default)
{
    var task = Result.Of( asyncFactory: async () =>
    {
        var fromCache:string? = await distributedCache.GetStringAsync(key, token);
        return string.IsNullOrEmpty(fromCache)
            ? throw new CacheMissException(key)
            : Deserialize<T>(fromCache) ?? throw new DeserializationFailException(key, fromCache);
    });
    return new ValueTask<Result<T>>(task);
}
```

Монада Result

```
3 usages  mrlldd
private static async Task<Result> ExecuteInParallelAsync<T>(IReadOnlyCachesCollection<T> caches,
    Func<IUnknownStoreCache<T>, CancellationToken, ValueTask<Result>> func, CancellationToken token)
{
    var tasksList = new List<Task<Result>>();
    var count:int = caches.Count;
    var fails = new List<Result>();
    for (var i = 0; i < count; i++)
    {
        var valueTask = func(caches.ElementAt(i), token);
        if (valueTask.IsCompletedSuccessfully)
        {
            if (!valueTask.Result.Successful) fails.Add(valueTask.Result);
            continue;
        }

        tasksList.Add(item: valueTask.AsTask());
    }

    var results:Result[]? = await Task.WhenAll(tasksList);
    var length:int = results.Length;
    for (var i = 0; i < length; i++)
    {
        var r:Result? = results[i];
        if (!r.Successful) fails.Add(r);
    }

    return fails.Any()
        ? new ParallelStrategyFailException(fails)
        : Result.Success;
}
```

Патерни ООП

```
internal class Cache<T> : ICache<T>
{
    mrlldd
    public Cache(IReadOnlyCachesCollection<T> instances)
    {
        Instances = instances;
    }

    9+3 usages
    public IReadOnlyCachesCollection<T> Instances { get; }

    mrlldd
    public Task<Result<T>> GetAsync(CancellationToken token = default)
    {
        return GetAsync(GetFirstSuccessfulStrategy.Instance, token);
    }

    mrlldd
    public Result<T> Get()
    {
        return Get(GetFirstSuccessfulStrategy.Instance);
    }

    1 usage  mrlldd
    public Task<Result<T>> GetAsync(ICacheGetStrategy strategy, Canc
    {
        return strategy.GetAsync(Instances, token);
    }

    1 usage  mrlldd
    public Result<T> Get(ICacheGetStrategy strategy)
    {
        return strategy.Get(Instances);
    }
```

Strategy

```
actionsAndPerfLoggingDecoratedServiceProvider = new ServiceCollection()
    .AddCaching(typeof(DependencyResolvingBenchmarks).Assembly)
    .WithActionsLogging<InVoid>()
    .WithPerformanceLogging<InVoid>() // ICachingServiceCollection
    .BuildServiceProvider() // ServiceProvider
    .CreateScope().ServiceProvider; // IScr
```

Decorator

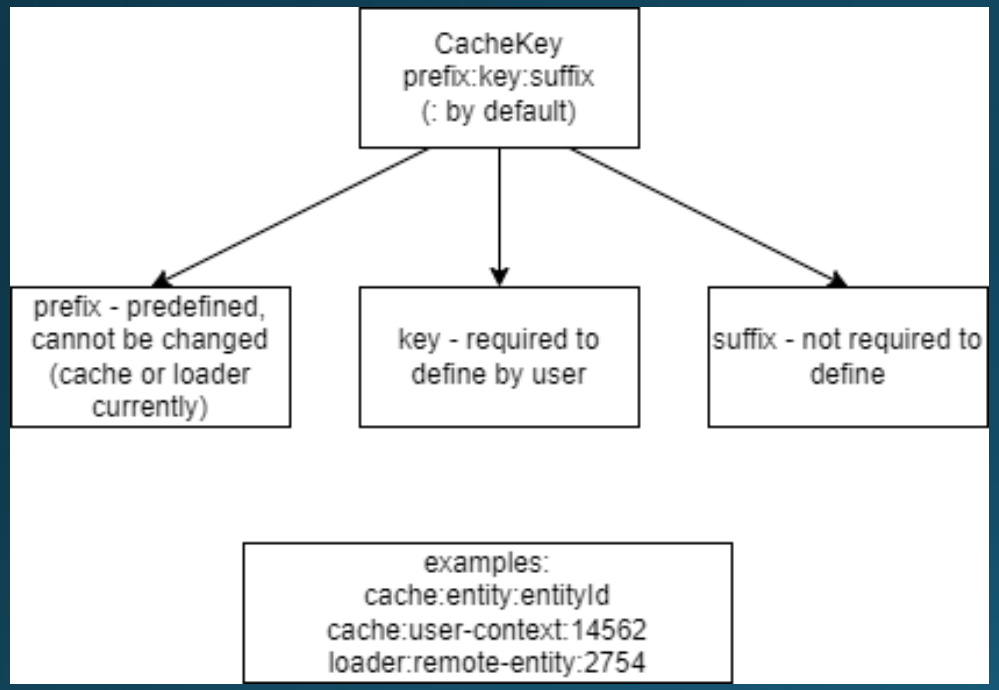
```
internal class StoreOperationProvider : IStoreOperationProvider
{
    private int currentId = 1;

    0+9 usages  mrlldd
    public ICacheStoreOperationMetadata Next(string cacheKeyDelimiter)
    {
        return new CacheStoreOperationMetadata(Interlocked.Increment(ref currentId), cacheKeyDelimiter);
    }
}
```

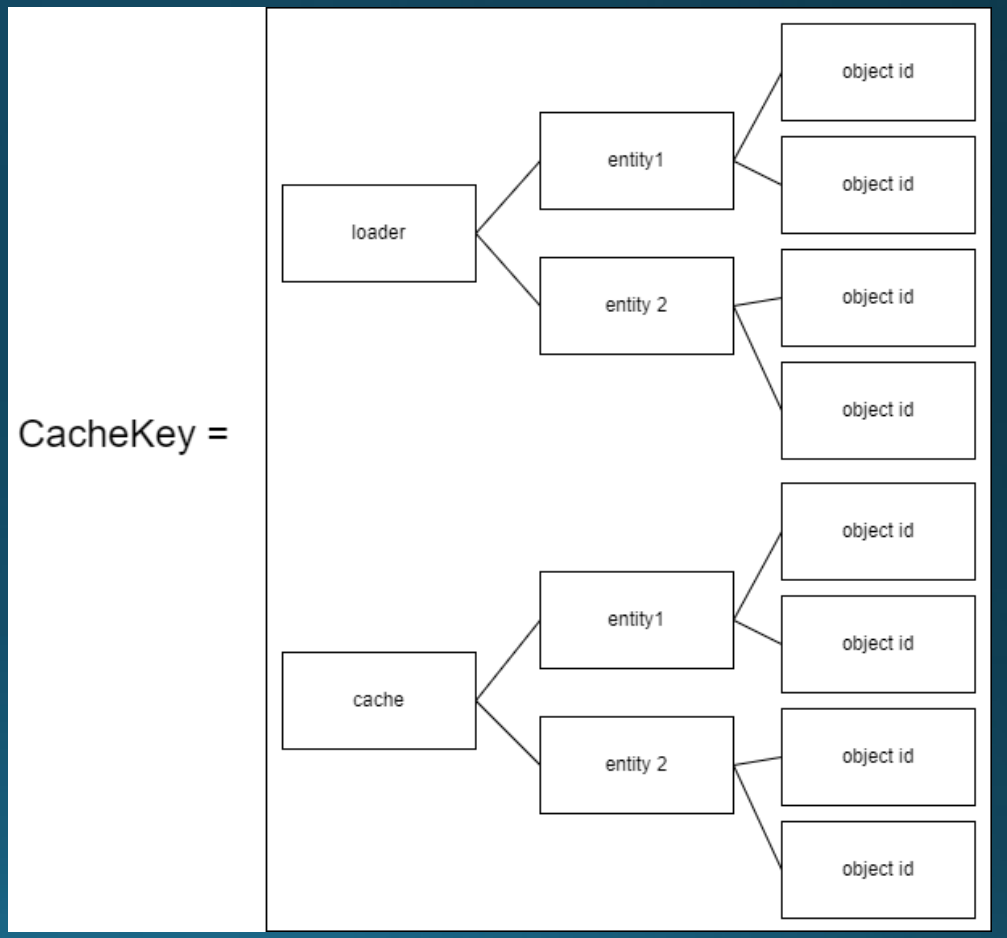
Builder

```
2 usages  mrlldd
internal class PerformanceLoggingCacheStore<TFlag> : ICacheStore<TFlag> where TFlag : CachingFlag
{
    private readonly ILogger<ICacheStore<TFlag>> logger;
    private readonly ICachingPerformanceLoggingOptions loggingOptions;
    private readonly ICacheStore<TFlag> sourceCacheStore;
    private readonly string storeLogPrefix;

    1 usage  mrlldd
    public PerformanceLoggingCacheStore(ICacheStore<TFlag> sourceCacheStore,
        ILogger<ICacheStore<TFlag>> logger,
        ICachingPerformanceLoggingOptions loggingOptions,
        string storeLogPrefix)
    {
        this.sourceCacheStore = sourceCacheStore;
        this.logger = logger;
        this.loggingOptions = loggingOptions;
        this.storeLogPrefix = storeLogPrefix;
    }

    0+16 usages  mrlldd
    public Result<T> Get<T>(string key, ICacheStoreOperationMetadata metadata)
    {
        return ThroughStopwatch(func: (s :ICacheStore<TFlag> , m :ICacheStoreOperationMetadata ) => s.Get<T>(key, m), metadata);
    }

    0+16 usages  mrlldd
    public ValueTask<Result<T>> GetAsync<T>(string key, ICacheStoreOperationMetadata metadata,
        CancellationToken token = default)
    {
        return ThroughStopwatchAsync((s :ICacheStore<TFlag> , m :ICacheStoreOperationMetadata ) => s.GetAsync<T>(key, m, token), metadata);
    }
```

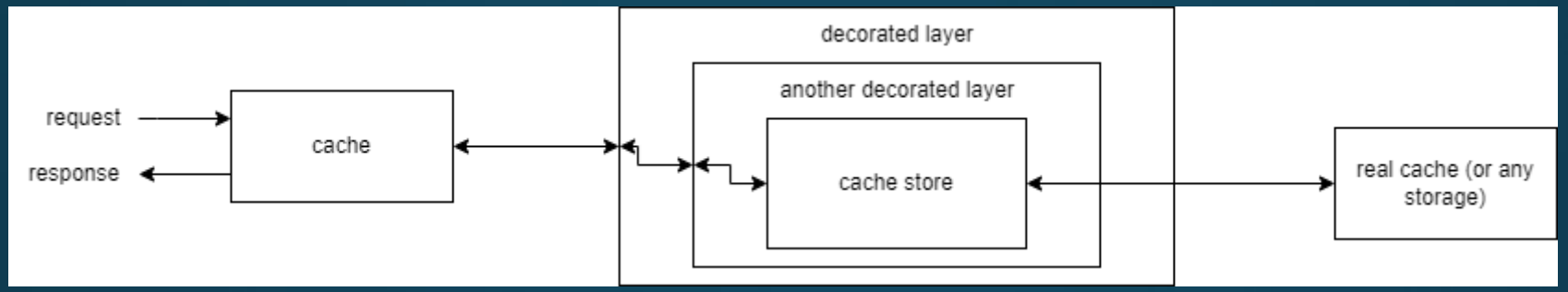
Factory

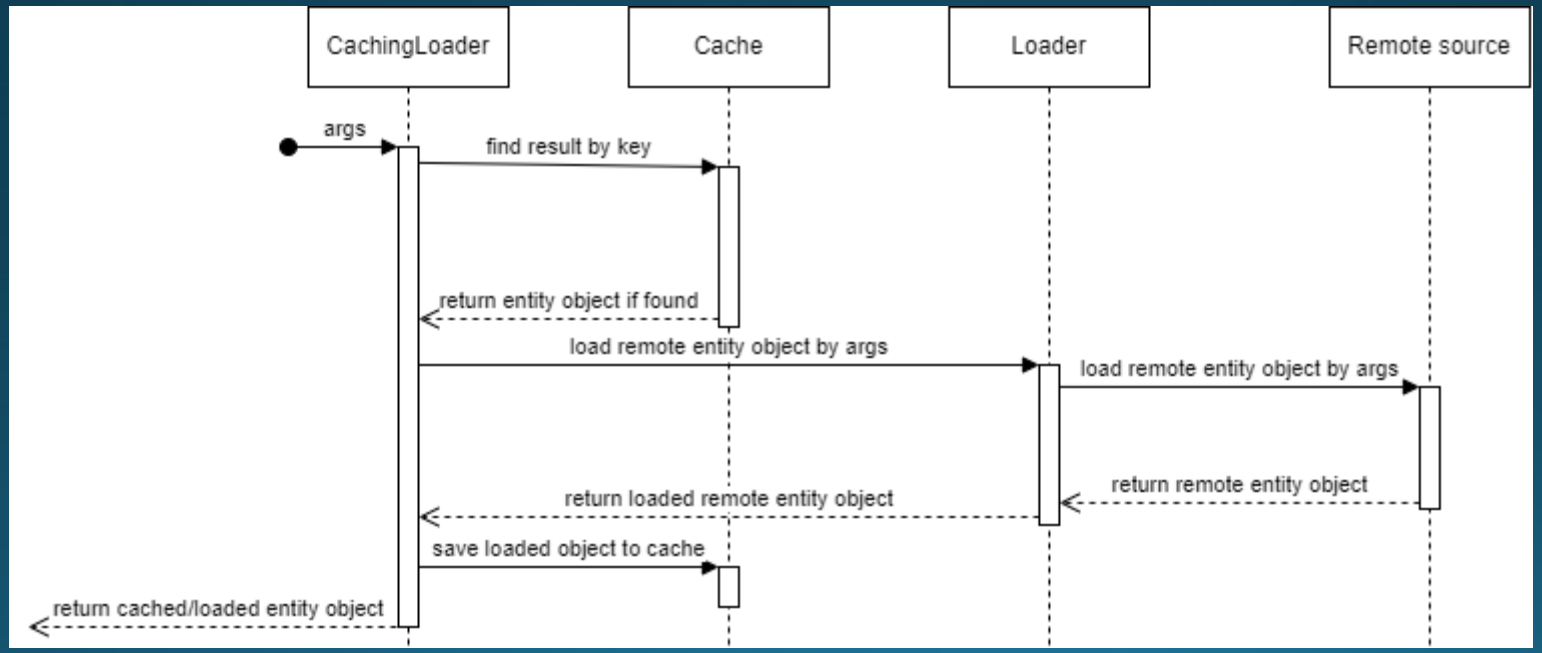
Категоризація ключів записів





Послідовність виконання запиту до кешу





Cache, CachingFlag та CacheStore

```
public sealed class InFile : CachingFlag
{
        private InFile()
        { }
}
```

Приклад реалізації CachingFlag

```
public class FileStore : ICacheStore<InFile>
{
        /*...*/
}
```

Приклад реалізації CachingStore

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddCaching(typeof(Startup).Assembly)
                .UseCachingStore<InFile, FileStore>();
}
```

Приклад додавання нового Store до контейнеру залежностей

```
public class ContextfulCache : Cache<UserContext, InDistributed>
{
        private readonly IUserContextProvider provider;

        protected override CachingOptions Options
                => CachingOptions.Enabled(TimeSpan.FromMinutes(20));

        protected override string CacheKey => "user-context";

        protected override string CacheKeySuffix => provider.SessionId;

        public ContextfulCache(IUserContextProvider provider)
                => this.provider = provider;
}
```

Приклад реалізації сервісу-кешу

```
public class IntCache : Cache<int, InMemory>
{
        protected override CachingOptions Options => CachingOptions.Enabled(TimeSpan.FromSeconds(15));
        protected override string CacheKey => "singleton-number";
}
```

Приклад реалізації сервісу-кешу з Singleton

CachingLoader

```
public class RemoteEntityLoader : ILoader<int, RemoteEntity>
{
        public async Task<RemoteEntity> LoadAsync(int args, CancellationToken token = default)
        {
                await Task.Delay(5000, token);
                return new RemoteEntity(args);
        }
}
```

Приклад реалізації Loader

```
public class RemoteEntityCachingLoader : CachingLoader<int, RemoteEntity, InMemory>
{
        protected override CachingOptions Options => CachingOptions.Enabled(TimeSpan.FromMinutes(5));
        protected override string CacheKey => nameof(RemoteEntity);

        protected override string CacheKeySuffixFactory(int args)
                => args.ToString();
}
```

Приклад реалізації «кешуючого» Loader

```
public class RemoteEntityStorage
{
    private readonly ICachingLoader<int, RemoteEntity, InMemory> cachingLoader;

    1 usage  new *
    public RemoteEntityStorage(ICachingLoader<int, RemoteEntity, InMemory> cachingLoader)
    {
        this.cachingLoader = cachingLoader;
    }

    1 usage  new *
    public async Task<RemoteEntity> LoadByIdAsync(int id, CancellationToken cancellationToken = default)
    {
        var loadingResult = await cachingLoader.GetOrLoadAsync(id, token: cancellationToken);
        if (!loadingResult.Successful)
        {
            throw loadingResult;
        }

        return loadingResult;
    }
}
```

Приклад використання «кешуючого» Loa

Декорація Store

```
public class WrappingDecorator<TStoreFlag> : ICacheStoreDecorator<TStoreFlag> where TStoreFlag : CachingFlag
{
        public ICacheStore<TStoreFlag> Decorate(ICacheStore<TStoreFlag> cacheStore)
                => new WrappedCacheStore<TStoreFlag>(cacheStore);

        public int Order => 2754;
}
```

Приклад реалізації Decorator

```
public class WrappedCacheStore<T>: ICacheStore<T> where T : CachingFlag
{
        /*...*/
}
```

Приклад реалізації «декорованого» Store

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddCaching(typeof(Startup).Assembly)
                .Decorators<InMemory>().Add<WrappingDecorator<InMemory>>()
                .Decorators<InDistributed>().Add<WrappingDecorator<InDistributed>>();
}
```

Додавання Decorator до контейнеру залежностей

```
public class RemoteEntityStorage
{
    private readonly ICache<RemoteEntity> cache;

    1 usage  new *
    public RemoteEntityStorage(ICache<RemoteEntity> cache)
    {
        this.cache = cache;
    }

    1 usage  new *
    public Task SetAsync(RemoteEntity remoteEntity, CancellationToken cancellationToken = default)
        => cache.SetAsync(remoteEntity, SequenceStrategy.Instance, cancellationToken);
}
```

Використання «безфлажних» кешів

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCaching(typeof(Startup).Assembly)
        .WithActionsLogging<InVoid>()
        .WithPerformanceLogging<InVoid>();
}
```

Додавання «логуючих» Decorator

Висновки

1. У результаті поєднання патернів об'єктно-орієнтованого та функціонального проектування створено концепцію модульної інтеграції кешів
2. Створено модель категоризованих ключів записів у кеші, що вирішує проблему дублювання та структурування ключів
3. На основі концепції модульної інтеграції кешів розроблено бібліотеку Caching у ASP .NET Core (C#) та «інтерфейс» для роботи з нею
4. Бібліотеку Caching повністю покрито Unit тестами та частково покрито Benchmark тестами, що доводить стабільність її використання у реальних web-додатках

ДЯКУЮ ЗА УВАГУ!