Hydrogen Sessions

#8 – Caching

Serviços de caching

A utilização de caching tem os seguintes benefícios:

- Reduz o processamento de CPU (reduzindo o processamento necessário para calcular determinado “valor” N vezes).
- Amortece a latência de rede (reduzindo o número de requests lentas necessárias para calcular determinado “valor”).
- Quando distribuída, essas vantagens são maiores entre “servidores” diferentes.

Mas tem potenciais problemas:

- Em memória, pode introduzir cenários de concorrência (entre threads).
- Em memória, introduz cenários de garbage collection mais complexos.
- Distribuída, pode introduzir instabilidade porque as cache REDIS são muito permeáveis a erros transientes.

O .NET tem essencialmente dois serviços de caching (complementares):

- IMemoryCache – cache em memória.
- IDistributedCache – cache distribuída.

No Hydrogen, há 2 componentes adicionais:

- IResilientDistributedCache – adiciona resiliência a IDistributedCache.
- Memoizer – cache estática em memória para resultados de funções (https://en.wikipedia.org/wiki/Memoization).

IDistributedCache

Este serviço é tipicamente registado usando uma implementação REDIS.

O Hydrogen tem um método de extensão para o registar que inclui opções para configurar a instância da cache:

```
/// <summary>
/// Adds the REDIS distributed caching services to the service collection.
/// </summary>
/// <param name="services">The service collection.</param>
/// <returns>
/// The <see cref="IServiceCollection"/> instance.
/// </returns>
7 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
public static IServiceCollection AddRedisCache(this IServiceCollection services)
{
    // Validation

    SmartGuard.NotNull(() => services, services);

    // Services

    services
        .AddLogging();

    // Configuration

    services
        .AddOptionsSnapshot<RedisCacheOptions>();

    // Resolve options to map to the implementation options

    IServiceProvider provider = services.BuildServiceProvider();
    RedisCacheOptions options = provider.GetRequiredService<RedisCacheOptions>();

    // Add service

    services
        .AddStackExchangeRedisCache(
            (redisOptions) =>
            {
                redisOptions.Configuration = options.ConnectionString;
                redisOptions.InstanceName = options.InstanceName;
            });

    // Result

    return services;
}
```

O método AddStackExchangeRedisCache é fornecido por Microsoft.Extensions.Caching.StackExchangeRedis:

```
/// <summary>
public static IServiceCollection AddStackExchangeRedisCache(this IServiceCollection services, Action<RedisCacheOptions> setupAction)
{
    if (services == null)
    {
        throw new ArgumentNullException("services");
    }
    if (setupAction == null)
    {
        throw new ArgumentNullException("setupAction");
    }
    services.AddOptions();
    services.Configure(setupAction);
    services.Add(ServiceDescriptor.Singleton<IDistributedCache, RedisCache>());
    return services;
}
```

REDIS cache

Com o método anterior, é muito fácil adicionar uma cache REDIS à aplicação. Para além de registar o serviço dessa forma, bastará definir a connection string por configuração, como por exemplo:

```
"RedisCacheOptions": {
    "ConnectionString": "elevation-dev.redis.cache.windows.net:6380,password=DTRj8hsIM5FVoc7kb/uu4gemQqrLbPGy/H16aR4QyKQ=,ssl=True,abortConnect=False"
},
```

É importante ter em atenção que as caches REDIS, por natureza, são muito suscetíveis a erros transientes – timeouts na conexão ou mesmo nas operações – pelo que é fundamental que as aplicações estejam preparadas para:

- Lidar com esses erros transientes com políticas de retry.
- Lidar com um fallback em que um erro na cache seja tratado pela aplicação da mesma forma que é tratado o caso em que o item não existe na cache (por exemplo).

Nos nossos serviços do Azure, estes erros transientes são muito comuns e são extremamente difíceis de diagnosticar. No caso mais recente foi necessário alterar as timeouts em todas as connection strings.

https://stackexchange.github.io/StackExchange.Redis/Configuration.html

https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-faq

https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-troubleshoot-client

https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-troubleshoot-timeouts#stackexchangeredis-timeout-exceptions

Outro aspeto importante é que os items não devem ser muito grandes (quantidade de bytes). A performance da cache degrada-se muito nesse caso.

Resilient Cache

O serviço IResilientDistributedCache fornece as ferramentas para adicionar esses comportamentos desejáveis às aplicações:

- Implementa uma política de retry para os erros transientes.
- Permite ao cliente indicar que os erros devem ser ignorados.

Do ponto de vista da aplicação, o serviço deve ser registado e depois utilizado da seguinte forma:

```
// REDIS cache (resilient)

services
    .AddRedisCache();

services
    .AddResilientDistributedCache();
```

```
using Microsoft.Extensions.Logging;
using Primavera.Hydrogen;
using Primavera.Hydrogen.Caching.Distributed.Resilience;
using Primavera.Hydrogen.Net.Email;
using Primavera.Hydrogen.TextMessaging;
```

```
1 reference | André Gonçalves, 4 days ago | 2 authors, 2 changes | 1 incoming change
private async Task<bool> AcquireLockAsync()
{
    if (!this.fieldLocked)
    {
        string lockExists = await this.DistributedCache.GetStringAsync(
            LockCacheKey,
            false)
            .ConfigureAwait(false);

        if (!lockExists.EqualsNoCase("true"))
        {
            await this.DistributedCache.SetStringAsync(
                LockCacheKey,
                "true",
                GetCacheOptions(),
                false)
                .ConfigureAwait(false);

            this.fieldLocked = true;
            return true;
        }
    }

    return false;
}
```

Note-se:

- O using, necessário para os métodos de extensão a IDistributedCache ficarem disponíveis.
- O valor do parâmetro “throwErrors” a false na invocação desses métodos de extensão.

Resilient Cache

O serviço e os métodos de extensão estão implementados em Primavera.Hydrogen.Caching.Distributed.

```
public static IServiceCollection AddResilientDistributedCache(this IServiceCollection services)
{
    // Validation

    SmartGuard.NotNull(() => services, services);

    // Services

    services
        .AddLogging();

    // Configuration

    services
        .AddOptionsSnapshot<ResilientCacheOptions>();

    // Ensure the original service registration

    services
        .AddDistributedMemoryCache();

    // Decorate original service

    services
        .AddTransientDecorator<IDistributedCache, ResilientDistributedCache>();

    // Result

    return services;
}
```

```
/// <inheritdoc />
[SuppressMessage("Microsoft.Naming", "CA1716:IdentifiersShouldNotMatchKeywords", Justification = "Imposed by IDistributedCache design.")]
1 reference | Hugo Ribeiro, 46 days ago | 2 authors, 2 changes | 1 incoming change
public virtual byte[] Get(string key)
{
    // Handle exceptions

    try
    {
        // Execute decorated service

        return this.RetryPolicy
            .Execute(
                () => this.Decorator.Instance.Get(key));
    }
    catch (Exception ex)
    {
        // Logging
        // Do not log exception because it will end up in the exceptions of AI (not only in the trace)

        this.Logger.LogWarning($"{nameof(IDistributedCache)} instance raised exception. Exception: {ex.GetType().FullName} - {ex.Message}.");

        // Throw

        throw;
    }
}
```

```
public static byte[] Get(this IDistributedCache cache, string key, bool throwErrors)
{
    // Validation

    SmartGuard.NotNull(() => cache, cache);
    SmartGuard.NotNullOrEmpty(() => key, key);

    // Execute

    try
    {
        byte[] result = cache.Get(key);
        if (result != null)
        {
            return result;
        }
    }
    catch (RedisTimeoutException)
    {
        // Re-throw?

        if (throwErrors)
        {
            throw;
        }
    }

    return default;
}
```

Caching e pipelines

O típico cenário de caching é algo teste tipo:

1. O "valor" existe em cache?
2. Se sim, retornar o valor da cache
3. Se não, determinar o valor
4. (processamento lento)
5. Adicionar o valor à cache para estar disponível na próxima execução

Este tipo de construção acaba por produzir muito código repetitivo e adapta-se particularmente bem ao comportamento suporto pelas Pipelines (e que é usado em praticamente todos os micro serviços).

```
/// <inheritdoc />
26 references | André Gonçalves, 4 days ago | 3 authors, 3 changes | 1 incoming change
async Task IPipelineHandler<GetEmailTemplateContext>.InvokeAsync(GetEmailTemplateContext context, PipelineDelegate<GetEmailTemplateContext> next)
{
    // NOTE:
    // This runs BEFORE the storage handler

    // Retrieve

    EmailTemplateData emailTemplate = await this.GetFromCacheAsync<EmailTemplateData>(
        context.Id)
        .ConfigureAwait(false);

    if (emailTemplate != null)
    {
        // Logging

        this.Logger.LogDebug($"Email template retrieved from cache.");

        // Set result

        context.Result = OperationResult<EmailTemplateData>.Success(emailTemplate);

        // Return

        return;
    }

    // Call next handler

    await next.Invoke(context).ConfigureAwait(false);

    // Add

    await this.SetInCacheAsync(
        context.Id,
        context.Result.Data)
        .ConfigureAwait(false);

    // Logging

    this.Logger.LogDebug($"Email template saved in cache.");
}
```

https://github.com/PrimaveraDeveloper/lithium/blob/master/ref/hydrogen-2.0/Pipelines.md

Caching e table storage

Há uma tendência natural para associar caching ao serviço de table storage (ou outros serviços de storage) com o intuito de reduzir os “round trips” de rede à storage. Mas isso deve ser ponderado com mais atenção.

Ambos os serviços – REDIS cache e Table Storage – são serviços acedidos através da rede no Azure (ver connection strings). Embora em teoria a comunicação com a instância REDIS seja TCP e por isso seja mais rápida que o acesso à table storage (HTTP), é preciso considerar outras circunstâncias:

- O processamento na table storage, quando as queries são sobre a partition key e a row key, é extremamente rápido. Provavelmente, neste caso, a instabilidade e o processamento adicional introduzidos pela cache não se justificam.
- Isso não é verdade, obviamente, quando as queries são sobre os dados não indexados.
- A cache REDIS tem a vantagem óbvia de ser distribuída.
- Deve considerar-se a vantagem de usar uma cache distribuída no lugar de uma em memória (em que a instabilidade é muito menor e o processamento muito mais rápido).

Boas práticas

1. Usar sempre o serviço IResilientDistributedCache.
2. Usar, sempre que possível, as extensões disponibilizadas no namespace Primavera.Hydrogen.Caching.Distributed.Resilience (throwErros = false).
3. Ponderar o uso de IDistributedCache ou IMemoryCache.
4. Ponderar o uso de pipelines (aceitam-se ideias para tornar a implementação de pipelines para caching menos “verbosas”!).
5. Garantir sempre a unicidade das chaves de cache (o CMS partilha muitas vezes a mesma cache por aplicações diferentes).
6. Reduzir ao máximo o tamanho dos items de cache.
7. Garantir resiliência lógica sobre a cache (a aplicação não pode falhar se a cache falhar, mas deve permitir diagnosticar facilmente esses casos).