Hydrogen Sessions

#4 – Configuração

Configuração nas aplicações

Importa começar pela forma como o sistema de configuração é inicializado nas aplicações (micro serviços).

O arranque da aplicação é controlado pelo HostBuilder gerado (usado para facilitar a geração de código e a sua customização, se necessário):

```
/// <summary>
/// Executes the application.
/// </summary>
/// <param name="args">The arguments.</param>
0 references | Hugo Lourenço | 1 author, 1 change
public static void Main(string[] args)
{
    // Create the host builder

    HostBuilder builder = new HostBuilder();

    // Build the host

    IHost host = builder.Build(args);

    // Run

    host.Run();
}
```

```
/// <summary>
/// Builds the <see cref="IHost" /> instance.
/// </summary>
/// <param name="args">The arguments.</param>
/// <returns>
/// The <see cref="IHost" /> instance.
/// </returns>
1 reference | Hugo Lourenço | 1 author, 1 change
public virtual IHost Build(string[] args)
{
    // Create the builder

    IHostBuilder builder = Host.CreateDefaultBuilder(args)
        .UseContentRoot(Directory.GetCurrentDirectory());

    // Setup host configuration

    builder = builder.ConfigureHostConfiguration(this.ConfigureHostConfiguration);

    // Setup application configuration

    builder = builder.ConfigureAppConfiguration(this.ConfigureAppConfiguration);

    // Setup logging

    builder = builder.ConfigureLogging(this.ConfigureLogging);

    // Setup Web host defaults

    builder = builder.ConfigureWebHostDefaults(this.ConfigureWebHostDefaults);

    // Result

    return builder.Build();
}
```

Configuração nas aplicações

Para a configuração, o método que interessa é ConfigureAppConfiguration().

Note-se que a configuração pode ser lida dos ficheiros JSON (pela ordem especificada, que permite “overrides” por ambiente), de variáveis de ambiente e de um KeyVault.

```
/// <summary>
/// Configures the application configuration.
/// </summary>
/// <param name="context">The host builder context.</param>
/// <param name="builder">The configuration builder.</param>
1 reference | Hugo Ribeiro, 97 days ago | 2 authors, 3 changes
protected virtual void ConfigureAppConfiguration(HostBuilderContext context, IConfigurationBuilder builder)
{
    // Environment

    IHostEnvironment env = context.HostingEnvironment;

    // Normalize environment name
    // NOTE: File names are case sensitive in Linux

    string environmentName = env.EnvironmentName.Transform().ToLowerCase();

    // JSON configuration

    builder
        .AddJsonFile($"GeneratedCode/appsettings.gen.json", optional: false, reloadOnChange: true)
        .AddJsonFile($"CustomCode/appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"GeneratedCode/appsettings-{environmentName}.gen.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"CustomCode/appsettings-{environmentName}.json", optional: true, reloadOnChange: true);

    // Environment variables configuration

    builder
        .AddEnvironmentVariables();

    // Secrets storage

    builder
        .AddAzureSecretsStorage();
}
```

Configuração nas aplicações

No arranque da aplicação propriamente dito, ocorre o passo “ConfigureServices”, onde são adicionadas as secções de configuração, logo no início.

```
/// <summary>
/// Called by the runtime when the application starts to allow the application to configure its services.
/// </summary>
/// <param name="services">The services collection.</param>
12 references | Hugo Ribeiro, 44 days ago | 2 authors, 4 changes
public virtual void ConfigureServices(IServiceCollection services)
{
    // Validation

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

    // Configuration

    HostConfiguration hostConfiguration = this.AddConfiguration(services);

    // Validate host configuration

    this.ValidateConfiguration(hostConfiguration);

    // Add cookie policy

    this.AddCookiePolicy(services, hostConfiguration);

    // MVC

    this.AddMvc(services, hostConfiguration);
```

```
/// <summary>
/// Called to add configuration to the service collection.
/// </summary>
/// <param name="services">The service collection.</param>
/// <returns>
/// An instance of <see cref="HostConfiguration"/> that can be used in the application c
/// </returns>
/// <remarks>
/// The method is called from <see cref="ConfigureServices(IServiceCollection)"/>.
/// </remarks>
5 references | Hugo Ribeiro, 23 days ago | 2 authors, 5 changes
protected virtual HostConfiguration AddConfiguration(IServiceCollection services)
{
    // Validation

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

    // Common options

    services
        .AddOptions()
        .Configure<HostConfiguration>(
            this.Configuration.GetSection(nameof(HostConfiguration)))
        .Configure<AzureInsightsTelemetryOptions>(
            this.Configuration.GetSection(nameof(AzureInsightsTelemetryOptions)));

    // REDIS cache options

    services
        .Configure<RedisCacheOptions>(
            this.Configuration.GetSection(nameof(RedisCacheOptions)))
        .Configure<RedisCacheOptions>(
            this.Configuration.GetSection(nameof(ResilientCacheOptions)));

    // Table storage options
```

Note-se como são adicionadas as secções (para fazerem match com os ficheiros JSON) e que praticamente todas as a secções têm o sufixo Options (exceção é a classe HostConfiguration, que não tem esse sufixo por razão de retrocompatibilidade e porque agora implicaria muitas alterações no código custom para corrigir).

IConfiguration, IOptions, IOptionsSnapshot

Podem entender em detalhe o funcionamento do sistema de configuração:

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration

Em particular o padrão das Options:

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options

Porque as secções de configuração são inicializadas com esse padrão, optou-se por criar um mecanismo no Hydrogen que permita injetar a classe diretamente (AzureTableStorageOptions) em vez das options (IOptions<AzureTableStorageOptions>) e permita suportar o conceito de snapshot (no link anterior). É usado, por exemplo, para a HostConfiguration, mas também para as classes de configuração do Hydrogen.

```
// Host configuration snapshot

services
    .AddOptionsSnapshot<HostConfiguration>();
```

```
// Configuration

services
    .AddOptionsSnapshot<AzureInsightsTelemetryOptions>();
```

NOTA: A solução Hydrogen inclui uma aplicação (WebApplication) que tem uma sample que ilustra aspetos da configuração.

AddOptionsSnapshot

Veja-se o que é método faz, ou seja, basicamente registar um serviço adicional – a classe de configuração – que resolve o respetivo IOptionsSnapshot e devolve o seu valor:

```
public static IServiceCollection AddOptionsSnapshot<TOptions>(this IServiceCollection services)
    where TOptions : class, new()
{
    // Validation

    if (services == null)
    {
        return null;
    }

    // Add transient

    services
        .AddOptions()
        .AddTransient(
            (provider) =>
            {
                using (IServiceScope scope = provider.CreateScope())
                {
                    return scope.ServiceProvider.GetRequiredService<IOptionsSnapshot<TOptions>>().Value;
                }
            });

    // Result

    return services;
}
```

Exemplo (AzureTableStorageOptions)

Para ilustrar como estas classes de configuração devem ser criadas e utilizadas, considere-se o exemplo do serviço ITableStorageService.

A classe de configuração/opções é um POCO simples:

```
/// <summary>
/// Defines the options used to configure new <see cref="AzureTableStorageService"/> instances.
/// </summary>
45 references | Hugo Ribeiro, 137 days ago | 2 authors, 2 changes | 1 incoming change
public partial class AzureTableStorageOptions
{
    #region Public Properties

    /// <summary>
    /// Gets or sets the connection string to connect to the storage service.
    /// </summary>
    18 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
    public string ConnectionString
    {
        get;
        set;
    }

    /// <summary>
    /// Gets or sets the timeout time when establishing a connection to the storage server.
    /// This setting is optional and the default value is 5 seconds.
    /// </summary>
    4 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
    public TimeSpan ServerConnectionTimeout
    {
        get;
        set;
    }

    /// <summary>
    /// Gets or sets the maximum execution time of the storage server requests (including all retries).
    /// This setting is optional and the default value is 60 seconds.
    /// </summary>
    8 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
    public TimeSpan MaximumExecutionTime
    {
        get;
        set;
    }
```

Exemplo (AzureTableStorageOptions)

Depois utilizada – configurada – nos métodos de extenção de IServiceCollection:

```
/// <summary>
/// Adds a service that allows creating instances of <see cref="ITableStorageService"/> using
/// configuration defined by <see cref="AzureTableStorageOptions"/>.
/// This service will use Azure Table Storage.
/// </summary>
/// <param name="services">The service collection.</param>
/// <returns>
/// The <see cref="IServiceCollection"/> instance.
/// </returns>
/// <remarks>
/// <see cref="ITableStorageService"/> is registered as a transient service.
/// </remarks>
12 references | Hugo Ribeiro, 23 days ago | 2 authors, 2 changes | 1 incoming change
public static IServiceCollection AddAzureTableStorage(this IServiceCollection services)
{
    // Validation

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

    // Logging

    services
        .AddLogging();

    // Configuration

    services
        .AddOptionsSnapshot<AzureTableStorageOptions>();

    // Add transient service

    services
        .TryAddTransient<ITableStorageService, AzureTableStorageService>();

    // Result

    return services;
}
```

As options podem também ser definidas/modificadas com um delegate, mas esse é um tema que será abordado quando se falar do tema da construção de serviços.

Boas práticas

#1 - Nome da classe

A classe deve ter um nome significativo, mas simples (porque vai ser necessário adicioná-lo nos ficheiros de app settings) e deve ter sempre o sufixo Options (prática utilizada pelas classes de options do .NET Core).

#2 - Namespace

O namespace deve ser o mesmo que o componente configurado.

Quando as opções forem de um serviço, o ficheiro deve ser adicionado na pasta DependencyInjection, junto dos métodos de extenção que adicionam o serviço.