Hydrogen Sessions

#10 – Background Services

Hosted services

O .NET Core disponibiliza o conceito de hosted services (IHostedService). Este conceito pode ser utilizado para:

- Realizar processamento assíncrono no arranque da aplicação.
  https://andrewlock.net/running-async-tasks-on-app-startup-in-asp-net-core-part-1/
- Implementar long-running tasks.
  https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services

A principal particularidade deste tipo de serviços é que eles são "reconhecidos" pelo host e o seu tempo de vida é gerido devidamente, nomeadamente, lançando-lhes um "sinal" sempre que seja necessário que parem o processamento (quando a aplicação está a terminar).

Isto é bem visível no interface de IHostedService:

```
Task StartAsync(CancellationToken cancellationToken);
Task StopAsync(CancellationToken cancellationToken);
```

BackgroundService

O tipo Primavera.Hydrogen.AspNetCore.Hosting.BackgroundService define uma classe base simples para construir background services.

A API é a seguinte:

```
public virtual Task StartAsync(CancellationToken cancellationToken);
public virtual async Task StopAsync(CancellationToken cancellationToken);
protected abstract Task ExecuteAsync(CancellationToken cancellationToken);
```

```
24 references | Hugo Lourenço | 1 author, 1 change | 1 incoming change
public virtual Task StartAsync(CancellationToken cancellationToken)
{
    // Logging

    this.Logger.LogDebug($"Starting the {this.Name} background service...");

    // Start the execution task and store it

    this.fieldExecutingTask = this.ExecuteAsync(this.fieldCancellationTokenSource.Token);

    // If the task is completed, return it
    // This is not expected and will forward cancellation and failure to the caller

    if (this.fieldExecutingTask.IsCompleted)
    {
        return this.fieldExecutingTask;
    }

    // The task is running

    return Task.CompletedTask;
}
```

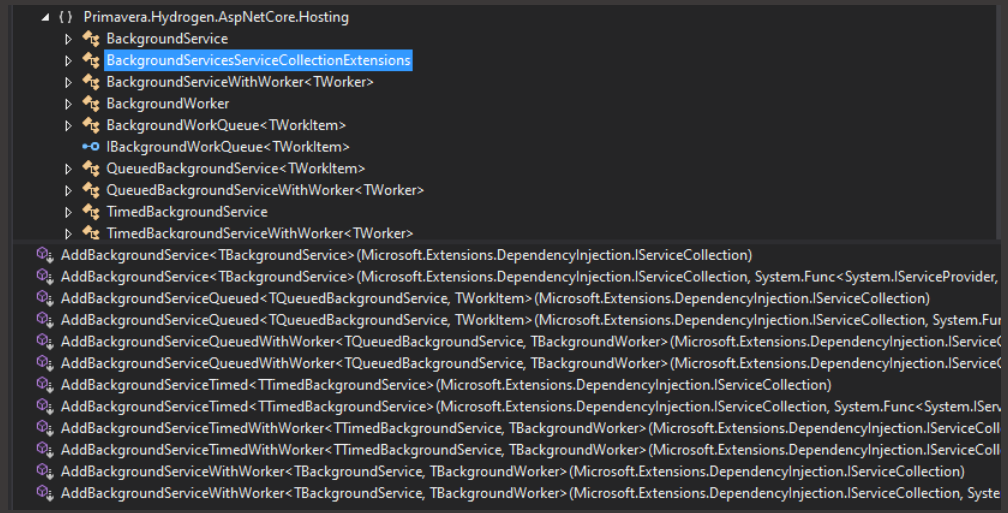
```
public virtual async Task StopAsync(CancellationToken cancellationToken)
{
    // Logging

    this.Logger.LogDebug($"Stopping the {this.Name} background service...");

    // If there is no executing task, than the background service was not started

    if (this.fieldExecutingTask == null)
    {
        return;
    }

    try
    {
        // Signal cancellation to the execution task

        this.fieldCancellationTokenSource.Cancel();
    }
    finally
    {
        // Wait until the task completes or the cancel token triggers

        await Task.WhenAny(this.fieldExecutingTask, Task.Delay(Timeout.Infinite, cancellationToken)).ConfigureAwait(false);
    }
}
```

Note-se que a implementação assegura a execução de uma task (o método ExecuteAsync) e a lógica de cancelamento a partir do StopAsync().

O método ExecuteAsync é abstrato para que as classes derivadas sejam “obrigadas” a implementar a lógica específica de cada serviço.

Registo dos background services

O Hydrogen também disponibiliza vários métodos para registar os background services:




```
public static IServiceCollection AddBackgroundService<TBackgroundService>(this IServiceCollection services)
    where TBackgroundService : BackgroundService
{
    // Validation

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

    // Ensure services

    services.AddLogging();

    // Add hosted service

    services.AddHostedService<TBackgroundService>();

    // Result

    return services;
}
```

No fim todos usam AddHostedService, de Microsoft.Extensions.Hosting.Abstractions.

NOTA: Embora o serviço seja registado como transiente, o seu tempo de vida é gerido como se tratasse de um singleton.

```
// Copyright (c) .NET Foundation. All rights reserved.
// Licensed under the Apache License, Version 2.0. See License.txt in the project root for license information.

using Microsoft.Extensions.Hosting;

namespace Microsoft.Extensions.DependencyInjection
{
    public static class ServiceCollectionHostedServiceExtensions
    {
        /// <summary>
        /// Add an <see cref="IHostedService"/> registration for the given type.
        /// </summary>
        /// <typeparam name="THostedService">An <see cref="IHostedService"/> to register.</typeparam>
        /// <param name="services">The <see cref="IServiceCollection"/> to register with.</param>
        /// <returns>The original <see cref="IServiceCollection"/>.</returns>
        public static IServiceCollection AddHostedService<THostedService>(this IServiceCollection services)
            where THostedService : class, IHostedService
            => services.AddTransient<IHostedService, THostedService>();
    }
}
```

Outros tipos de background service

O Hydrogen disponibiliza ainda outras classe base para construir background services:

- BackgroundServiceWithWorker<TWorker> – um BackgroundService em que o trabalho (ExecuteAsync) é realizado por uma instância de BackgroundWorker.
- TimedBackgroundService – um BackgroundService que executa indefinidamente (até ser cancelado) e invoca ExecuteWorkAsync() sempre que um período de tempo passa.
- TimedBackgroundService<TWorker> – um TimedBackgroundService em que o trabalho (ExecuteWorkAsync) é realizado por uma instância de BackgroundWorker.
- QueuedBackgroundService<TWorkItem> – um BackgroundService que executa indefinidamente e que invoca ExecuteWorkAsync() sempre que um item fica disponível numa queue (IBackgroundWorkQueue<TWorkItem>).
- QueuedBackgroundServiceWithWorker<TWorker> – um TimedBackgroundService<TWorker> em que os items da queue são instâncias do BackgroundWorker que deve ser executado.

Estes conceitos de TimedBackgroundService, QueuedBackgroundService e BackgroundWorker permitem suportar todos os cenários necessários às aplicações:

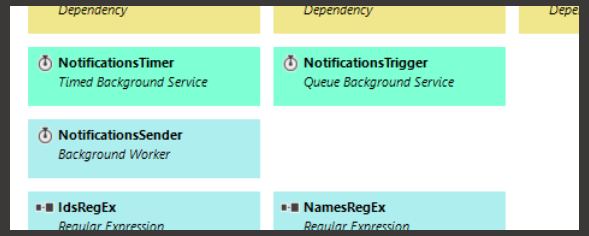
- Inicialização da aplicação – background service simples.
- Execução repetitiva de tarefas (ex.: envio de notificações) – timed background service.
- Reação rápida a “eventos” (ex.: envio de notificações urgentes) – queued background services.
- Mix dos dois anteriores (ex.: serviço de notificações) – timed e queued background services com um background worker partilhado.

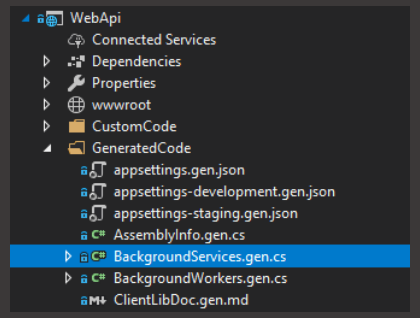
Service Designer

O Service Designer permite modelar exatamente todos esses tipos de background services.

A forma como o código é gerado para os serviços modelados, assegura que o developer implementa apenas o código absolutamente imprescindível.

NOTA: Este é um aspeto que pode melhorar no futuro, introduzindo nas classes base e/ou no código gerado determinados padrões (exemplo: reação ao stop, tratamento de erros, locks entre hosts diferentes, etc.).





Exemplo: background service simples

Uma operação comum nos micro serviços é configurar a table storage – criar as tabelas necessárias – quando a aplicação arranca pela primeira vez (e fazer upgrades depois, quando necessário). Isto é muito fácil de fazer com um background service.

Este exemplo está no serviço DLS (serviço ServiceSearch) (neste caso inicializa o serviço de search).

```
/// <summary>
/// Defines the base class for the <see cref="ServiceSearchService" /> background se
/// </summary>
[GeneratedCode("Lithium", "2.0")]
[SuppressMessage("Maintainability Rules", "SA1402:FileMayOnlyContainASingleType", Ju
3 references | Hugo Lourenço | 1 author, 1 change
internal abstract partial class ServiceSearchServiceBase : BackgroundService
{
    Code
}
```

```
/// <inheritdoc />
[SuppressMessage("Design", "CA1031:Do not catch general exception types", Justification
0 references | Hugo Lourenço | 1 author, 1 change
protected override async Task ExecuteAsync(CancellationToken cancellationToken)
{
    try
    {
        // Setup the search services

        await this.SetupSearchAsync(cancellationToken).ConfigureAwait(false);
    }
    catch (Exception ex)
    {
        this.Logger.LogError(ex, $"Search background service raised an exception.");
    }
}
```

```
1 reference | Hugo Lourenço | 1 author, 1 change
private async Task SetupSearchAsync(CancellationToken cancellationToken)
{
    // Logging

    this.Logger.LogDebug($"Configuring the search services...");

    // Create the data sources

    await this.CreateDataSourceAsync(
        SearchConstants.DataSources.VatNumbersIDB,
        StorageConstants.Tables.VatNumbersIDB.Name,
        cancellationToken)
        .ConfigureAwait(false);

    // Canceled?

    if (cancellationToken.IsCancellationRequested)
    {
        return;
    }

    // Create the indexes

    await this.CreateIndexAsync(
        SearchConstants.Indexes.VatNumbersIDB,
        GetVatNumbersIDBTableFields(),
        cancellationToken)
        .ConfigureAwait(false);
```

Exemplo: timed background service

O serviço NotificationsTimer no micro serviço NS, pesquisa uma table storage periodicamente para determinar as notificações a enviar.

```
/// <summary>
/// Defines the base class for the <see cref="NotificationsTimerService" /> background service.
/// </summary>
[GeneratedCode("Lithium", "2.0")]
[SuppressMessage("Maintainability Rules", "SA1402:FileMayOnlyContainASingleType", Justification = "Because of code generation design.")]
3 references | André Gonçalves, 7 days ago | 3 authors, 5 changes | 2 incoming changes
internal abstract partial class NotificationsTimerServiceBase : TimedBackgroundServiceWithWorker<NotificationsSenderWorker>
{
    Code
}
```

```
[SuppressMessage("StyleCop.CSharp.DocumentationRules", "SA1601: PartialElementsMustBeDocumented")]
[SuppressMessage("Microsoft.Performance", "CA1812:AvoidUninstantiatedInternalClasses", Justificati
6 references | Hugo Ribeiro, 3 days ago | 3 authors, 5 changes | 1 incoming change
internal partial class NotificationsTimerService
{
    Private Properties

    #region Public Properties

    /// <inheritdoc />
    0 references | André Gonçalves, 7 days ago | 2 authors, 2 changes | 1 incoming change
    public override TimeSpan WaitPeriod
    {
        get
        {
            return TimeSpan.FromSeconds(this.HostConfiguration.WorkerWaitInterval);
        }
    }

    /// <inheritdoc />
    0 references | André Gonçalves, 7 days ago | 2 authors, 2 changes | 1 incoming change
    public override NotificationsSenderWorker Worker
    {
        get
        {
            return new NotificationsSenderWorker(
                this.ServiceProvider,
                this.ServiceProvider.GetRequiredService<ILogger<NotificationsSenderWorker>>());
        }
    }
```

Exemplo: queued background service

O serviço NotificationsTrigger no micro serviço NS, reage a uma background queue para enviar notificações urgentes (que não podem esperar pelo timer).

```
/// <summary>
/// Defines the base class for the <see cref="NotificationsTriggerService" /> background service.
/// </summary>
[GeneratedCode("Lithium", "2.0")]
[SuppressMessage("Maintainability Rules", "SA1402:FileMayOnlyContainASingleType", Justification = "Because of code generation de
3 references | André Gonçalves, 7 days ago | 3 authors, 5 changes | 2 incoming changes
internal abstract partial class NotificationsTriggerServiceBase : QueuedBackgroundServiceWithWorker<NotificationsSenderWorker>
{
    Code
}
```

```
[SuppressMessage("StyleCop.CSharp.DocumentationRules", "SA1601: PartialElementsMustBeDocumented")]
[SuppressMessage("Microsoft.Performance", "CA1812:AvoidUninstantiatedInternalClasses", Justification = "Dependen
6 references | André Gonçalves, 7 days ago | 2 authors, 2 changes | 1 incoming change
internal partial class NotificationsTriggerService
{
    #region Public Properties

    /// <inheritdoc />
    0 references | André Gonçalves, 7 days ago | 2 authors, 2 changes | 1 incoming change
    public override IBackgroundWorkQueue<NotificationsSenderWorker> Queue
    {
        get
        {
            return this.ServiceProvider.GetRequiredService<IBackgroundWorkQueue<NotificationsSenderWorker>>();
        }
    }

    #endregion
}
```

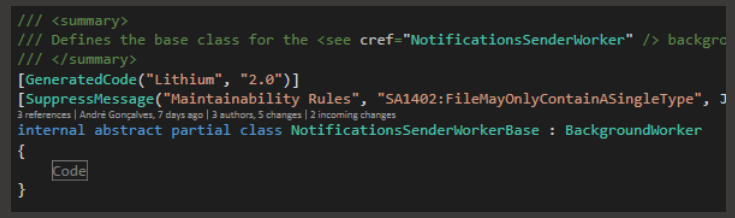
```
// High priority?

if (emailTemplate.SendPriority == SendPriority.High)
{
    // Enqueue

    this.WorkerQueue.Enqueue(
        new NotificationsSenderWorker(
            this.ServiceProvider,
            this.ServiceProvider.GetRequiredService<ILogger<NotificationsSenderWorker>>()));

    // Logging

    this.Logger.LogDebug($"Email notification enqueued for high priority processing.");
}
```

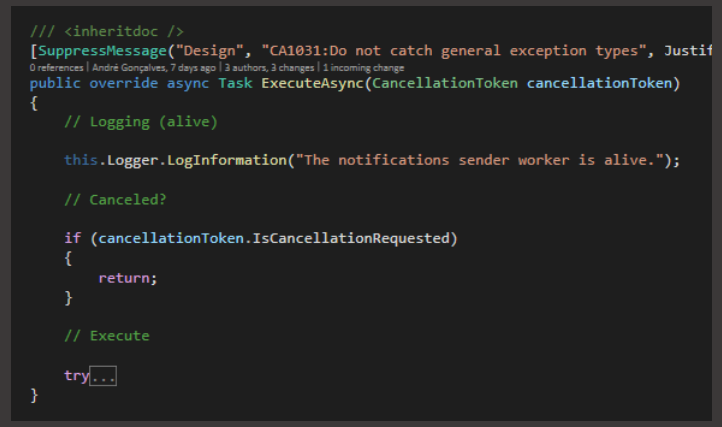
O trabalho é colocado na queue a partir de qualquer ponto da aplicação, através do serviço IBackgroundWorkQueue<NotificationsSenderWorker> (neste caso).

Exemplo: timed e queued com background worker

Os dois serviços partilham o mesmo worker e é assim que se pode facilmente combinar o comportamento de quaisquer dois background services (com workers).

O background worker é um tipo bastante simples.





Locking

Um dos cenários que é preciso considerar ao utilizar background services tem a ver com o escalonamento horizontal do host da aplicação (mais servidores).

Isto na prática significa ter N aplicações em paralelo, logo N background services em paralelo. Se eles partilharem o mesmo recurso (exemplo: a table storage) vão surgir os problemas de concorrência imediatamente.

Locking

Neste momento, o Hydrogen não fornece nenhuma funcionalidade para lidar com este problema e deixa ao developer a opção pelo melhor mecanismo. Nos serviços atuais, temos utilizado um mecanismo simples de pessimistic locking com cache REDIS. Funciona mas é obviamente ineficiente, para além de ter outros defeitos.

```
/// <inheritdoc />
[SuppressMessage("Design", "CA1031:Do not catch general exception types", Justification = "Required by design.")]
0 references | André Gonçalves, 7 days ago | 3 authors, 3 changes | 1 incoming change
public override async Task ExecuteAsync(CancellationToken cancellationToken)
{
    // Logging (alive)

    this.Logger.LogInformation("The notifications sender worker is alive.");

    // Canceled?

    if (cancellationToken.IsCancellationRequested)...

    // Execute

    try
    {
        // Acquire lock (pessimistic)

        if (await this.AcquireLockAsync().ConfigureAwait(false))...
        else
        {
            // Logging (alive)

            this.Logger.LogInformation("Another worker is performing work.");
        }

        // Canceled?

        if (cancellationToken.IsCancellationRequested)...
    }
    catch (Exception ex)
    {
        // Logging

        this.Logger.LogError(ex, $"The notifications sender worker raised exception: '{ex.Message}'. Continuing...");

        // Run forever...
    }
    finally
    {
        // Release lock

        await this.ReleaseLockAsync().ConfigureAwait(false);
    }
}
```

```
1 reference | André Gonçalves, 7 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;
}
```

```
private async Task ReleaseLockAsync()
{
    if (this.fieldLocked)
    {
        await this.DistributedCache.SetStringAsync(
            LockCacheKey,
            "false",
            GetCacheOptions(),
            false)
            .ConfigureAwait(false);

        this.fieldLocked = false;
    }
}
```

Tratamento de erros

Finalmente, outro aspeto importante tem a ver com o correto tratamento de erros. É preciso considerar:

- O serviço só inicia no arranque da aplicação. Se parar com uma exceção não tratada, será preciso reiniciar a aplicação para que ele reinicie.
- Um background service long-running não deve parar de executar se qualquer operação que realiza dá um erro. Por exemplo, se estiver a processar uma queue e parar porque o primeiro elemento dá erro, nunca processará os seguintes.
- Isso implica ter um mecanismo qualquer para deixar em determinado momento de tratar os elementos da queue que causam erro, porque senão também ficará a tentar infinitamente (e nunca processará os elementos seguintes na mesma).
- É fundamental que os erros sejam todos muito bem registados no log. Caso contrário, no limite, o serviço pode até não estar a conseguir fazer nada mas, porque não para de executar, nunca se chega a perceber que isso está a acontecer.

Mais uma vez, o Hydrogen não disponibiliza funcionalidades para tratar estes casos (aceitam-se ideias suficientes genéricas!).
Fica à responsabilidade do developer.

Notas:

- Passar a exceção em LogError garante que ela vai ser listada nas exceptions do Azure App Insights.
- Por isso é que a monitorização dos serviços no Azure deve ser sempre feita também sobre as exceções (o probe não chega).

```
// Execute

try
{
    // Acquire lock (pessimistic)

    if (await this.AcquireLockAsync().ConfigureAwait(false))...
    else
    {
        // Logging (alive)

        this.Logger.LogInformation("Another worker is performing work.");
    }

    // Canceled?

    if (cancellationToken.IsCancellationRequested)...
}
catch (Exception ex)
{
    // Logging

    this.Logger.LogError(ex, $"The notifications sender worker raised exception: '{ex.Message}'. Continuing...");

    // Run forever...
}
finally
{
    // Release lock

    await this.ReleaseLockAsync().ConfigureAwait(false);
}
```