Implementando o GeoCordinateWatcher como um serviço reativo

Este artigo explica como usar Rx para subscrever eventos de geolocalização.

Introdução

Com **Rx**, eventos são cidadãos de primeira classe que podem ser componíveis e passados a outros objetos de forma muito simples.

Implementando O GeoCoordinateWatcher Como Um Serviço Reativo

"As Extensões Reativas (Rx) são uma biblioteca para composição de programas assíncronos e baseados em eventos usando sequências observáveis e operadores estilo-LINQ. Usando Rx, os desenvolvedores **representam** correntes assíncronas de dados com Observáveis, ***consultam (query)*** correntes assíncronas de dados usando operadores LINQ e ***parametrizam*** a concorrência das correntes de dados assíncronas usando agendadores (Schedulers). Pondo de forma simples, Rx = Observables + LINQ + Schedulers." – da página da MSDN.

A biblioteca providencia também um considerável número de auxiliares que facilitam encapsular enventos em observáveis.

Encapsular o GeoCoordinateWatcher como um serviço reativo é muito simples. Tudo o que é necessário é expor os eventos como observáveis:

```
public class GeoCoordinateReactiveService : IGeoCoordinateReactiveService, IDisposable
{
    private readonly GeoCoordinateWatcher geoCoordinateWatcher = new GeoCoordinateWatcher();

    public GeoCoordinateReactiveService()
    {
        this.StatusObservable = Observable
            .FromEventPattern<GeoPositionStatusChangedEventArgs>(
                handler => geoCoordinateWatcher.StatusChanged += handler,
                handler => geoCoordinateWatcher.StatusChanged -= handler);

        this.PositionObservable = Observable
            .FromEventPattern<GeoPositionChangedEventArgs<GeoCoordinate>>(
                handler => geoCoordinateWatcher.PositionChanged += handler,
                handler => geoCoordinateWatcher.PositionChanged -= handler);
    }

    public IObservable<EventPattern<GeoPositionStatus> StatusObservable { get; private set; }

    public IObservable<EventPattern<GeoPosition<GeoCoordinate>> PositionObservable { get; private set; }
}
```

E assim, em vez dos eventos StatusChanged e PositionChanged temos respectivamente as correntes de instâncias de EventPattern<TEventArgs> **StatusObservable** e **PositionObservable**.

Mas a classe **EventPattern<TEventArgs>** inclui a fonte do evento e os seus argumentos em propriedades que são demasiado para as nossas necessidades. Usando normais operadores LINQ podemos converter as correntes de instâncias de **EventPattern<TEventArgs>** em correntes dos valores pretendidos.

```
public class GeoCoordinateReactiveService : IGeoCoordinateReactiveService, IDisposable
{
    private readonly GeoCoordinateWatcher geoCoordinateWatcher = new GeoCoordinateWatcher();

    public GeoCoordinateReactiveService()
    {
        this.StatusObservable = Observable
            .FromEventPattern<GeoPositionStatusChangedEventArgs>(
                handler => geoCoordinateWatcher.StatusChanged += handler,
                handler => geoCoordinateWatcher.StatusChanged -= handler)
            .Select(ep => ep.EventArgs.Status);

        this.PositionObservable = Observable
            .FromEventPattern<GeoPositionChangedEventArgs<GeoCoordinate>>(
                handler => geoCoordinateWatcher.PositionChanged += handler,
                handler => geoCoordinateWatcher.PositionChanged -= handler)
            .Select(ep => ep.EventArgs.Position);
    }

    public IObservable<GeoPositionStatus> StatusObservable { get; private set; }

    public IObservable<GeoPosition<GeoCoordinate>> PositionObservable { get; private set; }
}
```

Usando O GeoCoordinateReactiveService

Tendo criado o envelope Rx (wrapper) para o GeoCoordinateWatcher demonstrarei agora como pode ser usado numa aplicação simples.

A aplicação apresentará o estado do serviço, a posição e a distância percorrida.

Nesta aplicação simples o serviço será exposto através de uma propriedade singleton da classe App:

```
public partial class App : Application
```

 (C) Copyright Nokia 2013. All rights reserved.

```
{
    // ...

    public static IGeoCoordinateReactiveService GeoCoordinateService { get; private set; }

    public App()
    {
        // ...

        InitializePhoneApplication();

        // ...
    }

    // ...

    private void InitializePhoneApplication()
    {
        // ...

        GeoCoordinateService = new GeoCoordinateReactiveService();

        // ...

    }

    // ...

}
```

Obter o estado do serviço é muito simples. Apenas requere subscrever o **StatusObservable**. Uma vez que pretendemos apresentar o estado, precisamos de observá-lo no dispatcher:

```
App.GeoCoordinateService.StatusObservable
    .ObserveOnDispatcher()
    .Subscribe(this.OnStatusChanged);
```

Para a posição fazemos o mesmo com o **PositionObservable**:

```
App.GeoCoordinateService.PositionObservable
    .ObserveOnDispatcher()
    .Subscribe(this.OnPositionChanged);
```

Obter a distï¿½ncia percorrida parece ser mais complicado porque temos de manter registo da posiï¿½ï¿½o anterior e calcular a distï¿½ncia percorrida cada vez que a posiï¿½ï¿½o muda. Mas ï¿½ aqui que a **Rx** mostra as suas vantagens com os seus operadores. Se combinarmos a o observï¿½vel da posiï¿½ï¿½o com o observï¿½vel da posiï¿½ï¿½o saltando uma posiï¿½ï¿½o usando operador zip obtemos um observï¿½vel com a posiï¿½ï¿½o anterior e a atual. E se aplicarmos um seletor, obtemos apenas a distï¿½ncia percorrida:

```
App.GeoCoordinateService.PositionObservable
    .Zip(
        App.GeoCoordinateService.PositionObservable.Skip(1),
        (p1, p2) => p1.Location.GetDistanceTo(p2.Location))
    .ObserveOnDispatcher()
    .Subscribe(this.OnDistanceChanged);
```

A implementação complete pode ser encontrada aqui.

Recursos

- The GeoCoordinateWatcher As A Reactive Service Project
- The Reactive Extensions (Rx)... on MSDN
- Rx (Reactive Extensions) on CodePlex
- NuGet Pakages
  - Reactive Extensions - Interfaces Library
  - Reactive Extensions - Core Library
  - Reactive Extensions - Query Library
  - Reactive Extensions - Platform Services Library
  - Reactive Extensions - Main Library
  - Reactive Extensions - XAML Support Library
  - Reactive Extensions - Silverlight Helpers
- Using Rx
- Reactive Extensions (Rx) Forum
- Reactive Extensions Team Blog
- MS Open Tech Open Sources Rx (Reactive Extensions) – a Cure for Asynchronous Data Streams in Cloud Programming

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.