عنوان: **انجام پی در پی اعمال Async به کمک Iterators - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۴/۱۱ ۰۰:۰۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns

در قسمت قبل ایده‌ی اصلی و مفاهیم مرتبط با استفاده از Iterators مطرح شد. در این قسمت به یک مثال عملی در این مورد خواهیم پرداخت.

چندین کتابخانه و کلاس جهت مدیریت Coroutines در دات نت تهیه شده که لیست آن‌ها به شرح زیر است:

1) Using C# 2.0 iterators to simplify writing asynchronous code
2) Wintellect's Jeffrey Richter's PowerThreading Library
3) Rob Eisenberg's Build your own MVVM Framework codes

و ...

مورد سوم که توسط خالق اصلی کتابخانه‌ی Caliburn (یکی از فریم ورک‌های مشهور MVVM برای WPF و Silverlight) در کنفرانس MIX 2010 ارائه شد، این روزها در وبلاگ‌های مرتبط بیشتر مورد توجه قرار گرفته و تقریبا به یک روش استاندارد تبدیل شده است. این روش از یک اینترفیس و یک کلاس به شرح زیر تشکیل می‌شود:

```
using System;

namespace SLAsyncTest.Helper
{
   public interface IResult
   {
       void Execute();
       event EventHandler Completed;
   }
}
```

```
using System;
using System.Collections.Generic;

namespace SLAsyncTest.Helper
{
   public class ResultEnumerator
   {
       private readonly IEnumerator<IResult> _enumerator;

       public ResultEnumerator(IEnumerable<IResult> children)
       {
           _enumerator = children.GetEnumerator();
       }

       public void Enumerate()
       {
           childCompleted(null, EventArgs.Empty);
       }

       private void childCompleted(object sender, EventArgs args)
       {
           var previous = sender as IResult;

           if (previous != null)
               previous.Completed -= childCompleted;

           if (!_enumerator.MoveNext())
               return;

           var next = _enumerator.Current;
           next.Completed += childCompleted;
           next.Execute();
```

```
        }
    }
}
```

توضیحات:
مطابق توضیحات قسمت قبل، برای مدیریت اعمال همزمان به شکلی پی در پی، نیاز است تا یک IEnumerable را به همراه yield return در پایان هر مرحله از کار ایجاد کنیم. در اینجا این IEnumerable را از نوع اینترفیس IResult تعریف خواهیم کرد. متد Execute آن شامل کدهای عملیات Async خواهند شد و پس از پایان کار رخداد Completed صدا زده می‌شود. به این صورت کلاس ResultEnumerator به سادگی می‌تواند یکی پس از دیگری اعمال Async مورد نظر ما را به صورت متوالی فراخوانی نمائید. با هر بار فراخوانی رخداد Completed، متد MoveNext صدا زده شده و یک مرحله به جلو خواهیم رفت.
برای مثال کدهای ساده WCF Service زیر را در نظر بگیرید.

```
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.Threading;

namespace SLAsyncTest.Web
{
    [ServiceContract(Namespace = "")]
    [AspNetCompatibilityRequirements(RequirementsMode
        = AspNetCompatibilityRequirementsMode.Allowed)]
    public class TestService
    {
        [OperationContract]
        public int GetNumber(int number)
        {
            Thread.Sleep(2000);//Simulating a log running operation
            return number * 2;
        }
    }
}
```

قصد داریم در طی دو مرحله متوالی این WCF Service را در یک برنامه‌ی Silverlight فراخوانی کنیم. کدهای قسمت فراخوانی این سرویس بر اساس پیاده سازی اینترفیس IResult به صورت زیر درخواهند آمد:

```
using System;
using SLAsyncTest.Helper;

namespace SLAsyncTest.Model
{
    public class GetNumber : IResult
    {
        public int Result { set; get; }
        public bool HasError { set; get; }

        private int _num;
        public GetNumber(int num)
        {
            _num = num;
        }

        #region IResult Members
        public void Execute()
        {
            var srv = new TestServiceReference.TestServiceClient();
            srv.GetNumberCompleted += (sender, e) =>
            {
                if (e.Error == null)
                    Result = e.Result;
                else
                    HasError = true;

                Completed(this, EventArgs.Empty); //run the next IResult
            };
            srv.GetNumberAsync(_num);
        }
```

```
        public event EventHandler Completed;
        #endregion
    }
}
```

در متد Execute کار فراخوانی غیرهمزمان WCF Service به صورتی متداول انجام شده و در پایان متد Completed صدا زده می‌شود. همانطور که توضیح داده شد، این فراخوانی در کلاس ResultEnumerator یاد شده مورد استفاده قرار می‌گیرد.
اکنون قسمت‌های اصلی کدهای View Model برنامه به شکل زیر خواهند بود:

```
private void doFetch(object obj)
        {
            new ResultEnumerator(executeAsyncOps()).Enumerate();
        }

        private IEnumerable<IResult> executeAsyncOps()
        {
            FinalResult = 0;
            IsBusy = true; //Show BusyIndicator

            //Sequential Async Operations
            var asyncOp1 = new GetNumber(10);
            yield return asyncOp1;

            //using the result of the previous step
            if(asyncOp1.HasError)
            {
                IsBusy = false; //Hide BusyIndicator
                yield break;
            }

            var asyncOp2 = new GetNumber(asyncOp1.Result);
            yield return asyncOp2;

            FinalResult = asyncOp2.Result; //Bind it to the UI

            IsBusy = false; //Hide BusyIndicator
        }
```

در اینجا یک IEnumerable از نوع IResult تعریف شده است و در طی دو مرحله‌ی متوالی اما غیرهمزمان کار دریافت اطلاعات از WCF Service صورت می‌گیرد. ابتدا عدد 10 به WCF Service ارسال می‌شود و خروجی 20 خواهد بود. سپس این عدد در مرحله‌ی بعد مجددا به WCF Service ارسال گردیده و حاصل نهایی که عدد 40 می‌باشد در اختیار سیستم Binding قرار خواهد گرفت.
اگر از این روش استفاده نمی‌شد ممکن بود به این جواب برسیم یا خیر. ممکن بود مرحله‌ی دوم ابتدا شروع شود و سپس مرحله‌ی اول رخ دهد. اما با کمک Iterators و yield return به همراه کلاس ResultEnumerator موفق شدیم تا عملیات دوم همزمان را در حالت تعلیق قرار داده و پس از پایان اولین عملیات غیر همزمان، مرحله‌ی بعدی فراخوانی را بر اساس مقدار حاصل شده از WCF Service آغاز کنیم.
این روش برای برنامه نویس‌ها آشناتر است و همان سیستم فراخوانی A->B->C را تداعی می‌کند اما کلیه اعمال غیرهمزمان هستند و ترد اصلی برنامه قفل نخواهد شد.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: Majid325
تاریخ: ۱۳۸۹/۰۴/۱۲ ۱۳:۳۸:۲۱

خیلی عالی بود ، اتفاقا همین مشکل هفته گذشته واسه من به وجود امده بود.