عنوان: **انجام پی در پی اعمال Async به کمک Iterators - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۴/۱۰ ۲۰:۵۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns

تقریبا تمام اعمال کار با شبکه در Silverlight از مدل asynchronous programming پیروی می‌کنند؛ از فراخوانی یک متد وب سرویس تا دریافت اطلاعات از وب و غیره. اگر در سایر فناوری‌های موجود در دات نت فریم ورک برای مثال جهت کار با یک وب سرویس هر دو متد همزمان و غیرهمزمان در اختیار برنامه نویس هستند اما اینجا خیر. اینجا فقط روش‌های غیرهمزمان مرسوم هستند و بس. خیلی هم خوب. یک چارچوب کاری خوب باید روش استفاده‌ی صحیح از کتابخانه‌های موجود را نیز ترویج کند و این مورد حداقل در Silverlight اتفاق افتاده است.
برای مثال فراخوانی‌های زیر را در نظر بگیرید:

```
private int n1, n2;

private void FirstCall()
{
  Service.GetRandomNumber(10, SecondCall);
}

private void SecondCall(int number)
{
  n1 = number;
  Service.GetRandomNumber(n1, ThirdCall);
}

private void ThirdCall(int number)
{
  n2 = number;
  // etc
}
```

عموما در اعمال Async پس از پایان عملیات در تردی دیگر، یک متد فراخوانی می‌گردد که به آن callback delegate نیز گفته می‌شود. برای مثال توسط این سه متد قصد داریم اطلاعاتی را از یک وب سرویس دریافت و استفاده کنیم. ابتدا FirstCall فراخوانی می‌شود. پس از پایان کار آن به صورت خودکار متد SecondCall فراخوانی شده و این متد نیز یک عملیات Async دیگر را شروع کرده و الی آخر. در نهایت قصد داریم توسط مقادیر بازگشت داده شده منطق خاصی را پیاده سازی کنیم. همانطور که مشاهده می‌کنید این اعمال زیبا نیستند! چقدر خوب می‌شد مانند دوران synchronous programming (!) فراخوانی‌های این متدها به صورت ذیل انجام می‌شد:

```
private void FetchNumbers()
{
 int n1 = Service.GetRandomNumber(10);
 int n2 = Service.GetRandomNumber(n1);
}
```

در برنامه نویسی متداول همیشه عادت داریم که اعمال به صورت A -> B -> C انجام شوند. اما در Async programming ممکن است ابتدا C انجام شود، سپس A و بعد B یا هر حالت دیگری صرفنظر از تقدم و تاخر آن‌ها در حین معرفی متدهای مرتبط در یک قطعه کد. همچنین میزان خوانایی این نوع کدنویسی نیز مطلوب نیست. مانند مثال اول ذکر شده، یک عملیات به ظاهر ساده به چندین متد منقطع تقسیم شده است. البته به کمک lambda expressions مثال اول را به شکل زیر نیز می‌توان در طی یک متد ارائه داد اما اگر تعداد فراخوانی‌ها بیشتر بود چطور؟ همچنین آیا استفاده از عدد n2 بلافاصله پس از عبارت ذکر شده مجاز است؟ آیا عملیات واقعا به پایان رسیده و مقدار مطلوب به آن انتساب داده شده است؟

```
private void FetchNumbers()
{
  int n1, n2;

  Service.GetRandomNumber(10, result =>
                                {
```

```
                                    n1 = result;
                                    Service.GetRandomNumber(n1, secondResult =>
                                                                {
                                                                    n2 = secondResult;
                                                                });
                                });
}
```

به عبارتی می‌خواهیم کل اعمال انجام شده در متد FetchNumbers هنوز Async باشند (ترد اصلی برنامه را قفل نکنند) اما پی در پی انجام شوند تا مدیریت آن‌ها ساده‌تر شوند (هر لحظه دقیقا بدانیم که کجا هستیم) و همچنین کدهای تولیدی نیز خواناتر باشند.
روش استانداری که توسط الگوهای برنامه نویسی برای حل این مساله پیشنهاد می‌شود، استفاده از الگوی coroutines است.
توسط این الگو می‌توان چندین متد Async را در حالت معلق قرار داده و سپس در هر زمانی که نیاز به آن‌ها بود عملیات آن‌ها را از سر گرفت.
دات نت فریم ورک حالت ویژه‌ای از coroutines را توسط Iterators پشتیبانی می‌کند (از C# 2.0 به بعد) که در ابتدا نیاز است از دیدگاه این مساله مروری بر آن‌ها داشته باشیم. مثال بعد یک enumerator را به همراه yield return ارائه داده است:

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

namespace CoroutinesSample
{
  class Program
  {
      static void printAll()
      {
          foreach (int x in integerList())
          {
              Console.WriteLine(x);
          }
      }

      static IEnumerable<int> integerList()
      {
          yield return 1;
          Thread.Sleep(1000);
          yield return 2;
          yield return 3;
      }

      static void Main()
      {
          printAll();
      }
  }
}
```

کامپایلر سی شارپ در عمل یک state machine را برای پیاده سازی این عملیات به صورت خودکار تولید خواهد کرد:

```
private bool MoveNext()
{
  switch (this.<>1__state)
  {
      case 0:
          this.<>1__state = -1;
          this.<>2__current = 1;
          this.<>1__state = 1;
          return true;

      case 1:
          this.<>1__state = -1;
          Thread.Sleep(0x3e8);
          this.<>2__current = 2;
          this.<>1__state = 2;
          return true;

      case 2:
```

```
            this.<>1__state = -1;
            this.<>2__current = 3;
            this.<>1__state = 3;
            return true;

        case 3:
            this.<>1__state = -1;
            break;
    }
    return false;
}
```

در حین استفاده از یک IEnumerator ابتدا در وضعیت شیء Current آن قرار خواهیم داشت و تا زمانیکه متد MoveNext آن فراخوانی نشود هیچ اتفاق دیگری رخ نخواهد داد. هر بار که متد MoveNext این enumerator فرخوانی گردد (برای مثال توسط یک حلقه‌ی foreach) اجرای متد integerList ادامه خواهد یافت تا به yield return بعدی برسیم (سایر اعمال تعریف شده در حالت تعلیق قرار دارند) و همینطور الی آخر.
از همین قابلیت جهت مدیریت اعمال Async پی در پی نیز می‌توان استفاده کرد. State machine فوق تا پایان اولین عملیات تعریف شده صبر می‌کند تا به yield return برسد. سپس با فراخوانی متد MoveNext به عملیات بعدی رهنمون خواهیم شد. به این صورت دیدگاهی پی در پی از یک سلسه عملیات غیرهمزمان حاصل می‌گردد.

خوب ما الان نیاز به یک کلاس داریم که بتواند enumerator ایی از این دست را به صورت خودکار مرحله به مرحله آن هم پس از پایان واقعی عملیات Async قبلی (یا مرحله‌ی قبلی)، اجرا کند. قبل از اختراع چرخ باید متذکر شد که دیگران اینکار را انجام داده‌اند و کتابخانه‌های رایگان و یا سورس بازی برای این منظور موجود است.

ادامه دارد ...

## نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۰۸ ۱۶:۳۳:۴۸

خبر خوش اینکه انجام امور async در سی شارپ 5 به کمک واژه کلیدی await ، همانند مقصود دو مقاله فوق به سادگی در اختیار و کنترل برنامه نویس‌ها خواهد بود.

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۰۵/۱۵ ۱۸:۳۴

واقعاً عالی‌است!