عنوان: **درباره Iterator methodها و yield return در C#**
نویسنده: شهره مرتضوی
تاریخ: ۱۳۹۲/۰۶/۲۴ ۱۲:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#.NET, Iterator methods, IEnumerator, IEnumerable, Yeild return

هنگامیکه می‌خواهید در متدهای خود مقداری (از هر نوع datatype دلخواه) را return نمایید، در حالت عادی قادر خواهید بود که فقط از یک return در بدنه متد خود استفاده نمایید:

```
public int Sum(int a, int b)
{
     return a + b;
}
```

اما چنانچه از متدهای تکرار شونده استفاده نمایید، چطور؟

متدهای تکرار شونده یا Iterator methodها ، در داخل یک collection به صورت دلخواه iterate کرده یا به اصلاح پیمایش می‌کنند. این متدها از کلمه کلیدی Yield در هنگام return کردن مقادیر استفاده می‌کنند. (در C# از Yield return و در VB از Yield استفاده می‌شود) به عبارت دیگر یک متد با خروجی از نوع قابل پیمایش (مانند IEnumerable)، با استفاده از چند yield return، دارای قابلیت پیمایش و بازگرداندن چندین مقدار به جای یک مقدار واحد می‌گردد.

برای درک بهتر مسئله از مثالی برای ادامه توضیحات استفاده می‌کنم. متد پیمایش شونده (Iterate method) زیر را در نظر بگیرید که خروجی IEnumerable دارد:

```
public static IEnumerable SomeNumbers()
{
     yield return 3;
     yield return 5;
     yield return 8;
}
```

برای استفاده از مقادیر بازگشتی متد بالا از حلقه foreach زیر استفاده می‌نماییم:

```
static void Main()
{
    foreach (int number in SomeNumbers())
    {
        Console.Write(number.ToString() + " ");
    }
    // Output: 3 5 8
    Console.ReadKey();
}
```

حلقه foreach فوق ، در پایان اولین پیمایش، عدد 3 را باز گردانده و مکان این return را حفظ می‌کند. در چرخه بعدی عدد 5 را باز می‌گرداند و این نقطه را نیز نگه می‌دارد و در چرخه پایانی عدد 8 را برگردانده و سپس حلقه با رسیدن به نقطه پایانی متد، خاتمه می‌یابد.

برای خاتمه پیمایش در Iterator methodها ، میتوانید از foreach استفاده کنید و یا اینکه عبارت yield break را بعد از تمامی yield returnها به کار گیرید:

```
public static IEnumerable SomeNumbers()
{
   yield return 3;
   yield return 5;
   yield return 8;
   yeild break;
}
```

نکات:

- در هنگام ایجاد Iterator method ها، نوع مقادیر خروجی متد ، باید یکی از انواع IEnumerable, IEnumerable<T>, IEnumerator, و یا IEnumerator<T>. باشد.
- در هنگام declare کردن ، نمی‌توانید از پارامترهای ref و out استفاده نمایید.
- در Anonymous methodها (متدهای بی نام) و Unsafe blockها نمی‌توانید از yield return (yield در VB) استفاده نمایید.
- نمی‌توانید از Yield return در بلوکهای try-catch استفاده کندی. اما می‌توانید در قسمت try بلوک try-finally استفاده نمایید.
- از yield break می‌توانید در بلوک try و یا بلوک catch استفاده نمایید ، اما در بلوک finally خیر.
- هنگام بروز خطا در foreach هایی که خارج از iterator methodها استفاده می‌شوند، بلوک finally داخل این متدها اجرا می‌گردد.

مثالی دیگر با استفاده Iterator methodها و yield return جهت بازگرداندن روزهای هفته:

```
static void Main()
{
  DaysOfTheWeek days = new DaysOfTheWeek();
  foreach (string day in days)
    {
        Console.Write(day + " ");
    }
    // Output: Sun Mon Tue Wed Thu Fri Sat
    Console.ReadKey();
}

public class DaysOfTheWeek : IEnumerable
{
  private string[] days = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
  public IEnumerator GetEnumerator()
    {
        for (int index = 0; index < days.Length; index++)
        {
            // Yield each day of the week.
            yield return days[index];
        }
    }
}
```

منابع:

yield ، Iterators

نظرات خوانندگان

نویسنده: محسن جمشیدی
تاریخ: ۱۳۹۲/۰۶/۲۵ ۸:۴۵

گمان کنم "پیمایش" ترجمه مناسب‌تری نسبت "چرخش" باشد.
به نظر ترجمه Iterator Method به "متد تکرارشونده" منظور رو بدرستی منتقل نمی‌کنه اما کمله مناسبی به ذهنم نمی‌رسه

نویسنده: شهره مرتضوی
تاریخ: ۱۳۹۲/۰۶/۲۵ ۱۰:۲۵

با تشکر از حسن نظر شما. کلمه "پیمایش" رو مد نظر قرار خواهم دادم. در مورد "متد تکرار شونده" ، اگر دوستان کلمه بهتری به ذهنشون رسید ، بفرمایند تا تغییر بدم.

عنوان: **آشنایی با الگوی طراحی Iterator**
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۱/۲۹ ۱۸:۵۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, IEnumerable, Iterator methods

فرض کنید قبلا کلاسی بنام CollectionClass را داشته‌اید که در آن یک آرایه از نوع String[] تعریف کرده‌اید. همچنین n تا کلاس هم دارید که از آرایه‌ی تعریف شده‌ی در CollectionClass استفاده می‌کنند. تا اینجا مشکلی نیست. مشکل زمانی شروع می‌شود که متوجه می‌شوید دیگر این آرایه کارآیی ندارد و باید آن را با List<string> جایگزین کنید. واضح است که نمی‌توانید همه کلاس‌هایی را که از CollectionClass استفاده کرده‌اند، بیابید و آنها را تغییر دهید؛ چرا که شاید برخی از کلاس‌ها اصلا در دسترس شما نباشند یا هر دلیل دیگری.

راهگشای این مشکل، استفاده از الگوی طراحی Iterator است. در این الگو، باید کلاس CollectionClass ابتدا واسط IEnumerable را پیاده سازی نماید. این واسط متدی بنام GetEnumerator دارد که می‌توان به کمک آن، درون آرایه یا هر نوع کالکشن دیگری حرکت کرده و آیتم‌های آن را برگرداند.(مطالعه بیشتر)

اول این الگو را پیاده سازی می‌کنیم و در ادامه توضیح می‌دهیم که چگونه مشکل ما را حل میکند:

ابتدا باید کلاس CollectionClass واسط IEnumerable را پیاده سازی نماید. در ادامه بدنه متد GetEnumerator را می‌نویسیم:

```
public class CollectionClass : IEnumerable
    {
        private string[] mySet = { "Array of String 1", "Array of String 2", "Array of String 3" };
        public IEnumerator GetEnumerator()
        {
            //return arrayStrings.GetEnumerator();
            foreach (var element in mySet )
            {
                yield return element;
            }
        }
    }
```

در اینجا یک آرایه رشته‌ای را بنام mySet تعریف کرده‌ایم و مقادیر مختلفی را در آن قرار داده‌ایم. سپس در متد GetEnumerator اعضای این آرایه را خوانده و return می‌کنیم.(yield چیست؟)

وقتی از این کلاس می‌خواهیم استفاده کنیم، داریم:

```
CollectionClass c = new CollectionClass();
foreach (var element in c)
{
       Console.WriteLine(element);
}
```

در این حالت مهم نیست که مجموعه‌ی مورد نظر، آرایه هست یا هر نوع کالکشن دیگری. لذا وقتی بخواهیم نوع mySet را تغییر دهیم، نگران نخواهیم بود؛ چراکه فقط کافی‌است کلاس CollectionClass را تغییر دهیم. بصورت زیر:

```
public class CollectionClass : IEnumerable
    {
        //private readonly string[] arrayStrings = { "Array of String 1", "Array of String 2", "Array
of String 3" };
        private List<string> mySet= new List<string>() { "Array of String 1", "Array of String 2",
"Array of String 3" };
        public IEnumerator GetEnumerator()
        {
            foreach (var element in mySet )
            {
                yield return element;
            }
        }
    }
```