عنوان: **دات نت 4 و کلاس Lazy**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۰ ۲۱:۵۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

یکی از الگوهای برنامه نویسی شیء گرا، Lazy Initialization Pattern نام دارد که دات نت 4 پیاده سازی آن‌را سهولت بخشیده است.
در دات نت 4 کلاس جدیدی به فضای نام System اضافه شده است به نام Lazy و هدف از آن lazy initialization است؛ من ترجمه‌اش می‌کنم وهله سازی با تاخیر یا به آن on demand construction هم گفته‌اند (زمانی که به آن نیاز هست ساخته خواهد شد).
فرض کنید در برنامه‌ی خود نیاز به شیءایی دارید و ساخت این شیء بسیار پرهزینه است. نیازی نیست تا بلافاصله پس از تعریف، این شیء ساخته شود و تنها زمانیکه به آن نیاز است باید در دسترس باشد. کلاس Lazy جهت مدیریت اینگونه موارد ایجاد شده است. تنها کاری که در اینجا باید صورت گیرد، محصور کردن آن شیء هزینه‌بر توسط کلاس Lazy است:

```
Lazy<ExpensiveResource> ownedResource = new Lazy<ExpensiveResource>();
```

در این حالت برای دسترسی به شیء ساخته شده از ExpensiveResource ، می‌توان از خاصیت Value استفاده نمود (ownedResource.Value). تنها در حین اولین دسترسی به ownedResource.Value ، شیء ExpensiveResource ساخته خواهد شد و نه پیش از آن و نه در اولین جایی که تعریف شده است. پس از آن این حاصل cache شده و دیگر وهله سازی نخواهد شد.
ownedResource دارای خاصیت IsValueCreated نیز می‌باشد و جهت بررسی ایجاد آن شیء می‌تواند مورد استفاده قرار گیرد.
برای مثال قصد داریم اطلاعات ExpensiveResource را ذخیره کنیم اما تنها در حالتیکه یکبار مورد استفاده قرار گرفته باشد.
کلاس Lazy دارای دو متد سازنده‌ی دیگر نیز می‌باشد:

```
public Lazy(bool isThreadSafe);
public Lazy(Func<T> valueFactory, bool isThreadSafe);
```

و هدف از آن استفاده‌ی صحیح از این متد در محیط‌های چند ریسمانی است. بدیهی است در این نوع محیط ها علاقه‌ای نداریم که در یک لحظه توسط چندین ترد مختلف، سبب ایجاد وهله‌های ناخواسته‌ای از ExpensiveResource شویم و تنها یک مورد از آن کافی است یا به قولی thread safe, lazy initialization of expensive objects
بدیهی است اگر برنامه‌ی شما چند ریسمانی نیست می‌توانید این مکانیزم را کنسل کرده و اندکی کارآیی برنامه را با حذف قفل‌های همزمانی این کلاس بالا ببرید.

مثال اول:

```
using System;
using System.Threading;

namespace LazyExample
{
 class Program
 {
     static void Main()
     {
         Console.WriteLine("Before assignment");
         var slow = new Lazy<Slow>();
         Console.WriteLine("After assignment");

         Thread.Sleep(1000);

         Console.WriteLine(slow);
         Console.WriteLine(slow.Value);
```

```
        Console.WriteLine("Press a key...");
        Console.Read();
    }
}


class Slow
{
    public Slow()
    {
        Console.WriteLine("Start creation");
        Thread.Sleep(2000);
        Console.WriteLine("End creation");
    }
}
}
```

خروجی این برنامه به شرح زیر است:

```
Before assignment
After assignment
Value is not created.
Start creation
End creation
LazyExample.Slow
Press a key...
```

همانطور که ملاحظه می‌کنید تنها در حالت دسترسی به مقدار Value شیء slow ، عملا وهله‌ای از آن ساخته خواهد شد.

مثال دوم:
شاید نیاز به مقدار دهی خواص کلاس پرهزینه وجود داشته باشد. برای مثال علاقمندیم خاصیت SomeProperty کلاس ExpensiveClass را مقدار دهی کنیم. برای این منظور می‌توان به شکل ذیل عمل کرد (یک Func<t>را می‌توان به سازنده‌ی آن ارسال نمود):

```
using System;

namespace LazySample
{
class Program
{
    static void Main()
    {
        var expensiveClass =
            new Lazy<ExpensiveClass>
            (
                () =>
                {
                    var fobj = new ExpensiveClass
                                   {
                        SomeProperty = 100
                    };
                    return fobj;
                }
            );

        Console.WriteLine("expensiveClass has value yet {0}",
            expensiveClass.IsValueCreated);

        Console.WriteLine("expensiveClass.SomeProperty value {0}",
            (expensiveClass.Value).SomeProperty);

        Console.WriteLine("expensiveClass has value yet {0}",
            expensiveClass.IsValueCreated);

        Console.WriteLine("Press a key...");
        Console.Read();
    }
}
```

```
class ExpensiveClass
{
    public int SomeProperty { get; set; }

    public ExpensiveClass()
    {
        Console.WriteLine("ExpensiveClass constructed");
    }
}
}
```

کاربردها

:

- علاقمندیم تا ایجاد یک شیء هزینه‌بر تنها پس از انجام یک سری امور هزینه‌بر دیگر صورت گیرد. برای مثال آیا تابحال شده است که با یک سیستم ماتریسی کار کنید و نیاز به چند گیگ حافظه برای پردازش آن داشته باشید؟! زمانیکه از کلاس Lazy استفاده نمائید، تمام اشیاء مورد استفاده به یکباره تخصیص حافظه پیدا نکرده و تنها در زمان استفاده از آن‌ها کار تخصیص منابع صورت خواهد گرفت.

- ایجاد یک شیء بسیار پر هزینه بوده و ممکن است در بسیاری از موارد اصلا نیازی به ایجاد و یا حتی استفاده از آن نباشد. برای مثال یک شیء کارمند را درنظر بگیرید که یکی از خواص این شیء، لیستی از مکان‌هایی است که این شخص قبلا در آنجاها کار کرده است. این اطلاعات نیز به طور کامل از بانک اطلاعاتی دریافت می‌شود. اگر در متدی، استفاده کننده از شیء کارمند هیچگاه اطلاعات مکان‌های کاری قبلی او را مورد استفاده قرار ندهد، آیا واقعا نیاز است که این اطلاعات به ازای هر بار ساخت وهله‌ای از شیء کارمند از دیتابیس دریافت شده و همچنین در حافظه ذخیره شود؟

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۰۲/۲۱ ۰۹:۵۹:۵۱

کتاب مفیدی برای شروع کار و یا یادگیری برای بکار بردن C#4 بنظرتون میاد:

(Addison-Wesley Essential C# 4.0 (978 Pages

(Wrox Professional C# 4 and .NET 4 (1852 Pages

(OReilly C# 4.0 in a Nutshell 4th Edition Feb 2010 (1056 Pages

(Sams C# 4.0 How To Feb 2010 (669 Pges

(Effective C# 50 Specific Ways to Improve Your C# Second Edition (342 Pges

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۰۲/۲۱ ۱۰:۰۱:۵۵

راستی فکر کنم یواش یواش باید عنوان وبتون رو عوض کنید به ".Net 4.0 Tips and "

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۱ ۱۵:۰۹:۲۱

تنها کتاب اختصاصی برای C#4

C# Language Specification 4.0

این کتاب‌هایی که نام بردید تشکیل شده از تمام مؤلفه‌های دات نت. مثلا اون کتاب 1852 صفحه‌ای از WCF تا WPF را هم دارد. خود WPF جامع حدود 800 صفحه است. یا WCF راحت 600 صفحه است.

نویسنده: Pantee
تاریخ: ۱۳۸۹/۰۲/۲۴ ۲۰:۱۰:۲۳

آقا مطلبتو کپی کردن، محض اطلاع:

http://www.zabet.ir/lazy-initialization-pattern-dotnet-4.html

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۲۵ ۰۱:۰۵:۳۶

سلام، موفق باشید.