بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

چند نخی / multithreading در ASP.NET

مدرس : مهندس افشین رفوآ

چند نخی / multithreading در ASP.NET

Thread عبارتند از مسیر اجرای یک برنامه. هر thread یک جریان کنترل منحصر بفرد را تعریف می کند.

چنانچه برنامه ای شامل چندین فرایند سنگین و زمان بر مانند دسترسی به پایگاه داده یا عملیات ورودی/خروجی انبوه و فشرده می باشد، در آن صورت انتخاب مسیرها یا نخ های (thread) مختلف اجرا بسیار سودمند خواهد بود، به گونه ای که هر نخ به انجام کار معینی بپردازد.

Thread ها، فرایندهای lightweight (در سیستم‌های عامل به راه حل بینابینی از نخ‌های سطح کاربر و سطح هسته می‌گویند که الگوریتمی برای پیاده‌سازی سیستم‌های چندکارگی است ولی در سیستم‌های عامل تک پردازنده‌ای و قدیمی مثل سولاریس و یونیکس سیستم وی به نخ‌های سطح کاربر، پروسه سبک‌وزن می‌گویند) هستند. نخ در واقع کوچترین توالی از دستورالعمل‌های برنامه‌ریزی شده است که زمان‌بند سیستم‌عامل می‌تواند آنها را به شکل مستقل مدیریت کند.

یکی از کاربرد های امروزه ی نخ در پیاده سازی زمان بندی های همزمان است که توسط سیستم عامل های نوین و پیشرفته صورت می گیرد. استفاده از نخ ها از اتلاف چرخه یا cycle واحد پردازنده ی مرکزی (CPU Cycle) جلوگیری کرده و کارایی برنامه را بالا می برد.

تا به حال تنها برنامه هایی را کامپایل می کردیم که فقط یک نخ به عنوان یک فرایند اجرا می شد که همان نمونه ی در حال اجرای برنامه مورد نظر بود. از این طریق application تنها می تواند در یک بازه ی زمانی مشخص فقط یک کار را انجام دهد. برای ایجاد قابلیت اجرای چندین کار به طور همزمان، می توان برنامه را به نخ های کوچکتری تبدیل کرد.

در .NET. نخ کشی (Threading) از طریق فضای نام (namespace) 'System.Threading' اداره می شود. تعریف یک متغیر از نوع *System.Threading.Thread* به شما امکان ایجاد نخ های جدید و مجزا و نیز دسترسی به آن ها در برنامه را می دهد.

ایجاد thread

این کار از طریق ایجاد یک شئ Thread و دادن یک ارجاع ThreadStart به سازنده ی (constructor) آن انجام می گیرد.

```
ThreadStart childthreat = new ThreadStart(childthreadcall);
```

Thread life cycle (چرخه ی حیات نخ)

چرخه ی حیات نخ از زمانی آغاز می شود که یک شئ از کلاس System.Threading.Thread ایجاد شده و هنگامی پایان می یابد که نخ اجرای خود را به اتمام برساند.

زیر وضعیت های مختلف چرخه ی حیات نخ را مشاهده می کنید:

The Unstarted State: وضعیتی است که در آن نمونه (instance) نخ ایجاد شده ولی متد Start هنوز فراخوانی نشده است.

Ready State: وضعیتی است که نخ در آن به طور کامل آماده ی اجرا بوده و فقط منتظر CPU Cycle می باشد.

The Not Runnable State: نخ تحت شرایط زیر قابل اجرا نمی باشد:

تابع Sleep فراخوانی شده باشد.

تابع Wait صدا زده شده باشد.

توسط عملیات ورودی/خروجی یا I/O مسدود (block) شده باشد.

Dead State: وضعیتی است که نخ اجرای خود را به اتمام رسانده باشد یا به هر دلیلی ناگهان خاتمه (abort) داده شده باشد.

Thread Priority (سطح اولویت نخ ها)

خاصیت (property) Priority کلاس Thread، اولویت یک نخ را در رابط با نخ های دیگر سنجیده و تعیین می کند. .Net runtime، ready thread (نخ آماده ی اجرا) را به عنوان نخ دارای بالاترین سطح اولویت در نظر گرفته و انتخاب می کند:

سطح اولویت نخ ها را می توان به ترتیب زیر رده بندی کرد:

Above normal

Below normal

Highest

Lowest

Normal

یک نخ به مجرد اینکه ایجاد می گردد، سطح اولویت آن با استفاده از خاصیت Priority کلاس Thread تعیین می شود:

```
NewThread.Priority = ThreadPriority.Highest;
```

خاصیت ها و توابع thread

Property های کلاس thread به ترتیب با ذکر شرح برای شما فهرست شده:

خاصیت	شرح
CurrentContext	بستر یا context فعلی که نخ در آن در حال اجرا می باشد را بازیابی می کند.
CurrentCulture	این خاصیت culture (قالب پیش فرض نمایش تاریخ، اعداد، ارقام ، پول و غیره ..را تعیین می کند) نخ جاری را بازیابی یا مقداردهی می کند.
CurrentPrinciple	Principal جاری نخ را بازگردانده یا تنظیم می کند. Principal: موجودتی که توسط سیستم کامپیوتر یا شبکه (برای اهداف و مقاصد امنیتی) اعتبار سنجی می شود.
CurrentThread	نخ در حال اجرا را باز می گرداند.
CurrentUICulture	Culture جاری که توسط Resource Manager برای پیدا کردن منابع ویژه و مربوط به culture بکار می رود را بازگردانی کرده یا مقداردهی می کند.

ExecutionContext	یک شئ ExecutionContext بازمی گرداند که دربردانده ی اطلاعاتی درباره ی بسترهای (context) مختلف اجرا thread جاری را می باشد.
IsAlive	مقداری برگردانده که وضعیت اجرا (execution status) نخ جاری را مشخص می کند.
IsBackground	Thread یا پس زمینه است یا پیش زمینه. این خاصیت مقداری برگردانده یا تنظیم می کند که آن مقدار مشخص می کند آیا thread مورد نظراز نوع پس زمینه است یا خیر.
IsThreadPoolThread	مقداری بازگردانی می کند که توسط آن مقدار مشخص می شود آیا یک thread معین به مخزن thread های (thread pool) مدیریت شده تعلق دارد یا خیر.
ManagedThreadId	این خاصیت یک شناسه ی منحصر بفرد برای thread مدیریت شده ی جاری بازیابی می کند.
Name	اسم thread را بازگردانی کرده یا تنظیم می کند.
Priority	مقداری خوانده یا تنظیم می کند که اولویت زمان بندی و اجرای یک thread توسط آن مقدار تعیین می گردد.
ThreadState	مقداری بازیابی می کند که دربردارنده ی وضعیت های مختلف نخ (thread) جاری می باشد.

کلاس thread دارای توابعی می باشد که زیر فهرست شده است:

متد	شرح
Abort	این تابع باعث رخداد ThreadAbortException در نخی که تابع نام برده برای آن فراخوانی شده، می شود که فرایند خاتمه دادن به آن نخ را آغاز می کند. صدا زدن این متد همان طور که از اسم آن پیدا است باعث اتمام یافتن نخ می شود.

AllocateDataSlot	این متد یک data slot نام گذاری نشده به تمامی thread تخصیص می دهد. جهت افزایش کارایی توصیه می شود از فیلدهایی استفاده کنید که با خصیصه ی ThreadStaticAttribute نشانه گذاری شده اند.
AllocateNamedDataSlot	یک data slot نام گذاری شده به کلیه ی thread ها تخصیص می دهد. جهت افزایش کارایی توصیه می شود از فیلد هایی که با خصیصه ی ThreadStaticAttribute علامت گذاری شده اند استفاده کنید.
BeginCriticalRegion	به میزبان (host) اطلاع می دهد که اجرا (execution) در صدد وارد شدن به یک ناحیه ای از کد می باشد که در صورت رخداد خاتمه ی ناگهانی یا استثنای مدیریت نشده (unhandled exception) در آن، ممکن است دیگر task های برنامه صدمه دیده یا به خطر بیافتند.
BeginThreadAffinity	به host اطلاع می دهد که managed code در صدد اجرای دستورهایی است که به thread identity سیستم عامل جاری بستگی دارد.
EndCriticalRegion	به host اطلاع می دهد که اجرا (execution) در صدد ورود به آن بخشی از کد است که تاثیرات رخداد خاتمه ی ناگهانی thread یا استثنای مدیریت نشده (unhandled exception) تنها به task جاری محدود می شود.
FreeNamedDataSlot	ارتباط بین اسم و slot را برای کلیه ی نخ های موجود در فرایند حذف می کند. برای افزایش کارایی از فیلدهایی که با attribute ThreadStaticAttribute نشانه گذاری می شوند استفاده نمایید.
GetData	مقداری را که در slot جاری مشخص شده است بازیابی می کند. برای افزایش کارایی از فیلدهایی که با ThreadStaticAttribute attribute نشانه گذاری می شوند استفاده نمایید.
GetDomain	domain فعلی که thread جاری (در حال حاضر) درون آن در حال اجرا است را

	بازمی گرداند.
GetDomainID	یک شناسه ی منحصر بفرد application domain برمی گرداند.
GetNamedDataSlot	به دنبال data slot نام گذاری شده گشته و آن را بازیابی می کند. برای افزایش کارایی بهتر از ThreadStaticAttribute attribute استفاده نمایید.
Interrupt	در Thread ای که در وضعیت WaitSleepJoin به سر می برد وقفه ایجاد می کند.
ResetAbort	باعث می شود درخواست خاتمه ی ناگهانی thread جاری لغو گردد.
SetData	Data را در slot مشخص شده در domain مربوط به thread جاری قرار می دهد.برای افزایش کارایی بهتر از ThreadStaticAttribute attribute استفاده نمایید.
Start	این تابع یک thread جدید راه اندازی می کند.
Sleep	باعث می شود thread برای مدت زمان مشخصی (به طور موقت) متوقف شود.
SpinWait	باعث می شود یک thread بر اساس پارامتر تکرار (iteration) تعداد دفعات معینی به حالت انتظار در آید (تعداد دفعات معینی منتظر بماند). باعث می شود نخ به تعداد دفعاتی که توسط پارامتر تکرار تعریف شده منتظر بماند.
VolatileRead()	متد ()VolatileRead مقدار یک فیلد معین را می خواند. این مقدار آخرین مقداری است که توسط پردازنده های موجود در کامپیوتر نوشته شده، صرف نظر از اینکه حافظه ی نهان (cache) پردازنده در چه وضعیتی قرار دارد یا چه تعداد پردازنده در رایانه موجود می باشد. این متد نسخه های overload شده ی مختلفی دارد.
VolatileWrite()	این تابع یک مقدار را بلافاصله در یک فیلد write می کند تا بدین وسیله مقدار مذکور برای تمامی پردازنده ی های موجود در کامپیوتر قابل رویت (visible) باشد.

این متد نسخه های overload شده ی مختلفی دارد.	
باعث می شود calling thread اجرا را به نخ دیگری که آماده ی اجرا شدن بر روی پردازنده ی جاری می باشد، محول کند. سیستم عامل نخی که اجرا به آن محول می شود را تعیین می کند.	Yield

مثال:

مثال زیر کاربرد و نحوه ی استفاده از کلاس Thread را نمایش می دهد. صفحه ی مورد نظر دارای یک کنترل label برای نشان دادن پیغام های child thread می باشد. پیام های ارسال شده از برنامه ی اصلی با استفاده از متد Response.Write() به طور مستقیم در کنترل مذکور (بالای صفحه) نمایش داده می شوند.

Source file:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="threaddemo._Default"
%>
```

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs"
Inherits="multithreading.WebForm1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>
         Untitled Page
      </title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            <h3>Thread Example</h3>
         </div>
         <asp:Label ID="lblmessage" runat="server" Text="Label">
         </asp:Label>
      </form>
   </body>
</html>
```

Code behind file:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
```

```
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Threading;

namespace multithreading
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            ThreadStart childthreat = new ThreadStart(childthreadcall);
            Response.Write("Child Thread Started <br/>");
            Thread child = new Thread(childthreat);
            child.Start();
            Response.Write("Main sleeping  for 2 seconds.......<br/>");
            Thread.Sleep(2000);
            Response.Write("<br/>Main aborting child thread<br/>");
            child.Abort();
        }
        public void childthreadcall()
        {
            try
            {
                lblmessage.Text = "<br />Child thread started <br/>";
                lblmessage.Text += "Child Thread: Coiunting to 10";

                for (int i = 0; i < 10; i++)
                {
                    Thread.Sleep(500);
                    lblmessage.Text += "<br/> in Child thread </br>";
                }

                lblmessage.Text += "<br/> child thread finished";
            }
            catch (ThreadAbortException e)
            {

                lblmessage.Text += "<br /> child thread - exception";
            }
            finally
            {
                lblmessage.Text += "<br /> child thread - unable to catch the  exception";
            }
        }
    }

}
```

به نکات زیر توجه داشته باشید:

هنگامی که صفحه بارگذاری می شود به دنبال آن یک thread جدید با ارجاع (reference) متد childthreadcall() راه اندازی می شود. فعالیت های thread (thread activity) اصلی مسقیم روی صفحه نمایش داده می شوند.