Silverlight 4

پژوهش و نگارش: وحید نصیری
Vahid_Nasiri@Yahoo.Com
بهار ۱۳۸۹

فهرست مطالب

فصل ۲۴ – برنامه نویسی چند ریسمانی در Silverlight

مقدمه

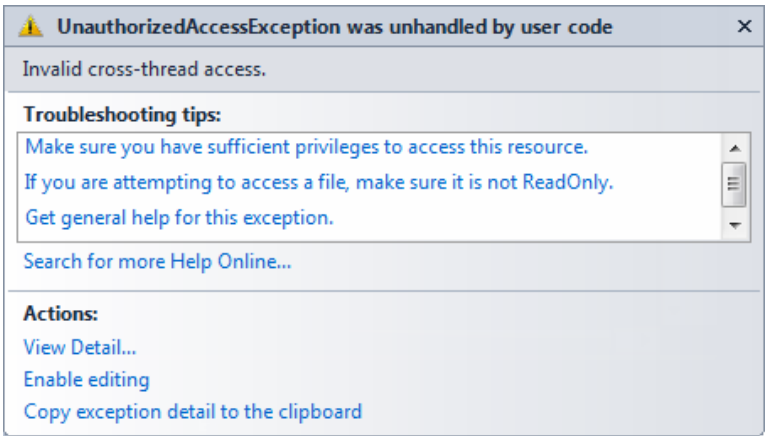
برنامه نویسی چند ریسمانی (Multithreading) مبحث مفصلی است و می‌تواند موضوع یک کتاب جامع باشد. در این فصل تنها قصد داریم با نکات ریزی که برنامه‌نویس‌ها حین استفاده از Threads در Silverlight باید به آن‌ها دقت داشته باشند، آشنا شویم. برای مثال حین کار با Threads چگونه باید رابط کاربری برنامه را به روز کرد یا نکات مرتبط با لیست‌های ویژه بکارگرفته شده در عملیات Binding در این زمینه کدامند و مباحثی از این دست.

به روز رسانی رابط کاربر برنامه از طریق یک Thread

همانند برنامه‌های WinForms و یا WPF ، برنامه‌های Silverlight نیز بر اساس مدل single-threaded apartment اجرا می‌شوند. در این حالت تنها یک Thread تمام برنامه را اجرا و مدیریت کرده و مالک اصلی کلیه‌ی اشیاء رابط کاربری برنامه نیز می‌باشد. به علاوه تمام این عناصر UI دارای thread affinity نیز می‌باشند؛ به این معنا که ریسمانی که آن‌ها را خلق کرده است، مالک آن‌ها بوده و سایر Threads نمی‌توانند به صورت مستقیم به آن‌ها دسترسی داشته باشند. اگر در برنامه‌های خود این قانون را رعایت نکنید، باید منتظر از کار افتادن آنی کل برنامه باشید. برای دسترسی به اشیاء رابط کاربری یک برنامه‌ی Silverlight باید از شیء Dispatcher کمک گرفت. Dispatcher مالک Thread اصلی برنامه بوده و صف آیتم‌های کاری آن‌را مدیریت می‌کند. Dispatcher در حین اجرای برنامه، درخواست‌های کاری جدید را پذیرفته و تنها یک کار را در هر لحظه انجام خواهد داد.

Dispatcher در حقیقت وهله‌ای است از کلاس System.Windows.Threading.Dispatcher که برای اولین بار در WPF معرفی گشت. برای دسترسی به آن از طریق هر المانی می‌توان به خاصیت Dispatcher آن رجوع کرد. کلاس Dispatcher دارای دو متد مهم است:

- CheckAccess : مشخص می‌سازد که آیا امکان دسترسی به Thread اصلی برنامه وجود دارد و آیا هم اکنون در این Thread قرار داریم یا خیر.
- BeginInvoke : امکان انتقال و اجرای کدهای یک Thread را به Thread اصلی برنامه که تحت کنترل Dispatcher است، فراهم می‌سازد.



شکل ۱– پیغام دسترسی غیرمجاز به عناصر UI از طریق یک ریسمان دیگر.

برای توضیحات بیشتر لطفا به مثالی در این زمینه دقت بفرمائید. کدهای XAML این مثال در ادامه ذکر شده‌اند:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication97.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
    "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <StackPanel>
        <StackPanel Orientation="Horizontal"
                    HorizontalAlignment="Center">
            <Button Content="BreakRules"
                Margin="5"
                Name="btnBreakRules"
                Click="btnBreakRules_Click"
                />
            <Button Content="FollowRules"
                Margin="5"
                Name="btnFollowRules"
                Click="btnFollowRules_Click"
                />
        </StackPanel>
        <TextBlock Name="txt"
                   Margin="10" HorizontalAlignment="Center" />
    </StackPanel>
</UserControl>
```

در این مثال دو دکمه را به همراه یک TextBlock بر روی صفحه قرار داده‌ایم. توسط یک دکمه روش نادرست به روز رسانی رابط کاربر از یک Thread دیگر نمایش داده شده و توسط دکمه‌ی دیگر روش صحیح به همراه استفاده از شیء Dispatcher بیان گردیده است.

MainPage.xaml.cs

```
using System.Threading;
using System.Windows;

namespace SilverlightApplication97
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void btnBreakRules_Click(object sender,
                    RoutedEventArgs e)
        {
            var thread = new Thread(updateTextWrong);
            thread.Start();
        }

        private void updateTextWrong()
        {
            txt.Text = "Here is some new text.";
        }

        private void btnFollowRules_Click(object sender,
                RoutedEventArgs e)
        {
            var thread = new Thread(updateTextRight);
            thread.Start();
        }

        private void updateTextRight()
        {
            this.Dispatcher.BeginInvoke((ThreadStart)(
               () => txt.Text = "Here is some new text."));
        }
    }
}
```

در این مثال اگر بر روی دکمه‌ی BreakRules کلیک شود، با پیغام خطای شکل ۱ برنامه خاتمه خواهد یافت.

نکته‌ی دیگری را که باید در همین زمینه به آن اشاره کرد، انجام اعمال غیرهمزمان همانند کار با Web Services است. در اینجا نیز در متد خاتمه‌ی کار باید از Dispatcher.BeginInvoke برای کار با رابط کاربری برنامه استفاده کرد. البته جهت بررسی این مطلب می‌توان از متد Dispatcher.CheckAccess نیز استفاده کرد؛ آیا هم اکنون در Thread اصلی برنامه قرار داریم یا خیر؟

C#

```
if(!Dispatcher.CheckAccess())
{
     Dispatcher.BeginInvoke(()=>resultBox.Text = e.Response);
}
else
{
     resultBox.Text = e.Response;
}
```

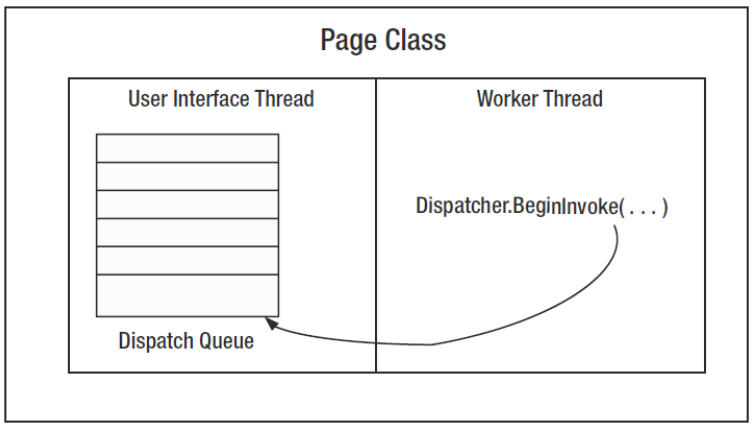
برای یادآوری مفاهیم زبان C# لازم به ذکر است که سه قطعه کد ذیل به یک مفهوم می‌باشند:

C#

```
// With Simple Lambda
Dispatcher.BeginInvoke(() => DoSomething());

// Also With Lambda
Dispatcher.BeginInvoke(() =>
  {
    DoSomething();
  });

// or with Anonymous Delegate
Dispatcher.BeginInvoke(delegate()
{
    DoSomething();
});
```

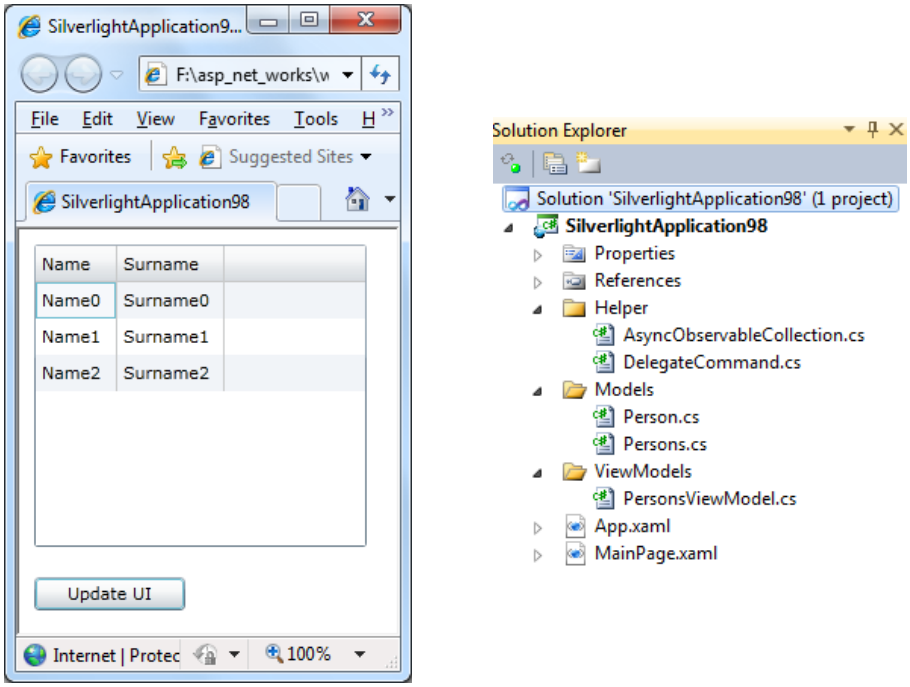


شکل ۲– استفاده از متد BeginInvoke جهت قرار دادن کدهای مورد نظر در صف اجرایی ریسمان اصلی برنامه

به روز رسانی یک شیء ObservableCollection از طریق ریسمانی دیگر

از آنجائیکه اشیاء ObservableCollection در حین عملیات Binding به صورت همزمان رابط کاربری برنامه را به روز می‌کنند، امکان تغییری در آن‌ها از یک Thread دیگر بجز Thread اصلی برنامه وجود ندارد و در صورت سعی به اینکار با همان خطای مطرح شده در مثال قبل مواجه خواهیم شد. برای این منظور یک کلاس کمکی جهت کار با اشیاء ObservableCollection در برنامه‌های چند ریسمانی تهیه شده است که در طی مثال بعد معرفی خواهد شد.

یک پروژه‌ی جدید Silverlight را آغاز نمائید (شکل ۳). در این مثال از همان کلاس معروف DelegateCommand معرفی شده در طی فصول قبل جهت انتقال رخدادها به ViewModel برنامه کمک خواهیم گرفت. کدهای کلاس جدید AsyncObservableCollection را در ادامه ملاحظه خواهید نمود. این کلاس دسترسی امنی را به شیء ObservableCollection از طریق ریسمان‌های دیگر برنامه مهیا می‌سازد. با کمک این کلاس تنها تغییری که در برنامه‌های شما باید انجام شود، استفاده از شیء AsyncObservableCollection بجای شیء متداول ObservableCollection می‌باشد.



شکل ۳– ساختار پوشه‌ها و فایل‌های پروژه به روز رسانی UI از طریق یک ریسمان دیگر

AsyncObservableCollection.cs

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
```

```
using System.ComponentModel;
using System.Threading;

namespace SilverlightApplication98.Helper
{
    public class AsyncObservableCollection<T> : ObservableCollection<T>
    {
        private SynchronizationContext _synchronizationContext =
            SynchronizationContext.Current;
        public AsyncObservableCollection()
        {
        }

        public AsyncObservableCollection(IEnumerable<T> list)
            : base(list)
        {
        }

        protected override void
              OnCollectionChanged(NotifyCollectionChangedEventArgs e)
        {
            if (SynchronizationContext.Current ==
                                    _synchronizationContext)
            {
          // Execute the CollectionChanged event on the current thread
                raiseCollectionChanged(e);
            }
            else
            {
          // Post the CollectionChanged event on the creator thread
                _synchronizationContext.Post(raiseCollectionChanged, e);
            }
        }

        private void raiseCollectionChanged(object param)
        {
          // We are in the creator thread, call the base
          //implementation directly
            base.OnCollectionChanged(
                      (NotifyCollectionChangedEventArgs)param);
        }

        protected override void
               OnPropertyChanged(PropertyChangedEventArgs e)
        {
            if (SynchronizationContext.Current ==
                                _synchronizationContext)
            {
```

```
            // Execute the PropertyChanged event on the current thread
            raisePropertyChanged(e);
        }
        else
        {
         // Post the PropertyChanged event on the creator thread
            _synchronizationContext.Post(raisePropertyChanged, e);
        }
    }

    private void raisePropertyChanged(object param)
    {
     // We are in the creator thread, call the base
     //implementation directly
        base.OnPropertyChanged((PropertyChangedEventArgs)param);
    }
  }
}
```

کدهای کلاس‌های مدل برنامه در ادامه ذکر شده‌اند:

Person.cs

```
using System.ComponentModel;

namespace SilverlightApplication98.Models
{
    public class Person : INotifyPropertyChanged
    {
        private string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name == value) return;
                _name = value;
                raisePropertyChanged("Name");
            }
        }

        private string _surname;
        public string Surname
        {
            get { return _surname; }
            set
            {
                if (_surname == value) return;
                _surname = value;
                raisePropertyChanged("Surname");
            }
```

```
        }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
        #endregion
    }
}
```

همانطور که در کدهای کلاس Persons ذیل نیز مشهود است، اینبار برای تهیه‌ی شیءایی که بیانگر لیستی از اشیاء کلاس Person است، از امکانات کلاس AsyncObservableCollection معرفی شده استفاده کرده‌ایم.

Persons.cs

```
using SilverlightApplication98.Helper;
namespace SilverlightApplication98.Models
{
    public class Persons : AsyncObservableCollection<Person>
    {
    }
}
```

کدهای ViewModel برنامه را در ادامه ملاحظه خواهید نمود. این کلاس لیستی از اشیاء Person را در اختیار View برنامه قرار می‌دهد. همچنین روال رویدادگردان کلیک بر روی دکمه‌ی به روز رسانی رابط کاربر را نیز مدیریت می‌کند.

PersonsViewModel.cs

```
using System.Threading;
using System.Windows.Input;
using SilverlightApplication98.Helper;
using SilverlightApplication98.Models;

namespace SilverlightApplication98.ViewModels
{
    public class PersonsViewModel
    {
        public Persons Persons { set; get; }
        public ICommand UpdateUiCommand { set; get; }

        public PersonsViewModel()
```

```
        {
            Persons = new Persons
                        {
                            new Person
                                {
                                    Name = "Name0",
                                    Surname = "Surname0"
                                }
                        };
            UpdateUiCommand = new DelegateCommand<Person>(
                addPerson, canAddPerson);
        }

        private bool canAddPerson(Person arg)
        {
            return true;
        }

        private void addPerson(Person obj)
        {
            startThread();
        }

        void startThread()
        {
            new Thread(work).Start();
        }

        private int _count;
        private void work()
        {
            _count++;
            var tmpPerson = new Person
                            {
                                Name = "Name" + _count,
                                Surname = "Surname" + _count
                            };
            Persons.Add(tmpPerson);
        }
    }
}
```

با توجه به اینکه شیء Persons تعریف شده از نوع AsyncObservableCollection است، دیگر نگرانی در مورد به روز رسانی آن از طریق یک ریسمان دیگر نخواهیم داشت و برنامه دچار مشکل نخواهد گردید.

کدهای XAML مرتبط با View برنامه در ادامه ذکر شده‌اند:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication98.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
    xmlns:
     sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
    xmlns:vm="clr-namespace:SilverlightApplication98.ViewModels">
    <UserControl.Resources>
        <vm:PersonsViewModel x:Key="vmPersonsViewModel" />
    </UserControl.Resources>
    <StackPanel
        DataContext=
           "{Binding Source={StaticResource vmPersonsViewModel}}">
        <sdk:DataGrid
            HorizontalAlignment="Left"
            VerticalAlignment="Top"
            AutoGenerateColumns="True"
            Height="200"
            Margin="10"
            ItemsSource="{Binding Persons}"
            Name="dataGrid1"
            Width="220" />
        <Button Content="Update UI"
                Width="100"
                Margin="10"
                HorizontalAlignment="Left"
                Command="{Binding UpdateUiCommand}"
                />
    </StackPanel>
</UserControl>
```

در اینجا یک کنترل DataGrid را از طریق جعبه ابزار VS.NET کشیده و بر روی فرم رها کرده‌ایم تا ارجاعات لازم به اسمبلی‌های مورد نیاز و همچنین فضاهای نام مرتبط با آن به صورت خودکار ایجاد شوند. کلیه ارتباطات این View با ViewModel برنامه از طریق عملیات Binding مدیریت می‌شود.

لازم به ذکر است از کلاس AsyncObservableCollection در برنامه‌های WPF خود نیز می‌توانید به همین ترتیب استفاده کنید.