WPF Dispatcher - Introduction and How to use?

WPF wpf

Before we learn what is dispatcher and why we need it, we need to understand what is the apartments of the Thread.

Thread Apartments

All objects in the process are grouped into Apartments.

There are two types of apartments in Threads:

- Single-Threaded Apartments
- Multi-Threaded Apartments

Single-Threaded Apartment (STA)

Single-threaded apartments contains only one thread. All objects in this apartment can receive method calls from only this thread. Objects does not need synchronization because all methods calls are comes synchronously from single thread.

Single-threaded apartment needs a message queue to handle calls from other threads. When other threads calls an object in STA thread then the method call are queued in the message queue and STA object will receive a call from that message queue.

Multi-Threaded Apartment (MTA)

Multi-threaded apartments contains one or more threads. All objects in this apartment can receive calls from any thread. All objects are self responsible for maintaining the synchronization of their data.

WPF Dispatcher

A WPF application must start in single-threaded apartment thread. STA have a message queue to synchronize method calls within his apartment. As well as other threads outside the apartment can't update the objects directly. They must place their method call into the message queue to update the objects in STA.

Dispatcher owns the message queue for the STA thread.

When you execute a WPF application, it automatically create a new Dispatcher object and calls its Run method. Run method is used for initializing the message queue.

When WPF application starts, it creates two threads:

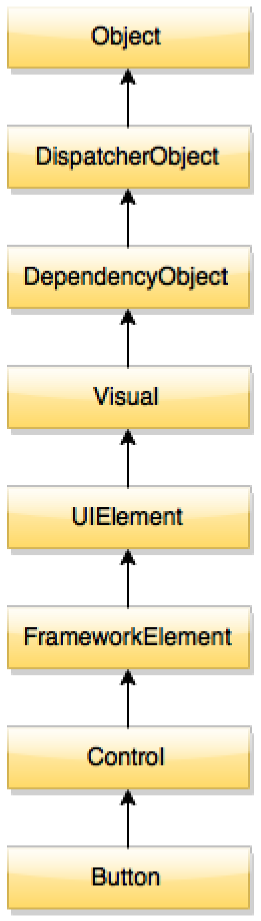
1. Render thread
2. UI thread

UI thread is responsible all the user inputs, handle events, paints screen and run the application code. Render threads runs in the background and used for render the WPF screen.

WPF Dispatcher is associated with the UI thread. The UI thread queues methods call inside the Dispatcher object. Whenever your changes the screen or any event executes, or call a method in the code-behind all this happen in the UI thread and UI thread queue the called method into the Dispatcher queue. Dispatcher execute its message queue into the synchronous order.

How all WPF objects refers to single Dispatcher?

Every WPF control whether it is Window, button or textbox inherits from DispatcherObject. Below is class hierarchy diagram.



When WPF creates an instance of Button, it calls the protected constructor of DispatcherObject. DispatcherObject contains a property of type Dispatcher. In the constructor, it save the reference of current thread Dispatcher to Dispatcher property of DispatcherObject.

Why we need Dispatcher?

WPF works with Dispatcher object behind the scenes and we don't need to work with Dispatcher when we are working on the UI thread.

When we create a new thread for offloading the work and want to update the UI from the other thread then we must need Dispatcher. Only Dispatcher can update the objects in the UI from non-UI thread.

Dispatcher provides two methods for registering method to execute into the message queue.

1. Invoke
2. BeginInvoke

Invoke

Invoke method takes an Action or Delegate and execute the method synchronously. That means it does not return until the Dispatcher complete the execution of the method.

Here is an example of Invoke:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        Task.Factory.StartNew(() =>
            {
                InvokeMethodExample();
            });
    }

    private void InvokeMethodExample()
    {
        Thread.Sleep(2000);
        Dispatcher.Invoke(() =>
            {
                btn1.Content = "By Invoke";
            });
    }
}
```

Above code will create a new thread using Task.Factory and immediately start the thread. In the InvokeMethodExample if we try to directly call to update the Content property of btn1 object. It will throws a *System.InvalidOperationException*. We have used Invoke method of Dispatcher. In the Invoke method, I pass the Action and update the Content property of Button object. It will not throws any error and successfully update the Content property.

BeginInvoke

BeginInvoke method take a Delegate but it executes the method asynchronously. That means it immediately returns before calling the method.

```
public MainWindow()
{
    InitializeComponent();
    Task.Factory.StartNew(() =>
        {
            BeginInvokeExample();
        });
}

private void BeginInvokeExample()
{
    DispatcherOperation op = Dispatcher.BeginInvoke(
            btn1.Content = "By BeginInvoke";
        }));
}
```

BeginInvoke returns a DispatcherOperation object. This object can be used for knowing the status of operation whether it is completed or not. It also provides two event Aborted and Completed.