高薪必备 -线程

1、线程分类

类型	描述
主线程（Main Thread）	应用程序启动时的第一个线程
后台线程（Background Thread）	不阻止程序退出
前台线程（Foreground Thread）	阻止程序退出，直到完成
池化线程（Thread Pool Threads）	由 .NET 线程池管理
显式线程（new Thread()）	手动创建和控制的线程

2、线程的生命周期状态

状态	含义
Unstarted	已创建但尚未启动
Running	正在运行
WaitSleepJoin	调用了 `Wait`、`Sleep`、`Join`
Suspended	已挂起（不推荐使用）
Aborted	线程已请求终止
Stopped	线程已结束

3、线程的常用属性和方法

`IsBackground`	是否为后台线程
`Priority`	设置线程优先级（High/AboveNormal/Normal 等）
`Start()` / `Join()` / `Abort()` / `Sleep()`	控制线程行为
`CurrentThread`	获取当前线程对象
`Name`	给线程命名，方便调试

4、Thread代码案例

□□□□□□□□□

```
Thread thread = new Thread(() =>
{
    Console.WriteLine("□□□□□□          ");
    Thread.Sleep(2000);
    Console.WriteLine("□□□□       ");
});
thread.IsBackground = false; // □□□□□□□
thread.Start();
thread.Join(); // □□□□□□□□□□
Console.WriteLine("□□□□□        ");



 □□□     Join() □□□□□□□□□□□□□□□□
```

□□□□□□□□□□

```
Thread thread = new Thread(obj =>
{
    string message = obj as string;
    Console.WriteLine($"□□□□       : {message}");
});
thread.Start("Hello from thread");
```

□□□□□□□□□□□□

```
Thread thread = new Thread(() =>
{
    Console.WriteLine($"□□□□□        : {Thread.CurrentThread.Priority}");
});
thread.Priority = ThreadPriority.Highest;
thread.Start();


□□□□□□□□□□□□□□□□□□
```

□□□□□□□□□□□

```
AppDomain.CurrentDomain.UnhandledException += (sender, args) =>
{
```

```
    Console.WriteLine("□□□□□□          " + (args.ExceptionObject as Exception)?.Message);
};

Thread thread = new Thread(() =>
{
    throw new InvalidOperationException("□□□□       ");
});
thread.Start();

// □□□□□□□□□□□□□□□□□□□□
```

□□□□□□□□□□□□ lock + Thread□

```
private static readonly object _sync = new object();
private static int _counter = 0;

Thread t1 = new Thread(() =>
{
    for (int i = 0; i < 1000; i++)
    {
        lock (_sync)
        {
            _counter++;
        }
    }
});

Thread t2 = new Thread(() =>
{
    for (int i = 0; i < 1000; i++)
    {
        lock (_sync)
        {
            _counter--;
        }
    }
});

t1.Start();
t2.Start();
```

```
t1.Join();
t2.Join();

Console.WriteLine(_counter); // □□□□       0
```

□□ ParameterizedThreadStart □□□□□□□

```
class Person
{
    public string Name { get; set; }
}

Thread thread = new Thread((obj) =>
{
    var person = obj as Person;
    Console.WriteLine($"□□□    {person.Name}");
});
thread.Start(new Person { Name = "□□   " });
```

□□ `Thread.Join(TimeSpan)` □□□□□□

```
Thread thread = new Thread(() =>
{
    Thread.Sleep(3000);
    Console.WriteLine("□□□□      ");
});

thread.Start();
bool finished = thread.Join(TimeSpan.FromSeconds(2));

if (!finished)
{
    Console.WriteLine("□□□□□□□□□             ");
}
```

□□ `AutoResetEvent` □ `ManualResetEvent` □□□□□□

```
AutoResetEvent waitHandle = new AutoResetEvent(false);
```

```
Thread worker = new Thread(() =>
{
    Console.WriteLine("□□□□      ...");
    waitHandle.WaitOne();
    Console.WriteLine("□□□□□□□□□□              ");
});

worker.Start();

Thread.Sleep(2000);
waitHandle.Set(); // □□□□



var signal = new ManualResetEvent(false);

Thread worker = new Thread(() =>
{
    Console.WriteLine("□□□□□□         ");
    Thread.Sleep(2000);
    Console.WriteLine("□□□□□□         ");
    signal.Set(); // □□□□
});

worker.Start();
Console.WriteLine("□□□□□        ...");
signal.WaitOne(); // □□□□
Console.WriteLine("□□□□□        ");
```

□□ `Thread.Abort()` □ `Thread.Interrupt()` □□□□□

```
Thread thread = new Thread(() =>
{
    try
    {
        Console.WriteLine("□□□□      ");
        Thread.Sleep(Timeout.Infinite); // □□□□□□□
    }
    catch (ThreadInterruptedException)
    {
        Console.WriteLine("□□□□□        ");
```

```
    }
    catch (ThreadAbortException)
    {
        Console.WriteLine("□□□□□        ");
        Thread.ResetAbort(); // □□□□□□□
    }
});

thread.Start();
thread.Interrupt(); // □□    Sleep/Wait □□
// thread.Abort(); // □□□□□□□□□
```

□□□□□□□□ TLS□ ——ThreadStatic / AsyncLocal

```
//□□    [ThreadStatic] □□□□□□□□
[ThreadStatic]
private static int _threadId;

Thread t1 = new Thread(() =>
{
    _threadId = 1;
    Console.WriteLine($"□□    1 ID: {_threadId}");
});

Thread t2 = new Thread(() =>
{
    _threadId = 2;
    Console.WriteLine($"□□    2 ID: {_threadId}");
});

t1.Start();
t2.Start();



//□□    AsyncLocal<T> □□□□□□□□□□□□□□□□                    async/await□
AsyncLocal<int> asyncLocal = new AsyncLocal<int>();

asyncLocal.Value = 1;
Console.WriteLine($"□□□□      : {asyncLocal.Value}");
```

```
Task.Run(() =>
{
    asyncLocal.Value = 2;
    Console.WriteLine($"□□□□□        : {asyncLocal.Value}");
}).Wait();
```

5□□□□□ ThreadPool □

□□□□□

```
ThreadPool.QueueUserWorkItem(state =>
{
    Console.WriteLine("□□□□□□□□              ");
});
```

□□□□□□□□□

```
int workerThreads, completionPortThreads;
ThreadPool.GetAvailableThreads(out workerThreads, out completionPortThreads);
Console.WriteLine($"□□□□□□□□□              {workerThreads}");
```

□□□□□□□□□□□□□□

```
ThreadPool.SetMinThreads(10, 10); // □□□□□□□
ThreadPool.SetMaxThreads(100, 100); // □□□□□□□
```

6□ Task□□□□□□□□□□□□

□□ CancellationToken □□□□

```
var cts = new CancellationTokenSource();
cts.CancelAfter(TimeSpan.FromSeconds(2));

try
{
    await Task.Run(async () =>
    {
        while (!cts.Token.IsCancellationRequested)
        {
            await Task.Delay(500);
```

```
            Console.WriteLine("□□□□      ");
        }
    }, cts.Token);
}
catch (OperationCanceledException)
{
    Console.WriteLine("□□□□□       ");
}
```

□□ ContinueWith() □□□□□

```
Task<int> task1 = Task.Run(() => 100);
task1.ContinueWith(prev => Console.WriteLine($"□□  : {prev.Result}"));
```

□□ ValueTask □□□□□□□□□□□□□□□

```
public ValueTask<int> ComputeAsync(int a, int b, CancellationToken ct = default)
{
    if (ct.IsCancellationRequested)
        return new ValueTask<int>(Task.FromCanceled<int>(ct));

    return new ValueTask<int>(a + b);
}
```

□□ ConfigureAwait(false) □□□□□□□

```
await SomeAsyncMethod().ConfigureAwait(false);



□□□□□□□□□□□□□□                    UI □□□□□□
```

□□ TaskCompletionSource<T> □□□□□□

```
public Task<int> DelayedResultAsync()
{
    var tcs = new TaskCompletionSource<int>();
    new Thread(() =>
    {
        Thread.Sleep(1000);
        tcs.SetResult(42);
```

```
    }).Start();
    return tcs.Task;
}
```

□□□ #2
□ □□ □□□ 4 □□ 2025 21:22:50
□ □□ □□□ 4 □□ 2025 22:07:54