线程池（Thread Pool）由System.Threading命名空间的ThreadPool实现的。

最大线程数是25的线程池执行100个任务的步骤？

1. 当线程池被创建后，里面就会创建5个空线程(和下限值相同)。
2. 当我们向线程池排入一个任务后，就会有一个空线程接手该任务，然后运行起来。随着我们不断地向线程池中排入任务，线程池中的空线程逐一接手任务并被执行。
3. 随着任务的增加，任务数量等于下限。再排入任务，线程池并不会立即创建新线程，而是等待约500毫秒，这么做的目的是看看在这段时间内是否有其他线程完成任务来接手这个请求，这样就可以避免因创建新线程而造成的消耗。如果这段时间内没有线程完成任务，就创建-一个新线程去执行新任务。
4. 任务数量超过下限后，每排入一个新任务，就增加一个新线程，这段期间，任务和线程数量都持续增加，直至线程数量达到上限值为止。
5. 线程数量达到上限，继续增加任务，线程数量将不再增加。比如你向线程池中排入100个任务，则只有25个进入线程池(和上限相同)，另外75个在线程池外排队等待。当线程池中的某个线程完成任务后，并不会被终止，而是从等待队列中选择一个任务继续执行，这样就减少了因创建和销毁线程而消耗的时间。
6. 当排入所有的任务后，随着线程池内的任务被逐步完成，线程池外部等候的任务被逐步调入线程池，任务的数量逐步减少，但线程的数量保持恒定，始终和上限值相同。
7. 随着任务被逐步完成，总有某一时刻，任务数量会小于上限值，这时线程池内多余的线程会在空闲2分钟后被释放并回收相关资源。线程数目逐步减少，直到达到下限值为止。
8. 当任务数量减小到下限值之下时，线程池中的线程数目保持不变(始终和下限值相同)，其中一部分在执行任务，另一部分处于空运行状态。

为什么使用线程池？

创建和销毁一个线程都需要系统开销，线程池能节省开销。本质是一个线程结束后能继续服务其他任务，（不是立刻销毁，创建线程需要申请栈，TCB数据结构等会消耗时间）这样系统中会一直有几个固定线程为新任务轮流服务，避免了频繁的创建和销毁线程。

什么情况最适合使用线程池？

并发若干持续时间不长，互不干扰的线程。

不适合使用线程池的情况？

1. 线程持续时间长。
2. 需要为线程设置详细的优先级
3. 线程执行过程中需要对线程进行操作和控制，如睡眠和挂起

ThreadPool类是静态类，常用方法：

GetMaxThreads( ) 获取线程池中线程数目的上限

GetMinThreads( ) 获取线程池中线程数目的下限

SetMax'Threads() 设置线程池中线程数目的上.限

SetMinThreads() 设置线程池中线程数日的下限

QueueUserWorkItem() 将工作任务排入线程池

一个应用程序最多只有一个线程池，线程池创建于首次向线程池排入工作函数的时候创建。

ThreadPool.QueueUserWorkItem(工作函数，工作函数的参数);

工作函数是一个WaitCallback委托

public delegate void WaitCallback( object o);

/\*程序功能：

\* 9个线程分别计算0-8的阶乘，存储于数组result并打印之

\*/

namespace ThreadPoolDemo {

class Program {

static int finishedThreadCount = 0;//运行结束的进程数量

static int[] result = new int[9 + 1];//保存计算结果

static void Main(string[] args) {

for(int i = 0; i < 9; ++i) {

ThreadPool.QueueUserWorkItem(new WaitCallback(factorial), i);

}

while (finishedThreadCount < 9) {

;//空，主线程等待

}

for(int i = 1; i <=9; ++i) {

Console.WriteLine("第" + i + "个线程的运算结果是" + result[i-1]);

}

}

public static void factorial(object n) {

result[0] = 1;

result[1] = 1;

for(int i = 2; i <= (int)n; ++i) {

result[i] = result[i - 1] \* i;

}

finishedThreadCount++;

}

}

}

运行结果：

第1个线程的运算结果是1

第2个线程的运算结果是1

第3个线程的运算结果是2

第4个线程的运算结果是6

第5个线程的运算结果是24

第6个线程的运算结果是120

第7个线程的运算结果是720

第8个线程的运算结果是5040

第9个线程的运算结果是40320