在.NET Framework4.0下，多线程代码选择：

|  |  |  |
| --- | --- | --- |
| 优先 | 次优先 | 不得以 |
| Parallel(包含PLinq)  Task | TreadPool  异步 | Thread |

使用Thread最重要的理由是控制线程的优先级。

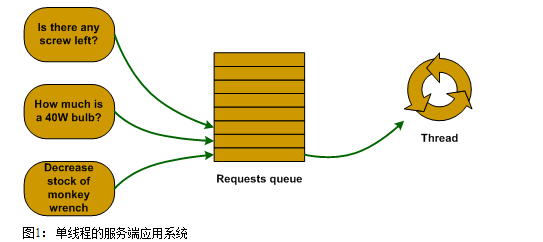
线程池的应用：

参考：

<http://www.cnblogs.com/wildman/archive/2008/08/22/1274170.html#commentform>

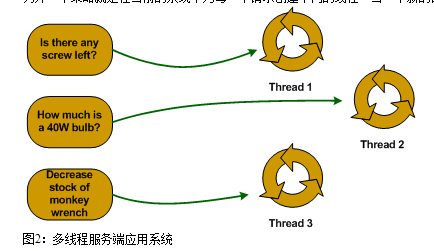
“硬件商店”的例子，客户端/服务器，服务端处理来自客户端的请求。

单线程：



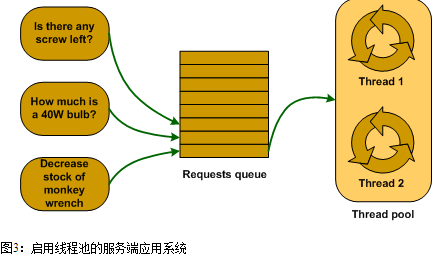
特点：每次执行一个请求，可能会出现很长的响应时间。

多线程：



特点：CPU利用率提高，但需要不停地创建和销毁线程，大多数时间浪费在上下文切换过程中。

线程池：



特点：不要为每个请求都创建线程，线程池可以通过增加/减少其中的线程的数量使系统能处理更多的请求或获得更多的CPU时间。

.NET Framework中的线程池。

在C#中，可以使用TreadPool来使用线程池。

例：程序threadpool\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 线程池

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace treadpool\_test1

{

class ThreadPoolTest

{

static void PooledFunc(object state)

{

Console.WriteLine("Processing request '{0}' " + "Is pool thread: {1}, Hash: {2}", (string)state,

Thread.CurrentThread.IsThreadPoolThread, Thread.CurrentThread.GetHashCode());

// Thread.Sleep(2000);

int ticks = Environment.TickCount;

while (Environment.TickCount - ticks < 2000) ;

Console.WriteLine("Request processed.");

}

static void Main(string[] args)

{

Console.WriteLine("Main thread. Is pool thread: {0}, Hash: {1}",

Thread.CurrentThread.IsThreadPoolThread, Thread.CurrentThread.GetHashCode());

// 线程池

WaitCallback call\_back1 = new WaitCallback(PooledFunc);

ThreadPool.QueueUserWorkItem(call\_back1, "Is there any screw left?");

ThreadPool.QueueUserWorkItem(call\_back1, "How much is a 4w bulb?");

ThreadPool.QueueUserWorkItem(call\_back1, "Decrease stock of monkey wrench");

Console.ReadLine();

}

}

}