C# PLINQ 内存列表查询优化历程 - 文章

原文出处： dengxi

产品中（基于ASP.NET MVC开发）需要经常对药品名称及名称拼音码进行下拉匹配及结果查询。为了加快查询的速度，所以我最开始就将其加入内存中（大约有六万五千条数据）。

下面附实体类。

```
public class drugInfo
{
    public int drug_nameid    { get; set; }
    public string drug_name    { get; set; }
    public string drug_search_code    { get; set; }
}
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();
key = key.ToLower();
var resultList = cacheList.Where(m => m.drug_name.ToLower().Contains(key) ||
m.drug_search_code.ToLower().Contains(key)).ToList();
stopWatch.Stop();
double eMseconds = Math.Max(0, stopWatch.Elapsed.TotalSeconds);
```

刷新页面几次，得到个平均用时约35MS左右。

第二次做法：

为了减少CPU的运算，我们将LINQ表达式中的转小写操作优化一下，先在缓存列表上做些动作，将名称和搜索码先转小写存储。

下面为改进过的实体类。

C#

```
public class drugInfo
{
    public int drug_nameid    { get; set; }
    public string drug_name    { get; set; }
    public string drug_search_code    { get; set; }
    public string lower_drug_name    { get; set; }
    public string lower_drug_search_code    { get; set; }
}
```

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();
key = key.ToLower();
var
resultList = cacheList.Where(m =>
m.lower_drug_name.Contains(key) ||
m.lower_drug_search_code.Contains(key)).ToList();
stopWatch.Stop();
double eMseconds = Math.Max(0, stopWatch.Elapsed.TotalSeconds);
ViewBag.useTime = string.Format("用时{0}秒rn", eMseconds);
```

刷新页面几次，得到个平均用时约16MS左右。

虽然这样做，内存列表中会多一些冗余数据，但是得到的性能提升有一倍了。

第三次做法：
启用PLINQ的并行计算，并行计算是NET4.0的特性，可以利用CPU多核的处理能力，提高运算效率，但是不一定是成倍的

LIST等泛型启用并行计算很简单，使用AsParallel()即可，改进如下：

C#

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();
key = key.ToLower();
var resultList = cacheList.AsParallel().Where(m => m.lower_drug_name.Contains(key) ||
m.lower_drug_search_code.Contains(key)).ToList();
stopWatch.Stop();
double eMseconds = Math.Max(0, stopWatch.Elapsed.TotalSeconds);
ViewBag.useTime = string.Format("用时{0}秒rn", eMseconds);
```

同样，我们多刷新页面几次，获得的平均时间为10MS左右。

当然，写到这里，大家以为这次的优化就结束了，至少我当时是这么想的。

但是事实上，碰到了一个大麻烦。

由于产品运行于服务器IIS上面，使用AsParallel并行特性时（默认情况下，到底使用多少个线程来执行PLINQ是在程序运行时由TPL决定的。但是，如果你需要限制执行PLINQ查询的线程数目（通常需要这么做的原因是有多个用户同时使用系统，为了服务器能同时服务尽可能多的用户，必须限制单个用户占用的系统资源），我们可以使用ParallelEnumerable. WithDegreeOfParallelism()扩展方法达到此目的。），客户端一个请求就占用了过多的系统资源，导致应用程序池假死。无法提供服务。

我也尝试过使用WithDegreeOfParallelism设置了一个相对较少的值，但是在使用LOADRUNNER来开启200个并发的时候，也会产生假死的情况，于是，不得不尝试下面第四步的办法。

第四次做法：

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();
key = key.ToLower();
ConcurrentBag resultList = new ConcurrentBag();
Parallel.For(0, cacheList.Count, new ParallelOptions { MaxDegreeOfParallelism = 4 }, (i) =>
{
            var item = cacheList[i];
            if (item.lower_drug_name.Contains(key) ||
item.lower_drug_search_code.Contains(key))
            {
                        resultList.Add(item);
             }
});
stopWatch.Stop();
double eMseconds = Math.Max(0, stopWatch.Elapsed.TotalSeconds);
ViewBag.useTime = string.Format("用时{0}秒rn", eMseconds);
```

时间与第三步没有什么区别，但是这样做解决了并发时，应用程序池假死的问题。至此，困扰两天的问题完美解决，虽然使用Parallel.For会带来结果乱序的问题，但是结果数量已经不多了，再次排序也没有什么关系了。

具体原因参见下面：

ParallelOptions.MaxDegreeOfParallelism指明一个并行循环最多可以使用多少个线程。TPL开始调度执行一个并行循环时，通常使用的是线程池中的线程，刚开始时,如果线程池中的线程很忙,那么,可以为并行循环提供数量少一些的线程（但此数目至少为1，否则并行任务无法执行，必须阻塞等待）。等到线程池中的线程完成了一些工作,则分配给此并行循环的线程数目就可以增加，从而提升整个任务完成的速度，但最多不会超过ParallelOptions.MaxDegreeOfParallelism所指定的数目。

PLINQ的WithDegreeOfParallelism()则不一样，它必须明确地指出需要使用多少个线程来完成工作。当PLINQ查询执行时，会马上分配指定数目的线程执行查询。

之所以PLINQ不允许动态改变线程的数目，是因为许多PLINQ查询是“级联”的，为保证得到正确的结果，必须同步参与的多个线程。如果线程数目不定，则要实现线程同步非常困难。

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子
头条 - 分享和发现有价值的内容与观点
相亲 - 为IT单身男女服务的征婚传播平台
资源 - 优秀的工具资源导航
翻译 - 翻译传播优秀的外文文章
文章 - 国内外的精选文章
设计 - UI,网页，交互和用户体验
iOS - 专注iOS技术分享
安卓 - 专注Android技术分享
前端 - JavaScript, HTML5, CSS
Java - 专注Java技术分享
Python - 专注Python技术分享