由于for和foreach是命令式语言的产物，所以递归在函数式编程中使用的相当普遍，但递归的问题在于它会导致堆栈的溢出，为了解决这个问题人们重新设计了一种递归的算法--尾递归，利用这种算法我们可以在编译时将CALL指令转化为JMP指令，这样递归调用就只会占用一个堆栈帧，但这样做的前提是必须首先对递归算法进行改造使之符合尾递归的标准，那么什么是符合尾递归的标准呢?

先看一个普通递归调用的例子:

```
class Program
{
    static void Main(string[] args)
    {
        var result = FactorialNormalRecursively(5);
    }

    public static int FactorialNormalRecursively(int n)
    {
        if (n == 1)
        {
            return 1;
        }
        return FactorialNormalRecursively(n - 1) * n;
    }
}
```

如果执行 FactorialNormalRecursively(5)那么具体的算法如下:

5 * （4 * （3*（2* （1））））

这种算法的问题在于我们每次都是先执行递归调用再执行*运算，在这种情况下我们是没有办法在编译时将CALL改成JMP的（请参考下文中汇编代码）。

以下是改造后的符合尾递归标准的递归:

```
class Program
{
    static void Main(string[] args)
    {
        var result = FactorialTailRecursively(5, 1);
    }

    public static int FactorialTailRecursively(int n, int result)
    {
        result = result * n;
        if (n == 1)
        {
            return result;
```

```
            }
            return FactorialTailRecursively(n - 1, result);
        }
    }
```

如果执行 FactorialTailRecursively(5，1)那么具体的算法如下：

（（（（5 * 4） * 3） * 2）* 1）

这个算法和之前算法最大的不同在于这个算法是先执行*计算再执行递归调用，在这种情况下我们就可以在编译时将CALL改成JMP从而只用消耗一个堆栈帧（请参考下文中汇编代码）。

具体的汇编代码分析如下：

//普通递归调用汇编源代码

```
int result = FactorialNormalRecursively(5);
00000035  mov         ecx,5【参数n通过ecx传递】
0000003a  call        FD3FB008
0000003f  mov         dword ptr [ebp-44h],eax【结果通过eax传回】
00000042  mov         eax,dword ptr [ebp-44h]
00000045  mov         dword ptr [ebp-40h],eax【保存r值】
public static int FactorialNormalRecursively(int n)
{
00000000  push        ebp
00000001  mov         ebp,esp
00000003  push        edi
00000004  push        esi
00000005  push        ebx
00000006  sub         esp,3Ch
00000009  mov         esi,ecx
0000000b  lea         edi,[ebp-38h]
0000000e  mov         ecx,0Bh
00000013  xor         eax,eax
00000015  rep stos    dword ptr es:[edi]
00000017  mov         ecx,esi
00000019  xor         eax,eax
0000001b  mov         dword ptr [ebp-1Ch],eax
0000001e  mov         dword ptr [ebp-3Ch],ecx
00000021  cmp         dword ptr ds:[009D9134h],0
00000028  je          0000002F
0000002a  call        76AE9109
0000002f  xor         edx,edx
00000031  mov         dword ptr [ebp-40h],edx
00000034  mov         dword ptr [ebp-44h],0
0000003b  nop
    if (n == 1)
    0000003c  cmp         dword ptr [ebp-3Ch],1
```

```
    00000040  setne       al
    00000043  movzx       eax,al
    00000046  mov         dword ptr [ebp-44h],eax
    00000049  cmp         dword ptr [ebp-44h],0
    0000004d  jne         0000005A
    {
    0000004f  nop
        return 1;
        00000050  mov         dword ptr [ebp-40h],1
        00000057  nop
        00000058  jmp         00000073
    }
    return FactorialNormalRecursively(n - 1) * n;
    0000005a  mov         ecx,dword ptr [ebp-3Ch]
    0000005d  dec         ecx
    0000005e  call        FD3FAF98【普通递归在Call自己之后会有一些代码，这是改造成尾递归的关键所在】
    00000063  mov         dword ptr [ebp-48h],eax
    00000066  mov         eax,dword ptr [ebp-3Ch]
    00000069  imul        eax,dword ptr [ebp-48h]
    0000006d  mov         dword ptr [ebp-40h],eax
    00000070  nop
    00000071  jmp         00000073
}
00000073  mov         eax,dword ptr [ebp-40h]
00000076  lea         esp,[ebp-0Ch]
00000079  pop         ebx
0000007a  pop         esi
0000007b  pop         edi
0000007c  pop         ebp
0000007d  ret
```

//符合尾递归的递归调用汇编源代码

```
int result = FactorialTailRecursively(5, 1);
00000035  mov         edx,1【参数result通过edx传递】
0000003a  mov         ecx,5【参数n通过ecx传递】
0000003f  call        FD3FB008
00000044  mov         dword ptr [ebp-44h],eax【结果通过eax传回】
00000047  mov         eax,dword ptr [ebp-44h]
0000004a  mov         dword ptr [ebp-40h],eax【保存r值】
public static int FactorialTailRecursively(int n, int result)
{
00000000  push        ebp
00000001  mov         ebp,esp
00000003  push        edi
00000004  push        esi
```

```
00000005  push        ebx
00000006  sub         esp,40h
00000009  mov         esi,ecx
0000000b  lea         edi,[ebp-38h]
0000000e  mov         ecx,0Bh
00000013  xor         eax,eax
00000015  rep stos    dword ptr es:[edi]
00000017  mov         ecx,esi
00000019  xor         eax,eax
0000001b  mov         dword ptr [ebp-1Ch],eax
0000001e  mov         dword ptr [ebp-3Ch],ecx
00000021  mov         dword ptr [ebp-40h],edx
00000024  cmp         dword ptr ds:[009D9134h],0
0000002b  je          00000032
0000002d  call        76AE9111
00000032  xor         edx,edx
00000034  mov         dword ptr [ebp-44h],edx
00000037  mov         dword ptr [ebp-48h],0
0000003e  nop
    result = result * n;
    0000003f  mov         eax,dword ptr [ebp-3Ch]
    00000042  imul        eax,dword ptr [ebp-40h]
    00000046  mov         dword ptr [ebp-40h],eax
    if (n == 1)
    00000049  cmp         dword ptr [ebp-3Ch],1
    0000004d  setne       al
    00000050  movzx       eax,al
    00000053  mov         dword ptr [ebp-48h],eax
    00000056  cmp         dword ptr [ebp-48h],0
    0000005a  jne         00000066
    {
    0000005c  nop
        return result;
        0000005d  mov         eax,dword ptr [ebp-40h]
        00000060  mov         dword ptr [ebp-44h],eax
        00000063  nop
        00000064  jmp         0000007E
    }
    return FactorialTailRecursively(n - 1,  result);
    00000066  mov         ecx,dword ptr [ebp-3Ch]
    00000069  dec         ecx
    0000006a  mov         edx,dword ptr [ebp-40h]【mov dword ptr [ebp-3Ch],ecx】
    0000006d  call        FD3FAFA0【jmp 0000003f】
    00000072  mov         dword ptr [ebp-4Ch],eax【Remove the line】
    00000075  mov         eax,dword ptr [ebp-4Ch]【Remove the line】
    00000078  mov         dword ptr [ebp-44h],eax【Remove the line】
    0000007b  nop          【Remove the line】
```

```
    0000007c  jmp         0000007E【Remove the line】
}
0000007e  mov         eax,dword ptr [ebp-44h]
00000081  lea         esp,[ebp-0Ch]
00000084  pop         ebx
00000085  pop         esi
00000086  pop         edi
00000087  pop         ebp
00000088  ret
```