赛艇队长

< 2018年3月 >

日	一	二	三	四	五	六
25	26	27	28	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

搜索

找找看

随笔分类

Android开发(21)
git(2)
Java基础(12)
Linux(2)
Python基础(3)
多线程技术(2)
计算机基础(1)
设计模式(2)
数据库学习(5)
算法与数据结构(1)
网络编程(5)
有趣的东西(1)
正则表达式(1)

最新评论

1. Re:http协议无状态中的 "状态" 到底指的是什么？！
感谢博主，已经转载，且在文章中标注了文章作者。
--敲代码的小哥
2. Re:http协议无状态中的 "状态" 到底指的是什么？！
感谢博主，已经转载，且在文章中标注了文章作者。
--ZeroOnes
3. Re:http协议无状态中的 "状态" 到底指的是什么？！
已经转载，且在文章中标注了文章作者。
--eatwhat
4. Re:http协议无状态中的 "状态" 到底指的是什么？！
好文！一股清流啊。
--eatwhat
5. Re:http协议无状态中的 "状态" 到底指的是什么？！
非常棒！
--Jation

阅读排行榜

1. git log命令全解析，打log还能这么随心所欲！(11201)
2. http协议无状态中的 "状态" 到底指的是什么？！(9630)
3. Android中的依赖问题（五种依赖、eclipse、AS、添加第三方库、jar）(6112)
4. 在Java中谈尾递归--尾递归和垃圾回收的比较(4599)
5. 由SOAP说开去 - - 谈谈WebServices、RMI、RPC、SOA、REST、XML、JSON(4156)

评论排行榜

1. http协议无状态中的 "状态" 到底指的是什么？！(10)
2. 华为手机Edittext光标（cursor）颜色修改(6)
3. Android中的依赖问题（五种依赖、eclipse、AS、添加第三方库、jar）(3)
4. Charles maplocal 时中文显示乱码问题(3)
5. 理解整数为什么存成补码的正确姿势！(2)

在Java中谈尾递归--尾递归和垃圾回收的比较

我不是故意在JAVA中谈尾递归的，因为在JAVA中谈尾递归真的是要绕好几个弯，只是我确实只有JAVA学得比较好，虽然确实C是在学校学过还考了90+，真学得没自学的JAVA好

不过也是因为要绕几个弯，所以才会有有意思的东西可写，另外还有我发现把尾递归如果跟JAVA中的GC比对一下，也颇有一些妙处（发现还没有人特地比较过）

（不过后来边写边整理思路，写出来又是另一个样子了）

一、首先我们讲讲递归

1. 递归的本质是，某个方法中调用了自身。本质还是调用一个方法，只是这个方法正好是自身而已
2. 递归因为是在自身中调用自身，所以会带来以下三个显著特点：
    1. 调用的是同一个方法
    2. 因为1，所以只需要写一个方法，就可以让你轻松调用无数次（不用一个个写，你定个n就能有n个方法），所以调用的方法数可能非常巨大
    3. 在自身中调用自身，是嵌套调用（栈帧无法回收，开销巨大）
3. 因为上面2和3两个特点，所以递归调用最大的诟病就是开销巨大，栈帧和堆一起爆掉，俗称内存泄露
    1. 一个误区，不是因为调用自身而开销巨大，而是嵌套加上轻易就能无数次调用，使得递归可以很容易开销巨大

既然会导致内存泄露如此，那肯定要想办法了，方法很简单，那就是尾递归优化

二、尾递归优化

1. 尾递归优化是利用上面的第一个特点"调用同一个方法"来进行优化的
2. 尾递归优化其实包括两个东西：1）尾递归的形式；2）编译器对尾递归的优化
    1. 尾递归的形式
        1. 尾递归其实只是一种对递归的特殊写法，这种写法原本并不会带来跟递归不一样的影响，它只是写法不一样而已，写成这样不会有任何优化效果，该爆的栈和帧都还会爆
        2. 具体不一样在哪里
            1. 前面说了，递归的本质是某个方法调用了自身，尾递归这种形式就要求：某个方法调用自身这件事，一定是该方法做的最后一件事（所以当有需要返回值的时候会是return f(n)，没有返回的话就直接是f(n)了）
        3. 要求很简单，就一条，但是有一些常见的误区
            1. 这个f(n)外不能加其他东西，因为这就不是最后一件事了，值返回来后还要再干点其他的活，变量空间还需要保留
                1. 比如如果有返回值的，你不能：乘个常数 return 3f(n)；乘个n return n*f(n)；甚至是 f(n)+f(n-1)
        4. 另外，使用return的尾递归还跟函数式编程有一点关系
    2. 编译器对尾递归的优化
        1. 上面说了，你光手动写成尾递归的形式，并没有什么卵用，要实现优化，还需要编译器中加入了对尾递归优化的机制
        2. 有了这个机制，编译的时候，就会自动利用上面的特点一来进行优化
        3. 具体是怎么优化的：
            1. 简单说就是重复利用同一个栈帧，不仅不用释放上一个，连下一个新的都不用开，效率非常高（有人做实验，这个比递推比迭代都要效率高）
    3. 为什么写成尾递归的形式，编译器就能优化了？或者说【编译器对尾递归的优化】的一些深层思想
        1. 说是深层思想，其实也是因为正好编译器其实在这里没做什么复杂的事，所以很简单
        2. 由于这两方面的原因，尾递归优化得以实现，而且效果很好
            1. 因为在递归调用自身的时候，这一层函数已经没有要做的事情了，虽然被递归调用的函数是在当前的函数里，但是他们之间的关系已经在传参的时候了断了，也就是这一层函数的所有变量什么的都不会再被用到了，所以当前函数虽然没有执行完，不能弹出栈，但它确实已经可以出栈了，这是一方面
            2. 另一方面，正因为调用的是自身，所以需要的存储空间是一毛一样的，那干脆重新刷新这些空间给下一层利用就好了，不用销毁再另开空间
        3. 有人对写成尾递归形式的说法是【为了告诉编译器这块要尾递归】，这种说法可能会导致误解，因为不是只告诉编译器就行，而是你需要做优化的前半部分，之后编译器做后半部分
    4. 所以总结：为了解决递归的开销大问题，使用尾递归优化，具体分两步：1）你把递归调用的形式写成尾递归的形式；2）编译器碰到尾递归，自动按照某种特定的方式进行优化编译

举例：

（没有使用尾递归的形式）

```
def recsum(x):
  if x == 1:
    return x
```

1 1

```
  else:
    return x + recsum(x - 1)
```

（使用尾递归的形式）

```
def tailrecsum(x, running_total=0):
  if x == 0:
    return running_total
  else:
    return tailrecsum(x - 1, running_total + x)
```

但不是所有语言的编译器都做了尾递归优化。比如C实现了，JAVA没有去实现
说到这里你很容易联想到JAVA中的自动垃圾回收机制，同是处理内存问题的机制，尾递归优化跟垃圾回收是不是有什么关系，这是不是就是JAVA不实现尾递归优化的原因？

三、所以下面要讲一下垃圾回收（**GC**）

1. 首先我们需要谈一下内存机制，这里我们需要了解内存机制的两个部分：栈和堆。下面虽然是在说JAVA，但是C也是差不多的
    1. 在Java中， JVM中的栈记录了线程的方法调用。每个线程拥有一个栈。在某个线程的运行过程中， 如果有新的方法调用，那么该线程对应的栈就会增加一个存储单元，即栈帧 (frame)。在frame 中，保存有该方法调用的参数、局部变量和返回地址
    2. Java的参数和局部变量只能是 基本类型 的变量(比如 int)，或者对象的引用(reference) 。因此，在栈中，只保存有基本类型的变量和对象引用。而引用所指向的对象保存在堆中。
2. 然后由栈和堆的空间管理方式的不同，引出垃圾回收的概念
    1. 当被调用方法运行结束时，该方法对应的帧将被删除，参数和局部变量所占据的空间也随之释放。线程回到原方法，继续执行。当所有的栈都清空时，程序也随之运行结束。
    2. 如上所述，栈 (stack)可以自己照顾自己。但堆必须要小心对待。堆是 JVM中一块可自由分配给对象的区域。当我们谈论垃圾回收 (garbage collection) 时，我们主要**回收堆(heap)的空间**。
    3. Java的普通对象存活在堆中。与栈不同，堆的空间不会随着方法调用结束而清空（即使它在栈上的引用已经被清空了）（也不知道为什么不直接同步清空）。因此，在某个方法中创建的对象，可以在方法调用结束之后，继续存在于堆中。这带来的一个问题是，如果我们不断的创建新的对象，内存空间将最终消耗殆尽。
    4. 如果没有垃圾回收机制的话，你就需要手动地显式分配及释放内存，如果你忘了去释放内存，那么这块内存就无法重用了（不管是什么局部变量还是其他的什么）。这块内存被占有了却没被使用，这种场景被称之为内存泄露
3. 所以不管是C还是JAVA，最原始的情况，都是需要手动释放堆中的对象，C到现在也是这样，所以你经常需要考虑对象的生存周期，但是JAVA则引入了一个自动垃圾回收的机制，它能智能地释放那些被判定已经没有用的对象

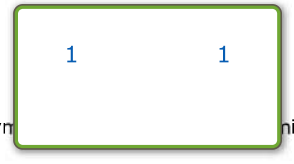
四、现在我们就可以比较一下尾递归优化和垃圾回收了

1. 他们最本质的区别是，尾递归优化解决的是内存溢出的问题，而垃圾回收解决的是内存泄露的问题
    1. 内存泄露：指程序中动态分配内存给一些临时对象，但是对象不会被GC所回收，它始终占用内存。即被分配的对象可达但已无用。
    2. 内存溢出：指程序运行过程中无法申请到足够的内存而导致的一种错误。内存溢出通常发生于OLD段或Perm段垃圾回收后，仍然无内存空间容纳新的Java对象的情况。
    3. 从定义上可以看出内存泄露是内存溢出的一种诱因，不是唯一因素。
2. 自动垃圾回收机制的特点是：
    1. 解决了所有情况下的内存泄露的问题，但还可以由于其他原因内存溢出
    2. 针对内存中的堆空间
    3. 正在运行的方法中的堆中的对象是不会被管理的，因为还有引用（栈帧没有被清空）
        1. 一般简单的自动垃圾回收机制是采用 引用计数 (reference counting)的机制。每个对象包含一个计数器。当有新的指向该对象的引用时，计数器加 1。当引用移除时，计数器减 1，当计数器为0时，认为该对象可以进行垃圾回收
3. 与之相对，尾递归优化的特点是：
    1. 优化了递归调用时的内存溢出问题
    2. 针对内存中的堆空间和栈空间
    3. 只在递归调用的时候使用，而且只能对于写成尾递归形式的递归进行优化
    4. 正在运行的方法的堆和栈空间正是优化的目标

最后可以解答一下前头提出的问题

1. 通过比较可以发现尾递归和GC是完全不一样的，JAVA不会是因为有GC所以不需要尾递归优化。那为什么呢，我看到有的说法是：JAVA编写组不实现尾递归优化是觉得麻烦又没有太大的必要，就懒得实现了（原话是：在日程表上，但是非常靠后），官方的建议是不使用递归，而是使用while循环，迭代，递推

参考资料：



http://it.deepinmind.com/jvm nization-and-java.html

http://book.51cto.com/art/201212/370096.htm

分类: Java基础

标签: 递归, 垃圾回收

好文要顶 关注我 收藏该文

赛艇队长
关注 - 7
粉丝 - 16
+加关注

« 上一篇：算法练习-排序算法
» 下一篇：回调的理解及其在应用中的多种写法——以安卓监听事件/线程启动为例

posted @ 2016-03-15 18:32 赛艇队长 阅读(4599) 评论(2) 编辑 收藏

发表评论

#1楼 2016-03-15 21:16 | 小米干饭

"所以递归调用最大的诟病就是开销巨大，栈帧和堆一起爆掉，俗称内存泄露"

这跟内存泄漏没什么关系。程序已经爆掉了，所占用的内存自然也就被操作系统回收了。

递归调用很容易造成堆栈溢出。在Java中，堆栈溢出是 Error。

```
1 public class StackOverflowError extends VirtualMachineError
```

在程序运行中如果出现了 Error，什么都不用说了，直接退出了事。程序退出后，它所占用的资源也随之被操作系统回收，不会有内存泄漏发生。

支持(2) 反对(0)

#2楼[楼主] 2016-03-16 13:51 | 赛艇队长

@ 小米干饭
是堆栈溢出而不是内存泄露，已经修改相关内容，感谢提出

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· ofo新一轮融资尘埃落定：倒向阿里，滴滴或出局
· 李国庆是创业者的典型反例
· 挖矿逐梦记：大浪无声 置身事外者也被卷入其中
· 过完年跳槽，要考虑哪些要素？
· 国产手机All in"刘海儿屏"：厂商无奈，用户悲哀
» 更多新闻...

最新知识库文章:
· 写给自学者的入门指南
· 和程序员谈恋爱
· 学会学习
· 优秀技术人的管理陷阱
· 作为一个程序员，数学对你到底有多重要
» 更多知识库文章...