

2.7旧版教程

```
>>> fact(1000)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
File "<stdin>", line 4, in fact
```

```
...
```

```
File "<stdin>", line 4, in fact
```

```
RuntimeError: maximum recursion depth exceeded in comparison
```

解决递归调用栈溢出的方法是通过尾递归优化，事实上尾递归和循环的效果是一样的，所以，把循环看成是一种特殊的尾递归函数也是可以的。

尾递归是指，在函数返回的时候，调用自身本身，并且，**return**语句不能包含表达式。这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。

上面的 `fact(n)` 函数由于 `return n * fact(n - 1)` 引入了乘法表达式，所以就不是尾递归了。要改成尾递归方式，需要多一点代码，主要是要把每一步的乘积传入到递归函数中：

```
def fact(n):
    return fact_iter(n, 1)

def fact_iter(num, product):
    if num == 1:
        return product
    return fact_iter(num - 1, num * product)
```

可以看到，`return fact_iter(num - 1, num * product)` 仅返回递归函数本身，`num - 1` 和 `num * product` 在函数调用前就会被计算，不影响函数调用。

`fact(5)` 对应的 `fact_iter(5, 1)` 的调用如下：

```
==> fact_iter(5, 1)
==> fact_iter(4, 5)
==> fact_iter(3, 20)
==> fact_iter(2, 60)
==> fact_iter(1, 120)
==> 120
```

尾递归调用时，如果做了优化，栈不会增长，因此，无论多少次调用也不会导致栈溢出。

遗憾的是，大多数编程语言没有针对尾递归做优化，Python解释器也没有做优化，所以，即使把上面的 `fact(n)` 函数改成尾递归方式，也会导致栈溢出。

小结

使用递归函数的优点是逻辑简单清晰，缺点是过深的调用会导致栈溢出。

针对尾递归优化的语言可以通过尾递归防止栈溢出。尾递归事实上和循环是等价的，没有循环语句的编程语言只能通过尾递归实现循环。

Python标准的解释器没有针对尾递归做优化，任何递归函数都存在栈溢出的问题。

练习

汉诺塔的移动可以用递归函数非常简单地实现。

请编写 `move(n, a, b, c)` 函数，它接收参数 `n`，表示3个柱子A、B、C中第1个柱子A的盘子数量，然后打印出把所有盘子从A借助B移动到C的方法，例如：

```
def move(n, a, b, c):  
  
    pass  
  
# 期待输出:  
# A --> C  
# A --> B  
# C --> B  
# A --> C  
# B --> A  
# B --> C  
# A --> C  
move(3, 'A', 'B', 'C')
```

▶ Run

参考源码

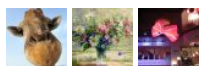
[recur.py](#)

感觉本站内容不错，读后有收获？

¥ 我要小额赞助，鼓励作者写出更好的教程

还可以分享给朋友

分享 [晨星_V5](#) , [哎哟不错耶](#) 等4人分享过



珠峰NODE.JS全栈开发

Angular React Gulp
Meteor Vue 微信开发

麦子学院 www.maiziedu.com

百万级python导师亲身指导

保你120天
变身python大牛

有时候，你需要的只是一句点拨

立即咨询



[函数的参数](#)

[高级特性](#)

评论

发表评论

Sign In to Make a Comment

