使用cProfile分析的结果可以输出到指定的文件中，但是文件内容是以二进制的方式保存的，用文本编辑器打开时乱码,所以，Python提供了一个pstats模块(Mac可用)，用来分析cProfile输出的文件内容,这也可以不用马丁说的图形化输出软件KCacheGrind就可以看到结果,而且这个软件只有linux和windows版。

这里我们可以使用pstats modules来分析生成文件

```
# 创建Stats对象
p = pstats.Stats("result.out")

# strip_dirs(): 去掉无关的路径信息
# sort_stats(): 排序，支持的方式和上述的一致
# print_stats(): 打印分析结果，可以指定打印前几行

# 和直接运行cProfile.run("test()")的结果是一样的
p.strip_dirs().sort_stats(-1).print_stats()

# 按照函数名排序，只打印前3行函数的信息, 参数还可为小数,表示前百分之几的函数信息
p.strip_dirs().sort_stats("name").print_stats(3)

# 按照运行时间和函数名进行排序
p.strip_dirs().sort_stats("cumulative", "name").print_stats(0.5)

# 如果想知道有哪些函数调用了sum_num
p.print_callers(0.5, "sum_num")

# 查看test()函数中调用了哪些函数
p.print_callees("test")
```

具体代码如下

```
import pstats
import sys

file_name = sys.argv[1]

p = pstats.Stats(file_name)
p.strip_dirs().sort_stats("cumulative", "name").print_stats(0.5)
```

生成样式

```
# us579 @ 192-168-1-103 in ~/Desktop/Beaver-Project/src/matcher on git:master x [13:46:24] C:146
$ python2 statics.py ctgov.cprof
Sat Apr  6 13:10:18 2019    ctgov.cprof

         5301401605 function calls (5274534309 primitive calls) in 11132.803 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 :1(<module>)
        1    0.000    0.000    0.000    0.000 :1(CallInfoAttributes)
        1    0.000    0.000    0.000    0.000 :1(ExceptionInfoAttributes)
        1    0.000    0.000    0.000    0.000 :1(FactoryAttributes)
```

ncalls：表示函数调用的次数；
tottime：表示指定函数的总的运行时间，除掉函数中调用子函数的运行时间；
percall：（第一个percall）等于 tottime/ncalls；
cumtime：表示该函数及其所有子函数的调用运行的时间，即函数开始调用到返回的时间；
percall：（第二个percall）即函数运行一次的平均时间，等于 cumtime/ncalls；
filename:lineno(function)：每个函数调用的具体信息；

注意：用Python2 运行的程序生成的cprofile文件只能用Python2中的pstats来分析,不然会报错