记录一次奇异bug的追踪过程

背景说明

智能出题项目 – 根据周边数据历史数据及近期数据的一个作业推荐服务
简单来说分 预测进度 + 周边学情 -》 作业组合 + 个人偏好 -》 排序 -》 出题
我们在进行服务的性能优化的时候，做了内存缓存优化后发现项目耗时极具增加

现场回放

优化推荐出题，尝试对班级章节 薄弱/遗漏/高频错题 知识点进行预生成 (qa)，（数据结构为dict 大致 如 weak = {(class_id,section_id):[knowid or questionid]}) 的100MB大dict，整个项目耗时约3g），预生成后，发现项目整体变慢，尤其pk模块耗时增加显著,约6-7倍,

```
@cost_count(True)
def queuePK1(r0,r1,rankLock):
    r0['package_type'] = 0
    r0['package_title'] = '口算练习'
    r1['package_type'] = 1
    r1['package_title'] = '基础训练'
    rz = []
    rz.append(r0)
    rz.append(r1)
    rz = pd.concat(rz)
    rz['max_s2'] = map(lambda x: max([int(i) for i in str(x).split(',')]),rz['coursesectio
n_2'])
    rz['combine_type'] = map(lambda x: int(math.ceil(float(x)/6)),rz['category_number'])
    s2list = rz['max_s2'].drop_duplicates().reset_index(drop = True)
    ctlist = [1,2,3,4]
    rz['queueId'] = 0
    rz['rank'] = 0
    idx = 1
    s2vk = {}
    for k,v in enumerate(s2list):
        s2vk[v]=k
    ctvk = {}
    for k,v in enumerate(ctlist):
        ctvk[v] = k + 1
    rz['queueId'] = map(lambda y,z:4*s2vk[y]+ctvk[z],rz['max_s2'],rz['combine_type'])
    rz['course'] = map(lambda x,y,z: y if z<=2 else x,rz['coursesection_2'],rz['coursesect
ion_3'],rz['combine_type'])
    rz.index = range(len(rz))
    all_queue = {}
    all_queue_publish = {}
    temp = rz.as_matrix()
    for index,row in enumerate(temp):
        queue_id = row[9]
        if all_queue.get(queue_id,None):
            all_queue[queue_id].append([index,row[2]])
        else:
            all_queue[queue_id] = [[index,row[2]]]
    sort_queue = []
    for index,value in all_queue.iteritems():
        value.sort(cmp=lambda x,y:cmp(x[1],y[1]),reverse=True)
```

```
            all_queue[index] = value
            sort_queue.append(value[0]+[index])
        sort_queue.sort(cmp=lambda x,y:cmp(x[1],y[1]),reverse=True)
        rankx = 1
        for i in range(1,101):
            if len(sort_queue)<=0:
                break
            if rankx > 100 - len(rankLock):
                break
            if rankx in rankLock:
                rankx += 1
                continue
            else:
                index_i = 0
                count = 0
                while len(sort_queue) >index_i+1:
                    if sort_queue[index_i][1]==sort_queue[index_i+1][1]:
                        index_i += 1
                        continue
                    break
                for i in xrange(index_i+1):
                    index = int(sort_queue[i][0])
                    rz.at[index,'rank'] = rankx
                    rz.at[index,'star'] = max(int(10-math.floor(float(rankx)/10)),1)
                    count += 1
                    queue_id = int(sort_queue[i][2])
                    all_queue_publish[queue_id] = all_queue_publish.get(queue_id,0)+1
                    times = all_queue_publish[queue_id]
                    while len(all_queue[queue_id]) > times:
                        if all_queue[queue_id][times][1] == all_queue[queue_id][times-1][1]:
                            all_queue_publish[queue_id] = all_queue_publish.get(queue_id,0)+1
                            index = all_queue[queue_id][times][0]
                            rz.at[index,'rank'] = rankx
                            #rz.at[index,'star'] = max(int(10-math.floor(float(rankx)/10)),1)
                            count += 1
                            print rankx,index,times
                            times += 1
                            continue
                        break
                    times = all_queue_publish[queue_id]
                    if len(all_queue[queue_id]) <= times:
                        continue
                    if isinstance(all_queue[queue_id][times][1],float) == False:
                        all_queue[queue_id][times][1] = 0.0
                    sort_queue.append([all_queue[queue_id][times][0],all_queue[queue_id][time
s][1]*(0.8**times),queue_id])
                for i in xrange(index_i+1):
                    sort_queue.pop(0)
                sort_queue.sort(cmp=lambda x,y:cmp(x[1],y[1]),reverse=True)
                rankx += count
        R = rz[rz['rank']>0][['rank','coursesection_2','coursesection_3','category_number','pa
ckage_type','package_title','scene']]
        R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
        return R
```

qa 有什么思路?

追踪凶手

那时的追凶心理– –内存变大引发 or 其他

round one

为验证是否是内存变大引起，是否我随机一个大量内存的数据集就会催发bug?
所以我缓存了一个 500mb的pandas 无效 数据集，再次运行发现 函数运行完好，完全没有变慢，排除内存变大的嫌疑、

round two

有缓存情况下，对pk变慢模块进行业务层级的代码耗时分析，耗时如下

```
1         4074   4074.0      0.6      R = rz[rz['rank']>0][['rank','coursesection_2','coursesection_3','category_number','
1       582604 582604.0     90.3      R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
1            4      4.0      0.0      return R
```

无缓存情况下

```
179         1        38355  38355.0     38.7      R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
180         1            6      6.0      0.0      return R
```

这个时候对业务代码改写，将耗时增加的行数改写，bug修复，囧
改写逻辑见上图 注释 部分

round three

尝试自己模拟环境，依次自己造了如下数据
A:

```
[5]: dic = dict.fromkeys(range(6*pow(10,6)), 'x')
```

```
[6]: dic = dict.fromkeys(range(6*pow(10,6)), 'x')
```

```
[7]: dic1 = dict.fromkeys(range(8*pow(10,5)), 'x')
```

B:

```
3]: import random
    a=[[i for i in range(random.randint(1,500))] for i in range(5*pow(10,5))]
    b=[i for i in range(5*pow(10,5))]
    c = dict(zip(b,a))
```

A变慢不明显，约20-30%，B变慢尤其明显，几十倍 且随数据集变大 运行更慢

round four

尝试抽离变慢模块

```
In [25]: R = queuePK1()

         58 175 1
         ('rz len:', 342)
```

```
In [*]:
def testMap(R):
    R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
    return R

%load_ext line_profiler
%lprun -s -f testMap testMap(R)
```

其中r的结构为（100行数据）

	rank	coursesection_2	coursesection_3	category_number	package_type	package_title	scene	star
0	1	80005	80006	4.0	0	AAAAA	日常作业	10
1	11	80166	80200, 80190, 80203, 80197, 80206	11.0	0	AAAAA	日常作业	9

第一次运行

```
Timer unit: 1e-06 s

Total time: 4.495 s
File: <ipython-input-26-9ef7da7f60f7>
Function: testMap at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def testMap(R):
     2         1      4495002 4495002.0    100.0      R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
     3         1            3      3.0      0.0      return R
```

第二次运行

```
Timer unit: 1e-06 s

Total time: 0.006583 s
File: <ipython-input-26-9ef7da7f60f7>
Function: testMap at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def testMap(R):
     2         1         6581   6581.0    100.0      R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
     3         1            2      2.0      0.0      return R
```

round five

因为速度快慢与内存大小无关 而与dict 健值对多少呈现一定的相关性 怀疑和gc 有关

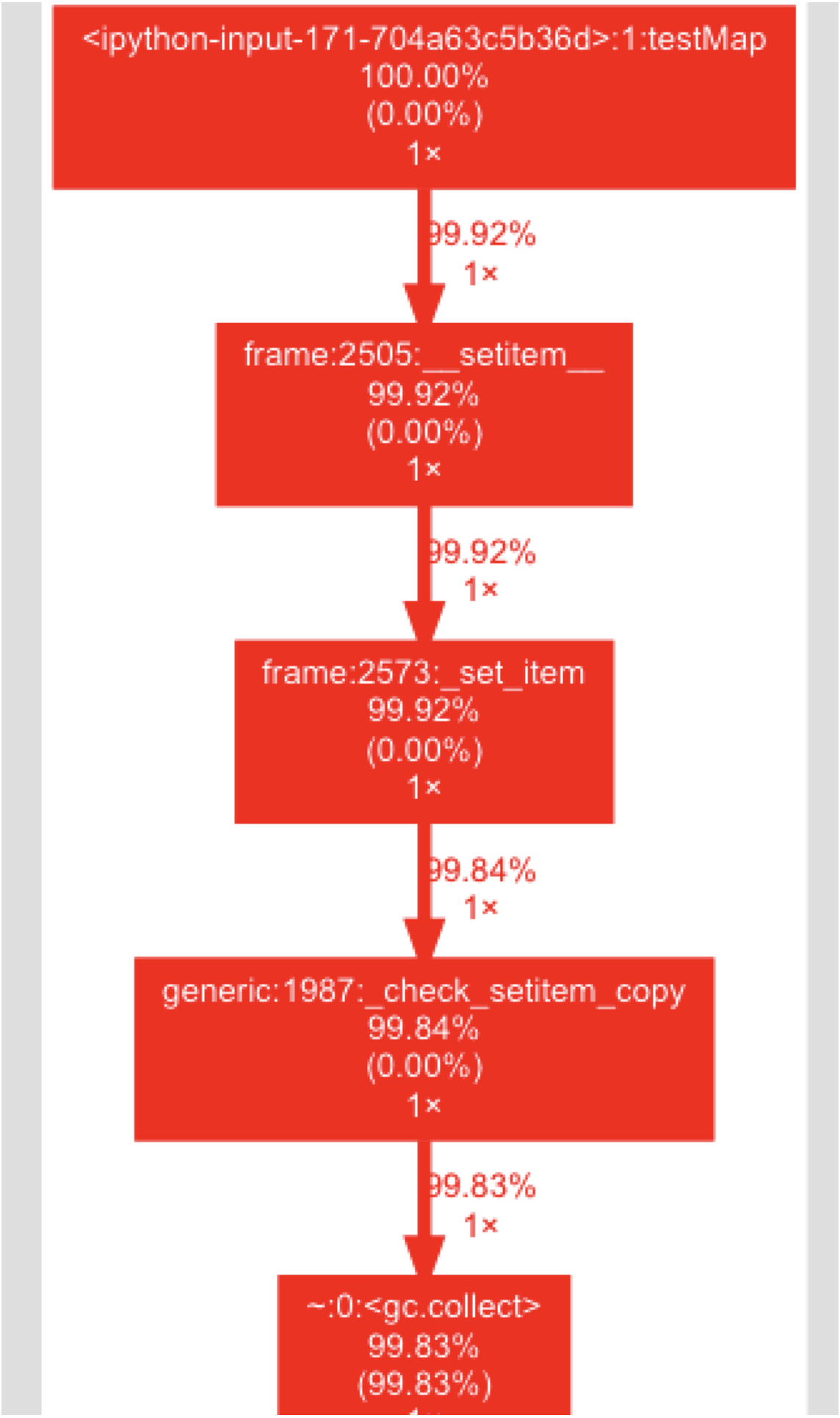
```
In [ ]:
```

```
In [3]: import gc
        gc.set_debug(gc.DEBUG_LEAK)
```

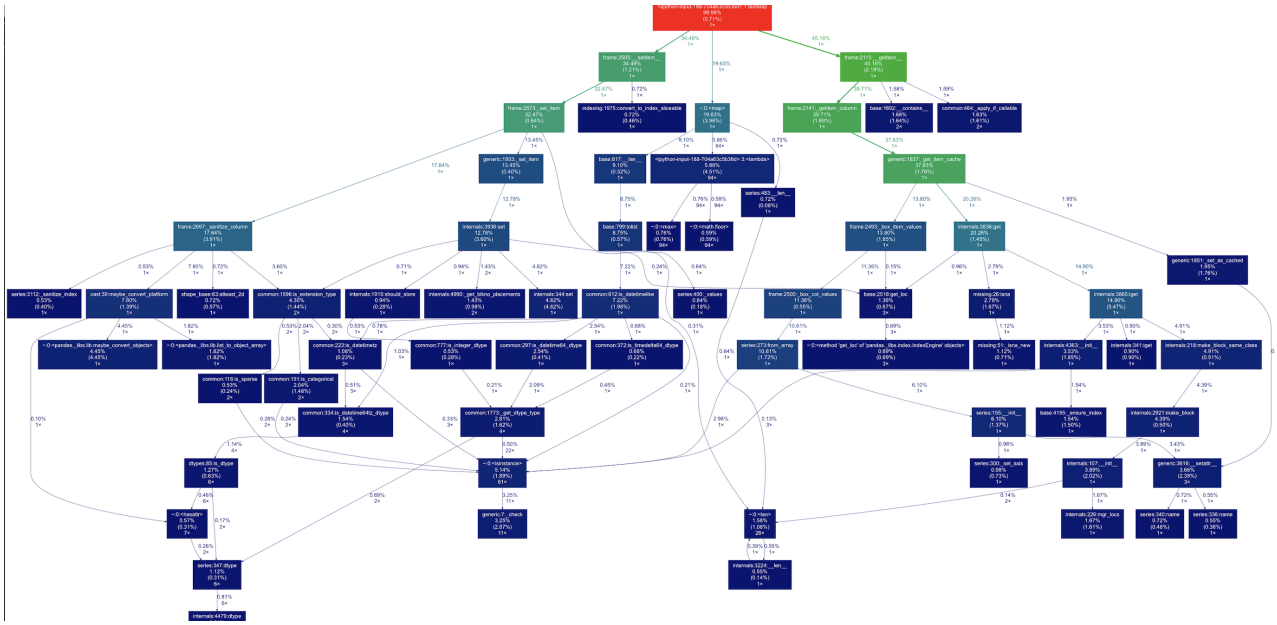
想通过gc日志得出相关线索，无果

round six

外部攻击感觉已经无法得出什么有效结论了，只能尝试 内部爆破
采取 cProfiler 进行性能分析 testMap 第一次运行 的cpu耗时如下


<ipython-input-171-704a63c5b36d>:1:testMap
100.00%
(0.00%)
1×
99.92%
1×
frame:2505:__setitem__
99.92%
(0.00%)
1×
99.92%
1×
frame:2573:_set_item
99.92%
(0.00%)
1×
99.84%
1×
generic:1987:_check_setitem_copy
99.84%
(0.00%)
1×
99.83%
1×
~:0:<gc.collect>
99.83%
(99.83%)

第二次



发现第一次几乎所有耗时全部因为 调用了 gc.collect (python 原生的对象收集函数)
而第二次则没有调用

分析原因

喜人的发现1

根据关键字 翻阅相关资料发现 pandas 为了保护对象的安全性 会主动去做gc收集，可以外围配置关闭收集选项

```
[2]: import pandas as pd
     pd.set_option('mode.chained_assignment', None)
```

https://github.com/pandas-dev/pandas/issues/11045
这个配置项可以直接提升项目缩短30%的耗时 整个接口响应由500ms -》300ms
不过作者不推荐该行为
— qa，真的搞定了吗？为啥第二次就好用了，为啥我模拟失败？

better choice

继续寻找相关资料，终于发现病根

```
R = rz[rz['rank']>0][['rank','coursesection_2','coursesection_3','category_number','packag
e_type','package_title','scene','star']]
R['star'] = map(lambda x: max(int(10-math.floor(float(x)/10)),1),R['rank'])
```

原因是这个数据集生成用了 链式索引，这是一种不安全的操作，为了弥补，后续对该数据集操作的时候 会进行 gc 相关操作 相关源码如下

```
def _check_setitem_copy(self, stacklevel=4, t='setting', force=False):
    if force or self._is_copy:

        value = config.get_option('mode.chained_assignment')
        if value is None:
            return

        # see if the copy is not actually referred; if so, then dissolve
        # the copy weakref
        try:
            gc.collect(2)
            if not gc.get_referents(self._is_copy()):
                self._is_copy = None
                return
        except Exception:
            pass
```

简单说说gc

python赋值相关 a = 1 啥意思?
引用回收与垃圾回收

```
def test2():
    a = [1]
    b = [2]
    a.append(b)
    b.append(a)
test2()
```

链式索引

```
As mentioned previously, chained indexing occurs whenever you use the indexers [], .loc, o
r .iloc twice in a row.
```

bad for pandas

- Two separate operations

```
    The first, and less important reason, is that two separate pandas operations will be c
alled instead of just one.
```

- SettingWithCopy warning on assignment

```
The major problem with chained indexing is when assigning new values to the subset, in whi
ch Pandas will usually emit the SettingWithCopy warning.
```

```
[1]: a = [1, 5, 10, 3, 99, 5, 8, 20, 40]

[2]: a[2:6][0]

[2]: 10

[9]: a[2:6][0] = 1000

[1]: a[2:6][0]

]: df['score'][df['age'] > 5] = 0 # view
   df[df['age'] > 10]['score'] = 99  #copy
```

解决方法

对链式索引的使用进行修改

```
#good
R = rz.loc[rz['rank']>0,
['rank','coursesection_2','coursesection_3','category_number','package_type','package_title','scene']]
#bad
R = rz[rz['rank']>0]
[['rank','coursesection_2','coursesection_3','category_number','package_type','package_title','scene']]
```

https://medium.com/dunder-data/selecting-subsets-of-data-in-pandas-part-4-c4216f84d388

200 150ms

工具推荐

各种外围试探，感觉无果，只能采取内部爆破。

jupyter

实验大杀器，简单好用，而且有一些魔法方法很好用 上面说的业务层代码耗时分析通过这个一行代码就可以

cProfiler

可做源码级别的性能分析，可得到原生函数执行的cpu耗时
结合其他相关工具可以生成相关的函数调用图
特别易用
用法1

```
30]: import cProfile

31]: cProfile.run('testMap(R)')
```

```
         6 function calls in 0.016 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.016    0.016 <ipython-input-33-5778420cef37>:10(profiled_func)
        1    0.000    0.000    0.016    0.016 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 UserDict.py:103(__contains__)
        1    0.000    0.000    0.000    0.000 UserDict.py:91(get)
        1    0.000    0.000    0.000    0.000 os.py:512(getenv)
        1    0.016    0.016    0.016    0.016 {method 'enable' of '_lsprof.Profiler' objects}
```

用法2

利用gprof2dot一行代码生成系统调用流程图片

```
gprof2dot -f pstats test.prof | dot -Tpng -o test.png
```

具体见文章 https://zhuanlan.zhihu.com/p/24495603

总结

工欲善其事必先利其器，好的工具还是会让追凶过程简单很多
多思考，所谓万变不离其宗，个人理解bug都是可以脱离工程项目去模拟复现的，复现简单化场景中更容易寻找破案的关键
工程中用相关包还是尽量多看看官方说明，了解各种用法的利弊，可以避免很多问题