

Создаем по 200 000 экземпляров каждого класса: с обычными атрибутами, со слотами и с атрибутами weakref и замеряем время создания пачки экземпляров.

Далее производим операции чтения/изменения и измеряем время этих операций:

```
egor@ubuntu01: ~/dev/10
egor@ubuntu01:~/dev/10$ python3 profiler1.py
Число повторов: 200000
Класс                Время создания
-----
С обычными атрибутами 0.222131
Со слотами            0.186183
С атрибутами weakref  0.312208

Число повторов: 200000
Класс                Чтение            Изменение
-----
С обычными атрибутами 0.019667          0.021724
Со слотами            0.015840          0.019002
С атрибутами weakref  0.022655          0.026087

egor@ubuntu01:~/dev/10$
```

Как мы видим, быстрее всего операции создания, чтения и изменения выполняются в классе со слотами, а медленнее всего – в классе с атрибутами weakref. Это может быть связано с тем, что при использовании слотов выделяется пространство для фиксированного числа атрибутов, что ускоряет доступ к атрибутам. Это уменьшает гибкость, но увеличивает скорость работы. В случае же с атрибутами weakref всё работает медленнее из-за дополнительных операций, требуемых для работы weakref, таких как создание внутренних объектов, в которых хранится слабая ссылка.

Далее выполняем профилирование с помощью cProfile:

```
egor@ubuntu01: ~/dev/10
egor@ubuntu01:~/dev/10$ python3 profiler1.py PROFILE
Число повторов: 200000
Класс          Время создания
-----
С обычными атрибутами  0.287421
Со слотами            0.236897
С атрибутами weakref   0.389442

600020 function calls in 0.914 seconds

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1   0.000   0.000   0.914   0.914 /home/egor/dev/10/profiler1.py:48(create_timing)
      1   0.000   0.000   0.389   0.389 /home/egor/dev/10/profiler1.py:44(create_weakref)
      1   0.278   0.278   0.389   0.389 /home/egor/dev/10/profiler1.py:45(<listcomp>)
      1   0.000   0.000   0.287   0.287 /home/egor/dev/10/profiler1.py:36(create_plain)
      1   0.163   0.163   0.287   0.287 /home/egor/dev/10/profiler1.py:37(<listcomp>)
      1   0.000   0.000   0.237   0.237 /home/egor/dev/10/profiler1.py:40(create_slotted)
      1   0.199   0.199   0.237   0.237 /home/egor/dev/10/profiler1.py:41(<listcomp>)
200000   0.124   0.000   0.124   0.000 /home/egor/dev/10/profiler1.py:18(__init__)
200000   0.111   0.000   0.111   0.000 /home/egor/dev/10/profiler1.py:32(__init__)
200000   0.038   0.000   0.038   0.000 /home/egor/dev/10/profiler1.py:26(__init__)
      6   0.000   0.000   0.000   0.000 {built-in method builtins.print}
      6   0.000   0.000   0.000   0.000 {built-in method time.time}
      1   0.000   0.000   0.000   0.000 {method 'disable' of '_lsprof.Profiler' objects}

Число повторов: 200000
Класс          Чтение          Изменение
-----
С обычными атрибутами  0.051171  0.050298
Со слотами            0.045053  0.050020
С атрибутами weakref   0.054920  0.058137

1200020 function calls in 0.310 seconds

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1   0.212   0.212   0.310   0.310 /home/egor/dev/10/profiler1.py:70(read_edit_timing)
600000   0.049   0.000   0.049   0.000 {built-in method builtins setattr}
600000   0.048   0.000   0.048   0.000 {built-in method builtins getattr}
      6   0.000   0.000   0.000   0.000 {built-in method builtins.print}
     12   0.000   0.000   0.000   0.000 {built-in method time.time}
      1   0.000   0.000   0.000   0.000 {method 'disable' of '_lsprof.Profiler' objects}

egor@ubuntu01:~/dev/10$
```

При 200 000 операций мы видим, что быстрее всего работает код на 26 строке – создание класса со слотами. Следующая по скорости 18 строка – создание класса с обычными атрибутами, а самая медленная – 32 строка, с созданием класса с weakref атрибутами, всё как и в предыдущем исследовании.

Далее профилируем память:

```
egor@ubuntu01: ~/dev/10
egor@ubuntu01:~/dev/10$ python3 -m memory_profiler profiler.py
Число повторов: 200000
Класс          Время создания
-----
С обычными атрибутами  21.529321
Со слотами            21.342555
С атрибутами weakref   21.770143

Число повторов: 200000
Класс          Чтение          Изменение
-----
С обычными атрибутами  41.081763      41.335148
Со слотами            41.920890      41.534691
С атрибутами weakref   42.130394      43.912094

Filename: profiler.py
Line #   Mem usage    Increment  Occurrences  Line Contents
=====
   36    20.613 MiB    20.613 MiB          1  @profile
   37                                     def create_plain(total_run):
   38    65.059 MiB    44.445 MiB      200003      return [PlainClass(f"Parent{__}", subclass) for _ in range(total_run)]

Filename: profiler.py
Line #   Mem usage    Increment  Occurrences  Line Contents
=====
   41    65.059 MiB    65.059 MiB          1  @profile
   42                                     def create_slotted(total_run):
   43    88.070 MiB    23.012 MiB      200003      return [SlottedClass(f"Parent{__}", subclass) for _ in range(total_run)]

Filename: profiler.py
Line #   Mem usage    Increment  Occurrences  Line Contents
=====
   46    88.070 MiB    88.070 MiB          1  @profile
   47                                     def create_weakref(total_run):
   48   132.453 MiB    44.383 MiB      200003      return [WeakRefClass(f"Parent{__}", subclass) for _ in range(total_run)]

Filename: profiler.py
Line #   Mem usage    Increment  Occurrences  Line Contents
=====
   51    20.613 MiB    20.613 MiB          1  @profile
   52                                     def create_timing(total_run, subcl):
   53    20.613 MiB     0.000 MiB          1      start_time = time.time()
   54    65.059 MiB    65.059 MiB          1      plain_child = create_plain(total_run)
   55    65.059 MiB     0.000 MiB          1      plain_time = time.time() - start_time
   56
   57    65.059 MiB     0.000 MiB          1      start_time = time.time()
   58    88.070 MiB    88.070 MiB          1      slotted_child = create_slotted(total_run)
   59    88.070 MiB     0.000 MiB          1      slotted_time = time.time() - start_time
   60
```

```
egor@ubuntu01: ~/dev/10
61 88.070 MiB 0.000 MiB 1 start_time = time.time()
62 132.453 MiB 132.453 MiB 1 weakref_child = create_weakref(total_run)
63 132.453 MiB 0.000 MiB 1 weakref_time = time.time() - start_time
64
65 132.453 MiB 0.000 MiB 1 print(f"Число повторов: {N}")
66 132.453 MiB 0.000 MiB 1 print("Класс          Время создания")
67 132.453 MiB 0.000 MiB 1 print("-----")
68 132.453 MiB 0.000 MiB 1 print(f"С обычными атрибутами {plain_time:.6f}")
69 132.453 MiB 0.000 MiB 1 print(f"Со слотами {slotted_time:.6f}")
70 132.453 MiB 0.000 MiB 1 print(f"С атрибутами weakref {weakref_time:.6f}\n\n")
71 132.453 MiB 0.000 MiB 1 return plain_child, slotted_child, weakref_child

Filename: profiler.py
Line #    Mem usage    Increment    Occurrences    Line Contents
=====
74 132.453 MiB 132.453 MiB 1 @profile
75 def read_edit_timing(num_to_run):
76     # замер времени доступа и чтения атрибутов для каждого экземпляра
77 132.453 MiB 0.000 MiB 1 start_time = time.time()
78 132.453 MiB 0.000 MiB 200001 for parent in plain_child:
79 132.453 MiB 0.000 MiB 200000     getattr(parent.loopback, "name")
80 132.453 MiB 0.000 MiB 1 plain_read = time.time() - start_time
81
82 132.453 MiB 0.000 MiB 1 start_time = time.time()
83 132.453 MiB 0.000 MiB 200001 for parent in plain_child:
84 132.453 MiB 0.000 MiB 200000     setattr(parent.loopback, "name", "New name pln")
85 132.453 MiB 0.000 MiB 1 plain_edit = time.time() - start_time
86
87 132.453 MiB 0.000 MiB 1 start_time = time.time()
88 132.453 MiB 0.000 MiB 200001 for parent in slotted_child:
89 132.453 MiB 0.000 MiB 200000     getattr(parent.loopback, "name")
90 132.453 MiB 0.000 MiB 1 slotted_read = time.time() - start_time
91
92 132.453 MiB 0.000 MiB 1 start_time = time.time()
93 132.453 MiB 0.000 MiB 200001 for parent in slotted_child:
94 132.453 MiB 0.000 MiB 200000     setattr(parent.loopback, "name", "New name slt")
95 132.453 MiB 0.000 MiB 1 slotted_edit = time.time() - start_time
96
97 132.453 MiB 0.000 MiB 1 start_time = time.time()
98 132.453 MiB 0.000 MiB 200001 for parent in weakref_child:
99 132.453 MiB 0.000 MiB 200000     getattr(parent.loopback(), "name")
100 132.453 MiB 0.000 MiB 1 weakref_read = time.time() - start_time
101
102 132.453 MiB 0.000 MiB 1 start_time = time.time()
103 132.453 MiB 0.000 MiB 200001 for parent in weakref_child:
104 132.453 MiB 0.000 MiB 200000     setattr(parent.loopback(), "name", "New name wkr")
105 132.453 MiB 0.000 MiB 1 weakref_edit = time.time() - start_time
106
107 132.453 MiB 0.000 MiB 1 print(f"Число повторов: {num_to_run}")
108 132.453 MiB 0.000 MiB 1 print("Класс          Чтение          Изменение")
109 132.453 MiB 0.000 MiB 1 print("-----")
110 132.453 MiB 0.000 MiB 1 print(f"С обычными атрибутами {plain_read:.6f} {plain_edit:.6f}")
111 132.453 MiB 0.000 MiB 1 print(f"Со слотами {slotted_read:.6f} {slotted_edit:.6f}")
112 132.453 MiB 0.000 MiB 1 print(f"С атрибутами weakref {weakref_read:.6f} {weakref_edit:.6f}\n\n")
```

Больше всего памяти используется при создании класса с обычными атрибутами, чуть меньше используется при создании класса с weakref атрибутами, а вот у класса со слотами потребление памяти меньше почти в 2 раза.

Вывод: с точки зрения скорости создания, чтения/изменения, профилирования вызовов и памяти лучше всего показал себя класс со слотами.