memory-tools Documentation

Release 0.0.2

Max Zheng

Jun 07, 2017

Contents

CHAPTER 1

memory-tools

A set of simple yet effective tools to troubleshoot memory leaks.

When debugging memory issues in Python 2.6, the author had tried memory_profiler and heapy, unfortunately neither worked. And so memory-tools was born with the goal of being simple - it should always work, yet effective - it is good at helping you find memory leaks.

CHAPTER 2

Quick Start Tutorial

Installation

```
$ pip install memory-tools
```

Show Memory Usage / Delta

Use the *show-mem* command to show system or process memory usage. When paired with watch, this becomes even more useful.

Show system memory:

```
$ show-mem

Commit Mem (MB):       27,852.80 total   17,278.42 used
Physical Mem (MB):     16,384.00 total   13,128.05 used
```

Re-run to show delta from last run:

```
$ show-mem

Commit Mem (MB):       27,852.80 total   17,888.59 used (delta: 310.15)
Physical Mem (MB):     16,384.00 total   13,126.40 used (delta: -1.65)
```

Show memory for process:

```
$ show-mem -p python

1 process matching "python":
  PID 26143 (MB):           4.79 rss          1.23 private

$ show-mem -p 26143
```

```
  PID 26143 (MB):           4.80 rss          1.24 private
```

Watch system/process memory using watch:

```
$ watch show-mem -s -p python

Commit Mem (MB):       27,852.80 total   17,888.59 used (delta: 310.15)
Physical Mem (MB):     16,384.00 total   13,126.40 used (delta: -1.65)

2 processes matching "python" (showing 1st & last):
  PID 26143 (MB):          40.79 rss         30.23 private
  PID 24118 (MB):           4.79 rss          1.23 private
```

Summarize / Save GC Objects

After running your program, view summary of gc.get_objects():

```
from memorytools import summarize_objects

summarize_objects()
```

And here is a sample output:

```
Objects count 3,790
Objects size 833,344

      Size Count Type
   476,864   296 <type 'dict'>
    76,320   954 <type 'wrapper_descriptor'>
    64,920   541 <type 'function'>
    ...

Count       Size Type
  954     76,320 <type 'wrapper_descriptor'>
  541     64,920 <type 'function'>
  515     37,080 <type 'builtin_function_or_method'>
  ...
```

Save all objects (along with the above summary) to a file:

```
from memorytools import save_objects

save_objects()

# Output: Wrote 3887 objects to /var/tmp/objects-45271 (882040 bytes)
```

Looping / Stress Testing

Use the *loop* command to run a command, module:method, or code in a forever loop to perform stress testing, which is useful in finding memory leaks. The command/code should, of course, act against a long running server for this to be useful.

Run a script in a loop:

```
$ loop show-mem 1

Physical Mem (MB):     16,384.00 total    9,415.50 used (delta: -190.67)
Physical Mem (MB):     16,384.00 total    9,415.27 used (delta: -0.23)
Physical Mem (MB):     16,384.00 total    9,415.85 used (delta: 0.58)
^C                   [ User CTRL-C here as it loops forever by default ]
Looped 3 times in 2.80 secs
```

Run a module:method in a loop - count of 10:

```
$ loop memorytools:summarize_objects 10 -c 10

# Results from summarize_objects() every 10 seconds

Looped 10 times in 100 secs
```

Run adhoc code in a loop - count of 2 and concurrency of 3:

```
$ loop 'print("Hello World!")' 0.1 -c 2 -cc 3
Hello World!
... 5 more times

Looped 2 times in 0.21 secs with concurrency of 3 (6 runs, 0.10 secs per loop, 0.03␣
↪secs per run)
```

Log Stack / Start Debugger on Signal

If you need to get a stacktrace of a running process, or start the debugger in specific situations to look at memory footprint, then a signal handler could help:

```
from memorytools import add_debug_handler

add_debug_handler(start_debugger_password='test')  # remove start_debugger_password␣
↪to skip rpdb2 debugger
```

The above will add a handler to SIGUSR2 that will log a stacktrace on trigger and also start the rpdb2 debugger.

CHAPTER 3

Links & Contact Info

Documentation: http://memory-tools.readthedocs.org

PyPI Package: https://pypi.python.org/pypi/memory-tools
GitHub Source: https://github.com/maxzheng/memory-tools
Report Issues/Bugs: https://github.com/maxzheng/memory-tools/issues

Connect: https://www.linkedin.com/in/maxzheng
Contact: maxzheng.os @t gmail.com

CHAPTER 4

API Documentation

Objects

CHAPTER 5

Change Log

Version 1.0.5

- Attempt to fix missing package

Version 1.0.4

- Reindent to 4 spaces and move source files to top level dir
- Use default locale instead of en_US
- Correct grammer

Version 1.0.3

- Make it compatible with Python 3.5

Version 1.0.2

- Add limit param to summarize_objects and do gc.collect() before calling gc.get_objects()

Version 1.0.1

- Update doc
- Fix intro sentence

Version 1.0.0

- Add add_debug_handler

Version 0.0.8

- Add loop command

Version 0.0.7

- Pass in correct used value

Version 0.0.6

- Subtract cached / buffers when showing used physical mem

Version 0.0.5

- Update docs

Version 0.0.4

- Update doc

Version 0.0.3

- Add docs
- Add show-mem to show system/process memory stats

Version 0.0.2

- Initial setup
- Initial commit

CHAPTER 6

Indices and tables

- genindex
- modindex
- search