27.7. `tracemalloc` — Trace memory allocations

New in version 3.4.

Source code: Lib/tracemalloc.py

The tracemalloc module is a debug tool to trace memory blocks allocated by Python. It provides the following information:

- Traceback where an object was allocated
- Statistics on allocated memory blocks per filename and per line number: total size, number and average size of allocated memory blocks
- Compute the differences between two snapshots to detect memory leaks

To trace most memory blocks allocated by Python, the module should be started as early as possible by setting the `PYTHONTRACEMALLOC` environment variable to `1`, or by using `-X tracemalloc` command line option. The `tracemalloc.start()` function can be called at runtime to start tracing Python memory allocations.

By default, a trace of an allocated memory block only stores the most recent frame (1 frame). To store 25 frames at startup: set the `PYTHONTRACEMALLOC` environment variable to `25`, or use the `-X tracemalloc=25` command line option.

27.7.1. Examples

27.7.1.1. Display the top 10

Display the 10 files allocating the most memory:

```
import tracemalloc

tracemalloc.start()

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')

print("[ Top 10 ]")
for stat in top_stats[:10]:
    print(stat)
```

Example of output of the Python test suite:

```
[ Top 10 ]
<frozen importlib._bootstrap>:716: size=4855 KiB, count=39328, average
<frozen importlib._bootstrap>:284: size=521 KiB, count=3199, average=1
/usr/lib/python3.4/collections/__init__.py:368: size=244 KiB, count=23
/usr/lib/python3.4/unittest/case.py:381: size=185 KiB, count=779, aver
/usr/lib/python3.4/unittest/case.py:402: size=154 KiB, count=378, aver
/usr/lib/python3.4/abc.py:133: size=88.7 KiB, count=347, average=262 B
<frozen importlib._bootstrap>:1446: size=70.4 KiB, count=911, average=
<frozen importlib._bootstrap>:1454: size=52.0 KiB, count=25, average=2
<string>:5: size=49.7 KiB, count=148, average=344 B
/usr/lib/python3.4/sysconfig.py:411: size=48.0 KiB, count=1, average=4
```

We can see that Python loaded `4855 KiB` data (bytecode and constants) from modules and that the `collections` module allocated `244 KiB` to build `namedtuple` types.

See `Snapshot.statistics()` for more options.

27.7.1.2. Compute differences

Take two snapshots and display the differences:

```
import tracemalloc
tracemalloc.start()
# ... start your application ...

snapshot1 = tracemalloc.take_snapshot()
# ... call the function leaking memory ...
snapshot2 = tracemalloc.take_snapshot()

top_stats = snapshot2.compare_to(snapshot1, 'lineno')

print("[ Top 10 differences ]")
for stat in top_stats[:10]:
    print(stat)
```

Example of output before/after running some tests of the Python test suite:

```
[ Top 10 differences ]
<frozen importlib._bootstrap>:716: size=8173 KiB (+4428 KiB), count=71
/usr/lib/python3.4/linecache.py:127: size=940 KiB (+940 KiB), count=81
/usr/lib/python3.4/unittest/case.py:571: size=298 KiB (+298 KiB), cour
<frozen importlib._bootstrap>:284: size=1005 KiB (+166 KiB), count=742
/usr/lib/python3.4/mimetypes.py:217: size=112 KiB (+112 KiB), count=13
/usr/lib/python3.4/http/server.py:848: size=96.0 KiB (+96.0 KiB), cour
/usr/lib/python3.4/inspect.py:1465: size=83.5 KiB (+83.5 KiB), count=1
/usr/lib/python3.4/unittest/mock.py:491: size=77.7 KiB (+77.7 KiB), co
```

```
/usr/lib/python3.4/urllib/parse.py:476: size=71.8 KiB (+71.8 KiB), cou
/usr/lib/python3.4/contextlib.py:38: size=67.2 KiB (+67.2 KiB), count=
```

We can see that Python has loaded `8173 KiB` of module data (bytecode and constants), and that this is `4428 KiB` more than had been loaded before the tests, when the previous snapshot was taken. Similarly, the `linecache` module has cached `940 KiB` of Python source code to format tracebacks, all of it since the previous snapshot.

If the system has little free memory, snapshots can be written on disk using the `Snapshot.dump()` method to analyze the snapshot offline. Then use the `Snapshot.load()` method reload the snapshot.

27.7.1.3. Get the traceback of a memory block

Code to display the traceback of the biggest memory block:

```
import tracemalloc

# Store 25 frames
tracemalloc.start(25)

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('traceback')

# pick the biggest memory block
stat = top_stats[0]
print("%s memory blocks: %.1f KiB" % (stat.count, stat.size / 1024))
for line in stat.traceback.format():
    print(line)
```

Example of output of the Python test suite (traceback limited to 25 frames):

```
903 memory blocks: 870.1 KiB
  File "<frozen importlib._bootstrap>", line 716
  File "<frozen importlib._bootstrap>", line 1036
  File "<frozen importlib._bootstrap>", line 934
  File "<frozen importlib._bootstrap>", line 1068
  File "<frozen importlib._bootstrap>", line 619
  File "<frozen importlib._bootstrap>", line 1581
  File "<frozen importlib._bootstrap>", line 1614
  File "/usr/lib/python3.4/doctest.py", line 101
    import pdb
  File "<frozen importlib._bootstrap>", line 284
  File "<frozen importlib._bootstrap>", line 938
  File "<frozen importlib._bootstrap>", line 1068
```

```
  File "<frozen importlib._bootstrap>", line 619
  File "<frozen importlib._bootstrap>", line 1581
  File "<frozen importlib._bootstrap>", line 1614
  File "/usr/lib/python3.4/test/support/__init__.py", line 1728
    import doctest
  File "/usr/lib/python3.4/test/test_pickletools.py", line 21
    support.run_doctest(pickletools)
  File "/usr/lib/python3.4/test/regrtest.py", line 1276
    test_runner()
  File "/usr/lib/python3.4/test/regrtest.py", line 976
    display_failure=not verbose)
  File "/usr/lib/python3.4/test/regrtest.py", line 761
    match_tests=ns.match_tests)
  File "/usr/lib/python3.4/test/regrtest.py", line 1563
    main()
  File "/usr/lib/python3.4/test/__main__.py", line 3
    regrtest.main_in_temp_cwd()
  File "/usr/lib/python3.4/runpy.py", line 73
    exec(code, run_globals)
  File "/usr/lib/python3.4/runpy.py", line 160
    "__main__", fname, loader, pkg_name)
```

We can see that the most memory was allocated in the `importlib` module to load data (bytecode and constants) from modules: `870.1 KiB`. The traceback is where the `importlib` loaded data most recently: on the `import pdb` line of the `doctest` module. The traceback may change if a new module is loaded.

27.7.1.4. Pretty top

Code to display the 10 lines allocating the most memory with a pretty output, ignoring `<frozen importlib._bootstrap>` and `<unknown>` files:

```
import linecache
import os
import tracemalloc

def display_top(snapshot, key_type='lineno', limit=10):
    snapshot = snapshot.filter_traces((
        tracemalloc.Filter(False, "<frozen importlib._bootstrap>"),
        tracemalloc.Filter(False, "<unknown>"),
    ))
    top_stats = snapshot.statistics(key_type)

    print("Top %s lines" % limit)
    for index, stat in enumerate(top_stats[:limit], 1):
        frame = stat.traceback[0]
        # replace "/path/to/module/file.py" with "module/file.py"
        filename = os.sep.join(frame.filename.split(os.sep)[-2:])
        print("#%s: %s:%s: %.1f KiB"
```

```
                % (index, filename, frame.lineno, stat.size / 1024))
        line = linecache.getline(frame.filename, frame.lineno).strip()
        if line:
            print('    %s' % line)

    other = top_stats[limit:]
    if other:
        size = sum(stat.size for stat in other)
        print("%s other: %.1f KiB" % (len(other), size / 1024))
    total = sum(stat.size for stat in top_stats)
    print("Total allocated size: %.1f KiB" % (total / 1024))

tracemalloc.start()

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
display_top(snapshot)
```

Example of output of the Python test suite:

```
Top 10 lines
#1: Lib/base64.py:414: 419.8 KiB
    _b85chars2 = [(a + b) for a in _b85chars for b in _b85chars]
#2: Lib/base64.py:306: 419.8 KiB
    _a85chars2 = [(a + b) for a in _a85chars for b in _a85chars]
#3: collections/__init__.py:368: 293.6 KiB
    exec(class_definition, namespace)
#4: Lib/abc.py:133: 115.2 KiB
    cls = super().__new__(mcls, name, bases, namespace)
#5: unittest/case.py:574: 103.1 KiB
    testMethod()
#6: Lib/linecache.py:127: 95.4 KiB
    lines = fp.readlines()
#7: urllib/parse.py:476: 71.8 KiB
    for a in _hexdig for b in _hexdig}
#8: <string>:5: 62.0 KiB
#9: Lib/_weakrefset.py:37: 60.0 KiB
    self.data = set()
#10: Lib/base64.py:142: 59.8 KiB
    _b32tab2 = [a + b for a in _b32tab for b in _b32tab]
6220 other: 3602.8 KiB
Total allocated size: 5303.1 KiB
```

See `Snapshot.statistics()` for more options.

27.7.2. API

27.7.2.1. Functions

`tracemalloc.`**`clear_traces`**()

Clear traces of memory blocks allocated by Python.

See also `stop()`.

`tracemalloc.`**`get_object_traceback`**(*obj*)

Get the traceback where the Python object *obj* was allocated. Return a `Traceback` instance, or `None` if the `tracemalloc` module is not tracing memory allocations or did not trace the allocation of the object.

See also `gc.get_referrers()` and `sys.getsizeof()` functions.

`tracemalloc.`**`get_traceback_limit`**()

Get the maximum number of frames stored in the traceback of a trace.

The `tracemalloc` module must be tracing memory allocations to get the limit, otherwise an exception is raised.

The limit is set by the `start()` function.

`tracemalloc.`**`get_traced_memory`**()

Get the current size and peak size of memory blocks traced by the `tracemalloc` module as a tuple: `(current: int, peak: int)`.

`tracemalloc.`**`get_tracemalloc_memory`**()

Get the memory usage in bytes of the `tracemalloc` module used to store traces of memory blocks. Return an `int`.

`tracemalloc.`**`is_tracing`**()

`True` if the `tracemalloc` module is tracing Python memory allocations, `False` otherwise.

See also `start()` and `stop()` functions.

`tracemalloc.`**`start`**(*nframe: int=1*)

Start tracing Python memory allocations: install hooks on Python memory allocators. Collected tracebacks of traces will be limited to *nframe* frames. By default, a trace of a memory block only stores the most recent frame: the limit is `1`. *nframe* must be greater or equal to `1`.

Storing more than `1` frame is only useful to compute statistics grouped by `'traceback'` or to compute cumulative statistics: see the `Snapshot.compare_to()` and `Snapshot.statistics()` methods.

Storing more frames increases the memory and CPU overhead of the `tracemalloc` module. Use the `get_tracemalloc_memory()` function to measure how much memory is used by the `tracemalloc` module.

The `PYTHONTRACEMALLOC` environment variable (`PYTHONTRACEMALLOC=NFRAME`) and the `-X tracemalloc=NFRAME` command line option can be used to start tracing at startup.

See also `stop()`, `is_tracing()` and `get_traceback_limit()` functions.

`tracemalloc.`**`stop`**()

Stop tracing Python memory allocations: uninstall hooks on Python memory allocators. Also clears all previously collected traces of memory blocks allocated by Python.

Call `take_snapshot()` function to take a snapshot of traces before clearing them.

See also `start()`, `is_tracing()` and `clear_traces()` functions.

`tracemalloc.`**`take_snapshot`**()

Take a snapshot of traces of memory blocks allocated by Python. Return a new `Snapshot` instance.

The snapshot does not include memory blocks allocated before the `tracemalloc` module started to trace memory allocations.

Tracebacks of traces are limited to `get_traceback_limit()` frames. Use the *nframe* parameter of the `start()` function to store more frames.

The `tracemalloc` module must be tracing memory allocations to take a snapshot, see the `start()` function.

See also the `get_object_traceback()` function.

27.7.2.2. DomainFilter

class `tracemalloc.`**`DomainFilter`**(*inclusive: bool*, *domain: int*)

Filter traces of memory blocks by their address space (domain).

New in version 3.6.

inclusive

If *inclusive* is `True` (include), match memory blocks allocated in the address space `domain`.

If *inclusive* is `False` (exclude), match memory blocks not allocated in the address space `domain`.

domain

Address space of a memory block (`int`). Read-only property.

27.7.2.3. Filter

class `tracemalloc.`**`Filter`**(*inclusive: bool*, *filename_pattern: str*, *lineno: int=None*, *all_frames: bool=False*, *domain: int=None*)

Filter on traces of memory blocks.

See the `fnmatch.fnmatch()` function for the syntax of *filename_pattern*. The `'.pyc'` file extension is replaced with `'.py'`.

Examples:

- `Filter(True, subprocess.__file__)` only includes traces of the `subprocess` module
- `Filter(False, tracemalloc.__file__)` excludes traces of the `tracemalloc` module
- `Filter(False, "<unknown>")` excludes empty tracebacks

Changed in version 3.5: The `'.pyo'` file extension is no longer replaced with `'.py'`.

Changed in version 3.6: Added the `domain` attribute.

domain

Address space of a memory block (`int` or `None`).

inclusive

If *inclusive* is `True` (include), only match memory blocks allocated in a file with a name matching `filename_pattern` at line number `lineno`.

If *inclusive* is `False` (exclude), ignore memory blocks allocated in a file with a name matching `filename_pattern` at line number `lineno`.

lineno

Line number (`int`) of the filter. If *lineno* is `None`, the filter matches any line number.

filename_pattern

Filename pattern of the filter (`str`). Read-only property.

all_frames

If *all_frames* is `True`, all frames of the traceback are checked. If *all_frames* is `False`, only the most recent frame is checked.

This attribute has no effect if the traceback limit is `1`. See the `get_traceback_limit()` function and `Snapshot.traceback_limit` attribute.

27.7.2.4. Frame

class `tracemalloc.`**Frame**

Frame of a traceback.

The `Traceback` class is a sequence of `Frame` instances.

filename

Filename (`str`).

lineno

Line number (`int`).

27.7.2.5. Snapshot

class `tracemalloc.`**Snapshot**

Snapshot of traces of memory blocks allocated by Python.

The `take_snapshot()` function creates a snapshot instance.

compare_to(*old_snapshot: Snapshot*, *key_type: str*, *cumulative: bool=False*)

Compute the differences with an old snapshot. Get statistics as a sorted list of `StatisticDiff` instances grouped by *key_type*.

See the `Snapshot.statistics()` method for *key_type* and *cumulative* parameters.

The result is sorted from the biggest to the smallest by: absolute value of `StatisticDiff.size_diff`, `StatisticDiff.size`, absolute value of `StatisticDiff.count_diff`, `Statistic.count` and then by `StatisticDiff.traceback`.

dump(*filename*)

Write the snapshot into a file.

Use `load()` to reload the snapshot.

filter_traces(*filters*)

Create a new `Snapshot` instance with a filtered `traces` sequence, *filters* is a list of `DomainFilter` and `Filter` instances. If *filters* is an empty list, return a new `Snapshot` instance with a copy of the traces.

All inclusive filters are applied at once, a trace is ignored if no inclusive filters match it. A trace is ignored if at least one exclusive filter matches it.

Changed in version 3.6: `DomainFilter` instances are now also accepted in *filters*.

classmethod **load**(*filename*)

Load a snapshot from a file.

See also `dump()`.

statistics(*key_type: str*, *cumulative: bool=False*)

Get statistics as a sorted list of `Statistic` instances grouped by *key_type*:

key_type	description
`'filename'`	filename
`'lineno'`	filename and line number
`'traceback'`	traceback

If *cumulative* is `True`, cumulate size and count of memory blocks of all frames of the traceback of a trace, not only the most recent frame. The cumulative mode can only be used with *key_type* equals to `'filename'` and `'lineno'`.

The result is sorted from the biggest to the smallest by: `Statistic.size`, `Statistic.count` and then by `Statistic.traceback`.

traceback_limit

Maximum number of frames stored in the traceback of `traces`: result of the `get_traceback_limit()` when the snapshot was taken.

traces

Traces of all memory blocks allocated by Python: sequence of `Trace` instances.

The sequence has an undefined order. Use the `Snapshot.statistics()` method to get a sorted list of statistics.

27.7.2.6. Statistic

class `tracemalloc.`**`Statistic`**

Statistic on memory allocations.

`Snapshot.statistics()` returns a list of `Statistic` instances.

See also the `StatisticDiff` class.

`count`

Number of memory blocks (`int`).

`size`

Total size of memory blocks in bytes (`int`).

`traceback`

Traceback where the memory block was allocated, `Traceback` instance.

27.7.2.7. StatisticDiff

class `tracemalloc.`**`StatisticDiff`**

Statistic difference on memory allocations between an old and a new `Snapshot` instance.

`Snapshot.compare_to()` returns a list of `StatisticDiff` instances. See also the `Statistic` class.

`count`

Number of memory blocks in the new snapshot (`int`): `0` if the memory blocks have been released in the new snapshot.

`count_diff`

Difference of number of memory blocks between the old and the new snapshots (`int`): `0` if the memory blocks have been allocated in the new snapshot.

`size`

Total size of memory blocks in bytes in the new snapshot (`int`): `0` if the memory blocks have been released in the new snapshot.

`size_diff`

Difference of total size of memory blocks in bytes between the old and the new snapshots (`int`): `0` if the memory blocks have been allocated in the new snapshot.

`traceback`

Traceback where the memory blocks were allocated, `Traceback` instance.

27.7.2.8. Trace

class `tracemalloc.`**`Trace`**

Trace of a memory block.

The `Snapshot.traces` attribute is a sequence of `Trace` instances.

`size`

Size of the memory block in bytes (`int`).

`traceback`

Traceback where the memory block was allocated, `Traceback` instance.

27.7.2.9. Traceback

class `tracemalloc.`**`Traceback`**

Sequence of `Frame` instances sorted from the most recent frame to the oldest frame.

A traceback contains at least `1` frame. If the `tracemalloc` module failed to get a frame, the filename `"<unknown>"` at line number `0` is used.

When a snapshot is taken, tracebacks of traces are limited to `get_traceback_limit()` frames. See the `take_snapshot()` function.

The `Trace.traceback` attribute is an instance of `Traceback` instance.

`format`(*limit=None*)

Format the traceback as a list of lines with newlines. Use the `linecache` module to retrieve lines from the source code. If *limit* is set, only format the *limit* most recent frames.

Similar to the `traceback.format_tb()` function, except that `format()` does not include newlines.

Example:

```
print("Traceback (most recent call first):")
for line in traceback:
    print(line)
```

Output:

```
Traceback (most recent call first):
  File "test.py", line 9
    obj = Object()
  File "test.py", line 12
    tb = tracemalloc.get_object_traceback(f())
```