# 使用 DTrace 和 SystemTap 檢測 CPython

**發🄵** ***3.12.4***

**Guido van Rossum and the Python development team**

**7 月 31, 2024**

**Python Software Foundation**
**Email: docs@python.org**

## Contents



**作者**
David Malcolm

**作者**
Łukasz Langa

DTrace 和 SystemTap 都是監測工具，各自都提供了一種檢查電腦系統之行程 (process) 正在執行什🄵操作的方法。它們都使用了領域限定 (domain-specific) 的語言，允許使用者編寫以下🄵本：

- 過濾要觀察的行程
- 收集來自感興趣之行程的資料
- 以這些資料🄵生報告

從 Python 3.6 開始，CPython 可以使用嵌入式「標記 (marker)」（也稱🄵「探針 (probe)」）建置，可以透過 DTrace 或 SystemTap 🄴本進行觀察，從而更輕松地監視系統之 CPython 行程正在執行的操作。

**CPython 實作細節：** DTrace 標記是 CPython 直譯器的實作細節。不保證 CPython 版本之間的探針相容性。更改 CPython 版本時，DTrace 🄴本可能會停止運作或錯誤地運作，而不會發出警告。

# 1 用態標記

macOS 建了對 DTrace 的支援。在 Linux 上，了建置帶有 SystemTap 嵌入標記的 CPython，必須安裝 SystemTap 開發工具。

在 Linux 機器上，這可以透過以下方式完成：

```
$ yum install systemtap-sdt-devel
```

或是：

```
$ sudo apt-get install systemtap-sdt-dev
```

然後 CPython 必須使用配置 `--with-dtrace` 選項：

```
checking for --with-dtrace... yes
```

在 macOS 上，你可以透過在後台運行 Python 行程列出 Python 發布者 (provider) 所提供的所有可用探針，以列出可用的 DTrace 探針：

```
$ python3.6 -q &
$ sudo dtrace -l -P python$!  # or: dtrace -l -m python3.6

   ID   PROVIDER            MODULE                          FUNCTION NAME
29564 python18035        python3.6          _PyEval_EvalFrameDefault function-entry
29565 python18035        python3.6             dtrace_function_entry function-entry
29566 python18035        python3.6          _PyEval_EvalFrameDefault function-return
29567 python18035        python3.6            dtrace_function_return function-return
29568 python18035        python3.6                           collect gc-done
29569 python18035        python3.6                           collect gc-start
29570 python18035        python3.6          _PyEval_EvalFrameDefault line
29571 python18035        python3.6                 maybe_dtrace_line line
```

在 Linux 上，你可以透過查看二進位建置檔案中是否包含".note.stapsdt" 部分來驗證 SystemTap 態標記是否存在。

```
$ readelf -S ./python | grep .note.stapsdt
[30] .note.stapsdt        NOTE         0000000000000000 00308d78
```

如果你已將 Python 建置共享函式庫（使用 `--enable-shared` 配置選項），則需要在共享函式庫中查找。例如：

```
$ readelf -S libpython3.3dm.so.1.0 | grep .note.stapsdt
[29] .note.stapsdt        NOTE         0000000000000000 00365b68
```

足現代化的 readelf 可以印出元資料 (metadata)：

```
$ readelf -n ./python

Displaying notes found at file offset 0x00000254 with length 0x00000020:
    Owner                 Data size          Description
    GNU                  0x00000010          NT_GNU_ABI_TAG (ABI version tag)
        OS: Linux, ABI: 2.6.32

Displaying notes found at file offset 0x00000274 with length 0x00000024:
    Owner                 Data size          Description
    GNU                  0x00000014          NT_GNU_BUILD_ID (unique build ID␣
```

(繼續下一頁)

(繼續上一頁)

```
↪bitstring)
        Build ID: df924a2b08a7e89f6e11251d4602022977af2670

Displaying notes found at file offset 0x002d6c30 with length 0x00000144:
    Owner                 Data size          Description
    stapsdt               0x00000031         NT_STAPSDT (SystemTap probe descriptors)
        Provider: python
        Name: gc__start
        Location: 0x00000000004371c3, Base: 0x0000000000630ce2, Semaphore:␣
↪0x00000000008d6bf6
        Arguments: -4@%ebx
    stapsdt               0x00000030         NT_STAPSDT (SystemTap probe descriptors)
        Provider: python
        Name: gc__done
        Location: 0x00000000004374e1, Base: 0x0000000000630ce2, Semaphore:␣
↪0x00000000008d6bf8
        Arguments: -8@%rax
    stapsdt               0x00000045         NT_STAPSDT (SystemTap probe descriptors)
        Provider: python
        Name: function__entry
        Location: 0x000000000053db6c, Base: 0x0000000000630ce2, Semaphore:␣
↪0x00000000008d6be8
        Arguments: 8@%rbp 8@%r12 -4@%eax
    stapsdt               0x00000046         NT_STAPSDT (SystemTap probe descriptors)
        Provider: python
        Name: function__return
        Location: 0x000000000053dba8, Base: 0x0000000000630ce2, Semaphore:␣
↪0x00000000008d6bea
        Arguments: 8@%rbp 8@%r12 -4@%eax
```

上述元資料包含 SystemTap 的資訊，描述了它如何修補策略性放置的機器碼指令以[F]用 SystemTap [F]本使用的追[F] hook。

## 2 [F]態 DTrace 探針

以下範例示範 DTrace [F]本可用於顯示 Python [F]本的呼叫/回傳階層結構，僅在名[F]"start" 的函式的呼叫[F]進行追[F]。[F]句話[F]，引入時的函式呼叫不會被列出：

```
self int indent;

python$target:::function-entry
/copyinstr(arg1) == "start"/
{
        self->trace = 1;
}

python$target:::function-entry
/self->trace/
{
        printf("%d\t%*s:", timestamp, 15, probename);
        printf("%*s", self->indent, "");
        printf("%s:%s:%d\n", basename(copyinstr(arg0)), copyinstr(arg1), arg2);
        self->indent++;
}
```

(繼續下一頁)

(繼續上一頁)

```
python$target:::function-return
/self->trace/
{
        self->indent--;
        printf("%d\t%*s:", timestamp, 15, probename);
        printf("%*s", self->indent, "");
        printf("%s:%s:%d\n", basename(copyinstr(arg0)), copyinstr(arg1), arg2);
}

python$target:::function-return
/copyinstr(arg1) == "start"/
{
        self->trace = 0;
}
```

可以這樣呼叫：

```
$ sudo dtrace -q -s call_stack.d -c "python3.6 script.py"
```

輸出如下所示：

```
156641360502280  function-entry:call_stack.py:start:23
156641360518804  function-entry: call_stack.py:function_1:1
156641360532797  function-entry:  call_stack.py:function_3:9
156641360546807 function-return:  call_stack.py:function_3:10
156641360563367 function-return: call_stack.py:function_1:2
156641360578365  function-entry: call_stack.py:function_2:5
156641360591757  function-entry:  call_stack.py:function_1:1
156641360605556  function-entry:   call_stack.py:function_3:9
156641360617482 function-return:   call_stack.py:function_3:10
156641360629814 function-return:  call_stack.py:function_1:2
156641360642285 function-return: call_stack.py:function_2:6
156641360656770  function-entry: call_stack.py:function_3:9
156641360669707 function-return: call_stack.py:function_3:10
156641360687853  function-entry: call_stack.py:function_4:13
156641360700719 function-return: call_stack.py:function_4:14
156641360719640  function-entry: call_stack.py:function_5:18
156641360732567 function-return: call_stack.py:function_5:21
156641360747370 function-return:call_stack.py:start:28
```

## 3 🄵態 SystemTap 標記

使用 SystemTap 整合的低階方法是直接使用🄵態標記。這會需要你明確聲明包含它們的二進位檔案。

例如，此 SystemTap 🄵本可用於顯示 Python 🄵本的呼叫/回傳階層結構：

```
probe process("python").mark("function__entry") {
     filename = user_string($arg1);
     funcname = user_string($arg2);
     lineno = $arg3;

     printf("%s => %s in %s:%d\\n",
            thread_indent(1), funcname, filename, lineno);
```

(繼續下一頁)

(繼續上一頁)

```
}

probe process("python").mark("function__return") {
    filename = user_string($arg1);
    funcname = user_string($arg2);
    lineno = $arg3;

    printf("%s <= %s in %s:%d\\n",
           thread_indent(-1), funcname, filename, lineno);
}
```

可以這樣呼叫：

```
$ stap \
  show-call-hierarchy.stp \
  -c "./python test.py"
```

輸出如下所示：

```
11408 python(8274):         => __contains__ in Lib/_abcoll.py:362
11414 python(8274):          => __getitem__ in Lib/os.py:425
11418 python(8274):           => encode in Lib/os.py:490
11424 python(8274):           <= encode in Lib/os.py:493
11428 python(8274):          <= __getitem__ in Lib/os.py:426
11433 python(8274):         <= __contains__ in Lib/_abcoll.py:366
```

其中的行 (column) 是：

- 自Ⓕ本開始以來的時間（以微秒Ⓕ單位）
- 可執行檔案的名稱
- 行程的 PID

其余部分表示Ⓕ本執行時的呼叫/回傳階層結構。

對於以 `--enable-shared` 建置的 CPython，標記被包含在 libpython 共享函式庫中，Ⓕ且探針的帶點路徑 (dotted path) 需要反映這一點。例如，上面範例中的這一列：

```
probe process("python").mark("function__entry") {
```

應該改Ⓕ讀取：

```
probe process("python").library("libpython3.6dm.so.1.0").mark("function__entry") {
```

（假設 CPython 3.6 的除錯建置版本）

## 4 可用的Ⓕ態標記

**`function__entry(str filename, str funcname, int lineno)`**

該標記表示 Python 函式的執行已經開始。它僅針對純 Python（位元組碼）函式觸發。

檔案名稱、函式名稱和列號作Ⓕ位置引數提供給追ⒻⒻ本，必須使用 `$arg1`、`$arg2`、`$arg3` 來存取：

- `$arg1`：`(const char *)` 檔案名稱，可使用 `user_string($arg1)` 存取
- `$arg2`：`(const char *)` 函式名稱，可使用 `user_string($arg2)` 存取

- `$arg3 : int` 列號

**function__return(str filename, str funcname, int lineno)**

該標記與 `function__entry()` 相反，表示 Python 函式的執行已結束（透過 `return` 或透過例外）。它僅針對純 Python（位元組碼）函式觸發。

引數與 `function__entry()` 相同

**line(str filename, str funcname, int lineno)**

該標記表示一列 Python 即將被執行。它相當於使用 Python 分析器來逐行追🄵。它不在 C 函式🄵觸發。

引數與 `function__entry()` 相同。

**gc__start(int generation)**

當 Python 直譯器開始垃圾回收 (garbage collection) 🄵期時觸發。`arg0` 是要掃描的一代 (generation)，如 `gc.collect()`。

**gc__done(long collected)**

當 Python 直譯器完成垃圾回收🄵期時觸發。`arg0` 是收集到的物件數量。

**import__find__load__start(str modulename)**

在 `importlib` 嘗試查找🄵載入模組之前觸發。`arg0` 是模組名稱。

在 3.7 版被加入.

**import__find__load__done(str modulename, int found)**

在呼叫 `importlib` 的 find_and_load 函式後觸發。`arg0` 是模組名稱，`arg1` 代表模組是否已成功載入。

在 3.7 版被加入.

**audit(str event, void *tuple)**

當呼叫 `sys.audit()` 或 `PySys_Audit()` 時觸發。`arg0` 是 C 字串形式的事件名稱，`arg1` 是指向元組 (tuple) 物件的 `PyObject` 指標。

在 3.8 版被加入.

# 5 SystemTap Tapsets

使用 SystemTap 整合的高階方法是使用"tapset"：SystemTap 相當於一個函式庫，它隱藏了🄵態標記的一些低階詳細資訊。

這是一個 tapset 檔案，是基於 CPython 的非共享建置版本：

```
/*
   Provide a higher-level wrapping around the function__entry and
   function__return markers:
 \*/
probe python.function.entry = process("python").mark("function__entry")
{
    filename = user_string($arg1);
    funcname = user_string($arg2);
    lineno = $arg3;
    frameptr = $arg4
}
probe python.function.return = process("python").mark("function__return")
{
    filename = user_string($arg1);
```

(繼續下一頁)

(繼續上一頁)

```
    funcname = user_string($arg2);
    lineno = $arg3;
    frameptr = $arg4
}
```

如果此檔案是安裝在 SystemTap 的 tapset 目🄵中（例如 `/usr/share/systemtap/tapset`），則這些額外的探測點將可被使用：

**`python.function.entry(str filename, str funcname, int lineno, frameptr)`**

該探測點表示 Python 函式的執行已經開始。它僅針對純 Python（位元組碼）函式觸發。

**`python.function.return(str filename, str funcname, int lineno, frameptr)`**

這個探測點與 `python.function.return` 相反，表示 Python 函式的執行已經結束（透過 `return` 或者透過例外）。它僅針對純 Python（位元組碼）函式觸發。

# 6 范例

此 SystemTap 🄵本使用上面的 tapset 來更清晰地實作上面給出的追🄵 Python 函式呼叫階層結構的範例，而無需直接命名🄵態標記：

```
probe python.function.entry
{
  printf("%s => %s in %s:%d\n",
         thread_indent(1), funcname, filename, lineno);
}

probe python.function.return
{
  printf("%s <= %s in %s:%d\n",
         thread_indent(-1), funcname, filename, lineno);
}
```

以下🄵本使用上面的 tapset 來提供所有正在運行之 CPython 程式碼的近乎最高層視角，顯示整個系統中每秒最常被進入的 20 個位元組碼幀 (bytecode frame)：

```
global fn_calls;

probe python.function.entry
{
    fn_calls[pid(), filename, funcname, lineno] += 1;
}

probe timer.ms(1000) {
    printf("\033[2J\033[1;1H") /* clear screen \*/
    printf("%6s %80s %6s %30s %6s\n",
           "PID", "FILENAME", "LINE", "FUNCTION", "CALLS")
    foreach ([pid, filename, funcname, lineno] in fn_calls- limit 20) {
        printf("%6d %80s %6d %30s %6d\n",
            pid, filename, lineno, funcname,
            fn_calls[pid, filename, funcname, lineno]);
    }
    delete fn_calls;
}
```