监控 C API

在 3.13 版中添加。

一个扩展可能需要与事件监视系统进行交互。 订阅事件和注册回调都可通过在 `sys.monitoring` 中暴露的 Python API 来完成。

生成执行事件

下面的函数使得扩展可以在模拟执行 Python 代码时触发监控事件。 这样的函数都接受一个 `PyMonitoringState` 结构体，其中包含有关事件激活状态的简明信息以及事件参数，此类参数包括代表代码对象的 `PyObject*`、指令偏移量，有时还包括额外的、事件专属的参数（请参阅 `sys.monitoring` 了解有关不同事件回调签名的详情）。 `codelike` 参数应为 `types.CodeType` 的实例或是某个模拟它的类型。

VM 会在触发事件时禁用跟踪，因此用户不需要额外的操作。

监控函数被调用时不应设置异常，除非是下面列出的用于处理当前异常的函数。

type **PyMonitoringState**

事件类型状态的表示形式。 它由用户分配，但其内容则由下文描述的监控 API 函数来维护。

下面的所有函数均在成功时返回 0 并在失败时返回 -1 (同时设置一个异常)。

请参阅 `sys.monitoring` 获取事件的描述。

int **PyMonitoring_FirePyStartEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

发出 `PY_START` 事件。

int **PyMonitoring_FirePyResumeEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

发出 `PY_RESUME` 事件。

int **PyMonitoring_FirePyReturnEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *retval)

发出 `PY_RETURN` 事件。

int **PyMonitoring_FirePyYieldEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *retval)

发出 `PY_YIELD` 事件。

int **PyMonitoring_FireCallEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *callable, PyObject *arg0)

发出 `CALL` 事件。

int **PyMonitoring_FireLineEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, int lineno)

发出 `LINE` 事件。

int **PyMonitoring_FireJumpEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *target_offset)

发出 `JUMP` 事件。

int **PyMonitoring_FireBranchLeftEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *target_offset)

发出 `BRANCH_LEFT` 事件。

int **PyMonitoring_FireBranchRightEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *target_offset)

发出 `BRANCH_RIGHT` 事件。

int **PyMonitoring_FireCReturnEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *retval)

发出 `C_RETURN` 事件。

int **PyMonitoring_FirePyThrowEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

使用当前（即 PyErr_GetRaisedException() 返回的）异常发出 `PY_THROW` 事件。

int **PyMonitoring_FireRaiseEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

使用当前（即 PyErr_GetRaisedException() 所返回的）异常发出 `RAISE` 事件。event with the current).

int **PyMonitoring_FireCRaiseEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

使用当前（即 PyErr_GetRaisedException() 所返回的）异常发出 `C_RAISE` 事件。

int **PyMonitoring_FireReraiseEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

使用当前（即 PyErr_GetRaisedException() 所返回的）异常发出 `RERAISE` 事件。

int **PyMonitoring_FireExceptionHandledEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

使用当前（即 PyErr_GetRaisedException() 所返回的）异常发出 `EXCEPTION_HANDLED` 事件。

int **PyMonitoring_FirePyUnwindEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset)

使用当前（即 PyErr_GetRaisedException() 所返回的）异常发出 `PY_UNWIND` 事件。

int **PyMonitoring_FireStopIterationEvent**(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *value)

发出 `STOP_ITERATION` 事件。 如果 `value` 是一个 `StopIteration` 实例，它将被使用。 在其他情况下，将新建一个 `StopIteration` 实例并以 `value` 作为其参数。

管理监控状态

监控状态可在监控作用域的协助下进行管理。 一个作用域通常对应一个 python 函数。

`int` **PyMonitoring_EnterScope**(`PyMonitoringState *state_array, uint64_t *version, const uint8_t *event_types, Py_ssize_t length`)

进入一个监控作用域。 `event_types` 是由可从该作用域发生事件的事件 ID 组成的数组。 例如，`PY_START` 事件的 ID 值为 `PY_MONITORING_EVENT_PY_START`，其对应数字等于 `sys.monitoring.events.PY_START` 的以 2 为底的对数。 `state_array` 是由对应 `event_types` 中每个事件的监控状态组成的数组，它由用户进行分配但是由 `PyMonitoring_EnterScope()` 使用事件激活状态相关信息填充。 `event_types` 的大小 (因而也是 `state_array` 的大小) 由 `length` 给出。

`version` 是一个指向应与 `state_array` 一起由用户分配的值并初始化为 0，然后仅由 `PyMonitoring_EnterScope()` 本身来设置。 它允许此函数确定事件状态自上次调用以来是否发生改变，并在它们未改变时立即返回。

这里所称的作用域是词法意义下的作用域：一个函数、类或方法。 `PyMonitoring_EnterScope()` 应当在进入词法作用域时被调用。 在模拟一个递归 Python 函数之类的情况下，作用域可被重进入，并重用相同的 *state_array* 和 *version*。 当某个代码执行暂停时，例如在模拟一个生成器时，此作用域需要被退出并重进入。

对应 *event_types* 的宏如下：

宏	事件
PY_MONITORING_EVENT_BRANCH_LEFT	`BRANCH_LEFT`
PY_MONITORING_EVENT_BRANCH_RIGHT	`BRANCH_RIGHT`
PY_MONITORING_EVENT_CALL	`CALL`
PY_MONITORING_EVENT_C_RAISE	`C_RAISE`
PY_MONITORING_EVENT_C_RETURN	`C_RETURN`
PY_MONITORING_EVENT_EXCEPTION_HANDLED	`EXCEPTION_HANDLED`
PY_MONITORING_EVENT_INSTRUCTION	`INSTRUCTION`
PY_MONITORING_EVENT_JUMP	`JUMP`
PY_MONITORING_EVENT_LINE	`LINE`
PY_MONITORING_EVENT_PY_RESUME	`PY_RESUME`

宏	事件
PY_MONITORING_EVENT_PY_RETURN	PY_RETURN
PY_MONITORING_EVENT_PY_START	PY_START
PY_MONITORING_EVENT_PY_THROW	PY_THROW
PY_MONITORING_EVENT_PY_UNWIND	PY_UNWIND
PY_MONITORING_EVENT_PY_YIELD	PY_YIELD
PY_MONITORING_EVENT_RAISE	RAISE
PY_MONITORING_EVENT_RERAISE	RERAISE
PY_MONITORING_EVENT_STOP_ITERATION	STOP_ITERATION

`int` **PyMonitoring_ExitScope**(`void`)

退出使用 `PyMonitoring_EnterScope()` 进入的上一个作用域。

`int` **PY_MONITORING_IS_INSTRUMENTED_EVENT**(`uint8_t ev`)

如果对应事件 ID *ev* 的事件属于 局部事件 则返回真值。

> *Added in version 3.13.*

> *自 3.14 版本弃用:* 此函数状态为 soft deprecated。