异常处理

本章描述的函数将让你处理和触发 Python 异常。 了解一些 Python 异常处理的基础知识是很重要的。 它的工作原理有点像 POSIX `errno` 变量: (每个线程) 有一个最近发生的错误的全局指示器。 大多数 C API 函数在成功执行时将不理会它。 大多数 C API 函数也会返回一个错误指示器，如果它们应当返回一个指针则会返回 `NULL`，或者如果它们应当返回一个整数则会返回 `-1` (例外情况: `PyArg_*` 函数返回 `1` 表示成功而 `0` 表示失败)。

具体地说，错误指示器由三个对象指针组成：异常的类型，异常的值，和回溯对象。如果没有错误被设置，这些指针都可以是 `NULL` （尽管一些组合使禁止的，例如，如果异常类型是 `NULL`，你不能有一个非 `NULL` 的回溯）。

当一个函数由于它调用的某个函数失败而必须失败时，通常不会设置错误指示器；它调用的那个函数已经设置了它。而它负责处理错误和清理异常，或在清除其拥有的所有资源后返回（如对象应用或内存分配）。如果不准备处理异常，则 *不* 应该正常地继续。如果是由于一个错误返回，那么一定要向调用者表明已经设置了错误。如果错误没有得到处理或小心传播，对 Python/C API的其它调用可能不会有预期的行为，并且可能会以某种神秘的方式失败。

备注: 错误指示器 **不是** `sys.exc_info()` 的执行结果。 前者对应于尚未捕获（因而仍在传播）的异常，而后者会在异常被捕获之后（因而已停止传播）返回它。

打印和清理

`void` **`PyErr_Clear`**`()`

属于 稳定 ABI.

清除错误指示器。如果没有设置错误指示器，则不会有作用。

`void` **`PyErr_PrintEx`**`(int set_sys_last_vars)`

属于 稳定 ABI.

将标准回溯打印到 `sys.stderr` 并清除错误指示器。**除非** 错误是 `SystemExit`，这种情况下不会打印回溯进程，且会退出 Python 进程，并显示 `SystemExit` 实例指定的错误代码。

只有在错误指示器被设置时才需要调用这个函数，否则这会导致错误！

如果 *set_sys_last_vars* 为非零值，则变量 `sys.last_exc` 将被设为要打印的异常。 出于向下兼容性考虑，已弃用的变量 `sys.last_type`, `sys.last_value` 和 `sys.last_traceback` 也会被分别设为该异常的类型, 值和回溯。

> *在 3.12 版本发生变更:* 增加了对 `sys.last_exc` 的设置。

`void` **`PyErr_Print`**`()`

属于 稳定 ABI.

`PyErr_PrintEx(1)` 的别名。

`void` **PyErr_WriteUnraisable**(PyObject *obj)

属于 稳定 ABI.

使用当前异常和 *obj* 参数调用 `sys.unraisablehook()`。

当异常已被设置但解释器不可能实际引发该异常时，这个工具函数会向 `sys.stderr` 打印一条警告消息。 例如，当异常发生在 `__del__()` 方法中时就会使用该函数。

该函数调用时将传入单个参数 *obj*，它标识发生不可引发的异常所在的上下文。 如果可能，*obj* 的表示形式将打印在警告消息中。 如果 *obj* 为 `NULL`，将只打印回溯。

调用此函数时必须设置一个异常。

> *在 3.4 版本发生变更:* 打印回溯信息。 如果 *obj* 为 `NULL` 将只打印回溯。

> *在 3.8 版本发生变更:* 使用 `sys.unraisablehook()`。

`void` **PyErr_FormatUnraisable**(`const char *format, ...`)

与 `PyErr_WriteUnraisable()` 类似，但 *format* 和后续的形参有助于格式化警告消息；它们的含义和值与 `PyUnicode_FromFormat()` 中的相同。 `PyErr_WriteUnraisable(obj)` 大致等价于 `PyErr_FormatUnraisable("Exception ignored in: %R", obj)`。 如果 *format* 为 `NULL`，则只打印回溯信息。

> *Added in version 3.13.*

`void` **PyErr_DisplayException**(PyObject *exc)

属于 稳定 ABI 自 3.12 版起.

将 `exc` 的标准回溯显示打印到 `sys.stderr`，包括链式异常和注释。

> *Added in version 3.12.*

抛出异常

这些函数可帮助你设置当前线程的错误指示器。为了方便起见，一些函数将始终返回 `NULL` 指针，以便用于 `return` 语句。

`void` **PyErr_SetString**(PyObject *type, `const char *message`)

属于 稳定 ABI.

这是设置错误指示器最常用的方式。 第一个参数指定异常类型；它通常为某个标准异常，例如 `PyExc_RuntimeError`。 你无需为其创建新的 strong reference (例如使用 `Py_INCREF()`)。 第二个参数是一条错误消息；它是用 `'utf-8'` 解码的。

`void` **PyErr_SetObject**(PyObject *type, PyObject *value)

属于 稳定 ABI.

此函数类似于 PyErr_SetString()，但是允许你为异常的"值"指定任意一个 Python 对象。

PyObject *PyErr_Format(PyObject *exception, const char *format, ...)

返回值：恒为 NULL。 属于 稳定 ABI.

这个函数设置了一个错误指示器并且返回了 NULL，*exception* 应当是一个 Python 中的异常类。*format* 和随后的形参会帮助格式化这个错误的信息；它们与 PyUnicode_FromFormat() 有着相同的含义和值。*format* 是一个ASCII编码的字符串。

PyObject *PyErr_FormatV(PyObject *exception, const char *format, va_list vargs)

返回值：恒为 NULL。 属于 稳定 ABI 自 3.5 版起.

和 PyErr_Format() 相同，但它接受一个 va_list 类型的参数而不是可变数量的参数集。

> *Added in version 3.5.*

void PyErr_SetNone(PyObject *type)

属于 稳定 ABI.

这是 PyErr_SetObject(type, Py_None) 的简写。

int PyErr_BadArgument()

属于 稳定 ABI.

这是 PyErr_SetString(PyExc_TypeError, message) 的简写，其中 *message* 指出使用了非法参数调用内置操作。它主要用于内部使用。

PyObject *PyErr_NoMemory()

返回值：恒为 NULL。 属于 稳定 ABI.

这是 PyErr_SetNone(PyExc_MemoryError) 的简写；它返回 NULL，以便当内存耗尽时，对象分配函数可以写 return PyErr_NoMemory(); 。

PyObject *PyErr_SetFromErrno(PyObject *type)

返回值：恒为 NULL。 属于 稳定 ABI.

这是一个便捷函数，当在 C 库函数返回错误并设置 C 变量 errno 时它会引发一个异常。 它构造了一个元组对象，其第一项是整数值 errno 而第二项是对应的错误信息（从 strerror() 获取），然后调用 PyErr_SetObject(type, object)。 在 Unix 上，当 errno 的值为 EINTR 时，表示有一个中断的系统调用，这将会调用 PyErr_CheckSignals()，如果它设置了错误指示符，则让其保持该设置。 该函数总是返回 NULL，因此当系统调用返回错误时该系统调用的包装函数可以写入 return PyErr_SetFromErrno(type); 。

PyObject *PyErr_SetFromErrnoWithFilenameObject(PyObject *type, PyObject *filenameObject)

返回值：恒为 NULL。 属于 稳定 ABI.

与 PyErr_SetFromErrno() 类似，但如果 *filenameObject* 不为 NULL，它将作为第三个参数传递给 *type* 的构造函数。 在 OSError 异常的情况下，它将被用于定义异常实例的 filename 属性。

PyObject *PyErr_SetFromErrnoWithFilenameObjects(PyObject *type, PyObject *filenameObject, PyObject *filenameObject2)

返回值：恒为 NULL。 属于 稳定 ABI 自 3.7 版起.

类似于 PyErr_SetFromErrnoWithFilenameObject()，但接受第二个 filename 对象，用于当一个接受两个 filename 的函数失败时触发错误。

> *Added in version 3.4.*

PyObject *PyErr_SetFromErrnoWithFilename(PyObject *type, const char *filename)

返回值：恒为 NULL。 属于 稳定 ABI.

类似于 PyErr_SetFromErrnoWithFilenameObject()，但文件名以 C 字符串形式给出。 *filename* 是用 filesystem encoding and error handler 解码的。

PyObject *PyErr_SetFromWindowsErr(int ierr)

返回值：恒为 NULL。 属于 稳定 ABI on Windows 自 3.7 版起.

这是一个用于引发 OSError 的便捷函数。 如果调用时传入的 *ierr* 值为 0，则会改用对 GetLastError() 的调用所返回的错误代码。 它将调用 Win32 函数 FormatMessage() 来获取 *ierr* 或 GetLastError() 所给出的错误代码的 Windows 描述，然后构造一个 OSError 对象，其中 winerror 属性将设为该错误代码，strerror 属性将设为相应的错误消息（从 FormatMessage() 获得），然后再调用 PyErr_SetObject(PyExc_OSError, object)。 该函数将总是返回 NULL。

Availability: Windows.

PyObject *PyErr_SetExcFromWindowsErr(PyObject *type, int ierr)

返回值：恒为 NULL。 属于 稳定 ABI on Windows 自 3.7 版起.

类似于 PyErr_SetFromWindowsErr()，额外的参数指定要触发的异常类型。

Availability: Windows.

PyObject *PyErr_SetFromWindowsErrWithFilename(int ierr, const char *filename)

返回值：恒为 NULL。 属于 稳定 ABI on Windows 自 3.7 版起.

与 PyErr_SetFromWindowsErr() 类似，额外的不同点是如果 *filename* 不为 NULL，则会使用文件系统编码格式 (os.fsdecode()) 进行解码并作为第三个参数传递给 OSError 的构造器用于定义异常实例的 filename 属性。

Availability: Windows.

PyObject *PyErr_SetExcFromWindowsErrWithFilenameObject(PyObject *type, int ierr, PyObject *filename)

返回值：恒为 NULL。 属于 稳定 ABI on Windows 自 3.7 版起.

与 PyErr_SetExcFromWindowsErr() 类似，额外的不同点是如果 *filename* 不为 NULL，它将作为第三个参数传递给 OSError 的构造器用于定义异常实例的 filename 属性。

Availability: Windows.

PyObject *PyErr_SetExcFromWindowsErrWithFilenameObjects(PyObject *type, int ierr, PyObject *filename, PyObject *filename2)

返回值：恒为 NULL。 属于 稳定 ABI on Windows 自 3.7 版起.

类似于 PyErr_SetExcFromWindowsErrWithFilenameObject()，但是接受第二个 filename 对象。

Availability: Windows.

> *Added in version 3.4.*

PyObject *PyErr_SetExcFromWindowsErrWithFilename(PyObject *type, int ierr, const char *filename)

返回值：恒为 NULL。 属于 稳定 ABI on Windows 自 3.7 版起.

类似于 PyErr_SetFromWindowsErrWithFilename()，额外参数指定要触发的异常类型。

Availability: Windows.

PyObject *PyErr_SetImportError(PyObject *msg, PyObject *name, PyObject *path)

返回值：恒为 NULL。 属于 稳定 ABI 自 3.7 版起.

这是触发 ImportError 的便捷函数。 *msg* 将被设为异常的消息字符串。 *name* 和 *path*，（都可以为 NULL），将用来被设置 ImportError 对应的属性 name 和 path。

> *Added in version 3.3.*

PyObject *PyErr_SetImportErrorSubclass(PyObject *exception, PyObject *msg, PyObject *name, PyObject *path)

返回值：恒为 NULL。 属于 稳定 ABI 自 3.6 版起.

和 PyErr_SetImportError() 很类似，但这个函数允许指定一个 ImportError 的子类来触发。

> *Added in version 3.6.*

void PyErr_SyntaxLocationObject(PyObject *filename, int lineno, int col_offset)

设置当前异常的文件，行和偏移信息。如果当前异常不是 SyntaxError，则它设置额外的属性，使异常打印子系统认为异常是 SyntaxError。

> *Added in version 3.4.*

`void` **PyErr_SyntaxLocationEx**(`const char *filename, int lineno, int col_offset`)

属于 稳定 ABI 自 3.7 版起.

类似于 `PyErr_SyntaxLocationObject()`，但 *filename* 是用 filesystem encoding and error handler 解码的字节串。

> *Added in version 3.2.*

`void` **PyErr_SyntaxLocation**(`const char *filename, int lineno`)

属于 稳定 ABI.

类似于 `PyErr_SyntaxLocationEx()`，但省略了 *col_offset* parameter 形参。

`void` **PyErr_BadInternalCall**()

属于 稳定 ABI.

这是 `PyErr_SetString(PyExc_SystemError, message)` 的缩写，其中 *message* 表示使用了非法参数调用内部操作（例如，Python/C API 函数）。它主要用于内部使用。

发出警告

这些函数可以从 C 代码中发出警告。它们仿照了由 Python 模块 `warnings` 导出的那些函数。它们通常向 *sys.stderr* 打印一条警告信息；当然，用户也有可能已经指定将警告转换为错误，在这种情况下，它们将触发异常。也有可能由于警告机制出现问题，使得函数触发异常。如果没有触发异常，返回值为 `0`；如果触发异常，返回值为 `-1`。（无法确定是否实际打印了警告信息，也无法确定异常触发的原因。这是故意为之）。如果触发了异常，调用者应该进行正常的异常处理（例如，`Py_DECREF()` 持有引用并返回一个错误值）。

`int` **PyErr_WarnEx**(`PyObject *category, const char *message, Py_ssize_t stack_level`)

属于 稳定 ABI.

发出一个警告信息。参数 *category* 是一个警告类别（见下面）或 `NULL`； *message* 是一个 UTF-8 编码的字符串。 *stack_level* 是一个给出栈帧数量的正数；警告将从该栈帧中当前正在执行的代码行发出。 *stack_level* 为 1 的是调用 `PyErr_WarnEx()` 的函数，2 是在此之上的函数，以此类推。

警告类别必须是 `PyExc_Warning` 的子类， `PyExc_Warning` 是 `PyExc_Exception` 的子类；默认警告类别是 `PyExc_RuntimeWarning`。标准 Python 警告类别作为全局变量可用，所有其名称见 警告类型。

有关警告控制的信息，参见模块文档 `warnings` 和命令行文档中的 `-W` 选项。没有用于警告控制的 C API。

`int` **PyErr_WarnExplicitObject**(`PyObject *category, PyObject *message, PyObject *filename, int lineno, PyObject *module, PyObject *registry`)

发出一个对所有警告属性进行显式控制的警告消息。 这是位于 Python 函数 `warnings.warn_explicit()` 外层的直接包装；请查看其文档了解详情。 *module* 和 *registry* 参数可被设为 `NULL` 以得到相关文档所描述的默认效果。

> *Added in version 3.4.*

`int` **PyErr_WarnExplicit**(`PyObject` *category, `const char` *message, `const char` *filename, `int` lineno, `const char` *module, `PyObject` *registry)

属于 稳定 ABI.

类似于 `PyErr_WarnExplicitObject()` 不过 *message* 和 *module* 是 UTF-8 编码的字符串，而 *filename* 是由 filesystem encoding and error handler 解码的。

`int` **PyErr_WarnFormat**(`PyObject` *category, `Py_ssize_t` stack_level, `const char` *format, ...)

属于 稳定 ABI.

类似于 `PyErr_WarnEx()` 的函数，但使用 `PyUnicode_FromFormat()` 来格式化警告消息。 *format* 是使用 ASCII 编码的字符串。

> *Added in version 3.2.*

`int` **PyErr_ResourceWarning**(`PyObject` *source, `Py_ssize_t` stack_level, `const char` *format, ...)

属于 稳定 ABI 自 3.6 版起.

类似于 `PyErr_WarnFormat()` 的函数，但 *category* 是 `ResourceWarning` 并且它会将 *source* 传给 `warnings.WarningMessage`。

> *Added in version 3.6.*

查询错误指示器

`PyObject` ***PyErr_Occurred**()

返回值：借入的引用。 属于 稳定 ABI.

测试是否设置了错误指示器。 如已设置，则返回异常 *type* (传给对某个 `PyErr_Set*` 函数或 `PyErr_Restore()` 的最后一次调用的第一个参数)。 如未设置，则返回 `NULL`。 你并不会拥有对返回值的引用，因此你不需要对它执行 `Py_DECREF()`。

调用方必须有已附加的线程状态 attached thread state。

> **备注：** 不要将返回值与特定的异常进行比较；请改为使用 `PyErr_ExceptionMatches()`，如下所示。 （比较很容易失败因为对于类异常来说，异常可能是一个实例而不是类，或者它可能是预期的异常的一个子类。）

`int` **PyErr_ExceptionMatches**(`PyObject` *exc)

属于 稳定 ABI.

等价于 `PyErr_GivenExceptionMatches(PyErr_Occurred(), exc)`。 此函数应当只在实际设置了异常时才被调用；如果没有任何异常被引发则将发生非法内存访问。

`int` **PyErr_GivenExceptionMatches**(PyObject *given, PyObject *exc)
属于 稳定 ABI.

如果 *given* 异常与 *exc* 中的异常类型相匹配则返回真值。 如果 *exc* 是一个类对象，则当 *given* 是一个子类的实例时也将返回真值。 如果 *exc* 是一个元组，则该元组（以及递归的子元组）中的所有异常类型都将被搜索进行匹配。

PyObject ***PyErr_GetRaisedException**(`void`)
返回值：新的引用。 属于 稳定 ABI 自 3.12 版起.

返回当前被引发的异常，同时清除错误指示器。 如果错误指示器尚未设置则返回 `NULL`。

此函数会被需要捕获异常的代码，或需要临时保存和恢复错误指示器的代码所使用。

例如：

```
{
   PyObject *exc = PyErr_GetRaisedException();

   /* ... 可能产生其他错误的代码 ... */

   PyErr_SetRaisedException(exc);
}
```

参见： PyErr_GetHandledException()，保存当前正在处理的异常。

Added in version 3.12.

`void` **PyErr_SetRaisedException**(PyObject *exc)
属于 稳定 ABI 自 3.12 版起.

将 *exc* 设为当前被引发的异常，如果已设置则清空现有的异常。

警告： 此调用将偷取一个对 *exc* 的引用，它必须是一个有效的异常。

Added in version 3.12.

`void` **PyErr_Fetch**(PyObject **ptype, PyObject **pvalue, PyObject **ptraceback)
属于 稳定 ABI.

自 3.12 版本弃用: 使用 PyErr_GetRaisedException() 代替。

将错误指示符提取到三个变量中并传递其地址。 如果未设置错误指示符，则将三个变量都设为 `NULL`。 如果已设置，则将其清除并且你将得到对所提取的每个对象的引用。 值和回溯对象可以为 `NULL` 即使类型对象不为空。

备注： 此函数通常只被需要捕获异常或临时保存和恢复错误指示符的旧式代码所使用。

例如：

```
{
   PyObject *type, *value, *traceback;
   PyErr_Fetch(&type, &value, &traceback);

   /* ... 可能产生其他错误的代码 ... */

   PyErr_Restore(type, value, traceback);
}
```

void **PyErr_Restore**(PyObject *type, PyObject *value, PyObject *traceback)

属于 稳定 ABI.

> *自 3.12 版本弃用:* 请改用 PyErr_SetRaisedException()。

根据 *type*, *value* 和 *traceback* 这三个对象设置错误指示符，如果已设置了错误指示符则先清除它。如果三个对象均为 NULL，则清除错误指示符。 请不要传入 NULL 类型和非 NULL 的值或回溯。 异常类型应当是一个类。 请不要传入无效的异常类型或值。 （违反这些规则将导致微妙的后继问题。） 此调用会带走对每个对象的引用：你必须在调用之前拥有对每个对象的引用并且在调用之后你将不再拥有这些引用。 （如果你不理解这一点，就不要使用此函数。 勿谓言之不预。）

备注： 此函数通常只被需要临时保存和恢复错误指示符的旧代码所使用。 请使用 PyErr_Fetch() 来保存当前的错误指示符。

void **PyErr_NormalizeException**(PyObject **exc, PyObject **val, PyObject **tb)

属于 稳定 ABI.

> *自 3.12 版本弃用:* 请改用 PyErr_GetRaisedException()，以避免任何可能的去正规化。

在特定情况下，下面 PyErr_Fetch() 所返回的值可以是“非正规化的”，即 *exc 是一个类对象而 *val 不是同一个类的实例。 在这种情况下此函数可以被用来实例化类。 如果值已经是正规化的，则不做任何操作。 实现这种延迟正规化是为了提升性能。

备注： 此函数 *不会* 隐式地在异常值上设置 __traceback__ 属性。 如果想要适当地设置回溯，还需要以下附加代码片段：

```
if (tb != NULL) {
  PyException_SetTraceback(val, tb);
}
```

PyObject *__PyErr_GetHandledException__(void)

属于 稳定 ABI 自 3.11 版起.

提取激活的异常实例，就如 sys.exception() 所返回的一样。 这是指一个 *已被捕获* 的异常，而不是刚被引发的异常。 返回一个指向该异常的新引用或者 NULL。 不会修改解释器的异常状态。 Does not modify the interpreter's exception state.

备注: 此函数通常不会被需要处理异常的代码所使用。 它可被使用的场合是当代码需要临时保存并恢复异常状态的时候。 请使用 PyErr_SetHandledException() 来恢复或清除异常状态。

Added in version 3.11.

void **PyErr_SetHandledException**(PyObject *exc)

属于 稳定 ABI 自 3.11 版起.

设置激活的异常，就是从 sys.exception() 所获得的。 这是指一个 *已被捕获* 的异常，而不是刚被引发的异常。 要清空异常状态，请传入 NULL。

备注: 此函数通常不会被需要处理异常的代码所使用。 它被使用的场合是在代码需要临时保存并恢复异常状态的时候。 请使用 PyErr_GetHandledException() 来获取异常状态。

Added in version 3.11.

void **PyErr_GetExcInfo**(PyObject **ptype, PyObject **pvalue, PyObject **ptraceback)

属于 稳定 ABI 自 3.7 版起.

提取旧式的异常信息表示形式，就是从 sys.exc_info() 所获得的。 这是指一个 *已被捕获* 的异常，而不是刚被引发的异常。 返回分别指向三个对象的新引用，其中任何一个都可以为 NULL。 不会修改异常信息的状态。 此函数是为了向下兼容而保留的。 更推荐使用 PyErr_GetHandledException()。

备注: 此函数通常不会被需要处理异常的代码所使用。 它被使用的场合是在代码需要临时保存并恢复异常状态的时候。 请使用 PyErr_SetExcInfo() 来恢复或清除异常状态。

Added in version 3.3.

void **PyErr_SetExcInfo**(PyObject *type, PyObject *value, PyObject *traceback)

属于 稳定 ABI 自 3.7 版起.

设置异常信息，就是从 sys.exc_info() 所获得的，这是指一个 *已被捕获* 的异常，而不是刚被引发的异常。 此函数会偷取对参数的引用。 要清空异常状态，请为所有三个参数传入 NULL。 此函数是为了向下兼容而保留的。 更推荐使用 PyErr_SetHandledException()。

备注: 此函数通常不会被需要处理异常的代码所使用。 它被使用的场合是在代码需要临时保存并恢复异常状态的情况。 请使用 PyErr_GetExcInfo() 来读取异常状态。

> *Added in version 3.3.*

> *在 3.11 版本发生变更:* `type` 和 `traceback` 参数已不再被使用并且可以为 NULL。 解释器现在会根据异常实例（即 `value` 参数）来推断出它们。 此函数仍然会偷取对所有三个参数的引用。

信号处理

`int` **PyErr_CheckSignals()**

属于 稳定 ABI.

这个函数与Python的信号处理交互。

如果在主 Python 解释器下从主线程调用该函数，它将检查是否向进程发送了信号，如果是，则唤起相应的信号处理器。 如果支持 `signal` 模块，则可以唤起以 Python 编写的信号处理器。

该函数会尝试处理所有待处理信号，然后返回 `0`。 但是，如果 Python 信号处理器引发了异常，则设置错误指示符并且函数将立即返回 `-1` (这样其他待处理信号可能还没有被处理：它们将在下次唤起 `PyErr_CheckSignals()` 时被处理)。

如果函数从非主线程调用，或在非主Python解释器下调用，则它不执行任何操作并返回0。

这个函数可以由希望被用户请求(例如按Ctrl-C)中断的长时间运行的C代码调用。

> **备注:** 针对 `SIGINT` 的默认 Python 信号处理器会引发 `KeyboardInterrupt` 异常。

`void` **PyErr_SetInterrupt()**

属于 稳定 ABI.

模拟一个 `SIGINT` 信号到达的效果。 这等价于 `PyErr_SetInterruptEx(SIGINT)`。

> **备注:** 此函数是异步信号安全的。 它可以在没有 attached thread state 的情况下以及从 C 信号处理器中被调用。

`int` **PyErr_SetInterruptEx**(`int` signum)

属于 稳定 ABI 自 3.10 版起.

模拟一个信号到达的效果。 当下次 `PyErr_CheckSignals()` 被调用时，将会调用针对指定的信号编号的 Python 信号处理器。

此函数可由自行设置信号处理，并希望 Python 信号处理器会在请求中断时（例如当用户按下 Ctrl-C 来中断操作时）按照预期被唤起的 C 代码来调用。

如果给定的信号不是由 Python 来处理的 (即被设为 `signal.SIG_DFL` 或 `signal.SIG_IGN`)，它将会被忽略。

如果 *signum* 在被允许的信号编号范围之外，将返回 `-1`。 在其他情况下，则返回 `0`。 错误指示符绝不会被此函数所修改。

备注： 此函数是异步信号安全的。 它可以在没有 attached thread state 的情况下以及从 C 信号处理器中被调用。

Added in version 3.10.

`int` **PySignal_SetWakeupFd**(`int fd`)

这个工具函数指定了一个每当收到信号时将被作为以单个字节的形式写入信号编号的目标的文件描述符。 *fd* 必须是非阻塞的。 它将返回前一个这样的文件描述符。

设置值 `-1` 将禁用该特性；这是初始状态。 这等价于 Python 中的 `signal.set_wakeup_fd()`，但是没有任何错误检查。 *fd* 应当是一个有效的文件描述符。 此函数应当只从主线程来调用。

在 3.5 版本发生变更: 在 Windows 上，此函数现在也支持套接字处理。

Exception 类

`PyObject` ***PyErr_NewException**(`const char *name, PyObject *base, PyObject *dict`)

返回值：新的引用。 属于 稳定 ABI.

这个工具函数会创建并返回一个新的异常类。 *name* 参数必须为新异常的名称，是 `module.classname` 形式的 C 字符串。 *base* 和 *dict* 参数通常为 `NULL`。 这将创建一个派生自 `Exception` 的类对象（在 C 中可以通过 `PyExc_Exception` 访问）。

新类的 `__module__` 属性将被设为 *name* 参数的前半部分（最后一个点号之前），而类名将被设为后半部分（最后一个点号之后）。 *base* 参数可被用来指定替代基类；它可以是一个类或是一个由类组成的元组。 *dict* 参数可被用来指定一个由类变量和方法组成的字典。

`PyObject` ***PyErr_NewExceptionWithDoc**(`const char *name, const char *doc, PyObject *base, PyObject *dict`)

返回值：新的引用。 属于 稳定 ABI.

和 `PyErr_NewException()` 一样，除了可以轻松地给新的异常类一个文档字符串：如果 *doc* 属性非空，它将用作异常类的文档字符串。

Added in version 3.2.

`int` **PyExceptionClass_Check**(`PyObject *ob`)

如果 *ob* 是异常类则返回非零，否则返回零。此函数总是会成功执行。

`const char` ***PyExceptionClass_Name**(`PyObject *ob`)

属于 稳定 ABI 自 3.8 版起.

返回异常类 *ob* 的 `tp_name`。

异常对象

PyObject *PyException_GetTraceback(PyObject *ex)
返回值：新的引用。 属于 稳定 ABI.

将与异常相关联的回溯作为一个新引用返回，可以通过 `__traceback__` 属性在 Python 中访问。 如果没有已关联的回溯，则返回 `NULL`。

int PyException_SetTraceback(PyObject *ex, PyObject *tb)
属于 稳定 ABI.

将异常关联的回溯设置为 *tb* 。使用 `Py_None` 清除它。

PyObject *PyException_GetContext(PyObject *ex)
返回值：新的引用。 属于 稳定 ABI.

将与异常相关联的上下文（在处理 *ex* 过程中引发的另一个异常实例）作为一个新引用返回，可以通过 `__context__` 属性在 Python 中访问。 如果没有已关联的上下文，则返回 `NULL`。

void PyException_SetContext(PyObject *ex, PyObject *ctx)
属于 稳定 ABI.

将与异常相关联的上下文设置为 *ctx*。 使用 `NULL` 来清空它。 没有用来确保 *ctx* 是一个异常实例的类型检查。 这将窃取一个指向 *ctx* 的引用。

PyObject *PyException_GetCause(PyObject *ex)
返回值：新的引用。 属于 稳定 ABI.

将与异常相关联的原因（一个异常实例，或为 `None`，由 `raise ... from ...` 设置）作为一个新引用返回，可通过 `__cause__` 属性在 Python 中访问。

void PyException_SetCause(PyObject *ex, PyObject *cause)
属于 稳定 ABI.

将与异常相关联的原因设为 *cause*。 使用 `NULL` 来清空它。 不存在类型检查用来确保 *cause* 是一个异常实例或为 `None`。 这个偷取一个指向 *cause* 的引用。

`__suppress_context__` 属性会被此函数隐式地设为 `True`。

PyObject *PyException_GetArgs(PyObject *ex)
返回值：新的引用。 属于 稳定 ABI 自 3.12 版起.

返回异常 *ex* 的 `args`。

void PyException_SetArgs(PyObject *ex, PyObject *args)
属于 稳定 ABI 自 3.12 版起.

将异常 *ex* 的 `args` 设为 *args*。

PyObject *PyUnstable_Exc_PrepReraiseStar(PyObject *orig, PyObject *excs)

这是 不稳定 API。它可在次发布版中不经警告地改变。

解释器的 except* 实现的具体实现部分。 *orig* 是被捕获的原始异常，而 *excs* 是需要被引发的异常组成的列表。 该列表包含 *orig* 可能存在的未被处理的部分，以及在 except* 子句中被引发的异常（因而它们具有与 *orig* 不同的回溯数据）和被重新引发的异常（因而它们具有与 *orig* 相同的回溯）。 返回需要被最终引发的 ExceptionGroup，或者如果没有要被引发的异常则返回 None。

> *Added in version 3.12.*

Unicode 异常对象

下列函数被用于创建和修改来自 C 的 Unicode 异常。

```
PyObject *PyUnicodeDecodeError_Create(const char *encoding, const char
*object, Py_ssize_t length, Py_ssize_t start, Py_ssize_t end, const char
*reason)
```

返回值：新的引用。 属于 稳定 ABI.

创建一个 UnicodeDecodeError 对象并附带 *encoding*, *object*, *length*, *start*, *end* 和 *reason* 等属性。 *encoding* 和 *reason* 为 UTF-8 编码的字符串。

```
PyObject *PyUnicodeDecodeError_GetEncoding(PyObject *exc)
PyObject *PyUnicodeEncodeError_GetEncoding(PyObject *exc)
```

返回值：新的引用。 属于 稳定 ABI.

返回给定异常对象的 *encoding* 属性

```
PyObject *PyUnicodeDecodeError_GetObject(PyObject *exc)
PyObject *PyUnicodeEncodeError_GetObject(PyObject *exc)
PyObject *PyUnicodeTranslateError_GetObject(PyObject *exc)
```

返回值：新的引用。 属于 稳定 ABI.

返回给定异常对象的 *object* 属性

```
int PyUnicodeDecodeError_GetStart(PyObject *exc, Py_ssize_t *start)
int PyUnicodeEncodeError_GetStart(PyObject *exc, Py_ssize_t *start)
int PyUnicodeTranslateError_GetStart(PyObject *exc, Py_ssize_t *start)
```

属于 稳定 ABI.

获取给定异常对象的 *start* 属性并将其放入 **start*。 *start* 必须不为 NULL。 成功时返回 0，失败时返回 -1。

如果 UnicodeError.object 是一个空序列，则得到的 *start* 将为 0。 在其他情况下，它会被剪切为 [0, len(object) - 1]。

参见: UnicodeError.start

```
int PyUnicodeDecodeError_SetStart(PyObject *exc, Py_ssize_t start)
int PyUnicodeEncodeError_SetStart(PyObject *exc, Py_ssize_t start)
```

```
int PyUnicodeTranslateError_SetStart(PyObject *exc, Py_ssize_t start)
```
属于 稳定 ABI.

将给定异常对象的 *start* 属性设为 *start*。 成功时返回 `0`，失败时返回 `-1`。

> **备注：** 虽然传入负的 *start* 不会引发异常，但是对应的获取器也不会把它当作是相对偏移量。

```
int PyUnicodeDecodeError_GetEnd(PyObject *exc, Py_ssize_t *end)
int PyUnicodeEncodeError_GetEnd(PyObject *exc, Py_ssize_t *end)
int PyUnicodeTranslateError_GetEnd(PyObject *exc, Py_ssize_t *end)
```
属于 稳定 ABI.

获取给定异常对象的 *end* 属性并将其放入 **end*。 *end* 必须不为 `NULL`。 成功时返回 `0`，失败时返回 `-1`。

如果 `UnicodeError.object` 是一个空序列，则得到的 *end* 将为 `0`。 在其他情况下，它会被剪切为 `[1, len(object)]`。

```
int PyUnicodeDecodeError_SetEnd(PyObject *exc, Py_ssize_t end)
int PyUnicodeEncodeError_SetEnd(PyObject *exc, Py_ssize_t end)
int PyUnicodeTranslateError_SetEnd(PyObject *exc, Py_ssize_t end)
```
属于 稳定 ABI.

将给定异常对象的 *end* 属性设为 *end*。 成功时返回 `0`，失败时返回 `-1`。

> **参见：** `UnicodeError.end`

```
PyObject *PyUnicodeDecodeError_GetReason(PyObject *exc)
PyObject *PyUnicodeEncodeError_GetReason(PyObject *exc)
PyObject *PyUnicodeTranslateError_GetReason(PyObject *exc)
```
返回值：新的引用。 属于 稳定 ABI.

返回给定异常对象的 *reason* 属性

```
int PyUnicodeDecodeError_SetReason(PyObject *exc, const char *reason)
int PyUnicodeEncodeError_SetReason(PyObject *exc, const char *reason)
int PyUnicodeTranslateError_SetReason(PyObject *exc, const char *reason)
```
属于 稳定 ABI.

将给定异常对象的 *reason* 属性设为 *reason*。 成功时返回 `0`，失败时返回 `-1`。

递归控制

这两个函数提供了一种在 C 层级上进行安全的递归调用的方式，在核心模块与扩展模块中均适用。当递归代码不一定会唤起 Python 代码（后者会自动跟踪其递归深度）时就需要用到它们。 它们对于 *tp_call* 实现来说也无必要因为 调用协议 会负责递归处理。

int **Py_EnterRecursiveCall**(const char *where)

属于 稳定 ABI 自 3.9 版起.

标记一个递归的 C 层级调用即将被执行的点位。

随后此函数将检查是否达到栈限制。 如果是的话，将设置一个 RecursionError 并返回一个非零值。 在其他情况下，将返回零。

where 应为一个 UTF-8 编码的字符串如 `" in instance check"`，它将与由递归深度限制所导致的 RecursionError 消息相拼接。

> *在 3.9 版本发生变更:* 此函数现在也在 受限 API 中可用。

void **Py_LeaveRecursiveCall**(void)

属于 稳定 ABI 自 3.9 版起.

结束一个 Py_EnterRecursiveCall()。 必须针对 Py_EnterRecursiveCall() 的每个 *成功的* 唤起操作执行一次调用。

> *在 3.9 版本发生变更:* 此函数现在也在 受限 API 中可用。

正确地针对容器类型实现 tp_repr 需要特别的递归处理。 在保护栈之外，tp_repr 还需要追踪对象以防止出现循环。 以下两个函数将帮助完成此功能。 从实际效果来说，这两个函数是 C 中对应 reprlib.recursive_repr() 的等价物。

int **Py_ReprEnter**(PyObject *object)

属于 稳定 ABI.

在 tp_repr 实现的开头被调用以检测循环。

如果对象已经被处理，此函数将返回一个正整数。 在此情况下 tp_repr 实现应当返回一个指明发生循环的字符串对象。 例如，dict 对象将返回 `{...}` 而 list 对象将返回 `[...]`。

如果已达到递归限制则此函数将返回一个负正数。 在此情况下 tp_repr 实现通常应当返回 `NULL`。

在其他情况下，此函数将返回零而 tp_repr 实现将可正常继续。

void **Py_ReprLeave**(PyObject *object)

属于 稳定 ABI.

结束一个 Py_ReprEnter()。 必须针对每个返回零的 Py_ReprEnter() 的唤起操作调用一次。

异常与警告类型

所有的标准 Python 异常和警告类别都可以用作全局变量，其名称为 `PyExc_` 加上 Python 异常名称。 这些类型是 PyObject* 类型；它们都是类对象。

为完整说明，以下是全部的变量：

异常类型

C 名称	Python 名称
PyObject *PyExc_BaseException *属于 稳定 ABI.*	BaseException
PyObject *PyExc_BaseExceptionGroup *属于 稳定 ABI 自 3.11 版起.*	BaseExceptionGroup
PyObject *PyExc_Exception *属于 稳定 ABI.*	Exception
PyObject *PyExc_ArithmeticError *属于 稳定 ABI.*	ArithmeticError
PyObject *PyExc_AssertionError *属于 稳定 ABI.*	AssertionError
PyObject *PyExc_AttributeError *属于 稳定 ABI.*	AttributeError
PyObject *PyExc_BlockingIOError *属于 稳定 ABI 自 3.7 版起.*	BlockingIOError
PyObject *PyExc_BrokenPipeError *属于 稳定 ABI 自 3.7 版起.*	BrokenPipeError
PyObject *PyExc_BufferError *属于 稳定 ABI.*	BufferError
PyObject *PyExc_ChildProcessError *属于 稳定 ABI 自 3.7 版起.*	ChildProcessError
PyObject *PyExc_ConnectionAbortedError *属于 稳定 ABI 自 3.7 版起.*	ConnectionAbortedError
PyObject *PyExc_ConnectionError *属于 稳定 ABI 自 3.7 版起.*	ConnectionError
PyObject *PyExc_ConnectionRefusedError *属于 稳定 ABI 自 3.7 版起.*	ConnectionRefusedError
PyObject *PyExc_ConnectionResetError *属于 稳定 ABI 自 3.7 版起.*	ConnectionResetError
PyObject *PyExc_EOFError *属于 稳定 ABI.*	EOFError
PyObject *PyExc_FileExistsError *属于 稳定 ABI 自 3.7 版起.*	FileExistsError
PyObject *PyExc_FileNotFoundError *属于 稳定 ABI 自 3.7 版起.*	FileNotFoundError

C 名称	Python 名称
PyObject *PyExc_FloatingPointError *属于 稳定 ABI.*	FloatingPointError
PyObject *PyExc_GeneratorExit *属于 稳定 ABI.*	GeneratorExit
PyObject *PyExc_ImportError *属于 稳定 ABI.*	ImportError
PyObject *PyExc_IndentationError *属于 稳定 ABI.*	IndentationError
PyObject *PyExc_IndexError *属于 稳定 ABI.*	IndexError
PyObject *PyExc_InterruptedError *属于 稳定 ABI 自 3.7 版起.*	InterruptedError
PyObject *PyExc_IsADirectoryError *属于 稳定 ABI 自 3.7 版起.*	IsADirectoryError
PyObject *PyExc_KeyError *属于 稳定 ABI.*	KeyError
PyObject *PyExc_KeyboardInterrupt *属于 稳定 ABI.*	KeyboardInterrupt
PyObject *PyExc_LookupError *属于 稳定 ABI.*	LookupError
PyObject *PyExc_MemoryError *属于 稳定 ABI.*	MemoryError
PyObject *PyExc_ModuleNotFoundError *属于 稳定 ABI 自 3.6 版起.*	ModuleNotFoundError
PyObject *PyExc_NameError *属于 稳定 ABI.*	NameError
PyObject *PyExc_NotADirectoryError *属于 稳定 ABI 自 3.7 版起.*	NotADirectoryError
PyObject *PyExc_NotImplementedError *属于 稳定 ABI.*	NotImplementedError
PyObject *PyExc_OSError *属于 稳定 ABI.*	OSError
PyObject *PyExc_OverflowError *属于 稳定 ABI.*	OverflowError
PyObject *PyExc_PermissionError	PermissionError

C 名称	Python 名称
属于 稳定 ABI 自 3.7 版起.	
PyObject *PyExc_ProcessLookupError *属于 稳定 ABI 自 3.7 版起.*	ProcessLookupError
PyObject *PyExc_PythonFinalizationError	PythonFinalizationError
PyObject *PyExc_RecursionError *属于 稳定 ABI 自 3.7 版起.*	RecursionError
PyObject *PyExc_ReferenceError *属于 稳定 ABI.*	ReferenceError
PyObject *PyExc_RuntimeError *属于 稳定 ABI.*	RuntimeError
PyObject *PyExc_StopAsyncIteration *属于 稳定 ABI 自 3.7 版起.*	StopAsyncIteration
PyObject *PyExc_StopIteration *属于 稳定 ABI.*	StopIteration
PyObject *PyExc_SyntaxError *属于 稳定 ABI.*	SyntaxError
PyObject *PyExc_SystemError *属于 稳定 ABI.*	SystemError
PyObject *PyExc_SystemExit *属于 稳定 ABI.*	SystemExit
PyObject *PyExc_TabError *属于 稳定 ABI.*	TabError
PyObject *PyExc_TimeoutError *属于 稳定 ABI 自 3.7 版起.*	TimeoutError
PyObject *PyExc_TypeError *属于 稳定 ABI.*	TypeError
PyObject *PyExc_UnboundLocalError *属于 稳定 ABI.*	UnboundLocalError
PyObject *PyExc_UnicodeDecodeError *属于 稳定 ABI.*	UnicodeDecodeError
PyObject *PyExc_UnicodeEncodeError *属于 稳定 ABI.*	UnicodeEncodeError
PyObject *PyExc_UnicodeError *属于 稳定 ABI.*	UnicodeError
PyObject *PyExc_UnicodeTranslateError	UnicodeTranslateError

C 名称	Python 名称
属于 稳定 ABI.	
PyObject *PyExc_ValueError *属于 稳定 ABI.*	ValueError
PyObject *PyExc_ZeroDivisionError *属于 稳定 ABI.*	ZeroDivisionError

> *Added in version 3.3:* PyExc_BlockingIOError, PyExc_BrokenPipeError, PyExc_ChildProcessError, PyExc_ConnectionError, PyExc_ConnectionAbortedError, PyExc_ConnectionRefusedError, PyExc_ConnectionResetError, PyExc_FileExistsError, PyExc_FileNotFoundError, PyExc_InterruptedError, PyExc_IsADirectoryError, PyExc_NotADirectoryError, PyExc_PermissionError, PyExc_ProcessLookupError and PyExc_TimeoutError 介绍如下 **PEP 3151**.

> *Added in version 3.5:* PyExc_StopAsyncIteration 和 PyExc_RecursionError.

> *Added in version 3.6:* PyExc_ModuleNotFoundError.

> *Added in version 3.11:* PyExc_BaseExceptionGroup.

OSError 别名

以下是 PyExc_OSError 的兼容性别名。

> *在 3.3 版本发生变更:* 这些别名曾经是单独的异常类型。

C 名称	Python 名称	备注
PyObject *PyExc_EnvironmentError *属于 稳定 ABI.*	OSError	
PyObject *PyExc_IOError *属于 稳定 ABI.*	OSError	
PyObject *PyExc_WindowsError *属于 稳定 ABI on Windows 自 3.7 版起.*	OSError	[win]

注:

[win] PyExc_WindowsError 仅在 Windows 上定义；通过测试是否定义了预处理器宏 MS_WINDOWS 来保护用到它的代码。

警告类型

C 名称	Python 名称
PyObject *PyExc_Warning	Warning

C 名称	Python 名称
属于 稳定 ABI.	
PyObject *PyExc_BytesWarning *属于 稳定 ABI.*	BytesWarning
PyObject *PyExc_DeprecationWarning *属于 稳定 ABI.*	DeprecationWarning
PyObject *PyExc_EncodingWarning *属于 稳定 ABI 自 3.10 版起.*	EncodingWarning
PyObject *PyExc_FutureWarning *属于 稳定 ABI.*	FutureWarning
PyObject *PyExc_ImportWarning *属于 稳定 ABI.*	ImportWarning
PyObject *PyExc_PendingDeprecationWarning *属于 稳定 ABI.*	PendingDeprecationWarning
PyObject *PyExc_ResourceWarning *属于 稳定 ABI 自 3.7 版起.*	ResourceWarning
PyObject *PyExc_RuntimeWarning *属于 稳定 ABI.*	RuntimeWarning
PyObject *PyExc_SyntaxWarning *属于 稳定 ABI.*	SyntaxWarning
PyObject *PyExc_UnicodeWarning *属于 稳定 ABI.*	UnicodeWarning
PyObject *PyExc_UserWarning *属于 稳定 ABI.*	UserWarning

> *Added in version 3.2:* PyExc_ResourceWarning.

> *Added in version 3.10:* PyExc_EncodingWarning.