极高层级 API

本章节的函数将允许你执行在文件或缓冲区中提供的 Python 源代码，但它们将不允许你在更细节化的方式与解释器进行交互。

这些函数中有几个可以接受特定的语法前缀符号作为形参。 可用的前缀符号有 `Py_eval_input`, `Py_file_input` 和 `Py_single_input`。 这些符号会在接受它们作为形参的函数文档中加以说明。

还要注意这些函数中有几个可以接受 `FILE*` 形参。 有一个需要小心处理的特别问题是针对不同 C 库的 `FILE` 结构体可能是不相同且不兼容的。 （至少是）在 Windows 中，动态链接的扩展实际上有可能会使用不同的库，所以应当特别注意只有在确定这些函数是由 Python 运行时所使用的相同的库创建的情况下才将 `FILE*` 形参传给它们。

`int` **PyRun_AnyFile**(`FILE *fp, const char *filename`)
这是针对下面 `PyRun_AnyFileExFlags()` 的简化版接口，将 *closeit* 设为 `0` 而将 *flags* 设为 `NULL`。

`int` **PyRun_AnyFileFlags**(`FILE *fp, const char *filename, PyCompilerFlags *flags`)
这是针对下面 `PyRun_AnyFileExFlags()` 的简化版接口，将 *closeit* 参数设为 `0`。

`int` **PyRun_AnyFileEx**(`FILE *fp, const char *filename, int closeit`)
这是针对下面 `PyRun_AnyFileExFlags()` 的简化版接口，将 *flags* 参数设为 `NULL`。

`int` **PyRun_AnyFileExFlags**(`FILE *fp, const char *filename, int closeit, PyCompilerFlags *flags`)
如果 *fp* 指向一个关联到交互设备（控制台或终端输入或 Unix 伪终端）的文件，则返回 `PyRun_InteractiveLoop()` 的值，否则返回 `PyRun_SimpleFile()` 的结果。 *filename* 会使用文件系统的编码格式 (`sys.getfilesystemencoding()`) 来解码。 如果 *filename* 为 `NULL`，此函数会使用 `"???"` 作为文件名。 如果 *closeit* 为真值，文件会在 `PyRun_SimpleFileExFlags()` 返回之前被关闭。

`int` **PyRun_SimpleString**(`const char *command`)
这是针对下面 `PyRun_SimpleStringFlags()` 的简化版接口，将 `PyCompilerFlags*` 参数设为 `NULL`。

`int` **PyRun_SimpleStringFlags**(`const char *command, PyCompilerFlags *flags`)
根据 *flags* 参数，在 `__main__` 模块中执行 Python 源代码。 如果 `__main__` 尚不存在，它将被创建。 成功时返回 `0`，如果引发异常则返回 `-1`。 如果发生错误，则将无法获得异常信息。 对于 *flags* 的含义，请参阅下文。

请注意如果引发了一个在其他场合下未处理的 `SystemExit`，此函数将不会返回 `-1`，而是退出进程，只要 `PyConfig.inspect` 为零就会这样。

`int` **PyRun_SimpleFile**(`FILE *fp, const char *filename`)
这是针对下面 `PyRun_SimpleFileExFlags()` 的简化版接口，将 *closeit* 设为 `0` 而将 *flags* 设为 `NULL`。

`int` **PyRun_SimpleFileEx**(`FILE *fp, const char *filename, int closeit`)
这是针对下面 `PyRun_SimpleFileExFlags()` 的简化版接口，将 *flags* 设为 `NULL`。

`int` **PyRun_SimpleFileExFlags**(`FILE *fp, const char *filename, int closeit, PyCompilerFlags *flags`)
类似于 `PyRun_SimpleStringFlags()`，但 Python 源代码是从 *fp* 读取而不是一个内存中的字符串。 *filename* 应为文件名，它将使用 filesystem encoding and error handler 来解码。 如果 *closeit* 为真值，则文件将在 `PyRun_SimpleFileExFlags()` 返回之前被关闭。

> **备注:** 在 Windows 上，*fp* 应当以二进制模式打开 (即 `fopen(filename, "rb")`)。 否则，Python 可能无法正确地处理使用 LF 行结束符的脚本文件。

`int` **PyRun_InteractiveOne**(`FILE *fp, const char *filename`)
这是针对下面 `PyRun_InteractiveOneFlags()` 的简化版接口，将 *flags* 设为 `NULL`。

`int` **PyRun_InteractiveOneFlags**(`FILE *fp, const char *filename, PyCompilerFlags *flags`)
根据 *flags* 参数读取并执行来自与交互设备相关联的文件的一条语句。 用户将得到使用 `sys.ps1` 和 `sys.ps2` 的提示。 *filename* 将使用 filesystem encoding and error handler 来解码。

当输入被成功执行时返回 `0`，如果引发异常则返回 `-1`，或者如果存在解析错误则返回来自作为 Python 的组成部分发布的 `errcode.h` 包括文件的错误代码。 (请注意 `errcode.h` 并未被 `Python.h` 所包括，因此如果需要则必须专门地包括。)

`int` **PyRun_InteractiveLoop**(`FILE *fp, const char *filename`)
这是针对下面 `PyRun_InteractiveLoopFlags()` 的简化版接口，将 *flags* 设为 `NULL`。

`int` **PyRun_InteractiveLoopFlags**(`FILE *fp, const char *filename, PyCompilerFlags *flags`)
读取并执行来自与交互设备相关联的语句直至到达 EOF。 用户将得到使用 `sys.ps1` 和 `sys.ps2` 的提示。 *filename* 将使用 filesystem encoding and error handler 来解码。 当位于 EOF 时将返回 `0`，或者当失败时将返回一个负数。

`int` (*__PyOS_InputHook__)(`void`)
属于 稳定 ABI.

可以被设为指向一个原型为 `int func(void)` 的函数。 该函数将在Python 的解释器提示符即将空闲并等待用户从终端输入时被调用。 返回值会被忽略。 重写这个钩子可被用来将解释器的提示符集成到其他事件循环中，就像 Python 码中 `Modules/_tkinter.c` 所做的那样。

> *在 3.12 版本发生变更:* 此函数只能被 主解释器 调用。

char *(*PyOS_ReadlineFunctionPointer)(FILE*, FILE*, const char*)

可以被设为指向一个原型为 `char *func(FILE *stdin, FILE *stdout, char *prompt)` 的函数，重写被用来读取解释器提示符的一行输入的默认函数。 该函数被预期为如果字符串 *prompt* 不为 `NULL` 就输出它，然后从所提供的标准输入文件读取一行输入，并返回结果字符串。 例如，`readline` 模块将这个钩子设置为提供行编辑和 tab 键补全等功能。

结果必须是一个由 `PyMem_RawMalloc()` 或 `PyMem_RawRealloc()` 分配的字符串，或者如果发生错误则为 `NULL`。

> *在 3.4 版本发生变更:* 结果必须由 `PyMem_RawMalloc()` 或 `PyMem_RawRealloc()` 分配，而不是由 `PyMem_Malloc()` 或 `PyMem_Realloc()` 分配。

> *在 3.12 版本发生变更:* 此函数只能被 主解释器 调用。

PyObject *PyRun_String(const char *str, int start, PyObject *globals, PyObject *locals)

返回值：新的引用。

这是针对下面 `PyRun_StringFlags()` 的简化版接口，将 *flags* 设为 `NULL`。

PyObject *PyRun_StringFlags(const char *str, int start, PyObject *globals, PyObject *locals, PyCompilerFlags *flags)

返回值：新的引用。

在由对象 *globals* 和 *locals* 指定的上下文中执行来自 *str* 的 Python 源代码并使用以 *flags* 指定的编译器旗标。 *globals* 必须是一个字典; *locals* 可以是任何实现了映射协议的对象。 形参 *start* 指定的起始符号必须是以下几种之一: `Py_eval_input`, `Py_file_input`, or `Py_single_input`。

返回将代码作为 Python 对象执行的结果，或者如果引发了异常则返回 `NULL`。

PyObject *PyRun_File(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals)

返回值：新的引用。

这是针对下面 `PyRun_FileExFlags()` 的简化版接口，将 *closeit* 设为 `0` 并将 *flags* 设为 `NULL`。

PyObject *PyRun_FileEx(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals, int closeit)

返回值：新的引用。

这是针对下面 `PyRun_FileExFlags()` 的简化版接口，将 *flags* 设为 `NULL`。

PyObject *PyRun_FileFlags(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals, PyCompilerFlags *flags)

返回值：新的引用。

这是针对下面 `PyRun_FileExFlags()` 的简化版接口，将 *closeit* 设为 `0`。

PyObject *PyRun_FileExFlags(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals, int closeit, PyCompilerFlags *flags)

返回值：新的引用。

类似于 PyRun_StringFlags()，但 Python 源代码是从 *fp* 读取而不是一个内存中的字符串。 *filename* 应为文件名，它将使用 filesystem encoding and error handler 来解码。 如果 *closeit* 为真值，则文件将在 PyRun_FileExFlags() 返回之前被关闭。

PyObject *Py_CompileString(const char *str, const char *filename, int start)

返回值：新的引用。 属于 稳定 ABI.

这是针对下面 Py_CompileStringFlags() 的简化版接口，将 *flags* 设为 NULL。

PyObject *Py_CompileStringFlags(const char *str, const char *filename, int start, PyCompilerFlags *flags)

返回值：新的引用。

这是针对下面 Py_CompileStringExFlags() 的简化版接口，将 *optimize* 设为 -1。

PyObject *Py_CompileStringObject(const char *str, PyObject *filename, int start, PyCompilerFlags *flags, int optimize)

返回值：新的引用。

解析并编译 *str* 中的 Python 源代码，返回结果代码对象。 起始符号由 *start* 给出；这可被用来约束能被编译的代码并且应当为 Py_eval_input, Py_file_input 或 Py_single_input。 由 *filename* 指定的文件名会被用来构造代码对象并可能出现在回溯信息或 SyntaxError 异常消息中。 如果代码无法被解析或编译则此函数将返回 NULL。

整数 *optimize* 指定编译器的优化级别；值 -1 将选择与 -O 选项相同的解释器优化级别。 显式级别为 0 (无优化；__debug__ 为真值)、1 (断言被移除，__debug__ 为假值) 或 2 (文档字符串也被移除)。

> *Added in version 3.4.*

PyObject *Py_CompileStringExFlags(const char *str, const char *filename, int start, PyCompilerFlags *flags, int optimize)

返回值：新的引用。

与 Py_CompileStringObject() 类似，但 *filename* 是以 filesystem encoding and error handler 解码出的字节串。

> *Added in version 3.2.*

PyObject *PyEval_EvalCode(PyObject *co, PyObject *globals, PyObject *locals)

返回值：新的引用。 属于 稳定 ABI.

这是针对 PyEval_EvalCodeEx() 的简化版接口，只附带代码对象，以及全局和局部变量。 其他参数均设为 NULL。

PyObject *PyEval_EvalCodeEx(PyObject *co, PyObject *globals, PyObject *locals, PyObject *const *args, int argcount, PyObject *const *kws, int kwcount, PyObject *const *defs, int defcount, PyObject *kwdefs, PyObject *closure)

返回值：新的引用。 属于 稳定 ABI.

对一个预编译的代码对象求值，为其求值给出特定的环境。 此环境由全局变量的字典，局部变量映射对象，参数、关键字和默认值的数组，仅限关键字 参数的默认值的字典和单元的封闭元组构成。

PyObject *PyEval_EvalFrame(PyFrameObject *f)

返回值：新的引用。 属于 稳定 ABI.

对一个执行帧求值。 这是针对 PyEval_EvalFrameEx() 的简化版接口，用于保持向下兼容性。

PyObject *PyEval_EvalFrameEx(PyFrameObject *f, int throwflag)

返回值：新的引用。 属于 稳定 ABI.

这是 Python 解释运行不带修饰的主函数。 与执行帧 *f* 相关联的代码对象将被执行，解释字节码并根据需要执行调用。 额外的 *throwflag* 形参基本可以被忽略 —— 如果为真值，则会导致立即抛出一个异常；这会被用于生成器对象的 throw() 方法。

> *在 3.4 版本发生变更:* 该函数现在包含一个调试断言，用以确保不会静默地丢弃活动的异常。

int PyEval_MergeCompilerFlags(PyCompilerFlags *cf)

此函数会修改当前求值帧的旗标，并在成功时返回真值，失败时返回假值。

int Py_eval_input

Python 语法中用于孤立表达式的起始符号；配合 Py_CompileString() 使用。

int Py_file_input

Python 语法中用于从文件或其他源读取语句序列的起始符号；配合 Py_CompileString() 使用。 这是在编译任意长的 Python 源代码时要使用的符号。

int Py_single_input

Python 语法中用于单独语句的起始符号；配合 Py_CompileString() 使用。 这是用于交互式解释器循环的符号。

struct PyCompilerFlags

这是用来存放编译器旗标的结构体。 对于代码仅被编译的情况，它将作为 `int flags` 传入，而对于代码要被执行的情况，它将作为 `PyCompilerFlags *flags` 传入。 在这种情况下，`from __future__ import` 可以修改 *flags*。

只要 `PyCompilerFlags *flags` 是 `NULL`，cf_flags 就会被视为等同于 `0`，而由于 `from __future__ import` 而产生的任何修改都会被丢弃。

int cf_flags

编译器旗标。

`int` **cf_feature_version**

cf_feature_version 是 Python 的小版本号。 它应当被初始化为 `PY_MINOR_VERSION`。

该字段默认会被忽略，当且仅当在 `cf_flags` 中设置了 `PyCF_ONLY_AST` 旗标时它才会被使用。

> *在 3.8 版本发生变更:* 增加了 *cf_feature_version* 字段。

现有的编译器旗标可作为宏来使用：

PyCF_ALLOW_TOP_LEVEL_AWAIT

PyCF_ONLY_AST

PyCF_OPTIMIZED_AST

PyCF_TYPE_COMMENTS

请参阅 `ast` Python 模块文档中的 编译器旗标，它们会将这些常量以相同的名称导出。

上述 "`PyCF`" 标志可与 "`CO_FUTURE`" 类标志如 `CO_FUTURE_ANNOTATIONS` 组合使用，以启用通常通过 future 语句 选择的功能特性。 完整标志列表请参见 代码对象标志。