

---

# What's New in Python

发行版本 3.13.3

A. M. Kuchling

四月 21, 2025

Python Software Foundation  
Email: [docs@python.org](mailto:docs@python.org)

## Contents

<b>1</b>	<b>摘要 -- 发布重点</b>	<b>3</b>
<b>2</b>	<b>新的特性</b>	<b>5</b>
2.1	更好的交互式解释器	5
2.2	改进的错误消息	5
2.3	自由线程的 CPython	6
2.4	实验性的即时 (JIT) 编译器	6
2.5	针对 <code>locals()</code> 的已定义修改语义	7
2.6	对移动平台的支持	8
<b>3</b>	<b>其他语言特性修改</b>	<b>8</b>
<b>4</b>	<b>新增模块</b>	<b>9</b>
<b>5</b>	<b>改进的模块</b>	<b>9</b>
5.1	<code>argparse</code>	9
5.2	<code>array</code>	9
5.3	<code>ast</code>	9
5.4	<code>asyncio</code>	10
5.5	<code>base64</code>	10
5.6	<code>compileall</code>	10
5.7	<code>concurrent.futures</code>	11
5.8	<code>configparser</code>	11
5.9	<code>copy</code>	11
5.10	<code>ctypes</code>	11
5.11	<code>dbm</code>	11
5.12	<code>dis</code>	11
5.13	<code>doctest</code>	12
5.14	<code>email</code>	12
5.15	<code>enum</code>	12
5.16	<code>fractions</code>	12
5.17	<code>glob</code>	12
5.18	<code>importlib</code>	12
5.19	<code>io</code>	12
5.20	<code>ipaddress</code>	13
5.21	<code>itertools</code>	13
5.22	<code>marshal</code>	13
5.23	<code>math</code>	13

5.24	mimetypes	13
5.25	mmap	13
5.26	multiprocessing	13
5.27	os	13
5.28	os.path	14
5.29	pathlib	14
5.30	pdb	14
5.31	queue	15
5.32	random	15
5.33	re	15
5.34	shutil	15
5.35	site	15
5.36	sqlite3	15
5.37	ssl	15
5.38	statistics	15
5.39	subprocess	16
5.40	sys	16
5.41	tempfile	16
5.42	time	16
5.43	tkinter	16
5.44	回溯	17
5.45	types	17
5.46	typing	17
5.47	unicodedata	17
5.48	venv	17
5.49	warnings	17
5.50	xml	17
5.51	zipimport	18
<b>6</b>	<b>性能优化</b>	<b>18</b>
<b>7</b>	<b>被移除的模块与 API</b>	<b>18</b>
7.1	PEP 594: 从标准库中移除“死电池”	18
7.2	2to3	19
7.3	builtins	20
7.4	configparser	20
7.5	importlib.metadata	20
7.6	locale	20
7.7	opcode	20
7.8	optparse	20
7.9	pathlib	20
7.10	re	20
7.11	tkinter.tix	20
7.12	turtle	21
7.13	typing	21
7.14	unittest	21
7.15	urllib	21
7.16	webbrowser	21
<b>8</b>	<b>新的弃用</b>	<b>21</b>
8.1	计划在 Python 3.14 中移除	23
8.2	Python 3.15 中的待移除功能	25
8.3	计划在 Python 3.16 中移除	26
8.4	计划在未来版本中移除	26
<b>9</b>	<b>CPython 字节码的变化</b>	<b>28</b>
<b>10</b>	<b>C API 的变化</b>	<b>28</b>
10.1	新的特性	28

10.2 被改变的 C API .....	31
10.3 受限 C API 的改变 .....	32
10.4 被移除的 C API .....	32
10.5 已弃用的 C API .....	34
<b>11 构建变化</b> .....	<b>36</b>
<b>12 移植到 Python 3.13</b> .....	<b>37</b>
12.1 Python API 的变化 .....	37
12.2 C API 的变化 .....	37
<b>13 回归测试的变化</b> .....	<b>39</b>
<b>14 3.13.1 中的重要变化</b> .....	<b>39</b>
14.1 sys .....	39
<b>索引</b> .....	<b>40</b>

## 编者

Adam Turner 和 Thomas Wouters

本文介绍了 Python 3.13 相比 3.12 增加的新特性。Python 3.13 已于 2024 年 10 月 7 日发布。要获取详细信息，可参阅 [更新日志](#)。

## 参见

[PEP 719](#) -- Python 3.13 发布计划

## 1 摘要 -- 发布重点

Python 3.13 是 Python 编程语言的最新稳定发布版，包含多项针对语言、实现和标准库的改变。最大的变化包括一个新的交互式解释器，以及对于在自由线程模式 ([PEP 703](#)) 下运行和即时编译器 ([PEP 744](#)) 的实验性支持。

错误消息继续得到改进，回溯信息现在默认使用彩色高亮显示。`locals()` 内置函数现在对于修改所返回的映射具有更细化的语法，并且类型形参现在支持设置默认值。

针对标准库的改变包括移除已弃用的 API 和模块，以及用户友好度和正确性方面的常规提升。一些旧式标准库模块自 Python 3.11 起被弃用 ([PEP 594](#)) 之后现在已被移除。

本文并不试图提供所有新特性的完整规范说明，而是提供一个方便的概览。要了解完整细节请参阅相应文档，如 [标准库参数](#) 和 [语言参考](#)。要了解某项改变的完整实现和设计理念，请参阅相应新特性的 PEP；但请注意一旦某项特性已完全实现则相应 PEP 通常不会再继续更新。请参阅 [迁移到 Python 3.13](#) 了解如何从较早 Python 进行升级的指导。

解释器的改进：

- 大幅改进的交互式解释器和改进的错误消息。
- [PEP 667](#): 现在 `locals()` 内置函数在修改被返回的映射时具有已定义语义。Python 调试器及类似的工具现在即使在并发代码执行期间也能更可靠地在已优化的作用域中更新局部变量。
- [PEP 703](#): CPython 3.13 具有对在运行时禁用 global interpreter lock 的实验性支持。请参阅 [自由线程 CPython](#) 了解详情。
- [PEP 744](#): 增加了一个基本的 JIT 编译器。目前默认是禁用的（但以后可能启用）。能够小幅提升性能 -- 我们预计在接下来的几个发布版中不断改进它。

- 在新的交互式解释器中,以及回溯信息和文档测试输出中的颜色支持。这可以通过 `PYTHON_COLORS` and `NO_COLOR` 环境变量来禁用。

对 Python 数据模型的改进:

- `__static_attributes__` 保存了可在一个类体的任何函数中通过 `self.x` 来访问的属性名称。
- `__firstlineno__` 记录了一个类定义的首行的行号。

标准库中的重大改进:

- 新增了 `PythonFinalizationError` 异常,当操作在最终化期间被阻塞时将被引发。
- 现在 `argparse` 模块可支持弃用命令行选项、位置参数和子命令。
- 新增的函数 `base64.z85encode()` 和 `base64.z85decode()` 支持对 **Z85 数据** 进行编码和解码。
- 现在 `copy` 模块有一个 `copy.replace()` 函数,支持许多内置类型和任何定义了 `__replace__()` 方法的类。
- 新的 `dbm.sqlite3` 模块现在是默认的 `dbm` 后端。
- `os` 模块增加了一套新函数用于处理 Linux 的定时器通知文件描述符。
- 现在 `random` 模块提供了一个命令行界面。

安全改进:

- `ssl.create_default_context()` 设置了 `ssl.VERIFY_X509_PARTIAL_CHAIN` 和 `ssl.VERIFY_X509_STRICT` 作为默认的旗标。

C API 的改进:

- 现在 `Py_mod_gil` 槽位被用来指明一个扩展模块支持在禁用 GIL 的情况下运行。
- 增加了 `PyTime` C API,提供了对系统时钟的访问。
- `PyMutex` 是新增的轻量级互斥锁,只占用一个字节。
- 新增了一套函数用于在 C API 中生成 **PEP 669** 监控事件。

新的类型标注特性:

- **PEP 696**: 类型形参 (`typing.TypeVar`, `typing.ParamSpec` 和 `typing.TypeVarTuple`) 现在可支持默认值。
- **PEP 702**: 新的 `warnings.deprecated()` 装饰器在类型系统和运行时中增加了对标记为弃用的支持。
- **PEP 705**: `typing.ReadOnly` 可被用来将 `typing.TypedDict` 的项标记为对类型检查器只读。
- **PEP 742**: `typing.TypeIs` 提供了更直观的类型细化行为,作为对 `typing.TypeGuard` 的替代。

平台支持:

- **PEP 730**: 现在 Apple 的 iOS 是官方支持的平台,处于 **第 3 层级**。
- **PEP 738**: 现在 Android 是官方支持的平台,处于 **第 3 层级**。
- 现在 `wasm32-wasi` 作为 **第 2 层级** 的平台受到支持。
- `wasm32-emscripten` 不再是受到官方支持的平台。

重要的移除:

- **PEP 594**: 剩余的 19 个“死电池”(老旧 `stdlib` 模块)已从标准库中移除: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `nntplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` 和 `xdrlib`。
- 移除了 **2to3** 工具和 `lib2to3` 模块(在 Python 3.11 中已被弃用)。
- 移除了 `tkinter.tix` 模块(在 Python 3.6 中已被弃用)。
- 移除了 `locale.resetlocale()` 函数。

- 移除了 `typing.io` 和 `typing.re` 命名空间。
- 移除了链式的 `classmethod` 描述器。

发布计划的变化：

**PEP 602** (“Annual Release Cycle for Python”) 已被更新为将新发布版的完整支持 (‘bugfix’) 期扩展至两年。这个更新的政策意味着：

- Python 3.9--3.12 有一年半的完整支持，另加三年半的安全修正。
- Python 3.13 及以后的版本有两年的完整支持，另加三年的安全修正。

## 2 新的特性

### 2.1 更好的交互式解释器

Python 现在默认会使用新的 `interactive shell`，它基于来自 **PyPy** 项目的代码。当使用从交互式终端启动 `REPL` 时，下列新特性将受到支持：

- 多行编辑并保留历史记录。
- 对 `REPL` 专属的命令如 `help`, `exit` 和 `quit` 的直接支持，无需以函数形式调用它们。
- 提示和回溯 默认启用彩色显示。
- 使用 `F1` 浏览交互式帮助并带有单独的命令历史。
- 使用 `F2` 浏览去除了输出以及 `>>>` 和 `...` 提示符的历史。
- 使用 `F3` 进入“粘贴模式”以更方便地粘贴大段代码（再次按 `F3` 返回常规提示符）。

要禁用新的交互式 `shell`，可设置 `PYTHON_BASIC_REPL` 环境变量。有关交互模式的详情，请参见 `tut-interac`。

（由 Pablo Galindo Salgado, Łukasz Langa 和 Lysandros Nikolaou 在 [gh-111201](#) 基于来自 PyPy 项目的代码贡献。Windows 支持由 Dino Viehland 和 Anthony Shaw 贡献。）

### 2.2 改进的错误消息

- 在终端里显示回溯时解释器现在会默认使用彩色。此特性可通过新的 `PYTHON_COLORS` 环境变量以及传统的 `NO_COLOR` 和 `FORCE_COLOR` 环境变量来进行控制。（由 Pablo Galindo Salgado 在 [gh-112730](#) 中贡献。）
- 一个常见错误是撰写的脚本和标准库中的某个模块重名。现在出现此类错误时会显示一条更有用的错误信息：

```
$ python random.py
Traceback (most recent call last):
  File "/home/me/random.py", line 1, in <module>
    import random
  File "/home/me/random.py", line 3, in <module>
    print(random.randint(5))
    ^^^^^^^^^^^^^^^^^
AttributeError: module 'random' has no attribute 'randint' (consider renaming '/home/me/
↳random.py' since it has the same name as the standard library module named 'random'
↳and prevents importing that standard library module)
```

类似地，如果一个脚本具有与它尝试导入的第三方模块相同的名称并因此导致错误，我们也会显示一条更有帮助的错误消息：

```
$ python numpy.py
Traceback (most recent call last):
  File "/home/me/numpy.py", line 1, in <module>
    import numpy as np
  File "/home/me/numpy.py", line 3, in <module>
```

(续下页)

(接上页)

```
np.array([1, 2, 3])
^^^^^^
AttributeError: module 'numpy' has no attribute 'array' (consider renaming '/home/me/
↳numpy.py' if it has the same name as a library you intended to import)
```

(由 Shantanu Jain 在 [gh-95754](#) 中贡献)。

- 现在当向一个函数传入不正确的关键字参数时错误消息会尝试提示正确的关键字参数。

```
>>> "Better error messages!".split(max_split=1)
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    "Better error messages!".split(max_split=1)
    ~~~~~^~~~~~
TypeError: split() got an unexpected keyword argument 'max_split'. Did you mean
↳'maxsplit'?
```

(由 Pablo Galindo Salgado 和 Shantanu Jain 在 [gh-107944](#) 中贡献。)

## 2.3 自由线程的 CPython

现在 CPython 具有对运行于禁用 global interpreter lock (GIL) 的自由线程模式的实验性支持。这是一个实验性的特性因而默认是不启用的。自由线程模式需要一个不同的可执行程序，通常名为 `python3.13t` 或 `python3.13t.exe`。标记为 *free-threaded* 的预构建二进制文件可作为官方 Windows 和 macOS 安装器的一部分被安装，或者可以附带 `--disable-gil` 选项使用源代码来构建 CPython。

自由线程模式的执行允许在可用的 CPU 核心上并行地运行线程从而充分利用可用的处理能力。虽然并非所有软件都能自动从中受益，但在设计时将线程纳入考虑的程序在多核心硬件上运行速度会更快。**自由线程模式是实验性的并且处于不断改进的过程中**：预计会出现一些程序错误并且在单线程场景下出现明显的性能损失。可以选择使用环境变量 `PYTHON_GIL` 或命令行选项 `-X gil=1` 让 CPython 的自由线程构建版支持在运行时启用 GIL。

要判断当前解释器是否支持自由线程，可检查 `python -VV` 和 `sys.version` 是否包含“experimental free-threading build”。新的 `sys._is_gil_enabled()` 函数可用于检查在运行进程中 GIL 是否确实被关闭。

C-API 扩展模块需要针对自由线程构建版专门进行构建。支持在禁用 GIL 的情况下运行的扩展应当使用 `Py_mod_gil` 槽位。使用单阶段初始化的扩展应当使用 `PyUnstable_Module_SetGIL()` 来指明它们是支持在禁用 GIL 的情况下运行。导入不使用这些机制的 C 扩展将导致 GIL 被启用，除非通过 `PYTHON_GIL` 环境变量或 `-X gil=0` 选项显式地禁用 GIL。需要 `pip 24.1` 或更新的版本才能在自由线程构建版中安装带有 C 扩展的软件包。

这项工作成为可能要感谢许多个人和组织，包括针对 Python 和第三方项目测试并启用自由线程支持的庞大的贡献者社区。重要的贡献者包括：Sam Gross, Ken Jin, Donghee Na, Itamar Oren, Matt Page, Brett Simmers, Dino Viehland, Carl Meyer, Nathan Goldbaum, Ralf Gommers, Lysandros Nikolaou 及其他许多人。有许多贡献者受雇于 Meta，该公司提供了大量的工程资源来支持此项目。

### ➡ 参见

**PEP 703** “Making the Global Interpreter Lock Optional in CPython” 包含了有关此项工作的理念和信息。  
[Porting Extension Modules to Support Free-Threading](#): 一份由社区维护的针对扩展开发者的移植指南。

## 2.4 实验性的即时 (JIT) 编译器

当 CPython 使用 `--enable-experimental-jit` 选项进行配置和构建时，会添加一个即时 (JIT) 编译器以加快某些 Python 程序的运行速度。在 Windows 上，可使用 `PCbuild/build.bat --experimental-jit` 启用 JIT 或使用 `--experimental-jit-interpreter` 启用第 2 层级解释器。构建要求和进一步的支持信息包含在 `Tools/jit/README.md` 中。



`--enable-experimental-jit` 选项接受这些（可选）值，如果不带可选值地预设 `--enable-experimental-jit` 则默认为 `yes`。

- `no`: 禁用整个第 2 层级和 JIT 管线。
- `yes`: 启用 JIT。要在运行时禁用 JIT，则传入环境变量 `PYTHON_JIT=0`。
- `yes-off`: 构建 JIT 但默认禁用它。要在运行时启用 JIT，则传入环境变量 `PYTHON_JIT=1`。
- `interpreter`: 启用第 2 层级解释器但是禁用 JIT。可以在运行时传入 `PYTHON_JIT=0` 来禁用该解释器。

其内部架构大致如下：

- 我们将从特化的第 1 层级字节码开始。请参阅 3.11 有什么新变化了解详情。
- 当第 1 层级字节码达到足够热度，它将被翻译为新的纯内部的中间表示形式 (IR)，称为第 2 层级 IR，有时也称为微操作码 (“uops”)。
- 第 2 层级 IR 使用与第 1 层级相同的基于栈的虚拟机，但其指令格式更适合被翻译为机器码。
- 在第 2 层级 IR 被解释或翻译为机器码之前，我们会预先应用一些优化通路。
- 虽然第 2 层级解释器存在，但它主要用于对优化管线的先前阶段进行调试。可通过为 Python 配置 `--enable-experimental-jit=interpreter` 选项启用第 2 层级解释器。
- 启用 JIT 时，经优化的第 2 层级 IR 将被翻译为机器码后再执行。
- 这个机器码翻译过程使用了名为 拷贝并打补丁的技巧。它没有运行时依赖，但增加了构建时对 LLVM 的依赖。

➡ 参见

PEP 744

(JIT 来自 Brandt Bucher 且受到 Haoran Xu 和 Fredrik Kjolstad 论文的启发。第 2 层级 IR 来自 Mark Shannon 和 Guido van Rossum。第 2 层级解释器来自 Ken Jin。)

## 2.5 针对 `locals()` 的已定义修改语义

在历史上，改变 `locals()` 的返回值的预期结果是留给具体的 Python 实现来定义的。从 Python 3.13 开始，**PEP 667** 标准化了 CPython 对于大多数代码执行作用域的历史行为，但也将已优化作用域 (函数、生成器、协程、推导式和生成器表达式) 修改为显式地返回当前已赋值的局部变量的独立快照，包括局部引用的在闭包中捕获的非局部变量。

在已优化作用域中对 `locals()` 语义的这项修改也会影响隐式地以 `locals()` 为目标的代码执行函数的默认行为，如果没有提供显式命名空间的话 (例如 `exec()` 和 `eval()` 等)。在之前的版本中，在调用代码执行函数后是否可以通过调用 `locals()` 访问更改情况取决于具体的实现。具体到 CPython 而言，此类代码通常会按预期工作，但有时可能会在基于其他代码 (包括调试器和代码执行跟踪工具) 的已优化作用域中失败，因为代码有可能重置该作用域中的共享快照。现在，代码在已优化作用域中将始终针对局部变量的独立快照运行，因为在后续调用 `locals()` 时将永远看不到更改。要访问在这些情况下所做的更改，现在必须将一个显式命名空间引用传递给相关的函数。或者，也可以更新受影响的代码以使用更高层级的代码执行 API 返回结果代码命名空间 (例如，当执行磁盘上的 Python 文件时使用 `runpy.run_path()` 函数)。

为确保调试器和类似工具能可靠地更新受到此变化影响的作用域中的局部变量，现在 `FrameType.f_locals` 将返回一个针对此种作用域中的帧的局部变量和在局部引用的非局部变量的直通写入代理对象，而不是返回一个非持续更新的具有规定的运行时语义的共享 dict 实例。

请参阅 **PEP 667** 了解详情，包括相关的 C API 更改和弃用。下文还针对受影响的 *Python API* 和 *C API* 提供了移植说明。

(PEP 和实现由 Mark Shannon 和 Tian Gao 在 [gh-74929](#) 中贡献。文档更新由 Guido van Rossum 和 Alyssa Coghlan 提供。)

## 2.6 对移动平台的支持

**PEP 730:** iOS 现在是 **PEP 11** 所支持的平台，包括第 3 层级的 `arm64-apple-ios` 和 `arm64-apple-ios-simulator` 等目标（分别为 2013 年后的 iPhone 和 iPad 设备以及运行于 Apple silicon 硬件的 Xcode iOS 模拟器）。`x86_64-apple-ios-simulator`（运行于较旧的 x86\_64 硬件的 Xcode iOS 模拟器）不是第 3 层级的受支持平台，但也将尽可能地支持。（PEP 撰写及实现由 Russell Keith-Magee 在 [gh-114099](#) 中贡献。）

**PEP 738:** Android 现在是 **PEP 11** 所支持的平台，包括位于第 3 层级的 `aarch64-linux-android` 和 `x86_64-linux-android` 等目标。32 位的目标 `arm-linux-androideabi` 和 `i686-linux-android` 不是第 3 层级的受支持平台，但也将尽可能地支持。（PEP 撰写及实现由 Malcolm Smith 在 [gh-116622](#) 中贡献。）

### 参见

**PEP 730, PEP 738**

## 3 其他语言特性修改

- 编译器现在将从文档字符串的每一行去除共有的前导空格。这会减少字节码缓存的大小（例如 `.pyc` 文件），例如在 SQLAlchemy 2.0 的 `sqlalchemy.orm.session` 中文件大小将减少约 5%。这项改变将影响各种使用了文档字符串的工具，如 `doctest`。

```
>>> def spam():
...     """
...         This is a docstring with
...         leading whitespace.
...
...         It even has multiple paragraphs!
...     """
...
>>> spam.__doc__
'\nThis is a docstring with\n leading whitespace.\n\nIt even has multiple paragraphs!\n'
```

（由 Inada Naoki 在 [gh-81283](#) 中贡献。）

- 类作用域内的标注作用域现在可以包含 `lambda` 和推导式。位于类作用域内的推导式不会内联到其父作用域中。

```
class C[T]:
    type Alias = lambda: T
```

（由 Jelle Zijlstra 在 [gh-109118](#) 和 [gh-118160](#) 中贡献。）

- `future` 语句不再会被 `__future__` 模块的相对导入触发，意味着 `from __future__ import ...` 形式的语句现在只是标准的相对导入，而不会激活任何特殊特性。（由 Jeremiah Gabriel Pascual 在 [gh-118216](#) 中贡献。）
- 现在 `global` 声明当其被用于 `else` 代码块中时也将被允许在 `except` 代码块中使用。在之前版本中这会错误地引发 `SyntaxError`。（由 Irit Katriel 在 [gh-111123](#) 中贡献。）
- 增加了新的环境变量 `PYTHON_FROZEN_MODULES`，它确定冻结模块是否会被导入机制所忽略，等价于 `-X frozen_modules` 命令行选项。（由 Yilei Yang 在 [gh-111374](#) 中贡献。）
- 通过新的环境变量 `PYTHON_PERF_JIT_SUPPORT` 和命令行选项 `-X perf_jit` 添加无需 帧指针 即可工作的对 `perf` 性能分析器的支持。（由 Pablo Galindo 在 [gh-118518](#) 中贡献。）
- 可通过新的 `PYTHON_HISTORY` 环境变量来更改 `.python_history` 文件的位置。（由 Levi Sabah, Zackery Spytz 和 Hugo van Kemenade 在 [gh-73965](#) 中贡献。）
- 类新增了一个 `__static_attributes__` 属性。这由编译器以类属性名称的元组来填充，这些名称是从类体中的任何函数通过 `self.<name>` 来赋值的。（由 Irit Katriel 在 [gh-115775](#) 中贡献。）



- 编译器现在会在类上创建一个 `__firstlineno__` 属性，其值为类定义第一行的行号。（由 Serhiy Storchaka 在 [gh-118465](#) 中贡献。）
- 现在 `exec()` 和 `eval()` 内置函数接受以关键字形式传入的 `globals` 和 `locals` 参数。（由 Raphael Gaschignard 在 [gh-105879](#) 中贡献。）
- 现在 `compile()` 内置函数接受一个新的旗标 `ast.PyCF_OPTIMIZED_AST`，它类似于 `ast.PyCF_ONLY_AST` 但区别在于返回的 AST 是根据 `optimize` 参数的值进行优化的。（由 Irit Katriel 在 [gh-108113](#) 中贡献。）
- 在 `property` 对象上增加了 `__name__` 属性。（由 Eugene Toder 在 [gh-101860](#) 中贡献。）
- 增加了新的异常 `PythonFinalizationError`，它派生自 `RuntimeError`，用于当操作在最终化期间被阻塞时发出信号。下列可调用对象现在将引发 `PythonFinalizationError`，而不是 `RuntimeError`：
  - `_thread.start_new_thread()`
  - `os.fork()`
  - `os.forkpty()`
  - `subprocess.Popen`
 （由 Victor Stinner 在 [gh-114570](#) 中贡献。）
- 允许 `str.replace()` 的 `count` 参数为关键字参数。（由 Hugo van Kemenade 在 [gh-106487](#) 中贡献。）
- 现在许多函数会对将布尔值作为文件描述符参数发出警告。这可以帮助尽早发现一些错误。（由 Serhiy Storchaka 在 [gh-82626](#) 中贡献。）
- 为 `bz2`, `lzma`, `tarfile` 和 `zipfile` 等模块中的已压缩和已归档文件型对象添加了 `name` 和 `mode` 属性。（由 Serhiy Storchaka 在 [gh-115961](#) 中贡献。）

## 4 新增模块

- `dbm.sqlite3`: 针对 `dbm` 的 SQLite 后端。（由 Raymond Hettinger 和 Erlend E. Aasland 在 [gh-100414](#) 中贡献。）

## 5 改进的模块

### 5.1 argparse

- 为 `add_argument()` 和 `add_parser()` 方法添加了 *deprecated* 形参，以允许弃用命令行选项、位置参数和子命令。（由 Serhiy Storchaka 在 [gh-83648](#) 中贡献。）

### 5.2 array

- 增加了 `'w'` 类型码 (`Py_UCS4`) 表示 Unicode 字符。它应被用来代替已弃用的 `'u'` 类型码。（由 Inada Naoki 在 [gh-80480](#) 中贡献。）
- 通过实现 `clear()` 方法将 `array.array` 注册为 `MutableSequence`。（由 Mike Zimin 在 [gh-114894](#) 中贡献。）

### 5.3 ast

- 现在 `ast` 模块中节点类型的构造器对其接受的参数要求更为严格，并在参数被省略时有更易理解的行为。

如果在构造实例时某个 AST 节点上的可选字段没有被作为参数包括在内，则该字段现在将被设为 `None`。类似地，如果某个列表字段被省略，则该字段现在将被设为空列表，而如果某个 `expr_context` 字段被省略，则它将默认为 `Load()`。（之前，在所有情况下，新构造的 AST 节点实例上的相应属性都将缺失。）

在所有其他情况下，当需要的参数被省略时，节点构造器将发出 `DeprecationWarning`。这在 Python 3.15 中将会引发异常。类似地，将关键字参数传入一个未映射到 AST 节点上的字段的构造器的做法现在已被弃用，并且在 Python 3.15 中将会引发异常。

这些更改将不会应用于用户自定义的 `ast.AST` 子类，除非该类选择通过设置 `AST._field_types` 映射的方式加入新的行为。

(由 Jelle Zijlstra 在 [gh-105858](#), [gh-117486](#) 和 [gh-118851](#) 中贡献。)

- 现在 `ast.parse()` 接受一个可选参数 `optimize`，它会被传递给 `compile()`。这使得获取已优化的 AST 成为可能。(由 Irit Katriel 在 [gh-108113](#) 中贡献。)

## 5.4 asyncio

- 现在 `asyncio.as_completed()` 将返回一个即是 `asynchronous iterator` 又是基本的产生可等待对象的 `iterator` 的对象。由异常迭代产生的可等待对象包括被传入的原始 `Task` 或 `Future` 对象，使得将结果与正在完成的任务相关联更为容易。(由 Justin Arthur 在 [gh-77714](#) 中贡献。)
- 现在当服务器被关闭时 `asyncio.loop.create_unix_server()` 会自动移除 Unix 套接字。(由 Pierre Ossman 在 [gh-111246](#) 中贡献。)
- 现在当附带一个空字节串对象调用时 `DatagramTransport.sendto()` 将发送零长度的数据报。现在当计算缓冲区大小时传输控制流还会将数据报标头纳入考量。(由 Jamie Phan 在 [gh-115199](#) 中贡献。)
- 增加了 `Queue.shutdown` 和 `QueueShutDown` 用于管理队列终结。(由 Laurie Opperman 和 Yves Duprat 在 [gh-104228](#) 中贡献。)
- 增加了 `Server.close_clients()` 和 `Server.abort_clients()` 方法，它们会以更强制的方式关闭 `asyncio` 服务器。(由 Pierre Ossman 在 [gh-113538](#) 中贡献。)
- 在 `StreamReader.readuntil()` 中接受一个由分隔符组成的元组，当遇到其中之一时就会停止。(由 Bruce Merry 在 [gh-81322](#) 中贡献。)
- 改进了 `TaskGroup` 在外部的取消操作与内部的取消操作发生冲突时的行为。例如，当嵌套两个任务分组并且两者同时在某个子任务中遇到异常时，外层的任务分组有可能被挂起，因为其内部的取消操作已由内层的任务分组进行处理。

对于任务分组在外部被取消时同时必须引发 `ExceptionGroup` 的情况，现在它将调用父任务的 `cancel()` 方法。这样可以确保 `CancelledError` 会在下一次 `await` 时被引发，因此取消操作不会丢失。，so the cancellation is not lost.

这些更改的一个附加好处是现在任务组会保留取消操作计数 (`cancelling()`)。

为了处理某些边界情况，现在 `uncancel()` 可以在取消操作计数达到零时重置未写入文档的 `_must_cancel` 旗标。

(受到由 Arthur Tacca 在 [gh-116720](#) 中报告的问题的启发。)

- 当在一个未激活的 `TaskGroup` 上调用 `TaskGroup.create_task()` 时，给定的协程将被关闭 (这将防止引发有关给定的协程从未被等待的 `RuntimeWarning`)。 (由 Arthur Tacca 和 Jason Zhang 在 [gh-115957](#) 中贡献。)

## 5.5 base64

- 增加了 `z85encode()` 和 `z85decode()` 函数用于将 `bytes` 编码为 `Z85 data` 和将 `Z85` 编码的数据解码为 `bytes`。(由 Matan Perelman 在 [gh-75299](#) 中贡献。)

## 5.6 compileall

- 工作线程和进程的默认数据现在是使用 `os.process_cpu_count()` 而不是 `os.cpu_count()` 来选择的。(由 Victor Stinner 在 [gh-109649](#) 中贡献。)

## 5.7 concurrent.futures

- 工作线程和进程的默认数据现在使用的是 `os.process_cpu_count()` 而不是 `os.cpu_count()` 来选择的。(由 Victor Stinner 在 [gh-109649](#) 中贡献。)

## 5.8 configparser

- 现在 `ConfigParser` 具有对未命名节的支持，这将允许使用最高层级的键值对。此特性可通过新增的 `allow_unnamed_section` 形参来启用。(由 Pedro Sousa Lacerda 在 [gh-66449](#) 中贡献。)

## 5.9 copy

- 新增的 `replace()` 函数和 `replace` 协议使得创建经修改的对象副本更为简单。这在操作不可变对象时特别有用。以下类型将支持 `replace()` 函数并实现了 `replace` 协议：

- `collections.namedtuple()`
- `dataclasses.dataclass`
- `datetime.datetime`, `datetime.date`, `datetime.time`
- `inspect.Signature`, `inspect.Parameter`
- `types.SimpleNamespace`
- 代码对象

任何用户自定义类也可以通过定义 `__replace__()` 方法来支持 `copy.replace()`。(由 Serhiy Storchaka 在 [gh-108751](#) 中贡献。)

## 5.10 ctypes

- 作为必要的内部重构的一个后果，内部元类的初始化现在将发生于 `__init__` 中而不是 `__new__` 中。这会影响子类化这些内部元类以提供自定义初始化的项目。一般而言：
  - 调用 `super().__new__` 之后在 `__new__` 中完成的自定义逻辑应当移至 `__init__`。
  - 要创建一个类，需调用相应的元类，而不仅是该元类的 `__new__` 方法。

请参阅 [gh-124520](#) 了解相关讨论和对某些受影响项目的修改的链接。

- `ctypes.Structure` 对象新增了一个 `_align_` 属性以允许显式地指定发往内存的结构体对齐方式。(由 Matt Sanderson 在 [gh-112433](#) 中贡献。)

## 5.11 dbm

- 增加 `dbm.sqlite3`，一个实现了 SQLite 后端的新模块，并将其设为默认的 `dbm` 后端。(由 Raymond Hettinger 和 Erlend E. Aasland 在 [gh-100414](#) 中贡献。)
- 允许通过新增的 `gdbm.clear()` 和 `ndbm.clear()` 方法移除数据库中的所有条目。(由 Donghee Na 在 [gh-107122](#) 中贡献。)

## 5.12 dis

- 将 `dis` 模块的函数的输出修改为显示跳转目标和异常处理器的逻辑标签，而不是偏移量。可以使用新的 `-O` 命令行选项或 `show_offsets` 参数来添加偏移量。(由 Irit Katriel 在 [gh-112137](#) 中贡献。)
- `get_instructions()` 不再将缓存条目表示为单独的指令。作为替代，它会将它们作为 `Instruction` 的组成部分返回，放在新的 `cache_info` 字段中。传给 `get_instructions()` 的 `show_caches` 参数已被弃用并且不再有任何效果。(由 Irit Katriel 在 [gh-112962](#) 中贡献。)

### 5.13 doctest

- 现在 doctest 输出默认是彩色的。此特性可通过新增的 `PYTHON_COLORS` 环境变量和传统的 `NO_COLOR` 和 `FORCE_COLOR` 环境变量来控制。另请参阅 [using-on-controlling-color](#)。（由 Hugo van Kemenade 在 [gh-117225](#) 中贡献。）
- 现在 `DocTestRunner.run()` 方法会统计已跳过测试的数量。增加了 `DocTestRunner.skips` 和 `TestResults.skipped` 属性。（由 Victor Stinner 在 [gh-108794](#) 中贡献。）

### 5.14 email

- 现在带有嵌入的换行符的标头在输出时会加上引号。现在 `generator` 会拒绝序列化（写入）不正确地折叠或分隔的标头，例如将被解析为多个标头或与相邻数据合并的标头等。如果你需要禁用此安全特性，请设置 `verify_generated_headers`。（由 Bas Bloemsaat 和 Petr Viktorin 在 [gh-121650](#) 中贡献。）
- 现在 `getaddresses()` 和 `parseaddr()` 会在更多遇到无效 email 地址的情况下返回 `('', '')` 对非可能不准确的值。这两个函数新增了可选的 `strict` 形参（默认为 `True`）。要获取旧版本的行为（接受错误格式的输入），请使用 `strict=False`。`getattr(email.utils, 'supports_strict_parsing', False)` 可被用于检查 `strict` 形参是否可用。（由 Thomas Dwyer 和 Victor Stinner 针对 [gh-102988](#) 贡献以改进 [CVE 2023-27043](#) 修正。）

### 5.15 enum

- `EnumDict` 被改为公有以更好的支持子类化 `EnumType`。

### 5.16 fractions

- 现在 `Fraction` 对象支持用于填充、对齐、正负号处理、最小宽度和分组的标准格式说明迷你语言规则。（由 Mark Dickinson 在 [gh-111320](#) 中贡献。）

### 5.17 glob

- 增加了 `translate()`，这是个用来将具有 shell 风格通配符的路径说明转换为正则表达式的函数。（由 Barney Gale 在 [gh-72904](#) 中贡献。）

### 5.18 importlib

- 现在 `importlib.resources` 中的下列函数允许访问资源目录（或树），并使用多个位置参数（现在在文本读取函数中的 `encoding` 和 `errors` 参数是仅限关键字参数）：

- `is_resource()`
- `open_binary()`
- `open_text()`
- `path()`
- `read_binary()`
- `read_text()`

这些函数将不再被弃用也不会被加入移除计划。（由 Petr Viktorin 在 [gh-116608](#) 中贡献。）

- `contents()` 仍然被弃用而应改用功能完整的 `Traversable` API。不过，目前还没有移除它的计划。（由 Petr Viktorin 在 [gh-116608](#) 中贡献。）

### 5.19 io

- 现在 `IOBase` 最终化器会使用 `sys.unraisablehook` 来将由 `close()` 方法引发的错误写入日志。在之前版本中，错误在默认情况下会被静默地忽略，而只有在 Python 开发模式或在使用 Python 调试构建版时才会被写入日志。（由 Victor Stinner 在 [gh-62948](#) 中贡献。）

## 5.20 ipaddress

- 增加了 `IPv4Address.ipv6_mapped` 特征属性，它将返回映射 IPv4 的 IPv6 地址。（由 Charles Machalow 在 [gh-109466](#) 中贡献。）
- 修正了 `IPv4Address`, `IPv6Address`, `IPv4Network` 和 `IPv6Network` 中 `is_global` 和 `is_private` 的行为。（由 Jakub Stasiak 在 [gh-113171](#) 中贡献。）

## 5.21 itertools

- `batched()` 新增了 `strict` 形参，它会在最后一批数据小于指定批准大小时引发 `ValueError`。（由 Raymond Hettinger 在 [gh-113202](#) 中贡献。）

## 5.22 marshal

- 在模块函数中增加了 `allow_code` 形参。传入 `allow_code=False` 将防止在 Python 各版本间不兼容的代码对象的序列化和反序列化。（由 Serhiy Storchaka 在 [gh-113626](#) 中贡献。）

## 5.23 math

- 新增函数 `fma()` 可执行合并的乘法-加法运算。此函数只需一轮操作即可计算  $x * y + z$ ，从而避免了任何中间步骤导致的精度损失。它包装了 C99 所提供的 `fma()` 函数，并且遵从针对特殊情况的 IEEE 754 “fusedMultiplyAdd” 运算规范。（由 Mark Dickinson 和 Victor Stinner 在 [gh-73468](#) 中贡献。）

## 5.24 mimetypes

- 增加了 `guess_file_type()` 函数用于根据文件系统路径来猜测 MIME 类型。在 `guess_type()` 中使用路径的做法现在已是 `soft deprecated`。（由 Serhiy Storchaka 在 [gh-66543](#) 中贡献。）

## 5.25 mmap

- 现在 `mmap` 在 Windows 上当被映射的内存由于文件系统错误或访问限制而不可访问时将获得保护以避免崩溃。（由 Jannis Weigend 在 [gh-118209](#) 中贡献。）
- `mmap` 具有新的 `seekable()` 方法将在需要可定位的文件型对象时被使用。现在 `seek()` 方法将返回一个新的绝对位置。（由 Donghee Na 和 Sylvie Liberman 在 [gh-111835](#) 中贡献。）
- `mmap` 新增了 UNIX 专属的 `trackfd` 形参用来控制文件描述符的复制；如为假值，则由 `fileno` 指定的文件描述符将不会被复制。（由 Zackery Spytz 和 Petr Viktorin 在 [gh-78502](#) 中贡献。）

## 5.26 multiprocessing

- 工作线程和进程的默认数据现在使用的是 `os.process_cpu_count()` 而不是 `os.cpu_count()` 来选择的。（由 Victor Stinner 在 [gh-109649](#) 中贡献。）

## 5.27 os

- 增加了 `process_cpu_count()` 函数用于获取当前进程的调用方线程可以使用的逻辑 CPU 核心数量。（由 Victor Stinner 在 [gh-109649](#) 中贡献。）
- `cpu_count()` 和 `process_cpu_count()` 可通过新的环境变量 `PYTHON_CPU_COUNT` 或新的命令行选项 `-X cpu_count` 来覆盖。此选项对于需要在不修改应用程序代码或容器本身的情况下限制一个容器系统的 CPU 资源的用户会很有用处。（由 Donghee Na 在 [gh-109595](#) 中贡献。）
- 通过 `timerfd_create()`, `timerfd_settime()`, `timerfd_gettime_ns()`, `timerfd_gettime()`, `timerfd_gettime_ns()`, `TFD_NONBLOCK`, `TFD_CLOEXEC`, `TFD_TIMER_ABSTIME` 和 `TFD_TIMER_CANCEL_ON_SET` 增加了针对 Linux 的 计算器文件描述符 的低层级接口。（由 Masaru Tsuchiyama 在 [gh-108277](#) 中贡献。）
- 在 Windows 上现在同时增加了对 `lchmod()` 和 `chmod()` 的 `follow_symlinks` 参数的支持。请注意在 Windows 上 `lchmod()` 中的 `follow_symlinks` 的默认值为 `False`。（由 Serhiy Storchaka 在 [gh-59616](#) 中贡献。）



- 在 Windows 上现在同时增加了 `fchmod()` 和对 `chmod()` 中的文件描述符的支持。(由 Serhiy Storchaka 在 [gh-113191](#) 中贡献。)
- 在 Windows 上, `mkdir()` 和 `makedirs()` 现在支持传入 *mode* 值 `0o700` 以对新目录应用访问控制。这会隐式地影响 `tempfile.mkdtemp()` 并可缓解 [CVE 2024-4030](#)。其他的 *mode* 值仍然会被忽略。(由 Steve Dower 在 [gh-118486](#) 中贡献。)
- 现在 `posix_spawn()` 可接受 `None` 作为 *env* 参数, 这将让新产生的进程使用当前进程的环境。(由 Jakub Kulik 在 [gh-113119](#) 中贡献。)
- 在支持 `posix_spawn_file_actions_addclosefrom_np()` 的平台上 `posix_spawn()` 现在可以在 *file\_actions* 形参中使用 `POSIX_SPAWN_CLOSEFROM` 属性。(由 Jakub Kulik 在 [gh-113117](#) 中贡献。)

## 5.28 os.path

- 增加了 `isreserved()` 用于检查一个路径在当前系统中是否为保留路径。此函数仅在 Windows 上可用。(由 Barney Gale 在 [gh-88569](#) 中贡献。)
- 在 Windows 上, `isabs()` 将不再把以恰好一个斜杠 (`\` 或 `/`) 开头的路径视为绝对路径。(由 Barney Gale 和 Jon Foster 在 [gh-44626](#) 中贡献。)
- 现在即使文件不可访问 `realpath()` 也能够解析 MS-DOS 风格的文件名。(由 Moonsik Park 在 [gh-82367](#) 中贡献。)

## 5.29 pathlib

- 增加了 `UnsupportedOperation`, 它会在一个路径操作不受支持时代替 `NotImplementedError` 被引发。(由 Barney Gale 在 [gh-89812](#) 中贡献。)
- 新增了一个用于根据“file”URI (`file:///`) 来创建 `Path` 对象的构造器 `Path.from_uri()`。(由 Barney Gale 在 [gh-107465](#) 中贡献。)
- 增加了 `PurePath.full_match()` 用于匹配带有 `shell` 风格通配符的路径, 包括递归通配符“`***`”。(由 Barney Gale 在 [gh-73435](#) 中贡献。)
- 增加了 `PurePath.parser` 类属性以存储用于低层级路径解析与合并的 `os.path` 实现。这可以是 `posixpath` 或 `ntpath`。
- 为 `Path.glob()` 和 `rglob()` 增加了 *recurse\_symlinks* 仅限关键字参数。(由 Barney Gale 在 [gh-77609](#) 中贡献。)
- 现在当给出以“`***`”结束的模式时 `Path.glob()` 和 `rglob()` 将返回文件和目录。在之前版本中, 仅会返回目录。(由 Barney Gale 在 [gh-70303](#) 中贡献。)
- 为 `Path.is_file`, `Path.is_dir`, `Path.owner()` 和 `Path.group()` 增加了 *follow\_symlinks* 仅限关键字参数。(由 Barney Gale 在 [gh-105793](#) 中, 以及 Kamil Turek 在 [gh-107962](#) 中贡献。)

## 5.30 pdb

- 现在 `breakpoint()` 和 `set_trace()` 会立即进入调试器而不是在被执行代码的下一行进入。这一更改可防止当 `breakpoint()` 位于上下文末尾时调试器在上下文以外被中断。(由 Tian Gao 在 [gh-118579](#) 中贡献。)
- 当设置了 `sys.flags.safe_path` 时 `sys.path[0]` 将不会再被替换为被调试脚本的目录。(由 Tian Gao 和 Christian Walther 在 [gh-111762](#) 中贡献。)
- 现在支持将 `zipapp` 作为调试目标。(由 Tian Gao 在 [gh-118501](#) 中贡献。)
- 添加了在 `pm()` 中进行事后调试期间使用 `Pdb` 新增的 `exceptions [exc_number]` 命令在串连的异常之间移动的能力。(由 Matthias Bussonnier 在 [gh-106676](#) 中贡献。)
- 以一条 `pdb` 命令打头的表达式和语句现在会被正确地标识并执行。(由 Tian Gao 在 [gh-108464](#) 中贡献。)



### 5.31 queue

- 增加了 `Queue.shutdown` 和 `ShutDown` 用于管理队列的终结。(由 Laurie Opperman 和 Yves Duprat 在 [gh-104750](#) 中贡献。)

### 5.32 random

- 增加了一个 命令行接口。(由 Hugo van Kemenade 在 [gh-118131](#) 中贡献。)

### 5.33 re

- 将 `re.error` 重命名为 `PatternError` 以改善准确性。`re.error` 仍被保留用于向下兼容。

### 5.34 shutil

- 在 `chown()` 中增加了对 `dir_fd` 和 `follow_symlinks` 关键字参数的支持。(由 Berker Peksag 和 Tahia K 在 [gh-62308](#) 中贡献。)

### 5.35 site

- 现在 `.pth` 文件将先使用 UTF-8 来解码, 如果 UTF-8 解码失败再使用 locale encoding。(由 Inada Naoki 在 [gh-117802](#) 中贡献。)

### 5.36 sqlite3

- 现在当一个 `Connection` 对象未被显式地 关闭时将发出 `ResourceWarning`。(由 Erlend E. Aasland 在 [gh-105539](#) 中贡献。)
- 为 `Connection.iterdump()` 增加了 `filter` 仅限关键字形参用于过滤要转储的数据库对象。(由 Mariusz Felisiak 在 [gh-91602](#) 中贡献。)

### 5.37 ssl

- 现在 `create_default_context()` API 将在其默认旗标中包括 `VERIFY_X509_PARTIAL_CHAIN` 和 `VERIFY_X509_STRICT`。

#### 备注

`VERIFY_X509_STRICT` 可能会拒绝下层 OpenSSL 实现本来会接受的 [RFC 5280](#) 以前的证书或格式错误的证书。虽然不建议禁用此功能, 但你可以使用以下方式禁用它:

```
import ssl

ctx = ssl.create_default_context()
ctx.verify_flags &= ~ssl.VERIFY_X509_STRICT
```

(由 William Woodruff 在 [gh-112389](#) 贡献。)

### 5.38 statistics

- 增加了用于核密度估计的 `kde()`。这使得根据固定数量的离散样本估计连续概率密度函数成为可能。(由 Raymond Hettinger 在 [gh-115863](#) 中贡献。)
- 增加了 `kde_random()` 用来从 `kde()` 创建的估计概率密度函数进行取样。(由 Hettinger 在 [gh-115863](#) 中贡献。)

## 5.39 subprocess

- 现在 `subprocess` 模块会在更多场合下使用 `posix_spawn()` 函数。

需要注意的是，当 `close_fds` 为 `True` 时（默认值），则将在 C 库提供了 `posix_spawn_file_actions_addclosefrom_np()` 时使用 `posix_spawn()`，这包括近期的 Linux, FreeBSD 和 Solaris 版本。在 Linux，其性能应当与现有的 Linux `vfork()` 基础代码类似。

如果你需要强制 `subprocess` 绝不使用 `posix_spawn()` 可以将私有的控制节点 `subprocess._USE_POSIX_SPAWN` 设为 `False`。如果你这样设置的话请在 [issue tracker](#) 中报告你的理由和平台相关的细节以便我们能够为大家改进 API 的选择逻辑。（由 Jakub Kulik 在 [gh-113117](#) 中贡献。）

## 5.40 sys

- 增加了 `_is_interned()` 函数用于检测字符串是否被内部化。此函数不保证在所有的 Python 实现中均存在。（由 Serhiy Storchaka 在 [gh-78573](#) 中贡献。）

## 5.41 tempfile

- 在 Windows 上，`tempfile.mkdtemp()` 所使用的默认模式 `0o700` 由于 `os.mkdir()` 的更改现在将限制对新目录的访问。这是对 [CVE 2024-4030](#) 的缓解措施。（由 Steve Dower 在 [gh-118486](#) 中贡献。）

## 5.42 time

- 在 Windows 上，`monotonic()` 现在将使用精度为 1 微秒的 `QueryPerformanceCounter()` 时钟，而不是精度只有 15.6 毫秒的 `GetTickCount64()` 时钟。（由 Victor Stinner 在 [gh-88494](#) 中贡献。）
- 在 Windows 上，`time()` 现在将使用精度为 1 微秒的 `GetSystemTimePreciseAsFileTime()` 时钟，代替精度为 15.6 毫秒的 `GetSystemTimeAsFileTime()` 时钟。（由 Victor Stinner 在 [gh-63207](#) 中贡献。）

## 5.43 tkinter

- 增加了 tkinter 控件方法: `tk_busy_hold()`, `tk_busy_configure()`, `tk_busy_cget()`, `tk_busy_forget()`, `tk_busy_current()` 和 `tk_busy_status()`。（由 Miguel, klappnase 和 Serhiy Storchaka 在 [gh-72684](#) 中贡献。）
- 现在 tkinter 控件 `wm_attributes()` 接受不带负号前缀的属性名称来获取窗口属性，例如 `w.wm_attributes('alpha')` 并允许指定属性和值以关键字参数形式来设置，例如 `w.wm_attributes(alpha=0.5)`。（由 Serhiy Storchaka 在 [gh-43457](#) 中贡献。）
- 通过使用新的可选关键字形参 `return_python_dict`，现在 `wm_attributes()` 可将属性作为 `dict` 返回。（由 Serhiy Storchaka 在 [gh-43457](#) 中贡献。）
- 现在当使用新的可选仅限关键字形参 `return_ints` 时 `Text.count()` 可以返回一个简单的 `int`。在其他情况下，将以 1 个元素的元组形式返回单个计数值或者 `None`。（由 Serhiy Storchaka 在 [gh-97928](#) 中贡献。）
- 在 `tkinter.ttk.Style` 的 `element_create()` 方法中增加了对“vsapi”元素类型的支持。（由 Serhiy Storchaka 在 [gh-68166](#) 中贡献。）
- 为 Tkinter 的控件增加了 `after_info()` 方法。（由 Cheryl Sabella 在 [gh-77020](#) 中贡献。）
- 为 `PhotoImage` 新增 `copy_replace()` 方法用于将一个图像的某个区域拷贝到另一个图像，可能带有像素缩放、子采样，或两者皆有。（由 Serhiy Storchaka 在 [gh-118225](#) 中贡献。）
- 为 `PhotoImage` 的方法 `copy()`, `zoom()` 和 `subsample()` 增加了 `from_coords` 形参。为 `PhotoImage` 的方法 `copy()` 增加了 `zoom` 和 `subsample` 形参。（由 Serhiy Storchaka 在 [gh-118225](#) 中贡献。）
- 增加了 `PhotoImage` 方法 `read()` 用于从文件读取图像以及 `data()` 用于获取图像数据。为 `write()` 方法增加了 `background` 和 `grayscale` 形参。（由 Serhiy Storchaka 在 [gh-118271](#) 中贡献。）

## 5.44 回溯

- 为 `TracebackException` 增加了 `exc_type_str` 属性，它用于保存 `exc_type` 字符串表示。弃用了 `exc_type` 属性，它用于保存类型对象本身。增加了 `save_exc_type` 形参 (默认值为 `True`) 用于指明 `exc_type` 是否应当被保存。(由 Irit Katriel 在 [gh-112332](#) 中贡献。)
- 为 `TracebackException.format_exception_only()` 增加了新的 `show_group` 仅限关键字形参用于 (递归地) 格式化 `BaseExceptionGroup` 实例中嵌套的异常。(由 Irit Katriel 在 [gh-105292](#) 中贡献。)

## 5.45 types

- 现在 `SimpleNamespace` 可以接受单个位置参数来初始化命名空间的各个参数值。该参数必须为映射或键值对的可迭代对象。(由 Serhiy Storchaka 在 [gh-108191](#) 中贡献。)

## 5.46 typing

- **PEP 705**: 增加 `ReadOnly`，一个针对类型检查器的特殊类型结构，用于将 `TypedDict` 的项标记为只读。
- **PEP 742**: 增加 `TypeIs`，一个可被用于指示类型检查器如何细化类型的类型结构。
- 增加 `NoDefault`，一个用于代表 `typing` 模块中某些形参的默认值的哨兵对象。(由 Jelle Zijlstra 在 [gh-116126](#) 中贡献。)
- 增加 `get_protocol_members()` 用于返回定义一个 `typing.Protocol` 的成员的集合。(由 Jelle Zijlstra 在 [gh-104873](#) 中贡献。)
- 增加 `is_protocol()` 用于检查一个类是否属于 `Protocol`。(由 Jelle Zijlstra 在 [gh-104873](#) 中贡献。)
- 现在 `ClassVar` 可以被嵌套在 `Final` 中，反之亦然。(由 Mehdi Drissi 在 [gh-89547](#) 中贡献。)

## 5.47 unicodedata

- 将 Unicode 数据库更新到 15.1.0 版。(由 James Gerity 在 [gh-109559](#) 中贡献。)

## 5.48 venv

- 增加了对在虚拟环境目录中添加源码控制管理 (SCM) 忽略文件的支持。在默认情况下，`Git` 已受到支持。此特性是以可被扩展为支持其他 SCM 的通过 API 选择启用 (`EnvBuilder` 和 `create()`)，并通过 CLI 使用 `--without-scm-ignore-files` 选择禁用的方式实现的。(由 Brett Cannon 在 [gh-108125](#) 中贡献。)

## 5.49 warnings

- **PEP 702**: 新的 `warnings.deprecated()` 装饰器提供了一种将弃用消息传送给 `static type checker` 并在使用已弃用的类和函数时发出警告的方式。当被装饰的函数或类在运行时被使用时也可以发出 `DeprecationWarning`。(由 Jelle Zijlstra 在 [gh-104003](#) 中贡献。)

## 5.50 xml

- 通过添加五个新方法允许控制 `Expat >=2.6.0` 重解析延迟 (**CVE 2023-52425**):

```
- xml.etree.ElementTree.XMLParser.flush()
- xml.etree.ElementTree.XMLPullParser.flush()
- xml.parsers.expat.xmlparser.GetReparseDeferralEnabled()
- xml.parsers.expat.xmlparser.SetReparseDeferralEnabled()
- xml.sax.expatreader.ExpatParser.flush()
```

(由 Sebastian Pipping 在 [gh-115623](#) 中贡献。)

- 为 `iterparse()` 所返回的迭代器增加了 `close()` 方法用于执行显式的清理。(由 Serhiy Storchaka 在 [gh-69893](#) 中贡献。)

## 5.51 zipimport

- 增加了对 ZIP64 格式的文件的支持。大家都喜欢更庞大的数据，对吧？(由 Tim Hatch 在 [gh-94146](#) 中贡献。)

## 6 性能优化

- 一些标准库模块的导入时间得到了显著改善。例如，`typing` 模块的导入时间通过移除对 `re` 和 `contextlib` 的依赖而减少了大约三分之一。其他获得导入时间加速的模块包括 `email.utils`, `enum`, `functools`, `importlib.metadata` 和 `threading`。(由 Alex Waygood, Shantanu Jain, Adam Turner, Daniel Hollas 等人在 [gh-109653](#) 中贡献。)
- 现在对于大量输入 `textwrap.indent()` 相比之前可提速大约 30%。(由 Inada Naoki 在 [gh-107369](#) 中贡献。)
- 现在 `subprocess` 模块会在更多场合下使用 `posix_spawn()` 函数，包括在许多现代系统平台上当 `close_fds` 为 `True` (默认值) 的时候。当在 `FreeBSD` 和 `Solaris` 上启动进程时这应该能提供显著的性能提升。请参阅上面的 [subprocess](#) 小节了解详情。(由 Jakub Kulik 在 [gh-113117](#) 中贡献。)

## 7 被移除的模块与 API

### 7.1 PEP 594: 从标准库中移除“死电池”

**PEP 594** 提议从标准库移除 19 个模块，它们因其古旧、过时或不安全的状态而被非正式地称呼为‘死电池’。下列所有模块在 Python 3.11 中被弃用，现在已被移除：

- `aifc`
  - `standard-aifc`: 使用 PyPI 上的重新分发版 `aifc` 库。
- `audioop`
  - `audioop-lts`: 使用 PyPI 上的 `audioop-lts` 库。
- `chunk`
  - `standard-chunk`: 使用 PyPI 上的重新分发版 `chunk` 库。
- `cgi` 和 `cgitb`
  - 对于 GET 和 HEAD 请求 `cgi.FieldStorage` 通常可以用 `urllib.parse.parse_qs()` 来替换，而对于 POST 和 PUT 请求则可以用 `email.message` 模块或 `multipart` 库。
  - `cgi.parse()` 可被替换为在想要的查询字符串上直接调用 `urllib.parse.parse_qs()`，除非输入为 `multipart/form-data`，它应当如下文针对 `cgi.parse_multipart()` 所描述的那样被替换。
  - `cgi.parse_header()` 可被 `email` 包中的功能所替换，它实现了相同的 MIME RFC。例如，使用 `email.message.EmailMessage`:

```
from email.message import EmailMessage

msg = EmailMessage()
msg['content-type'] = 'application/json; charset="utf8"'
main, params = msg.get_content_type(), msg['content-type'].params
```

- `cgi.parse_multipart()` 可以用 `email` 包中的功能来替换，它实现了相同的 MIME RFC，也可以用 `multipart` 库。例如，`email.message.EmailMessage` 和 `email.message.Message` 类。
- `standard-cgi`: 和 `standard-cgitb`: 使用 PyPI 上的重新分发版 `cgi` 和 `cgitb` 库。

- `crypt` 以及私有的 `_crypt` 扩展。`hashlib` 模块在仅需对值执行哈希时是一个适当的替代物。在其他情况下，可以使用 PyPI 上的一些第三方库：
  - `bcrypt`: 用于软件和服务器的现代密码哈希算法。
  - `passlib`: 支持超过 over 30 种方案的综合密码哈希算法框架。
  - `argon2-cffi`: 安全的 Argon2 密码哈希算法。
  - `legacycrypt`: 针对 POSIX 加密库调用和相关功能的 `ctypes` 包装器。
  - `crypt_r`: `crypt` 模块的分叉，针对 `crypt_r(3)` 库调用和相关功能和包装器。
  - `standard-crypt` 和 `deprecated-crypt-alternative`: 使用 PyPI 上的重新分发版 `crypt` 和重新实现的 `_crypt` 库。
- `imghdr`: 应当使用 `filetype`, `puremagic` 或 `python-magic` 等库作为替代。例如，对于 `imghdr` 所支持的所有文件格式 `puremagic.what()` 函数可被用来替代 `imghdr.what()` 函数。
  - `standard-imghdr`: 使用 PyPI 上的重新分发版 `imghdr` 库。
- `mailcap`: 改用 `mimetypes` 模块。
  - `standard-mailcap`: 使用 PyPI 上的重新分发版 `mailcap` 库。
- `msilib`
- `nis`
- `nntplib`: 改用 PyPI 上的 `pynntp` 库。
  - `standard-nntplib`: 使用 PyPI 上的重新分发版 `nntplib` 库。
- `ossaudiodev`: 对于音频回放，改用 PyPI 上的 `pygame` 库。
- `pipes`: 改用 `subprocess` 模块。使用 `shlex.quote()` 来替代未写入文档的 `pipes.quote` 函数。
  - `standard-pipes`: 使用 PyPI 上的重新分发版 `pipes` 库。
- `sndhdr`: 应当使用 `filetype`, `puremagic` 或 `python-magic` 库作为替代。
  - `standard-sndhdr`: 使用 PyPI 上的重新分发版 `sndhdr` 库。
- `spwd`: 改用 PyPI 上的 `python-pam` 库。
- `sunau`
  - `standard-sunau`: 使用 PyPI 上的重新分发版 `sunau` 库。
- `telnetlib`, 改用 PyPI 上的 `telnetlib3` 或 `Exscript` 库。
  - `standard-telnetlib`: 使用 PyPI 上的重新分发版 `telnetlib` 库。
- `uu`: 改用 `base64` 模块，作为一款现代化的替代。
  - `standard-uu`: 使用 PyPI 上的重新分发版 `uu` 库。
- `xdrlib`
  - `standard-xdrlib`: 使用 PyPI 上的重新分发版 `xdrlib` 库。

(由 Victor Stinner 和 Zachary Ware 在 [gh-104773](#) 和 [gh-104780](#) 中贡献。)

## 7.2 2to3

- 移除 `2to3` 程序和 `lib2to3` 模块，此前已在 Python 3.11 中被弃用。(由 Victor Stinner 在 [gh-104780](#) 中贡献。)

## 7.3 builtins

- 移除了对串联 `classmethod` 描述器的支持 (在 [gh-63272](#) 中引入)。它们不能再被用来包装其他描述器, 如 `property`。此特性的核心设计存在缺陷并导致了一些问题。要“穿过”一个 `classmethod`, 请考虑使用在 Python 3.10 中增加的 `__wrapped__` 属性。(由 Raymond Hettinger 在 [gh-89519](#) 中贡献。)
- 当在被挂起的帧上调用 `frame.clear()` 时将引发 `RuntimeError`。(就如正在执行的帧一直以来的情况那样)。(由 Irit Katriel 在 [gh-79932](#) 中贡献。)

## 7.4 configparser

- 移除了未写入文档的 `LegacyInterpolation` 类, 它自 Python 3.2 起已在文档字符串中声明弃用, 并且自 Python 3.11 起在运行时也已弃用。(由 Hugo van Kemenade 在 [gh-104886](#) 中贡献。)

## 7.5 importlib.metadata

- 移除了已弃用的针对 `EntryPoint` 对象的下标 (`__getitem__()`) 访问。(由 Jason R. Coombs 在 [gh-113175](#) 中贡献。)

## 7.6 locale

- 移除了 `locale.resetlocale()` 函数, 它已在 Python 3.11 中被弃用。请改用 `locale.setlocale(locale.LC_ALL, "")`。(由 Victor Stinner 在 [gh-104783](#) 中贡献。)

## 7.7 opcode

- 将 `opcode.ENABLE_SPECIALIZATION` 移至 `_opcode.ENABLE_SPECIALIZATION`。这个字段是在 3.12 中增加的, 它从未被写入文档, 也无意开放给外部使用。(由 Irit Katriel 在 [gh-105481](#) 中贡献。)
- 移除了 `opcode.is_pseudo()`, `opcode.MIN_PSEUDO_OPCODE` 和 `opcode.MAX_PSEUDO_OPCODE`, 它们是在 Python 3.12 中增加了, 但从未被写入文档也未通过 `dis` 对外公开, 并且不应当在外部被使用。(由 Irit Katriel 在 [gh-105481](#) 中贡献。)

## 7.8 optparse

- 此模块已不再被设为 `soft deprecated`。虽然对于不使用第三方命令行参数处理库的新项目 `argparse` 仍是推荐的选择, 但在某些方面 `argparse` 的特征意味着较低层级的 `optparse` 模块为编写参数处理库以及实现相比 `argparse` 与多种 Unix 命令行处理规范绑定更严格、源自 C `getopt()` 函数行为的命令行应用程序提供了坚实的基础。(由 Alyssa Coghlan 和 Serhiy Storchaka 在 [gh-126180](#) 中贡献。)

## 7.9 pathlib

- 移除了使用 `Path` 对象作为上下文管理器的能力。此功能自 Python 3.9 起已被弃用并设为空操作。(由 Barney Gale 在 [gh-83863](#) 中贡献。)

## 7.10 re

- 移除了未被写入文档、已被弃用且已不能工作的 `re.template()` 函数和 `re.TEMPLATE / re.T` 标志。(由 Serhiy Storchaka 和 Nikita Sobolev 在 [gh-105687](#) 中贡献。)

## 7.11 tkinter.tix

- 移除了 `tkinter.tix` 模块, 它在 Python 3.6 中已被弃用。该模块所包装的第三方库 `Tix` 已不再维护。(由 Zachary Ware 在 [gh-75552](#) 中贡献。)



## 7.12 turtle

- 移除了 `RawTurtle.settiltangle()` 方法，它自 Python 3.1 起已在文档中声明弃用并自 Python 3.11 起在运行时声明弃用。（由 Hugo van Kemenade 在 [gh-104876](#) 中贡献。）

## 7.13 typing

- 移除了 `typing.io` 和 `typing.re` 命名空间，它们自 Python 3.8 起已被弃用。这些命名空间中的条目可从 `typing` 模块直接导入。（由 Sebastian Rittau 在 [gh-92871](#) 中贡献。）
- 移除了创建 `TypedDict` 类型的关键字参数方法，它在 Python 3.11 中已被弃用。（由 Tomas Roun 在 [gh-104786](#) 中贡献。）

## 7.14 unittest

- 移除了下列 `unittest` 函数，它们在 Python 3.11 中已被弃用：

- `unittest.findTestCases()`
- `unittest.makeSuite()`
- `unittest.getTestCaseNames()`

请改用 `TestLoader` 方法：

- `loadTestsFromModule()`
- `loadTestsFromTestCase()`
- `getTestCaseNames()`

（由 Hugo van Kemenade 在 [gh-104835](#) 中贡献。）

- 移除了未经测试且未写入文档的 `TestProgram.usageExit()` 方法，它在 Python 3.11 中已被弃用。（由 Hugo van Kemenade 在 [gh-104992](#) 中贡献。）

## 7.15 urllib

- 移除了 `urllib.request.urlopen()` 函数的 `cafile`、`capath` 和 `cadefault` 等形参，它们在 Python 3.6 中已弃用。应改为向 `context` 形参传入一个 `SSLContext` 实例。可以使用 `ssl.SSLContext.load_cert_chain()` 函数来加载指定的证书，或是让 `ssl.create_default_context()` 来选择操作系统的受信任证书颁发机构 (CA) 证书。（由 Victor Stinner 在 [gh-105382](#) 中贡献。）

## 7.16 webbrowser

- 移除了未经测试且未写入文档的 `MacOSX` 类，它在 Python 3.11 中已弃用。请改用 `MacOSXOSAScript` 类（在 Python 3.2 中引入）。（由 Hugo van Kemenade 在 [gh-104804](#) 中贡献。）
- 移除了已被弃用的 `MacOSXOSAScript._name` 属性。请改用 `MacOSXOSAScript.name` 属性。（由 Nikita Sobolev 在 [gh-105546](#) 中贡献。）

## 8 新的弃用

- 用户自定义函数：
  - 弃用对函数的 `__code__` 属性的赋值，其中新代码对象的类型与函数的类型不匹配。不同的类型有：普通函数、生成器、异步生成器和协程。（由 Irit Katriel 在 [gh-81137](#) 中贡献。）
- `array`：
  - 在运行时弃用 `'u'` 格式代码 (`wchar_t`)。此格式代码自 Python 3.3 起在文档中已被弃用，并将在 Python 3.16 中被移除。对于 Unicode 字符，请使用 `'w'` 格式代码 (`Py_UCS4`)。（由 Hugo van Kemenade 在 [gh-80480](#) 中贡献。）
- `ctypes`：

- 弃用了未写入文档的 `SetPointerType()` 函数，并将在 Python 3.15 中移除。（由 Victor Stinner 在 [gh-105733](#) 中贡献。）
- 软弃用 `ARRAY()` 函数并改用 `type * length` 乘法运算。（由 Victor Stinner 在 [gh-105733](#) 中贡献。）
- decimal:
  - 弃用了非标准且未写入文档的 `Decimal` 格式说明符 `'N'`，它仅在 `decimal` 模块的 C 实现中受到支持。（由 Serhiy Storchaka 在 [gh-89902](#) 中贡献。）
- dis:
  - 弃用了 `HAVE_ARGUMENT` 分隔符。改为在 `hasarg` 中的成员检测。（由 Irit Katriel 在 [gh-109319](#) 中贡献。）
- gettext:
  - 在 `gettext` 模块中，弃用非整数作为考虑复数形式的函数和方法的参数，即使没有找到翻译时也不可以。（由 Serhiy Storchaka 在 [gh-88434](#) 中贡献。）
- glob:
  - 弃用未写入文档的 `glob0()` 和 `glob1()` 函数。改为使用 `glob()` 并传递一个指定根目录的 `path-like object` 到 `root_dir` 形参。（由 Barney Gale 在 [gh-117337](#) 中贡献。）
- http.server:
  - 弃用 `CGIHTTPRequestHandler` 类，将在 Python 3.15 中移除。基于进程的 CGI HTTP 服务器已经过时很久了。该代码已经过时，无人维护，而且很少使用。它极有可能出现安全和功能方面的程序错误。（由 Gregory P. Smith 在 [gh-109096](#) 中贡献。）
  - 弃用 `program:python -m http.server` 命令行接口的 `option:!--cgi` 标志位，将在 Python 3.15 中移除。（由 Gregory P. Smith 在 [gh-109096](#) 中提供。）
- mimetypes:
  - 软弃用传给 `guess_type()` 的文件路径参数，改用 `guess_file_type()`。（由 Serhiy Storchaka 在 [gh-66543](#) 中贡献。）
- re:
  - 弃用将可选的 `maxsplit`, `count` 或 `flags` 参数以位置参数形式传给模块级 `split()`, `sub()` 和 `subn()` 函数的做法。这些形参将在未来的 Python 版本中成为 仅限关键字形参。（由 Serhiy Storchaka 在 [gh-56166](#) 中贡献。）
- pathlib:
  - 弃用 `PurePath.is_reserved()`，将在 Python 3.15 中移除。请使用 `os.path.isreserved()` 来检测 Windows 上的保留路径。（由 Barney Gale 在 [gh-88569](#) 中贡献。）
- platform:
  - 弃用了 `java_ver()`，并将在 Python 3.15 中移除。此函数仅对 Jython 支持有用，具有令人困惑的 API，并且大部分未经测试。（由 Nikita Sobolev 在 [gh-116349](#) 中贡献。）
- pydoc:
  - 弃用未写入文档的 `ispackage()` 函数。（由 Zackery Spytz 在 [gh-64020](#) 中贡献。）
- sqlite3:
  - 弃用向 `connect()` 函数和 `Connection` 构造器传入多个位置参数的做法。其余的形参在 Python 3.15 中将成为 仅限关键字形参。（由 Erlend E. Aasland 在 [gh-107948](#) 中贡献。）
  - 弃用将名称、参数数量和可调用对象作为 `Connection.create_function()` 和 `Connection.create_aggregate()` 的关键字参数传入的做法。这些形参在 Python 3.15 中将成为 仅限位置形参。（由 Erlend E. Aasland 在 [gh-108278](#) 中贡献。）

- 弃用将 `callback` 可调用对象作为 `set_authorizer()`, `set_progress_handler()` 和 `set_trace_callback()` 等 `Connection` 方法的关键字参数传入的做法。`callback` 可调用对象在 Python 3.15 中将成为仅限位置参数。(由 Erlend E. Aasland 在 [gh-108278](#) 中贡献。)
- `sys`:
  - 弃用 `_enablelegacywindowsfsencoding()` 函数, 并将在 Python 3.16 中移除。请改用 `PYTHONLEGACYWINDOWSFSENCODING` 环境变量。(由 Inada Naoki 在 [gh-73427](#) 中贡献。)
- `tarfile`:
  - 弃用了未写入文档也未被使用的 `TarFile.tarfile` 属性, 并将在 Python 3.16 中移除。(在 [gh-115256](#) 中贡献。)
- `traceback`:
  - 已弃用 `TracebackException.exc_type` 属性。请改用 `TracebackException.exc_type_str`。(由 Irit Katriel 在 [gh-112332](#) 中贡献。)
- `typing`:
  - 弃用了未写入文档的用于创建 `NamedTuple` 类的关键字参数语法 (例如 `Point = NamedTuple("Point", x=int, y=int)`), 将在 Python 3.15 中移除。请改用基于类的语法或函数式语法。(由 Alex Waygood 在 [gh-105566](#) 中贡献。)
  - 弃用当创建 `NamedTuple` 或 `typing.TypedDict` 类时省略 `fields` 形参的做法, 并弃用将 `None` 传给这两种类型的 `fields` 形参的做法。Python 3.15 将要求以一个有效的序列作为 `fields` 形参。要创建具有零个字段的 `NamedTuple` 类, 请使用 `class NT(NamedTuple): pass` 或 `NT = NamedTuple("NT", ())`。要创建具有零个字段的 `TypedDict` 类, 请使用 `class TD(TypedDict): pass` 或 `TD = TypedDict("TD", {})`。(由 Alex Waygood 在 [gh-105566](#) 和 [gh-105570](#) 中贡献。)
  - 弃用 `typing.no_type_check_decorator()` 装饰器函数, 将在 Python 3.15 中被移除。存在于 `typing` 模块中八年之后, 它仍未被任何主要类型检查器所支持。(由 Alex Waygood 在 [gh-106309](#) 中贡献。)
  - 弃用 `typing.AnyStr`。在 Python 3.16 中, 它将从 `typing.__all__` 移除, 当它被导入或被访问时将会发出 `DeprecationWarning`。它将在 Python 3.18 中被完全移除。请改用新的类型形参语法。(由 Michael The 在 [gh-107116](#) 中贡献。)
- `wave`:
  - 弃用 `Wave_read` 和 `Wave_write` 类的 `getmark()`, `setmark()` 和 `getmarkers()` 方法, 将在 Python 3.15 中移除。(由 Victor Stinner 在 [gh-105096](#) 中贡献。)

## 8.1 计划在 Python 3.14 中移除

- `argparse`: `argparse.BooleanOptionalAction` 的 `type`, `choices` 和 `metavar` 形参已被弃用并将在 3.14 中移除。(由 Nikita Sobolev 在 [gh-92248](#) 中贡献。)
- `ast`: 以下特性自 Python 3.8 起已在文档中声明弃用, 现在当运行时如果它们被访问或使用时将发出 `DeprecationWarning`, 并将在 Python 3.14 中移除:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`
 请改用 `ast.Constant`。(由 Serhiy Storchaka 在 [gh-90953](#) 中贡献。)
- `asyncio`:

- 子监视器类 `MultiLoopChildWatcher`, `FastChildWatcher`, `AbstractChildWatcher` 和 `SafeChildWatcher` 已被弃用并将在 Python 3.14 中移除。(由 Kumar Aditya 在 [gh-94597](#) 中贡献。)
- `asyncio.set_child_watcher()`、`asyncio.get_child_watcher()`、`asyncio.AbstractEventLoopPolicy.set_child_watcher()` 和 `asyncio.AbstractEventLoopPolicy.get_child_watcher()` 已弃用，并将在 Python 3.14 中移除。(由 Kumar Aditya 在 [gh-94597](#) 中贡献。)
- 现在默认事件循环策略的 `get_event_loop()` 方法在当前事件循环未设置并决定创建一个时将发出 `DeprecationWarning`。(由 Serhiy Storchaka 和 Guido van Rossum 在 [gh-100160](#) 中贡献。)
- `collections.abc`: 已弃用 `ByteString`。推荐改用 `Sequence` 或 `Buffer`。用于类型标注时，则推荐并集运算符，如 `bytes | bytearray`，或 `collections.abc.Buffer`。(由 Shantanu Jain 在 [gh-91896](#) 中贡献。)
- `email`: 已弃用 `email.utils.localtime()` 中的 *isdst* 形参。(由 Alan Williams 在 [gh-72346](#) 中贡献。)
- `importlib.abc` 中已弃用的类:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`
 使用 `importlib.resources.abc` 类代替:
  - `importlib.resources.abc.Traversable`
  - `importlib.resources.abc.TraversableResources`
 (由 Jason R. Coombs 和 Hugo van Kemenade 在 [gh-93963](#) 中贡献。)
- `itertools` 具有对 `copy`, `deepcopy` 和 `pickle` 等操作的未写入文档的、低效的、历史上充满问题的且不稳定的支持。这将在 3.14 中移除以显著减少代码量和维护负担。(由 Raymond Hettinger 在 [gh-101588](#) 中贡献。)
- `multiprocessing`: 默认的启动方法在目前默认使用 'fork' 的 Linux, BSD 和其他非 macOS POSIX 平台上将改为更安全的方法 ([gh-84559](#))。为此添加运行时警告将带来糟糕的体验因为大部分代码并不会关心这个问题。当你的代码需要 'fork' 时请使用 `get_context()` 或 `set_start_method()` API 来显式地指明。参见 [multiprocessing-start-methods](#)。
- `pathlib`: `is_relative_to()` 和 `relative_to()`: 传入额外参数的做法已被弃用。
- `pkgutil`: 现在 `find_loader()` 和 `get_loader()` 将引发 `DeprecationWarning`; 请改用 `importlib.util.find_spec()`。(由 Nikita Sobolev 在 [gh-97850](#) 中贡献。)
- `pty`:
  - `master_open()`: 使用 `pty.openpty()`。
  - `slave_open()`: 使用 `pty.openpty()`。
- `sqlite3`:
  - `version` 和 `version_info`。
  - 如果使用了命名占位符且 *parameters* 是一个序列而不是 `dict` 则选择 `execute()` 和 `executemany()`。
- `typing`: `ByteString` 自 Python 3.9 起已被弃用，现在当被使用时将会发出 `DeprecationWarning`。
- `urllib`: `urllib.parse.Quoter` 已被弃用: 它不应被作为公有 API。(由 Gregory P. Smith 在 [gh-88168](#) 中贡献。)

## 8.2 Python 3.15 中的待移除功能

- 导入系统:
  - 当设置 `__spec__.cached` 失败时在模块上设置 `__cached__` 的做法已被弃用。在 Python 3.15 中, `__cached__` 将不会再被导入系统或标准库纳入考虑。(gh-97879)
  - 当设备 `__spec__.parent` 失败时在模块上设置 `__package__` 的做法已被弃用。在 Python 3.15 中, `__package__` 将不会再被导入系统或标准库纳入考虑。(gh-97879)
- ctypes:
  - 未写入文档的 `ctypes.SetPointerType()` 函数自 Python 3.13 起已被弃用。
- http.server:
  - 过时且很少被使用的 `CGIHTTPRequestHandler` 自 Python 3.13 起已被弃用。不存在直接的替代品。对于建立带有请求处理器的 Web 服务程序来说 任何东西都比 CGI 要好。
  - 用于 `python -m http.server` 命令行界面的 `--cgi` 旗标自 Python 3.13 起已被弃用。
- importlib:
  - `load_module()` 方法: 改用 `exec_module()`。
- locale:
  - `getdefaultlocale()` 函数自 Python 3.11 起已被弃用。最初计划在 Python 3.13 中移除它 (gh-90817), 但已被推迟至 Python 3.15。请改用 `getlocale()`, `setlocale()` 和 `getencoding()`。(由 Hugo van Kemenade 在 gh-111187 中贡献。)
- pathlib:
  - `PurePath.is_reserved()` 自 Python 3.13 起已被弃用。请使用 `os.path.isreserved()` 来检测 Windows 上的保留路径。
- platform:
  - `java_ver()` 自 Python 3.13 起已被弃用。此函数仅对 Jython 支持有用, 具有令人困惑的 API, 并且大部分未经测试。
- sysconfig:
  - `sysconfig.is_python_build()` 的 `check_home` 参数自 Python 3.12 起已被弃用。
- threading:
  - 在 Python 3.15 中 `RLock()` 将不再接受参数。传入参数的做法自 Python 3.14 起已被弃用, 因为 Python 版本不接受任何参数, 而 C 版本允许任意数量的位置或关键字参数, 但会忽略所有参数。
- types:
  - `types.CodeType`: 访问 `co_lnotab` 的做法自 3.10 起已根据 **PEP 626** 被弃用并曾计划在 3.12 中移除, 但在 3.12 中实际仅设置了 `DeprecationWarning`。可能会在 3.15 中移除。(由 Nikita Sobolev 在 gh-101866 中贡献。)
- typing:
  - 未写入文档的用于创建 `NamedTuple` 类的关键字参数语法 (例如 `Point = NamedTuple("Point", x=int, y=int)`) 自 Python 3.13 起已被弃用。请改用基于类的语法或函数语法。
  - `typing.no_type_check_decorator()` 装饰器自 Python 3.13 起已被弃用。存在于 `typing` 模块八年之后, 它仍未被任何主要类型检查器所支持。
- wave:
  - `Wave_read` 和 `Wave_write` 类的 `getmark()`, `setmark()` 和 `getmarkers()` 方法自 Python 3.13 起已被弃用。

## 8.3 计划在 Python 3.16 中移除

- 导入系统:
  - 当设置 `__spec__.loader` 失败时在模块上设置 `__loader__` 的做法已被弃用。在 Python 3.16 中, `__loader__` 将不会再被设置或是被导入系统或标准库纳入考虑。
- `array`:
  - 'u' 格式代码 (`wchar_t`) 自 Python 3.3 起已在文档中弃用并自 Python 3.13 起在运行时弃用。对于 Unicode 字符请改用 'w' 格式代码 (`Py_UCS4`)。
- `asyncio`:
  - `asyncio.iscoroutinefunction()` 已被弃用并将在 Python 3.16 中移除, 请改用 `inspect.iscoroutinefunction()`。(由 Jiahao Li 和 Kumar Aditya 在 [gh-122875](#) 中贡献。)
- `builtins`:
  - 对布尔类型 `~True` 或 `~False` 执行按位取反的操作自 Python 3.12 起已被弃用, 因为它会产生奇怪和不直观的结果 (`-2 and -1`)。请改用 `not x` 来对布尔值执行逻辑否操作。对于需要对下层整数执行按位取反操作的少数场合, 请显式地将其转换为 `int(~int(x))`。
- `shutil`:
  - `ExecError` 异常自 Python 3.14 起已被弃用。它自 Python 3.4 起就未被 `shutil` 中的任何函数所使用, 现在是 `RuntimeError` 的一个别名。
- `symtable`:
  - `Class.get_methods` 方法自 Python 3.14 起被弃用。
- `sys`:
  - `_enablelegacywindowsfsencoding()` 函数自 Python 3.13 起被弃用。请改用 `PYTHONLEGACYWINDOWSFSENCODING` 环境变量。
- `tarfile`:
  - 未写入文档也未被使用的 `TarFile.tarfile` 属性自 Python 3.13 起被弃用。

## 8.4 计划在未来版本中移除

以下 API 将会被移除, 尽管具体时间还未确定。

- `argparse`: 嵌套参数分组和嵌套互斥分组的做法已被弃用。
- `array` 的 'u' 格式代码 ([gh-57281](#))
- `builtins`:
  - `bool(NotImplemented)`。
  - 生成器: `throw(type, exc, tb)` 和 `athrow(type, exc, tb)` 签名已被弃用: 请改用 `throw(exc)` 和 `athrow(exc)`, 即单参数签名。
  - 目前 Python 接受数字类字面值后面紧跟关键字的写法, 例如 `0in x, 1or x, 0if 1else 2`。它允许像 `[0x1for x in y]` 这样令人困惑且有歧义的表达式 (它可以被解读为 `[0x1 for x in y]` 或者 `[0x1f or x in y]`)。如果数字类字面值后面紧跟关键字 `and`, `else`, `for`, `if`, `in`, `is` 和 `or` 中的一个将会引发语法警告。在未来的版本中它将改为语法错误。([gh-87999](#))
  - 对 `__index__()` 和 `__int__()` 方法返回非 `int` 类型的支持: 将要求这些方法必须返回 `int` 的子类的实例。
  - 对 `__float__()` 方法返回 `float` 的子类的支持: 将要求这些方法必须返回 `float` 的实例。
  - 对 `__complex__()` 方法返回 `complex` 的子类的支持: 将要求这些方法必须返回 `complex` 的实例。
  - 将 `int()` 委托给 `__trunc__()` 方法。



- 传入一个复数作为 `complex()` 构造器中的 *real* 或 *imag* 参数的做法现在已被弃用；它应当仅作为单个位置参数被传入。（由 Serhiy Storchaka 在 [gh-109218](#) 中贡献。）
- `calendar`: `calendar.January` 和 `calendar.February` 常量已被弃用并由 `calendar.JANUARY` 和 `calendar.FEBRUARY` 替代。（由 Prince Roshan 在 [gh-103636](#) 中贡献。）
- `codeobject.co_lnotab`: 改用 `codeobject.co_lines()` 方法。
- `datetime`:
  - `utcnow()`: 使用 `datetime.datetime.now(tz=datetime.UTC)`。
  - `utcfromtimestamp()`: 使用 `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`。
- `gettext`: 复数值必须是一个整数。
- `importlib`:
  - `cache_from_source()` *debug\_override* 形参已被弃用：改用 *optimization* 形参。
- `importlib.metadata`:
  - `EntryPoint` 元组接口。
  - 返回值中隐式的 `None`。
- `logging`: `warn()` 方法自 Python 3.3 起已被弃用，请改用 `warning()`。
- `mailbox`: 对 `StringIO` 输入和文本模式的使用已被弃用，改用 `BytesIO` 和二进制模式。
- `os`: 在多线程的进程中调用 `os.register_at_fork()`。
- `pydoc.ErrorDuringImport`: 使用元组值作为 *exc\_info* 形参的做法已被弃用，应使用异常实例。
- `re`: 现在对于正则表达式中的数字分组引用和分组名称将应用更严格的规则。现在只接受 ASCII 数字序列作为数字引用。字节串模式和替换字符串中的分组名称现在只能包含 ASCII 字母和数字以及下划线。（由 Serhiy Storchaka 在 [gh-91760](#) 中贡献。）
- `sre_compile`, `sre_constants` 和 `sre_parse` 模块。
- `shutil.rmtree()` 的 *onerror* 形参在 Python 3.12 中已被弃用；请改用 *onexc* 形参。
- `ssl` 选项和协议：
  - `ssl.SSLContext` 不带 `protocol` 参数的做法已被弃用。
  - `ssl.SSLContext`: `set_npn_protocols()` 和 `selected_npn_protocol()` 已被弃用：请改用 `ALPN`。
  - `ssl.OP_NO_SSL*` 选项
  - `ssl.OP_NO_TLS*` 选项
  - `ssl.PROTOCOL_SSLv3`
  - `ssl.PROTOCOL_TLS`
  - `ssl.PROTOCOL_TLSv1`
  - `ssl.PROTOCOL_TLSv1_1`
  - `ssl.PROTOCOL_TLSv1_2`
  - `ssl.TLSVersion.SSLv3`
  - `ssl.TLSVersion.TLSv1`
  - `ssl.TLSVersion.TLSv1_1`
- `threading` 的方法：
  - `threading.Condition.notifyAll()`: 使用 `notify_all()`。
  - `threading.Event.isSet()`: 使用 `is_set()`。

- `threading.Thread.isDaemon()`, `threading.Thread.setDaemon()`: 使用 `threading.Thread.daemon` 属性。
- `threading.Thread.getName()`, `threading.Thread.setName()`: 使用 `threading.Thread.name` 属性。
- `threading.currentThread()`: 使用 `threading.current_thread()`。
- `threading.activeCount()`: 使用 `threading.active_count()`。
- `typing.Text` ([gh-92332](#))。
- `unittest.IsolatedAsyncioTestCase`: 从测试用例返回不为 `None` 的值的做法已被弃用。
- `urllib.parse` 函数已被弃用: 改用 `urlparse()`
  - `splitattr()`
  - `splithost()`
  - `splitnport()`
  - `splitpasswd()`
  - `splitport()`
  - `splitquery()`
  - `splittag()`
  - `splitttype()`
  - `splituser()`
  - `splitvalue()`
  - `to_bytes()`
- `urllib.request`: 发起请求的 `URLopener` 和 `FancyURLopener` 方式已被弃用。改用更新 `urlopen()` 函数和方法。
- `wsgiref.SimpleHandler.stdout.write()` 不应执行部分写入。
- `xml.etree.ElementTree`: 对 `Element` 的真值测试已被弃用。在未来的发布版中它将始终返回 `True`。建议改用显式的 `len(elem)` 或 `elem is not None` 测试。
- `zipimport.zipimporter.load_module()` 已被弃用: 请改用 `exec_module()`。

## 9 CPython 字节码的变化

- 现在 `YIELD_VALUE` 的操作数在 `yield` 是 `yield-from` 或 `await` 的一部分时为 1, 否则为 0。RESUME 的操作数被修改为增加一个比特位来指明 `except-depth` 是否为 1, 这是优化生成器的关闭所需要的。(由 Irit Katriel 在 [gh-111354](#) 中贡献。)

## 10 C API 的变化

### 10.1 新的特性

- 增加 `PyMonitoring` C API 用于生成 **PEP 669** 监控事件:
  - `PyMonitoringState`
  - `PyMonitoring_FirePyStartEvent()`
  - `PyMonitoring_FirePyResumeEvent()`
  - `PyMonitoring_FirePyReturnEvent()`
  - `PyMonitoring_FirePyYieldEvent()`

- PyMonitoring\_FireCallEvent()
- PyMonitoring\_FireLineEvent()
- PyMonitoring\_FireJumpEvent()
- PyMonitoring\_FireBranchEvent()
- PyMonitoring\_FireCReturnEvent()
- PyMonitoring\_FirePyThrowEvent()
- PyMonitoring\_FireRaiseEvent()
- PyMonitoring\_FireCRaiseEvent()
- PyMonitoring\_FireReraiseEvent()
- PyMonitoring\_FireExceptionHandledEvent()
- PyMonitoring\_FirePyUnwindEvent()
- PyMonitoring\_FireStopIterationEvent()
- PyMonitoring\_EnterScope()
- PyMonitoring\_ExitScope()

(由 Irit Katriel 在 [gh-111997](#) 中贡献。)

- 增加了 PyMutex，它是占用一个字节的轻量级互斥锁，以及新的 PyMutex\_Lock() 和 PyMutex\_Unlock() 函数。如果操作需要阻塞则 PyMutex\_Lock() 将释放 (当前持有的) GIL。(由 Sam Gross 在 [gh-108724](#) 中贡献。)

- 增加了 PyTime C API 以提供对系统时钟的访问：

- PyTime\_t。
- PyTime\_MIN 和 PyTime\_MAX。
- PyTime\_AsSecondsDouble()。
- PyTime\_Monotonic()。
- PyTime\_MonotonicRaw()。
- PyTime\_PerfCounter()。
- PyTime\_PerfCounterRaw()。
- PyTime\_Time()。
- PyTime\_TimeRaw()。

(由 Victor Stinner 和 Petr Viktorin 在 [gh-110850](#) 中贡献。)

- 增加了 PyDict\_ContainsString() 函数，其行为与 PyDict\_Contains() 相同，但 key 被指定为一个 const char\* UTF-8 编码的字节串，而不是 PyObject\*。(由 Victor Stinner 在 [gh-108314](#) 中贡献。)
- 增加了 PyDict\_GetItemRef() 和 PyDict\_GetItemStringRef() 函数，其行为类似于 PyDict\_GetItemWithError()，但将返回一个 strong reference 而不是 borrowed reference。此外，这些函数在出错时将返回 -1，因而不必再检测 PyErr\_Occurred()。(由 Victor Stinner 在 [gh-106004](#) 中贡献。)
- 增加了 PyDict\_SetDefaultRef() 函数，其行为类似于 PyDict\_SetDefault()，但将返回一个 strong reference 而不是 borrowed reference。此函数在出错时将返回 -1，插入时返回 0，而在键已存在于字典中时返回 1。(由 Sam Gross 在 [gh-112066](#) 中贡献。)
- 增加了 PyDict\_Pop() 和 PyDict\_PopString() 函数用于从字典移除键并可选择返回被移除的值。这类似于 dict.pop()，但是没有默认值，且对缺失的键不会引发 KeyError。(由 Stefan Behnel 和 Victor Stinner 在 [gh-111262](#) 中贡献。)

- 增加了 `PyMapping_GetOptionalItem()` 和 `PyMapping_GetOptionalItemString()` 函数分别作为 `PyObject_GetItem()` 和 `PyMapping_GetItemString()` 的替代。这些新函数在映射中缺失所请求的键时不会引发 `KeyError`。这些变体形式在键缺失不应被视为执行失败的场合下更为方便和快速。(由 Serhiy Storchaka 在 [gh-106307](#) 中贡献。)
- 增加了 `PyObject_GetOptionalAttr()` 和 `PyObject_GetOptionalAttrString()` 函数分别作为 `PyObject_GetAttr()` 和 `PyObject_GetAttrString()` 的替代。这些新函数在对象中未找到所请求的属性时不会引发 `AttributeError`。这些变体形式在属性缺失不应被视为执行失败的场合下更为方便和快速。(由 Serhiy Storchaka 在 [gh-106521](#) 中贡献。)
- 增加了 `PyErr_FormatUnraisable()` 函数作为对 `PyErr_WriteUnraisable()` 的扩展，它允许自定义警告消息。(由 Serhiy Storchaka 在 [gh-108082](#) 中贡献。)
- 作为 [PEP 667](#) 的一部分，增加了一组针对帧的 `locals`, `globals` 和 `builtins` 返回 `strong reference` 而不是 `borrowed reference` 的函数：
  - `PyEval_GetFrameBuiltins()` 替代 `PyEval_GetBuiltins()`
  - `PyEval_GetFrameGlobals()` 替代 `PyEval_GetGlobals()`
  - `PyEval_GetFrameLocals()` 替代 `PyEval_GetLocals()`
 (由 Mark Shannon 和 Tian Gao 在 [gh-74929](#) 中贡献。)
- 增加了 `Py_GetConstant()` 和 `Py_GetConstantBorrowed()` 函数用来获取对常量的强引用或借入引用。例如，`Py_GetConstant(Py_CONSTANT_ZERO)` 将返回一个对常量零的强引用。(由 Victor Stinner 在 [gh-115754](#) 中贡献。)
- 增加了 `PyImport_AddModuleRef()` 函数作为 `PyImport_AddModule()` 的替代，它将返回一个 `strong reference` 而不是 `borrowed reference`。(由 Victor Stinner 在 [gh-105922](#) 中贡献。)
- 增加了 `Py_IsFinalizing()` 函数用于检测主 Python 解释器是否正在关闭。(由 Victor Stinner 在 [gh-108014](#) 中贡献。)
- 增加了 `PyList_GetItemRef()` 函数作为 `PyList_GetItem()` 的替代，它将返回一个 `strong reference` 而不是 `borrowed reference`。(由 Sam Gross 在 [gh-114329](#) 中贡献。)
- 增加了 `PyList_Extend()` 和 `PyList_Clear()` 函数，对应于 Python `list.extend()` 和 `list.clear()` 方法。(由 Victor Stinner 在 [gh-111138](#) 中贡献。)
- 增加了 `PyLong_AsInt()` 函数。其行为类似于 `PyLong_AsLong()`，但会将结果存储于一个 `C int` 而不是 `C long`。(由 Victor Stinner 在 [gh-108014](#) 中贡献。)
- 增加了 `PyLong_AsNativeBytes()`，`PyLong_FromNativeBytes()` 和 `PyLong_FromUnsignedNativeBytes()` 等函数以简化原生整数类型与 Python `int` 对象之间的转换。(由 Steve Dower 在 [gh-111140](#) 中贡献。)
- 增加了 `PyModule_Add()` 函数，它类似于 `PyModule_AddObjectRef()` 和 `PyModule_AddObject()`，但总是会偷取一个对值的引用。(由 Serhiy Storchaka 在 [gh-86493](#) 中贡献。)
- 增加了实现 Python 对象的默认哈希函数的 `PyObject_GenericHash()` 函数。(由 Serhiy Storchaka 在 [gh-113024](#) 中贡献。)
- 增加了 `Py_HashPointer()` 函数用于对原始指针执行哈希运算。(由 Victor Stinner 在 [gh-111545](#) 中贡献。)
- 增加了 `PyObject_VisitManagedDict()` 和 `PyObject_ClearManagedDict()` 函数。它们必须由一个使用 `Py_TPFLAGS_MANAGED_DICT` 旗标的类型的遍历和清理函数来调用。[pythoncapi-compat project](#) 可被用于在 Python 3.11 和 3.12 中使用这些函数。(由 Victor Stinner 在 [gh-107073](#) 中贡献。)
- 增加了 `PyRefTracer_SetTracer()` 和 `PyRefTracer_GetTracer()` 函数，它们会以与 `tracemalloc` 模块相同的方式启用对象创建和销毁的追踪。(由 Pablo Galindo 在 [gh-93502](#) 中贡献。)
- 增加 `PySys_AuditTuple()` 函数作为 `PySys_Audit()` 的替代，它将接受一个 Python `tuple` 对象作为事件参数。(由 Victor Stinner 在 [gh-85283](#) 中贡献。)

- 增加 `PyThreadState_GetUnchecked()` 函数作为 `PyThreadState_Get()` 的替代，它在结果为 `NULL` 时不会杀死进程并报告致命错误。调用方要负责检查结果是否为 `NULL`。（由 Victor Stinner 在 [gh-108867](#) 中贡献。）
- 增加 `PyType_GetFullyQualifiedName()` 函数用来获取类型的完整限定名称。如果 `type.__module__` 是一个字符串且不等于 `'builtins'` 或 `'__main__'` 则会在开头添加模块名称。（由 Victor Stinner 在 [gh-111696](#) 中贡献。）
- 增加了 `PyType_GetModuleName()` 函数用来获取类型的模块名称。这等价于获取 `type.__module__` 属性。（由 Eric Snow 和 Victor Stinner 在 [gh-111696](#) 中贡献。）
- 添加了 `PyUnicode_EqualToUTF8AndSize()` 和 `PyUnicode_EqualToUTF8()` 函数以将 `Unicode` 对象与 `const char*` `UTF-8` 编码的字符串进行比较并在它们相等时返回 1 否则返回 0。这些函数不会引发异常。（由 Serhiy Storchaka 在 [gh-110289](#) 中贡献。）
- 增加 `PyWeakref_GetRef()` 函数作为 `PyWeakref_GetObject()` 的替代，它将返回一个 `strong reference` 或是在引用对象不再存活时返回 `NULL`。（由 Victor Stinner 在 [gh-105927](#) 中贡献。）
- 增加了静默地忽略错误的函数的已修正变体形式：
  - `PyObject_HasAttrWithError()` 替代 `PyObject_HasAttr()`。
  - `PyObject_HasAttrStringWithError()` 替代 `PyObject_HasAttrString()`。
  - `PyMapping_HasKeyWithError()` 替代 `PyMapping_HasKey()`。
  - `PyMapping_HasKeyStringWithError()` 替代 `PyMapping_HasKeyString()`。

这些新函数将返回 -1 表示错误而返回标准的 1 表示真值以及 0 表示假值。

（由 Serhiy Storchaka 在 [gh-108511](#) 中贡献。）

## 10.2 被改变的 C API

- 现在 `PyArg_ParseTupleAndKeywords()` 和 `PyArg_VaParseTupleAndKeywords()` 的 `keywords` 形参类型在 C 中为 `char *const*` 而在 C++ 中为 `const char *const*`，而不是 `char**`。在 C++ 中，这将使这些函数与类型为 `const char *const*`、`const char**` 或 `char *const*` 的参数保持兼容而不必使用显式的类型转换。在 C 中，这些函数仅支持类型为 `char *const*` 的参数。这可以通过 `PY_CXX_CONST` 宏来覆盖。（由 Serhiy Storchaka 在 [gh-65210](#) 中贡献。）
- 现在 `PyArg_ParseTupleAndKeywords()` 支持非 ASCII 关键字形参名称。（由 Serhiy Storchaka 在 [gh-110815](#) 中贡献。）
- 现在 `PyCode_GetFirstFree()` 函数属于非稳定 API 并被命名为 `PyUnstable_Code_GetFirstFree()`。（由 Bogdan Romanyuk 在 [gh-115781](#) 中贡献。）
- 现在当调用 `PyDict_GetItem()`、`PyDict_GetItemString()`、`PyMapping_HasKey()`、`PyMapping_HasKeyString()`、`PyObject_HasAttr()`、`PyObject_HasAttrString()` 和 `PySys_GetObject()` 等函数时如果使用 `sys.unraisablehook()` 报告这些错误它们将清空所有已发生的错误。你可以用本文档中推荐的其他函数来替换它们。（由 Serhiy Storchaka 在 [gh-106672](#) 中贡献。）
- 为 `PyUnicode_FromFormat()` 增加了 `%T`、 `%#T`、`%N` 和  `%#N` 格式的支持：
  - `%T`: 获取一个对象类型的完整限定名称
  - `%#T`: 同上，但使用冒号作为分隔符
  - `%N`: 获取一个类型的完整限定名称
  - `%#N`: 同上，但使用冒号作为分隔符

请参阅 [PEP 737](#) 了解详情。（由 Victor Stinner 在 [gh-111696](#) 中贡献。）

- 当在格式编解码器中使用 `#` 时包括 `Python.h` 之前你不必再定义 `PY_SSIZE_T_CLEAN` 宏。接受格式编解码器的 API 总是会使用 `Py_ssize_t` 作为 `#` 格式。（由 Inada Naoki 在 [gh-104922](#) 中贡献。）
- 如果 Python 是使用调试模式或附带断言构建的，`PyTuple_SET_ITEM()` 和 `PyList_SET_ITEM()` 现在将通过一个断言来检查 `index` 参数。（由 Victor Stinner 在 [gh-106168](#) 中贡献。）

## 10.3 受限 C API 的改变

- 下列函数现在被包括在受限 C API 中：

- PyMem\_RawMalloc()
- PyMem\_RawCalloc()
- PyMem\_RawRealloc()
- PyMem\_RawFree()
- PySys\_Audit()
- PySys\_AuditTuple()
- PyType\_GetModuleByDef()

(由 Victor Stinner 在 [gh-85283](#), [gh-85283](#) 和 [gh-116936](#) 中贡献。)

- 使用 `--with-trace-refs` (跟踪引用) 构建的 Python 现在支持受限 API。(由 Victor Stinner 在 [gh-108634](#) 中贡献。)

## 10.4 被移除的 C API

- 移除了一些名称带有 `_Py` 或 `_PY` 前缀 (即被视为私有) 的函数、宏和变量。如果你的项目受到了此项修改的影响并且你认为这些被移除的 API 应当保持可用, 请发起一个新事项以请求提供公有 C API 并向该事项添加 `cc: @vstinner` 来通知 Victor Stinner。(由 Victor Stinner 在 [gh-106320](#) 中贡献。)
- 移除在 Python 3.0 中已弃用的旧缓冲区协议。改用 `bufferobjects`。

- `PyObject_CheckReadBuffer()`: 使用 `PyObject_CheckBuffer()` 来测试对象是否支持缓冲区协议。请注意 `PyObject_CheckBuffer()` 并不保证 `PyObject_GetBuffer()` 会成功。要测试对象是否确实可读, 参见下面的 `PyObject_GetBuffer()` 示例。
- `PyObject_AsCharBuffer()`, `PyObject_AsReadBuffer()`: 改用 `PyObject_GetBuffer()` 和 `PyBuffer_Release()`:

```
Py_buffer view;
if (PyObject_GetBuffer(obj, &view, PyBUF_SIMPLE) < 0) {
    return NULL;
}
// 使用 `view.buf` 和 `view.len` 从缓冲区读取。
// 你可能需要将 `buf` 转换为 `(const char*)view.buf`。
PyBuffer_Release(&view);
```

- `PyObject_AsWriteBuffer()`: 改用 `PyObject_GetBuffer()` 和 `PyBuffer_Release()`:

```
Py_buffer view;
if (PyObject_GetBuffer(obj, &view, PyBUF_WRITABLE) < 0) {
    return NULL;
}
// 使用 `view.buf` 和 `view.len` 向缓冲区写入。
PyBuffer_Release(&view);
```

(由 Inada Naoki 在 [gh-85275](#) 中贡献。)

- 删除了在 Python 3.9 中弃用的各种函数:

- `PyEval_CallObject()`, `PyEval_CallObjectWithKeywords()`: 改用 `PyObject_CallNoArgs()` 或 `PyObject_Call()`。



### 警告

在 `PyObject_Call()` 中，位置参数必须是 `tuple` 且不可为 `NULL`，而关键字参数必须是 `dict` 或 `NULL`，但是被移除的函数会检查参数类型且接受 `NULL` 位置参数和关键字参数。要将 `PyEval_CallObjectWithKeywords(func, NULL, kwargs)` 替换为 `PyObject_Call()`，请使用 `PyTuple_New(0)` 传入一个空元组作为位置参数。

- `PyEval_CallFunction()`: 改用 `PyObject_CallFunction()`。
- `PyEval_CallMethod()`: 改用 `PyObject_CallMethod()`。
- `PyCFunction_Call()`: 改用 `PyObject_Call()`。

(由 Victor Stinner 在 [gh-105107](#) 中贡献。)

- 移除了下列用于配置 Python 初始化的旧函数，它们在 Python 3.11 中已被弃用：

- `PySys_AddWarnOptionUnicode()`: 改用 `PyConfig.warnoptions`。
- `PySys_AddWarnOption()`: 改用 `PyConfig.warnoptions`。
- `PySys_AddXOption()`: 改用 `PyConfig.xoptions`。
- `PySys_HasWarnOptions()`: 改用 `PyConfig.xoptions`。
- `PySys_SetPath()`: 改为设置 `PyConfig.module_search_paths`。
- `Py_SetPath()`: 改为设置 `PyConfig.module_search_paths`。
- `Py_SetStandardStreamEncoding()`: 改为设置 `PyConfig.stdio_encoding`，或者也可以设置 `PyConfig.legacy_windows_stdio` (在 Windows 上)。
- `_Py_SetProgramFullPath()`: 改为设置 `PyConfig.executable`。

改用新的 Python 初始化配置的 `PyConfig` API ([PEP 587](#))，在 Python 3.8 已添加。(由 Victor Stinner 在 [gh-105145](#) 中贡献。)

- 移除 `PyEval_AcquireLock()` 和 `PyEval_ReleaseLock()` 函数，它们在 Python 3.2 中已被弃用。它们不会更新当前线程状态。它们可被替换为：

- `PyEval_SaveThread()` 和 `PyEval_RestoreThread()`；
- 低层级的 `PyEval_AcquireThread()` 和 `PyEval_RestoreThread()`；
- 或者 `PyGILState_Ensure()` 和 `PyGILState_Release()`。

(由 Victor Stinner 在 [gh-105182](#) 中贡献。)

- 删除了在 Python 3.9 弃用的 `c:func:!PyEval_ThreadsInitialized` 函数。自 Python 3.7 起，`Py_Initialize()` 总是创建全局解释器锁：调用 `c:func:!PyEval_InitThreads` 不做任何事情，而 `c:func:!PyEval_ThreadsInitialized` 总返回非零值。(由 Victor Stinner 在 [gh-105182](#) 中贡献。)
- 移除了 `PyInterpreterState_Get()` 的别名 `_PyInterpreterState_Get()`，它曾经因要向下兼容 Python 3.8 而被保留。在 Python 3.8 及更旧版本中可以使用 [pythoncapi-compat project](#) 来获得 `PyInterpreterState_Get()`。(由 Victor Stinner 在 [gh-106320](#) 中贡献。)
- 移除了私有的 `_PyObject_FastCall()` 函数：请使用自 Python 3.8 起增加的 `PyObject_Vectorcall()` ([PEP 590](#))。(由 Victor Stinner 在 [gh-106023](#) 中贡献。)
- 移除了 `cpython/pytime.h` 头文件，它仅包含私有函数。(由 Victor Stinner 在 [gh-106316](#) 中贡献。)
- 从受限 C API 移除了未写入文档的 `PY_TIMEOUT_MAX` 常量。(由 Victor Stinner 在 [gh-110014](#) 中贡献。)
- 移除了旧的 `trashcan` 宏 `Py_TRASHCAN_SAFE_BEGIN` 和 `Py_TRASHCAN_SAFE_END`。将两者替换为新的宏 `Py_TRASHCAN_BEGIN` 和 `Py_TRASHCAN_END`。(由 Irit Katriel 在 [gh-105111](#) 中贡献。)

## 10.5 已弃用的 C API

- 已弃用旧的 Python 初始化函数:

- `PySys_ResetWarnOptions()`: 改为清除 `sys.warnoptions` 和 `warnings.filters`。
- `Py_GetExecPrefix()`: 改为获取 `sys.exec_prefix`。
- `Py_GetPath()`: 改为获取 `sys.path`。
- `Py_GetPrefix()`: 改为获取 `sys.prefix`。
- `Py_GetProgramFullPath()`: 改为获取 `sys.executable`。
- `Py_GetProgramName()`: 改为获取 `sys.executable`。
- `Py_GetPythonHome()`: 改为获取 `PyConfig.home` 或 `PYTHONHOME` 环境变量。

(由 Victor Stinner 在 [gh-105145](#) 中贡献。)

- 软弃用 `PyEval_GetBuiltins()`, `PyEval_GetGlobals()` 和 `PyEval_GetLocals()` 函数, 它们会返回 borrowed reference。(作为 [PEP 667](#) 的一部分被软弃用。)
- 弃用了 `PyImport_ImportModuleNoBlock()` 函数, 它自 Python 3.3 起就只是 `PyImport_ImportModule()` 的别名。(由 Victor Stinner 在 [gh-105396](#) 中贡献。)
- 软弃用 `PyModule_AddObject()` 函数。它应被替换为 `PyModule_Add()` 或 `PyModule_AddObjectRef()`。(由 Serhiy Storchaka 在 [gh-86493](#) 中贡献。)
- 软弃用旧的 `Py_UNICODE` 和 `PY_UNICODE_TYPE` 类型和 `Py_UNICODE_WIDE` 定义。改为直接使用 `wchar_t` 类型。自 Python 3.3 起, `Py_UNICODE` 和 `PY_UNICODE_TYPE` 就只是 `wchar_t` 的别名。(由 Victor Stinner 在 [gh-105156](#) 中贡献。)
- 弃用 `PyWeakref_GetObject()` 和 `PyWeakref_GET_OBJECT()` 函数, 它们都是返回 borrowed reference。将它们替换为新的 `PyWeakref_GetRef()` 函数, 它是返回 strong reference。在 Python 3.12 或更旧的版本中可以使用 [pythoncapi-compat project](#) 来获取 `PyWeakref_GetRef()`。(由 Victor Stinner 在 [gh-105927](#) 中贡献。)

### 计划在 Python 3.14 中移除

- `PyDictObject` 中的 `ma_version_tag` 字段用于扩展模块 ([PEP 699](#); [gh-101193](#))。
- 创建 immutable types 的可变基础 ([gh-95388](#))。
- 用于配置 Python 的初始化的函数, 在 Python 3.11 中已弃用:
  - `PySys_SetArgvEx()`: 改为设置 `PyConfig.argv`。
  - `PySys_SetArgv()`: 改为设置 `PyConfig.argv`。
  - `Py_SetProgramName()`: 改为设置 `PyConfig.program_name`。
  - `Py_SetPythonHome()`: 改为设置 `PyConfig.home`。

`Py_InitializeFromConfig()` API 应与 `PyConfig` 一起使用。

- 全局配置变量:

- `Py_DebugFlag`: 改用 `PyConfig.parser_debug`。
- `Py_VerboseFlag`: 改用 `PyConfig.verbose`。
- `Py_QuietFlag`: 改用 `PyConfig.quiet`。
- `Py_InteractiveFlag`: 改用 `PyConfig.interactive`。
- `Py_InspectFlag`: 改用 `PyConfig.inspect`。
- `Py_OptimizeFlag`: 改用 `PyConfig.optimization_level`。
- `Py_NoSiteFlag`: 改用 `PyConfig.site_import`。
- `Py_BytesWarningFlag`: 改用 `PyConfig.bytes_warning`。

- `Py_FrozenFlag`: 改用 `PyConfig.pathconfig_warnings`。
- `Py_IgnoreEnvironmentFlag`: 改用 `PyConfig.use_environment`。
- `Py_DontWriteBytecodeFlag`: 改用 `PyConfig.write_bytecode`。
- `Py_NoUserSiteDirectory`: 改用 `PyConfig.user_site_directory`。
- `Py_UnbufferedStdioFlag`: 改用 `PyConfig.buffered_stdio`。
- `Py_HashRandomizationFlag`: 改用 `PyConfig.use_hash_seed` 和 `PyConfig.hash_seed`。
- `Py_IsolatedFlag`: 改用 `PyConfig.isolated`。
- `Py_LegacyWindowsFSEncodingFlag`: 改用 `PyPreConfig.legacy_windows_fs_encoding`。
- `Py_LegacyWindowsStdioFlag`: 改用 `PyConfig.legacy_windows_stdio`。
- `Py_FileSystemDefaultEncoding`: 改用 `PyConfig.filesystem_encoding`。
- `Py_HasFileSystemDefaultEncoding`: 改用 `PyConfig.filesystem_encoding`。
- `Py_FileSystemDefaultEncodeErrors`: 改用 `PyConfig.filesystem_errors`。
- `Py_UTF8Mode`: 改用 `PyPreConfig.utf8_mode`。(参见 `Py_PreInitialize()`)

`Py_InitializeFromConfig()` API 应与 `PyConfig` 一起使用。

### Python 3.15 中的待移除功能

- 捆绑的 `libmpdecimal` 副本。
- `The PyImport_ImportModuleNoBlock()`: 改用 `PyImport_ImportModule()`。
- `PyWeakref_GetObject()` 和 `PyWeakref_GET_OBJECT()`: 改用 `PyWeakref_GetRef()`。
- `Py_UNICODE` 类型和 `Py_UNICODE_WIDE` 宏: 改用 `wchar_t`。
- Python 初始化函数
  - `PySys_ResetWarnOptions()`: 改为清除 `sys.warnoptions` 和 `warnings.filters`。
  - `Py_GetExecPrefix()`: 改为获取 `sys.base_exec_prefix` 和 `sys.exec_prefix`。
  - `Py_GetPath()`: 改为获取 `sys.path`。
  - `Py_GetPrefix()`: 改为获取 `sys.base_prefix` 和 `sys.prefix`。
  - `Py_GetProgramFullPath()`: 改为获取 `sys.executable`。
  - `Py_GetProgramName()`: 改为获取 `sys.executable`。
  - `Py_GetPythonHome()`: 改为获取 `PyConfig.home` 或 `PYTHONHOME` 环境变量。

### 计划在未来版本中移除

以下 API 已被弃用，将被移除，但目前尚未确定移除日期。

- `Py_TPFLAGS_HAVE_FINALIZE`: 自 Python 3.8 起不再需要。
- `PyErr_Fetch()`: 改用 `PyErr_GetRaisedException()`。
- `PyErr_NormalizeException()`: 改用 `PyErr_GetRaisedException()`。
- `PyErr_Restore()`: 改用 `PyErr_SetRaisedException()`。
- `PyModule_GetFilename()`: 改用 `PyModule_GetFilenameObject()`。
- `PyOS_AfterFork()`: 改用 `PyOS_AfterFork_Child()`。
- `PySlice_GetIndicesEx()`: 改用 `PySlice_Unpack()` 和 `PySlice_AdjustIndices()`。
- `PyUnicode_AsDecodedObject()`: 改用 `PyCodec_Decode()`。
- `PyUnicode_AsDecodedUnicode()`: 改用 `PyCodec_Decode()`。

- `PyUnicode_AsEncodedObject()`: 改用 `PyCodec_Encode()`。
- `PyUnicode_AsEncodedUnicode()`: 改用 `PyCodec_Encode()`。
- `PyUnicode_READY()`: 自 Python 3.12 起不再需要
- `PyErr_Display()`: 改用 `PyErr_DisplayException()`。
- `_PyErr_ChainExceptions()`: 改用 `_PyErr_ChainExceptions1()`。
- `PyBytesObject.ob_shash` 成员: 改为调用 `PyObject_Hash()`。
- `PyDictObject.ma_version_tag` 成员。
- 线程本地存储 (TLS) API:
  - `PyThread_create_key()`: 改用 `PyThread_tss_alloc()`。
  - `PyThread_delete_key()`: 改用 `PyThread_tss_free()`。
  - `PyThread_set_key_value()`: 改用 `PyThread_tss_set()`。
  - `PyThread_get_key_value()`: 改用 `PyThread_tss_get()`。
  - `PyThread_delete_key_value()`: 改用 `PyThread_tss_delete()`。
  - `PyThread_ReInitTLS()`: 自 Python 3.7 起不再需要。

## 11 构建变化

- 现在 `arm64-apple-ios` 和 `arm64-apple-ios-simulator` 都是 **PEP 11** 第 3 层级的平台。(PEP 730 由 Russell Keith-Magee 编写并在 [gh-114099](#) 中贡献实现。)
- 现在 `aarch64-linux-android` 和 `x86_64-linux-android` 都是 **PEP 11** 第 3 层级的平台。(PEP 738 由 Malcolm Smith 编写并在 [gh-116622](#) 中贡献实现。)
- 现在 `wasm32-wasi` 是 **PEP 11** 第 2 层级的平台。(由 Brett Cannon 在 [gh-115192](#) 中贡献。)
- `wasm32-emsripten` 不再是 **PEP 11** 的受支持平台。(由 Brett Cannon 在 [gh-115192](#) 中贡献。)
- 现在构建 CPython 需要带有 C11 atomic 库支持的编译器、GCC 内置 atomic 函数或 MSVC 互锁内生函数。
- 现在需要有 `autoconf 2.71` 和 `aclocal 1.16.5` 才能重新生成 `configure` 脚本。(由 Christian Heimes 在 [gh-89886](#) 中并由 Victor Stinner 在 [gh-112090](#) 中贡献。)
- 需要 SQLite 3.15.2 或更新的版本才能构建 `sqlite3` 扩展模块。(由 Erlend Aasland 在 [gh-105875](#) 中贡献。)
- 现在 CPython 默认会捆绑 `mimalloc library`。它使用 MIT 许可证提供许可; 请参阅 `mimalloc license`。捆绑的 `mimalloc` 带有定制的修改, 详情参见 [gh-113141](#)。(由 Dino Viehland 在 [gh-109914](#) 中贡献。)
- 现在 `configure` 选项 `--with-system-libmpdec` 默认为 `yes`。捆绑的 `libmpdecimal` 副本将在 Python 3.15 中被移除。
- 使用 `configure --with-trace-refs` (跟踪引用) 构建的 Python 现在与 Python 发布构建版和 调试构建版是 ABI 兼容的。(由 Victor Stinner 在 [gh-108634](#) 中贡献。)
- 在 POSIX 系统上, `pkg-config(.pc)` 文件名现在会包括 ABI 旗标。例如, 自由线程构建版将生成 `python-3.13t.pc` 而调试构建版将生成 `python-3.13d.pc`。
- 现在 `errno`, `fcntl`, `grp`, `md5`, `pwd`, `resource`, `termios`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshm`, `_scproxy`, `_stat`, `_statistics`, `_testconsole`, `_testimportmultiple` 和 `_uuid` C 扩展是使用受限 C API 构建的。(由 Victor Stinner 在 [gh-85283](#) 中贡献。)

## 12 移植到 Python 3.13

本节列出了先前描述的更改以及可能需要更改代码的其他错误修正。

### 12.1 Python API 的变化

- **PEP 667** 引入了对 `locals()` 和 `f_locals` 语义的多项更改：
  - 在 **optimized scope** 中调用 `locals()` 时，每次调用都会生成一个独立的快照，因此不再隐式更新之前返回的引用。要获得以往版本 CPython 的行为，现需显式调用以使用后来调用 `func:locals` 得到的结果来更新初次调用返回的字典。隐式使用 `func:locals` 的代码执行函数（如 `exec` 和 `eval`）必须传入一个显式命名空间，才能在已优化的作用域中访问其结果。（此更改为 **PEP 667** 的一部分。）
  - 从模块作用域或类作用域的推导式中调用 `locals`（包括 `exec()` 或 `eval` 导致的间接调用）时，行为再次表现得像是推导式作为独立的嵌套函数运行（即不包含外部作用域的局部变量）。在 Python 3.12 中，此场景下的行为曾按照 **PEP 709** 更改为包含存放推导式的作用域的局部变量。（此更改为 **PEP 667** 的一部分。）
  - 在 **optimized scope** 中访问 `FrameType.f_locals` 现在会返回一个写入代理，而不是一个在不明时间更新的快照。如果需要快照，必须使用 `dict` 或该代理的 `.copy()` 方法显式地创建。（此修改是 **PEP 667** 的一部分。）
- 当作为方法使用时，`functools.partial` 现在会发出 `FutureWarning`。该行为将在 Python 的未来版本中更改。如果想保留旧行为，请将其包装在 `staticmethod()` 中。（由 Serhiy Storchaka 在 [gh-121027](#) 贡献。）
- 现在当无法获取到用户名时 `getpass.getuser()` 将会引发 `OSError`，而不是在非 Unix 平台上引发 `ImportError` 或因密码数据库为空在 Unix 平台上引发 `KeyError`。
- `gzip.GzipFile` 的 `mode` 属性值现在是字符串（`'rb'` 或 `'wb'`）而不是整数（1 或 2）。`zipfile.ZipFile.open()` 返回的可读文件型对象的 `mode` 属性值现在是 `'rb'`，而非 `'r'`。（由 Serhiy Storchaka 在 [gh-115961](#) 贡献。）
- `mailbox.Maildir` 现在会忽略以点（`.`）开头的文件。（由 Zackery Spytz 在 [gh-65559](#) 贡献。）
- `pathlib.Path.glob()` 和 `rglob()` 现在会在匹配模式以 `“**”` 结尾时同时返回文件和目录，而不仅仅是目录。若要保持之前的行为，仅匹配目录，请添加尾部斜杠。
- `threading` 模块现在期望 `_thread` 模块具有 `_is_main_interpreter()` 函数。该函数不接受任何参数，如果当前解释器是主解释器，则返回 `True`。  
与模块的其他“私有”属性一样，任何提供自定义 `_thread` 模块的库或应用都必须提供 `_is_main_interpreter()`。（[gh-112826](#)。）

### 12.2 C API 的变化

- `Python.h` 不再包括 `<ieeefp.h>` 标准头文件。它曾经为了 `finite()` 函数被包括，该函数现在是由 `<math.h>` 头文件提供的。现在如有必要应当显式地包括它。此外还移除了 `HAVE_IEEEFP_H` 宏。（由 Victor Stinner 在 [gh-108765](#) 中贡献。）
- `Python.h` 不再包括这些标准头文件：`<time.h>`，`<sys/select.h>` 和 `<sys/time.h>`。如果需要，它们应当被显式地包括。例如，`<time.h>` 提供 `clock()` 和 `gmtime()` 函数，`<sys/select.h>` 提供 `select()` 函数，而 `<sys/time.h>` 提供 `the futimes()`，`gettimeofday()` 和 `setitimer()` 函数。（由 Victor Stinner 在 [gh-108765](#) 中贡献。）
- 在 Windows 上，`Python.h` 不再包括 `<stddef.h>` 标准头文件。如果需要，它应当被显式地包括。例如，它提供 `offsetof()` 函数，以及 `size_t` 和 `ptrdiff_t` 类型。显式地包括 `<stddef.h>` 在所有其他平台都已经是必须的，`HAVE_STDDEF_H` 宏仅在 Windows 上被定义。（由 Victor Stinner 在 [gh-108765](#) 中贡献。）
- 如果 `Py_LIMITED_API` 宏已被定义，则 `Py_BUILD_CORE`，`Py_BUILD_CORE_BUILTIN` 和 `Py_BUILD_CORE_MODULE` 宏现在会被 `<Python.h>` 撤销定义。（由 Victor Stinner 在 [gh-85283](#) 中贡献。）

- 旧的 `trashcan` 宏 `Py_TRASHCAN_SAFE_BEGIN` 和 `Py_TRASHCAN_SAFE_END` 已被移除。它们应该被替换为新的宏 `Py_TRASHCAN_BEGIN` 和 `Py_TRASHCAN_END`。

带有旧版宏的 `tp_dealloc` 函数，例如：

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_SAFE_BEGIN(p);
    ...
    Py_TRASHCAN_SAFE_END
}
```

应当按照以下方式迁移到新版宏：

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    Py_TRASHCAN_END
}
```

请注意 `Py_TRASHCAN_BEGIN` 具有第二个参数，其值应为它所在的析构函数。这些新宏是在 Python 3.8 中添加而旧的宏是在 Python 3.11 中弃用的。（由 Irit Katriel 在 [gh-105111](#) 中贡献。）

- [PEP 667](#) 引入了对帧相关函数的多项更改：
  - 在一个 `optimized scope` 中修改从 `PyEval_GetLocals()` 的字典的效果已被改变。以这种方式新加的字典条目现在将 仅对该帧内的后续 `PyEval_GetLocals()` 调用是可见的，因为 `PyFrame_GetLocals()`、`locals()` 和 `FrameType.f_locals` 不会再访问相同的底层已缓存字典。针对实际变量名称所做的条目修改和通过直通写入代理接口添加的名称在该帧内对 `PyEval_GetLocals()` 的后续调用中被覆盖。推荐的代码更新取决于该函数是如何被使用的，因此请参考函数的弃用通知了解详情。
  - 在 `optimized scope` 中调用 `PyFrame_GetLocals()` 现在将返回一个直通写入代理而不是一个在不明确的时刻更新的快照。如果需要快照，必须显式地创建它（例如使用 `PyDict_Copy()`），或是通过调用新的 `PyEval_GetFrameLocals()` API。
  - `PyFrame_FastToLocals()` 和 `PyFrame_FastToLocalsWithError()` 将不再有任何效果。自 Python 3.11 起调用这些函数是多余的，因为该版本首次引入了 `PyFrame_GetLocals()`。
  - `PyFrame_LocalsToFast()` 将不再有任何效果。现在调用此函数是多余的因为 `PyFrame_GetLocals()` 将为已优化作用域返回一个直通写入代理。
- Python 3.13 移除了许多私有函数。其中一些可以用下列的替代：
  - `_PyDict_Pop()`: `PyDict_Pop()` 或 `PyDict_PopString()`;
  - `_PyDict_GetItemWithError()`: `PyDict_GetItemRef()`;
  - `_PyErr_WriteUnraisableMsg()`: `PyErr_FormatUnraisable()`;
  - `_PyEval_SetTrace()`: `PyEval_SetTrace()` 或 `PyEval_SetTraceAllThreads()`;
  - `_PyList_Extend()`: `PyList_Extend()`;
  - `_PyLong_AsInt()`: `PyLong_AsInt()`;
  - `_PyMem_RawStrdup()`: `strdup()`;
  - `_PyMem_Strdup()`: `strdup()`;
  - `_PyObject_ClearManagedDict()`: `PyObject_ClearManagedDict()`;
  - `_PyObject_VisitManagedDict()`: `PyObject_VisitManagedDict()`;



- `_PyThreadState_UncheckedGet()`: `PyThreadState_GetUnchecked()`;
- `_PyTime_AsSecondsDouble()`: `PyTime_AsSecondsDouble()`;
- `_PyTime_GetMonotonicClock()`: `PyTime_Monotonic()` 或 `PyTime_MonotonicRaw()`;
- `_PyTime_GetPerfCounter()`: `PyTime_PerfCounter()` 或 `PyTime_PerfCounterRaw()`;
- `_PyTime_GetSystemClock()`: `PyTime_Time()` 或 `PyTime_TimeRaw()`;
- `_PyTime_MAX`: `PyTime_MAX`;
- `_PyTime_MIN`: `PyTime_MIN`;
- `_PyTime_t`: `PyTime_t`;
- `_Py_HashPointer()`: `Py_HashPointer()`;
- `_Py_IsFinalizing()`: `Py_IsFinalizing()`。

在 Python 3.12 和更旧的版本中可以使用 [pythoncapi-compat project](#) 来充分利用这些新函数。

## 13 回归测试的变化

- 现在使用 `configure --with-pydebug` 编译的 Python 将支持 `-X presite=package.module` 命令行选项。如果被使用，它指明一个模块应当在解释器生命周期开始时，即 `site.py` 被执行之前被导入。（由 Łukasz Langa 在 [gh-110769](#) 中贡献。）

## 14 3.13.1 中的重要变化

### 14.1 sys

- 之前未写入文档的特殊函数 `sys.getobjects()`，它仅存在于某些专用的 Python 构建版，现在可以从其他解释器而非调用它的解释器返回对象。

# 索引

## 非字母

### 环境变量

PYTHON\_BASIC\_REPL, 5  
PYTHON\_COLORS, 4, 5, 12  
PYTHON\_CPU\_COUNT, 13  
PYTHON\_FROZEN\_MODULES, 8  
PYTHON\_GIL, 6  
PYTHON\_HISTORY, 8  
PYTHON\_PERF\_JIT\_SUPPORT, 8  
PYTHONHOME, 34, 35  
PYTHONLEGACYWINDOWSFSENCODING, 23, 26

## C

### Common Vulnerabilities and Exposures

CVE 2023-27043, 12  
CVE 2023-52425, 17  
CVE 2024-4030, 14, 16

## P

Python 增强建议; PEP 11, 8, 36  
Python 增强建议; PEP 11#tier-2, 4  
Python 增强建议; PEP 11#tier-3, 4  
Python 增强建议; PEP 587, 33  
Python 增强建议; PEP 590, 33  
Python 增强建议; PEP 594, 3, 18  
Python 增强建议; PEP 602, 5  
Python 增强建议; PEP 626, 25  
Python 增强建议; PEP 667, 3, 7, 34, 37  
Python 增强建议; PEP 669, 4, 28  
Python 增强建议; PEP 696, 4  
Python 增强建议; PEP 699, 34  
Python 增强建议; PEP 702, 4, 17  
Python 增强建议; PEP 703, 3, 6  
Python 增强建议; PEP 705, 4, 17  
Python 增强建议; PEP 709, 37  
Python 增强建议; PEP 719, 3  
Python 增强建议; PEP 730, 4, 8  
Python 增强建议; PEP 737, 31  
Python 增强建议; PEP 738, 4, 8  
Python 增强建议; PEP 742, 4, 17  
Python 增强建议; PEP 744, 3, 7  
PYTHON\_BASIC\_REPL, 5  
PYTHON\_COLORS, 4, 5, 12  
PYTHON\_CPU\_COUNT, 13  
PYTHON\_FROZEN\_MODULES, 8  
PYTHON\_GIL, 6  
PYTHON\_HISTORY, 8  
PYTHON\_PERF\_JIT\_SUPPORT, 8  
PYTHONHOME, 34, 35  
PYTHONLEGACYWINDOWSFSENCODING, 23, 26

## R

### RFC

RFC 5280, 15