
What's New in Python

发布 3.12.0rc3

A. M. Kuchling

十月 01, 2023

Python Software Foundation
Email: docs@python.org

Contents

1	摘要 -- 发布重点	3
2	新的特性	4
2.1	PEP 695: 类型形参语法	4
2.2	PEP 701: f-字符串的句法形式化	5
2.3	PEP 684: 解释器级 GIL	6
2.4	PEP 669: 针对 CPython 的低影响监控	7
2.5	PEP 688: 使缓冲区协议在 Python 中可访问	7
2.6	PEP 709: 推导式内联	7
2.7	改进的错误消息	7
3	有关类型提示的新增特性	8
3.1	PEP 692: 使用 TypedDict 进行更精确的 **kwargs 类型标注	8
3.2	PEP 698: 覆盖静态类型的装饰器	9
4	其他语言特性修改	9
5	新增模块	10
6	改进的模块	10
6.1	array	10
6.2	asyncio	10
6.3	calendar	11
6.4	csv	11
6.5	dis	11
6.6	fractions	11
6.7	importlib.resources	11
6.8	inspect	11
6.9	itertools	12
6.10	math	12
6.11	os	12
6.12	os.path	12
6.13	pathlib	12
6.14	pdb	13
6.15	random	13
6.16	shutil	13
6.17	sqlite3	13
6.18	statistics	13

6.19	sys	14
6.20	tempfile	14
6.21	threading	14
6.22	tkinter	14
6.23	tokenize	14
6.24	types	15
6.25	typing	15
6.26	unicodedata	15
6.27	unittest	16
6.28	uuid	16
7	性能优化	16
8	CPython 字节码的改变	16
9	演示和工具	17
10	弃用	17
10.1	计划在 Python 3.13 中移除	19
10.2	计划在 Python 3.14 中移除	20
10.3	计划在未来版本中移除	21
11	移除	21
11.1	asynchat 和 asyncore	21
11.2	configparser	22
11.3	distutils	22
11.4	ensurepip	22
11.5	enum	22
11.6	ftplib	22
11.7	gzip	22
11.8	hashlib	23
11.9	importlib	23
11.10	imp	23
11.11	io	24
11.12	locale	24
11.13	sqlite3	24
11.14	ssl	25
11.15	unittest	25
11.16	webbrowser	26
11.17	xml.etree.ElementTree	26
11.18	zipimport	26
11.19	其他事项	26
12	移植到 Python 3.12	26
12.1	Python API 的变化	26
13	构建变化	27
14	C API 的变化	28
14.1	新的特性	28
14.2	移植到 Python 3.12	30
14.3	弃用	31
14.4	移除	35
	索引	36

本文介绍 Python 3.12 相比 3.11 增加的新特性。Python 3.12 将于 2023 年 10 月 2 日发布。要获取详细信息，可参阅 changelog。

参见：

[PEP 693](#) -- Python 3.12 发布计划

1 摘要 -- 发布重点

Python 3.12 is the latest stable release of the Python programming language, with a mix of changes to the language and the standard library. The library changes focus on cleaning up deprecated APIs, usability, and correctness. Of note, the `distutils` package has been removed from the standard library. Filesystem support in `os` and `pathlib` has seen a number of improvements, and several modules have better performance.

The language changes focus on usability, as f-strings have had many limitations removed and 'Did you mean ...' suggestions continue to improve. The new *type parameter syntax* and `type` statement improve ergonomics for using generic types and type aliases with static type checkers.

This article doesn't attempt to provide a complete specification of all new features, but instead gives a convenient overview. For full details, you should refer to the documentation, such as the Library Reference and Language Reference. If you want to understand the complete implementation and design rationale for a change, refer to the PEP for a particular new feature; but note that PEPs usually are not kept up-to-date once a feature has been fully implemented.

新的语法特性：

- [PEP 695](#)，类型形参语法和 `type` 语句

新的语法特性：

- [PEP 701](#)，语法中的 f-字符串

解释器的改进：

- [PEP 684](#)，解释器级的单独 GIL
- [PEP 669](#)，低开销的监控
- 针对 `NameError`, `ImportError` 和 `SyntaxError` 异常改进了 'Did you mean ...' 提示消息。

对 Python 数据模型的改进：

- [PEP 688](#)，使用 Python 的 缓冲区协议

标准库中的重大改进：

- `pathlib.Path` 类现在支持子类化
- `os` 模块获得了多项针对 Windows 支持的改进
- 在 `sqlite3` 模块中添加了 命令行界面。
- 基于 运行时可检测协议的 `isinstance()` 检测获得了 2 至 20 倍的提速
- `asyncio` 包的性能获得了多项改进，一些基准测试显示有 75% 的提速。
- A command-line interface has been added to the `uuid` module
- Due to the changes in [PEP 701](#), producing tokens via the `tokenize` module is up to up to 64% faster.

安全改进：

- Replace the builtin `hashlib` implementations of SHA1, SHA3, SHA2-384, SHA2-512, and MD5 with formally verified code from the [HACL*](#) project. These builtin implementations remain as fallbacks that are only used when OpenSSL does not provide them.

C API 的改进：

- [PEP 697](#), unstable C API tier
- [PEP 683](#), immortal objects

CPython 实现的改进：

- [PEP 709](#), comprehension inlining
- CPython support for the Linux perf profiler
- Implement stack overflow protection on supported platforms

新的类型标注特性：

- [PEP 692](#), using TypedDict to annotate ****kwargs**
- [PEP 698](#), `typing.override()` decorator

重要的弃用、移除或限制：

- **PEP 623**: Remove `wstr` from Unicode objects in Python's C API, reducing the size of every `str` object by at least 8 bytes.
- **PEP 632**: Remove the `distutils` package. See [the migration guide](#) for advice replacing the APIs it provided. The third-party [Setuptools](#) package continues to provide `distutils`, if you still require it in Python 3.12 and beyond.
- [gh-95299](#): Do not pre-install `setuptools` in virtual environments created with `venv`. This means that `distutils`, `setuptools`, `pkg_resources`, and `easy_install` will no longer be available by default; to access these run `pip install setuptools` in the activated virtual environment.
- The `asynchat`, `asyncore`, and `imp` modules have been removed, along with several `unittest.TestCase` [method aliases](#).

2 新的特性

2.1 PEP 695: 类型形参语法

PEP 484 下的泛型类和函数是使用详细语法声明的，这使得类型参数的范围不明确，并且需要显式声明方差。

PEP 695 引入了一种新的、更紧凑、更明确的方式来创建 泛型类和 函数：

```
def max[T](args: Iterable[T]) -> T:
    ...

class list[T]:
    def __getitem__(self, index: int, /) -> T:
        ...

    def append(self, element: T) -> None:
        ...
```

此外，该 PEP 引入了一种新的方法来使用 `type` 语句声明 类型别名，该语句会创建 `TypeAliasType` 的实例：

```
type Point = tuple[float, float]
```

类型别名也可以是 generic：

```
type Point[T] = tuple[T, T]
```

新语法允许声明 `TypeVarTuple` 和 `ParamSpec` 形参，以及带边界或约束的 `TypeVar` 形参：


```
>>> print(f"This is the playlist: {\n".join(songs)}")
This is the playlist: Take me back to Eden
Alkaline
Ascensionism
>>> print(f"This is the playlist: {\n{BLACK HEART SUIT}}".join(songs)}")
This is the playlist: Take me back to Eden♥Alkaline♥Ascensionism
```

更多细节请参见 [PEP 701](#)。

实现此功能的一个正面的附带影响是(通过使用 [PEG 解析器](#) 来解析 f-字符串), 现在 f-字符串的错误消息会更加精确, 包括错误的确切位置。例如, 在 Python 3.11 中, 下面的 f-字符串将引发一个 `SyntaxError`:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
File "<stdin>", line 1
    (x z y)
      ^^^
SyntaxError: f-string: invalid syntax. Perhaps you forgot a comma?
```

但是错误消息不包括错误在行中的确切位置, 而且表达式被人为地用括号括起来。在 Python 3.12 中, 由于 f-字符串是用 [PEG 解析器](#) 解析的, 因此错误消息可以更精确, 并显示整行:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
File "<stdin>", line 1
    my_string = f"{x z y}" + f"{1 + 1}"
                  ^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

(由 Pablo Galindo、Batuhan Taskaya、Lysandros Nikolaou、Cristián Maureira-Fredes 和 Marta Gómez 在 [gh-102856](#) 中贡献。PEP 由 Pablo Galindo、Batuhan Taskaya、Lysandros Nikolaou 和 Marta Gómez 撰写)。

2.3 PEP 684: 解释器级 GIL

[PEP 684](#) 引入了解释器级的 GIL, 使得现在可以创建带有独立的解释器级 GIL 的子解释器。这将允许 Python 程序充分利用多个 CPU 核心。此特性目前仅能通过 C-API 使用, 不过相应的 Python API [预计将在 3.13 中添加](#)。

使用新的 `Py_NewInterpreterFromConfig()` 函数来创建具有单独 GIL 的解释器:

```
PyInterpreterConfig config = {
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tstate, &config);
if (PyStatus_Exception(status)) {
    return -1;
}
/* The new interpreter is now active in the current thread. */
```

有关如何将 C-API 用于子解释器和解释器级 GIL 的更多示例, 请参见 `Modules/_xxsubinterpretersmodule.c`。

(由 Eric Snow 在 [gh-104210](#) 等中贡献。)

2.4 PEP 669: 针对 CPython 的低影响监控

PEP 669 定义了一个新的 API 用于性能分析器、调试器和其他在 CPython 中监控事件的工具。它覆盖了大范围的事件，包括调用、返回、行、异常、跳转等等。这意味着你将只为你所使用的东西付出开销，提供了对近乎零开销的调试器和覆盖工具的支持。请参阅 `sys.monitoring` 了解详情。

(Contributed by Mark Shannon in [gh-103082](#).)

2.5 PEP 688: 使缓冲区协议在 Python 中可访问

PEP 688 引入了一种在 Python 代码中使用 缓冲区协议的方法。实现 `__buffer__()` 方法的类现在可以作为缓冲区类型使用。

新的 `collections.abc.Buffer ABC` (抽象基类) 提供了一种表示缓冲区对象的标准方法，例如在类型注释中。新的 `inspect.BufferFlags` 枚举表示可用于自定义缓冲区创建的标志。(由 Jelle Zijlstra 在 [gh-102500](#) 中贡献。)

2.6 PEP 709: 推导式内联

字典、列表和集合推导式现在都是内联的，而不是为每次执行推导式都创建一个新的一次性函数对象。这样可以将推导式的执行速度提高最多两倍。更多细节请参阅 **PEP 709**。

推导式迭代变量将保持隔离而不会覆盖外作用域中的同名变量，在离开推导式后也不再可见。内联确实会导致一些可见的行为变化：

- 回溯中的推导式不再有单独的帧，跟踪/评测也不再将推导式显示为函数调用。
- `symtable` 模块将不再为每个推导式产生子符号表；取而代之的是，推导式的 `locals` 将包括在父函数的符号表中。
- 在推导式内部调用 `locals()` 现在包括该推导式外部外部的变量，而不再包括推导式“参数”导致的 `.0` 合成变量。
- 一个直接迭代 `locals()` 的推导式 (例如 `[k for k in locals()]`) 在启动追踪 (例如检测代码覆盖度) 的情况下运行时可能导致 `RuntimeError: dictionary changed size during iteration`。此行为与现有的 `for k in locals():` 等代码保持一致。要避免此错误，可先创建一个由键组成的列表用于迭代: `keys = list(locals()); [k for k in keys]`。

(由 Carl Meyer 和 Vladimir Matveev 在 **PEP 709** 中贡献。)

2.7 改进的错误消息

- 当引发的 `NameError` 传播到最高层级时，解释器显示的错误消息可能将标准库中的模块作为建议的一部分。(由 Pablo Galindo 在 [gh-98254](#) 中贡献。)

```
>>> sys.version_info
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined. Did you forget to import 'sys'?
```

- 改进针对实例的 `NameError` 异常的错误建议。现在如果在方法中引发了 `NameError` 而实例具有与异常中的名称完全相同的属性，建议将会包括 `self.<NAME>` 而不是方法作用域中最接近的匹配项。(由 Pablo Galindo 在 [gh-99139](#) 中贡献。)

```
>>> class A:
...     def __init__(self):
...         self.blech = 1
...
...     def foo(self):
```

(下页继续)

(续上页)

```
...     somethin = blech
...
>>> A().foo()
File "<stdin>", line 1
    somethin = blech
    ^^^^^
NameError: name 'blech' is not defined. Did you mean: 'self.blech'?
```

- 改进了当用户输入 `import x from y` 而不是 `from y import x` 时产生的 `SyntaxError` 错误消息。(由 Pablo Galindo 在 [gh-98931](#) 中贡献。)

```
>>> import a.y.z from b.y.z
File "<stdin>", line 1
    import a.y.z from b.y.z
    ^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Did you mean to use 'from ... import ...' instead?
```

- 由失败的 `from <module> import <name>` 语句引发的 `ImportError` 异常现在会包括根据 `<module>` 中的可用名称对 `<name>` 的值提出的建议。(由 Pablo Galindo 在 [gh-91058](#) 中贡献。)

```
>>> from collections import chainmap
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'chainmap' from 'collections'. Did you mean:
↳ 'ChainMap'?
```

3 有关类型提示的新增特性

本节介绍了影响 **类型提示** 和 `typing` 模块的主要更改。

3.1 PEP 692: 使用 `TypedDict` 进行更精确的 `**kwargs` 类型标注

在函数签名中的 `**kwargs` 类型标注 (由 [PEP 484](#) 引入) 只允许在所有 `**kwargs` 都属于同一类型的情况下进行有效标注。

PEP 692 通过依赖类型化的字典规定了一种更精确的针对 `**kwargs` 的类型标注方式:

```
from typing import TypedDict, Unpack

class Movie(TypedDict):
    name: str
    year: int

def foo(**kwargs: Unpack[Movie]): ...
```

更多细节请参见 [PEP 692](#)。

(由 Franek Magiera 在 [gh-103629](#) 中贡献。)

3.2 PEP 698: 覆盖静态类型的装饰器

一个新的装饰器 `typing.override()` 已添加到 `typing` 模块中。它向类型检查器指示该方法旨在重写超类中的方法。这允许类型检查器在打算重写基类中的某个方法实际上没有重写的情况下捕获错误。

示例:

```
from typing import override

class Base:
    def get_color(self) -> str:
        return "blue"

class GoodChild(Base):
    @override # ok: overrides Base.get_color
    def get_color(self) -> str:
        return "yellow"

class BadChild(Base):
    @override # type checker error: does not override Base.get_color
    def get_colour(self) -> str:
        return "red"
```

更多细节参见 [PEP 698](#)。

(由 Steven Troxler 在 [gh-101561](#) 中贡献。)

4 其他语言特性修改

- 解析器现在在解析包含空字节的源代码时引发 `SyntaxError`。(由 Pablo Galindo 在 [gh-96670](#) 中贡献。)
- 不是有效转义序列的反斜杠加字符组合现在会生成 `SyntaxWarning`, 而不是 `DeprecationWarning`。例如, `re.compile("\d+\\.\\d+")` 现在会发出 `SyntaxWarning("\d" 是一个无效的转义序列, 请使用原始字符串来表示正则表达式: re.compile(r"\d+\\.\\d+")`)。在未来的 Python 版本中, 最终将引发 `SyntaxError`, 而不是 `SyntaxWarning`。(由 Victor Stinner 在 [gh-98401](#) 中贡献。)
- 值大于 `0o377` (例如: `"\477"`) 的八进制转义序列, 在 Python 3.11 中已弃用, 现在会产生 `SyntaxWarning`, 而不是 `DeprecationWarning`。在未来的 Python 版本中, 它们最终将是 `SyntaxError`。(由 Victor Stinner 在 [gh-98401](#) 中贡献。)
- 未存储在推导式目标部分中的变量现在可以在赋值表达式 `(:=)` 中使用。例如, 在 `[(b := 1) for a, b.prop in some_iter]` 中, 现在允许对 `b` 进行赋值。请注意, 根据 [PEP 572](#), 仍然不允许向存储在推导式目标部分中的变量 (如 `a`) 赋值。(由 Nikita Sobolev 在 [gh-100581](#) 中贡献。)
- 在类或类型对象的 `__set_name__` 方法中引发的异常不再由 `RuntimeError` 来包装。上下文信息将作为 [PEP 678](#) 注释添加到异常中。(由 Irit Katriel 在 [gh-77757](#) 中贡献。)
- 当 `try-except*` 构造处理整个 `ExceptionGroup` 并引发另一个异常时, 该异常不再封装在 `ExceptionGroup` 中。在 3.11.4 版中也进行了更改。(由 Irit Katriel 在 [gh-103590](#) 中贡献。)
- 垃圾回收器现在只在 Python 字节码评估循环的 `eval-breaker` 机制上运行, 而不是在对象分配上运行。垃圾回收也可以在调用 `PyErr_CheckSignals()` 时运行, 因此需要长时间运行而不执行任何 Python 代码的 C 扩展也有机会定期执行垃圾回收。(由 Pablo Galindo 在 [gh-97922](#) 中贡献。)
- 所有期望布尔参数的内置和扩展可调函数现在都接受任何类型的参数, 而不仅仅是 `bool` 和 `int`。(由 Serhiy Storchaka 在 [gh-60203](#) 中贡献。)
- `memoryview` now supports the half-float type (the "e" format code). (Contributed by Donghee Na and Antoine Pitrou in [gh-90751](#).)

- `slice` 对象现在是可哈希的，允许它们用作字典的键和集合项。(由 Will Bradshaw、Furkan Onder 和 Raymond Hettinger 在 [gh-101264](#) 中贡献。)
- `sum()` now uses Neumaier summation to improve accuracy and commutativity when summing floats or mixed ints and floats. (Contributed by Raymond Hettinger in [gh-100425](#).)
- `ast.parse()` 现在会在解析包含空字节的源代码时引发 `SyntaxError` 而不是 `ValueError`。(由 Pablo Galindo 在 [gh-96670](#) 中贡献。)
- `tarfile` 中的提取方法和 `shutil.unpack_archive()` 有一个新的 *filter* 参数，它允许限制可能令人惊讶或危险的 `tar` 功能，例如在目标目录之外创建文件。相关细节请参阅 `tarfile` 提取过滤器。在 Python 3.14 中。默认值将切换为 `'data'`。(由 Petr Viktorin 在 [PEP 706](#) 中贡献。)
- 如果底层映射是可哈希的，那么 `types.MappingProxyType` 实例现在是可哈希的。(由 Serhiy Storchaka 在 [gh-87995](#) 中贡献。)
- 通过新的环境变量 `PYTHONPERFSUPPORT` 和命令行选项 `-X perf` 以及新的 `sys.activate_stack_trampoline()`、`sys.deactivate_stack_trampoline()` 和 `sys.is_stack_trampoline_active()` 函数添加了对 `perf` 性能分析器的支持。(由 Pablo Galindo 设计。由 Pablo Galindo 和 Christian Heimes 在 [gh-96123](#) 中贡献并包含来自 Gregory P. Smith [Google] 和 Mark Shannon 的帮助。)

5 新增模块

- 无。

6 改进的模块

6.1 array

- `array.array` 类现在支持下标，使其成为 generic type。(由 Jelle Zijlstra 在 [gh-98658](#) 中贡献。)

6.2 asyncio

- 在 `asyncio` 中写入套接字的性能得到了显著提高。`asyncio` 现在可以避免在写入套接字时进行不必要的复制，并在平台支持的情况下使用 `sendmsg()`。(由 Kumar Aditya 在 [gh-91166](#) 中贡献。)
- Add `asyncio.eager_task_factory()` and `asyncio.create_eager_task_factory()` functions to allow opting an event loop in to eager task execution, making some use-cases 2x to 5x faster. (Contributed by Jacob Bower & Itamar Oren in [gh-102853](#), [gh-104140](#), and [gh-104138](#))
- 在 Linux 上，如果 `os.pidfd_open()` 可用且能工作则 `asyncio` 默认会使用 `asyncio.PidfdChildWatcher` 而不是 `asyncio.ThreadedChildWatcher`。(由 Kumar Aditya 在 [gh-98024](#) 中贡献。)
- The event loop now uses the best available child watcher for each platform (`asyncio.PidfdChildWatcher` if supported and `asyncio.ThreadedChildWatcher` otherwise), so manually configuring a child watcher is not recommended. (Contributed by Kumar Aditya in [gh-94597](#).)
- 为 `asyncio.run()` 添加了形参 *loop_factory*，以允许指定自定义事件循环工厂。(由 Kumar Aditya 在 [gh-99388](#) 中贡献。)
- 添加了 `asyncio.current_task()` 的 C 实现以实现 4 - 6 倍的加速。(由 Itamar Oren 和 Pranav Thulasiram Bhat 在 [gh-100344](#) 中贡献。)
- `asyncio.iscoroutine()` 现在为生成器返回 `False`，因为 `asyncio` 不支持传统的基于生成器的协程。(由 Kumar Aditya 在 [gh-102748](#) 中贡献。)

- `asyncio.wait()` 和 `asyncio.as_completed()` 现在接受生成器 `yield` 任务。(由 Kumar Aditya 在 [gh-78530](#) 中贡献。)

6.3 calendar

- 添加了枚举 `calendar.Month` 和 `calendar.Day` 来定义年份中的每一月和星期中的每一日。(由 Prince Roshan 在 [gh-103636](#) 中贡献。)

6.4 csv

- 增加了 `csv.QUOTE_NOTNULL` 和 `csv.QUOTE_STRINGS` 旗标以通过 `csv.writer` 对象来提供对 `None` 和空字符串的更细粒度控制。

6.5 dis

- 伪指令操作码 (由编译器使用但不会出现在可执行字节码中) 现在将暴露在 `dis` 模块中。`HAVE_ARGUMENT` 仍然与实际的操作码相关, 但对伪指令来说没有用处。请改用新的 `dis.hasarg` 多项集。(由 Irit Katriel 在 [gh-94216](#) 中贡献。)
- Add the `dis.hasexc` collection to signify instructions that set an exception handler. (Contributed by Irit Katriel in [gh-94216](#).)

6.6 fractions

- 类型为 `fractions.Fraction` 的对象现在支持浮点格式。(由 Mark Dickinson 在 [gh-100161](#) 中贡献。)

6.7 importlib.resources

- `importlib.resources.as_file()` 现在将支持资源目录。(由 Jason R. Coombs 在 [gh-97930](#) 中贡献。)

6.8 inspect

- 添加 `inspect.markcoroutinefunction()` 来标记返回 `coroutine` 的同步函数, 以便与 `inspect.iscoroutinefunction()` 一起使用。(由 Carlton Gibson 在 [gh-99247](#) 中贡献。)
- Add `inspect.getasyncgenstate()` and `inspect.getasyncgenlocals()` for determining the current state of asynchronous generators. (Contributed by Thomas Krennwallner in [gh-79940](#).)
- `inspect.getattr_static()` 的性能得到了相当大的改进。对函数的大多数调用应该比 Python 3.11 中至少快 2 倍, 有些可能快 6 倍甚至更多。(由 Alex Waygood 在 [gh-103193](#) 中贡献。)

6.9 itertools

- Add `itertools.batched()` for collecting into even-sized tuples where the last batch may be shorter than the rest. (Contributed by Raymond Hettinger in [gh-98363](#).)

6.10 math

- Add `math.sumprod()` for computing a sum of products. (Contributed by Raymond Hettinger in [gh-100485](#).)
- Extend `math.nextafter()` to include a *steps* argument for moving up or down multiple steps at a time. (By Matthias Goergens, Mark Dickinson, and Raymond Hettinger in [gh-94906](#).)

6.11 os

- 增加了 `os.PIDFD_NONBLOCK` 以在非阻塞模式下打开具有 `os.pidfd_open()` 的进程的文件描述符。(由 Kumar Aditya 在 [gh-93312](#) 中贡献。)
- `os.DirEntry` 现在包括一个 `os.DirEntry.is_junction()` 方法来检查该条目是否为目录联接。(由 Charles Machalow 在 [gh-99547](#) 中贡献。)
- 在 Windows 版中添加 `os.listdirrives()`、`os.listvolumes()` 和 `os.listmounts()` 函数，用于枚举驱动器、卷和挂载点。(由 Steve Dower 在 [gh-102519](#) 中贡献。)
- `os.stat()` 和 `os.lstat()` 现在在 Windows 系统上更准确了。`st_birthtime` 字段现在将使用文件的创建时间，`st_ctime` 已弃用，但仍包含创建时间（但为了与其他平台保持一致，将来将返回最后一次元数据更改时间）。`st_dev` 可以高达 64 位，`st_ino` 可以高达 128 位，具体取决于你的文件系统，并且 `st_rdev` 始终设置为零，而非不正确的值。这两个函数在较新版本的 Windows 上将会明显更快。(由 Steve Dower 在 [gh-99726](#) 中贡献。)

6.12 os.path

- 添加 `os.path.isjunction()` 以检查给定路径是否为目录联接。(由 Charles Machalow 在 [gh-99547](#) 中贡献。)
- 添加 `os.path.splitroot()` 以将路径拆分为三元组 (`drive`, `root`, `tail`)。(由 Barney Gale 在 [gh-101000](#) 中贡献。)

6.13 pathlib

- 增加对子类化 `pathlib.PurePath` 和 `pathlib.Path`，加上它们的 Posix 和 Windows 专属变体形式的支持。子类可以重载 `pathlib.PurePath.with_segments()` 方法来在路径实例之间传递信息。
- 添加 `pathlib.Path.walk()` 用于遍历目录树并生成其中的所有文件或目录名，类似于 `os.walk()`。(由 Stanislav Zmiev 在 [gh-90385](#) 中贡献。)
- Add *walk_up* optional parameter to `pathlib.PurePath.relative_to()` to allow the insertion of `..` entries in the result; this behavior is more consistent with `os.path.relpath()`. (Contributed by Domenico Ragusa in [gh-84538](#).)
- 添加 `pathlib.Path.is_junction()` 作为 `os.path.isjunction()` 的代理。(由 Charles Machalow 在 [gh-99547](#) 中贡献。)
- 为 `pathlib.Path.glob()`、`pathlib.Path.rglob()` 和 `pathlib.PurePath.match()` 添加可选形参 *case_sensitive*，以匹配路径的大小写敏感性，从而对匹配过程进行更精确的控制。

6.14 pdb

- 添加便利变量以临时保存调试会话的值，并提供对当前帧或返回值等值的快速访问。（由高天在 [gh-103693](#) 中贡献。）

6.15 random

- Add `random.binomialvariate()`. (Contributed by Raymond Hettinger in [gh-81620](#).)
- Add a default of `lamdb=1.0` to `random.expovariate()`. (Contributed by Raymond Hettinger in [gh-100234](#).)

6.16 shutil

- `shutil.make_archive()` 现在将 `rootdir` 参数传递给支持它的自定义存档程序。在这种情况下，它不再临时将进程的当前工作目录更改为 `rootdir` 来执行存档。（由 Serhiy Storchaka 在 [gh-74696](#) 中贡献。）
- `shutil.rmtree()` 现在接受一个新的参数 `onexc`，它是一个类似 `onerror` 的错误处理器，但它需要一个异常实例，而不是一个 `(typ, val, tb)` 三元组。`onerror` 已弃用，并将在 Python 3.14 中移除。（由 Irit Katriel 在 [gh-102828](#) 中贡献。）
- `shutil.which()` 现在即使给定的 `cmd` 包含目录组件，在 Windows 系统上也会参考 `PATHEXT` 环境变量在 `PATH` 中查找匹配项。（由 Charles Machalow 在 [gh-103179](#) 中贡献。）

`shutil.which()` 将在 Windows 上查询可执行文件时调用 `NeedCurrentDirectoryForExePathW`，以确定是否应将当前工作目录预先设置为搜索路径。（由 Charles Machalow 在 [gh-103179](#) 中贡献。）

在 Windows 上 `shutil.which()` 将在搜索路径的其他地方直接匹配之前返回 `cmd` 与来自 `PATHEXT` 的组件相匹配的路径。（由 Charles Machalow 在 [gh-103179](#) 中贡献。）

6.17 sqlite3

- 增加了一个命令行接口。（由 Erlend E. Aasland 在 [gh-77617](#) 中贡献。）
- 向 `sqlite3.Connection` 添加 `sqlite3.Connection.autocommit` 属性并向 `sqlite3.connect()` 添加 `autocommit` 形参用于控制兼容 [PEP 249](#) 的事务处理。（由 Erlend E. Aasland 在 [gh-83638](#) 中贡献。）
- 向 `sqlite3.Connection.load_extension()` 添加 `entrypoint` 仅限关键字形参，用于覆盖 SQLite 扩展入口点。（由 Erlend E. Aasland 在 [gh-103015](#) 中贡献。）
- 向 `sqlite3.Connection` 添加 `sqlite3.Connection.getconfig()` 和 `sqlite3.Connection.setconfig()` 用于对数据库连接进行配置修改。（由 Erlend E. Aasland 在 [gh-103489](#) 中贡献。）

6.18 statistics

- Extend `statistics.correlation()` to include as a ranked method for computing the Spearman correlation of ranked data. (Contributed by Raymond Hettinger in [gh-95861](#).)

6.19 sys

- Add the `sys.monitoring` namespace to expose the new [PEP 669](#) monitoring API. (Contributed by Mark Shannon in [gh-103082](#).)
- 增加了 `sys.activate_stack_trampoline()` 和 `sys.deactivate_stack_trampoline()` 用于激活和停用栈性能分析器 `trampoline`，以及 `sys.is_stack_trampoline_active()` 用于查询栈性能分析器 `trampoline` 是否激活。（基于 Gregory P. Smith [Google] 和 Mark Shannon 的贡献由 Pablo Galindo 和 Christian Heimes 在 [gh-96123](#) 中贡献。）
- 增加了 `sys.last_exc` 用于保存最新引发的未处理异常（针对事后调试的应用场景）。弃用了以三个字段来保存相同信息的旧形式：`sys.last_type`、`sys.last_value` 和 `sys.last_traceback`。（由 Irit Katriel 在 [gh-102778](#) 中贡献。）
- 现在 `sys._current_exceptions()` 将返回从线程 ID 到异常实例的映射，而不是到 `(typ, exc, tb)` 元组的映射。（由 Irit Katriel 在 [gh-103176](#) 中贡献。）
- `sys.setrecursionlimit()` 和 `sys.getrecursionlimit()`。递归限制现在只应用于 Python 代码。内置函数不使用该递归限制，但受到另一种可防止递归导致虚拟机崩溃的机制保护。

6.20 tempfile

- `tempfile.NamedTemporaryFile` 函数增加了一个新的可选形参 `delete_on_close`。（由 Evgeny Zorin 在 [gh-58451](#) 中贡献。）
- `tempfile.mkdtemp()` 现在将总是返回一个绝对路径，即使提供给 `dir` 形参的参数是一个相对路径。

6.21 threading

- 增加了 `threading.settrace_all_threads()` 和 `threading.setprofile_all_threads()` 以允许在所运行的全部线程中设置追踪和性能分析函数而不是只在调用方线程中。（由 Pablo Galindo 在 [gh-93503](#) 中贡献。）

6.22 tkinter

- 现在 `tkinter.Canvas.coords()` 会展开其参数。它现在不仅接受单独参数形式的坐标 `(x1, y1, x2, y2, ...)` 以及由坐标组成的序列 `[(x1, y1, x2, y2, ...)]`，也接受成对分组 `((x1, y1), (x2, y2), ...)` 和 `[(x1, y1), (x2, y2), ...]` 形式的坐标，就像 `create_*` 方法一样。（由 Serhiy Storchaka 在 [gh-94473](#) 中贡献。）

6.23 tokenize

- `tokenize` 模块包括了 [PEP 701](#) 所引入的更改。（由 Marta Gómez Macías 和 Pablo Galindo 在 [gh-102856](#) 中贡献。）请参阅[移植到 Python 3.12](#)了解有关对 `tokenize` 模块的更改详情。

6.24 types

- 增加了 `types.get_original_bases()` 以允许在子类化时继续对 `user-defined-generics` 进行内省。(由 James Hilton-Balfe 和 Alex Waygood 在 [gh-101827](#) 中贡献。)

6.25 typing

- 针对 运行时可检测协议的 `isinstance()` 检测现在会使用 `inspect.getattr_static()` 而不是 `hasattr()` 来查找属性是否存在。这意味着描述器和 `__getattr__()` 方法在针对运行时可检测协议的 `isinstance()` 检测期间不会被意外地求值。但是，这也意味着某些原来被视为运行时可检测协议的实例的对象在 Python 3.12+ 上将不再被视为运行时可检测协议的实例，反之亦然。大部分用户都不太可能受到这一改变的影响。(由 Alex Waygood 在 [gh-102433](#) 中贡献。)
- 现在运行时可检测协议的成员在运行时一旦创建了相应的类将被视为“已冻结”。作用于运行时可检测协议的猴子补丁属性将仍然可用，但不会再影响将对象与协议进行比较的 `isinstance()` 检测中。例如：

```
>>> from typing import Protocol, runtime_checkable
>>> @runtime_checkable
... class HasX(Protocol):
...     x = 1
...
>>> class Foo: ...
...
>>> f = Foo()
>>> isinstance(f, HasX)
False
>>> f.x = 1
>>> isinstance(f, HasX)
True
>>> HasX.y = 2
>>> isinstance(f, HasX) # unchanged, even though HasX now also has a "y"
↪attribute
True
```

应用这项改变是为了提高针对运行时可检测协议的 `isinstance()` 检测速度。

- 针对 运行时可检测协议的 `isinstance()` 检测的性能表现有显著的改进。对于具有少量成员的协议的 `isinstance()` 检测相比 3.11 应当至少有 2x 的提速，有些可能会有 20x 或更多的提速。但是，对于具有十四个或更多成员的协议的 `isinstance()` 检测可能会慢于 Python 3.11。(由 Alex Waygood 在 [gh-74690](#) 和 [gh-103193](#) 中贡献。)
- 现在所有 `typing.TypedDict` 和 `typing.NamedTuple` 类都具有 `__orig_bases__` 属性。(由 Adrian Garcia Badaracco 在 [gh-103699](#) 中贡献。)
- 向 `typing.dataclass_transform()` 添加了 `frozen_default` 形参。(由 Erik De Bonte 在 [gh-99957](#) 中贡献。)

6.26 unicodedata

- Unicode 数据库已更新到 15.0.0 版。(由 Benjamin Peterson 在 [gh-96734](#) 中贡献。)

6.27 unittest

Add a `--durations` command line option, showing the N slowest test cases:

```
python3 -m unittest --durations=3 lib.tests.test_threading
.....
Slowest test durations
-----
1.210s      test_timeout (Lib.test.test_threading.BarrierTests)
1.003s      test_default_timeout (Lib.test.test_threading.BarrierTests)
0.518s      test_timeout (Lib.test.test_threading.EventTests)

(0.000 durations hidden.  Use -v to show these durations.)
-----
Ran 158 tests in 9.869s

OK (skipped=3)
```

(Contributed by Giampaolo Rodola in [gh-48330](#))

6.28 uuid

- 增加了一个 命令行接口。(由 Adam Chhina 在 [gh-88597](#) 中贡献。)

7 性能优化

- Remove `wstr` and `wstr_length` members from Unicode objects. It reduces object size by 8 or 16 bytes on 64bit platform. (**PEP 623**) (Contributed by Inada Naoki in [gh-92536](#).)
- Add experimental support for using the BOLT binary optimizer in the build process, which improves performance by 1-5%. (Contributed by Kevin Modzelewski in [gh-90536](#) and tuned by Donghee Na in [gh-101525](#))
- 对于包含分组引用的替换字符串的正则表达式替换 (包括 `re.sub()` 和 `re.subn()` 函数及对应的 `re.Pattern` 方法) 可加速 2--3 倍。(由 Serhiy Storchaka 在 [gh-91524](#) 中贡献。)
- 通过推迟高消耗的字符串格式化来加速 `asyncio.Task` 的创建的。(由 Itamar Oren 在 [gh-103793](#) 中贡献。)
- 作为在 `tokenize` 模块中应用 **PEP 701** 所要求的更改的附带效果, `tokenize.tokenize()` 和 `tokenize.generate_tokens()` 函数可加速至多 64%。(由 Marta Gómez Macías 和 Pablo Galindo 在 [gh-102856](#) 中贡献。)
- 通过新的 `LOAD_SUPER_ATTR` 指令加速 `super()` 方法调用和属性加载。(由 Carl Meyer 和 Vladimir Matveev 在 [gh-103497](#) 中贡献。)

8 CPython 字节码的改变

- 移除了 `LOAD_METHOD` 指令。它已被合并至 `LOAD_ATTR`。现在如果设置了 `LOAD_ATTR` 的 `oparg` 比特位则它的行为将类似原来的 `LOAD_METHOD`。(由 Ken Jin 在 [gh-93429](#) 中贡献。)
- 移除了 `JUMP_IF_FALSE_OR_POP` 和 `JUMP_IF_TRUE_OR_POP` 指令。(由 Irit Katriel 在 [gh-102859](#) 中贡献。)
- Remove the `PRECALL` instruction. (Contributed by Mark Shannon in [gh-92925](#).)
- Add the `BINARY_SLICE` and `STORE_SLICE` instructions. (Contributed by Mark Shannon in [gh-94163](#).)
- Add the `CALL_INTRINSIC_1` instructions. (Contributed by Mark Shannon in [gh-99005](#).)
- Add the `CALL_INTRINSIC_2` instruction. (Contributed by Irit Katriel in [gh-101799](#).)

- Add the `CLEANUP_THROW` instruction. (Contributed by Brandt Bucher in [gh-90997](#).)
- Add the `END_SEND` instruction. (Contributed by Mark Shannon in [gh-103082](#).)
- 增加了 `LOAD_FAST_AND_CLEAR` 指令作为 **PEP 709** 的实现的组成部分。(由 Carl Meyer 在 [gh-101441](#) 中贡献。)
- Add the `LOAD_FAST_CHECK` instruction. (Contributed by Dennis Sweeney in [gh-93143](#).)
- 增加了 `LOAD_FROM_DICT_OR_DEREF`, `LOAD_FROM_DICT_OR_GLOBALS` 和 `LOAD_LOCALS` 操作码作为 **PEP 695** 的组成部分。移除了 `LOAD_CLASSDEREF` 操作码，它可以用 `LOAD_LOCALS` 加 `LOAD_FROM_DICT_OR_DEREF` 来代替。(由 Jelle Zijlstra 在 [gh-103764](#) 中贡献。)
- 增加了 `LOAD_SUPER_ATTR` 指令。(由 Carl Meyer 和 Vladimir Matveev 在 [gh-103497](#) 中贡献。)
- Add the `RETURN_CONST` instruction. (Contributed by Wenyang Wang in [gh-101632](#).)

9 演示和工具

- 移除了包含旧演示脚本的 `Tools/demo/` 目录。其副本可在 [old-demos project](#) 中找到。(由 Victor Stinner 在 [gh-97681](#) 中贡献。)
- 移除了 `Tools/scripts/` 目录下过时的示例脚本。其副本可在 [old-demos project](#) 中找到。(由 Victor Stinner 在 [gh-97669](#) 中贡献。)

10 弃用

- `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 `ast` 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`:
 - The `child watcher` classes `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` and `asyncio.SafeChildWatcher` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in [gh-94597](#).)
 - `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。)
- `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。)

- `collections.abc`: 已弃用 `collections.abc.ByteString`。推荐改用 `Sequence` 或 `collections.abc.Buffer`。在类型标中, 推荐改用并集, 如 `bytes | bytearray` 或 `collections.abc.Buffer`。(由 Shantanu Jain 在 [gh-91896](#) 中贡献。)
- `datetime`: `datetime.datetime` 的 `utcnow()` 和 `utcfromtimestamp()` 已被弃用并将在未来的版本中移除。请改用可感知时区的对象以 UTC 来表示日期时间: 分别调用 `now()` 和 `fromtimestamp()` 并设置 `tz` 形参为 `datetime.UTC`。(由 Paul Ganssle 在 [gh-103857](#) 中贡献。)
- `email`: Deprecate the `isdst` parameter in `email.utils.localtime()`. (Contributed by Alan Williams in [gh-72346](#).)
- `importlib.abc`: 已弃用下列类, 计划在 Python 3.14 中移除:

- `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`: Deprecate the support for `copy`, `deepcopy`, and `pickle` operations, which is undocumented, inefficient, historically buggy, and inconsistent. This will be removed in 3.14 for a significant reduction in code volume and maintenance burden. (Contributed by Raymond Hettinger in [gh-101588](#).)
- `multiprocessing`: In Python 3.14, the default `multiprocessing` start method will change to a safer one on Linux, BSDs, and other non-macOS POSIX platforms where `'fork'` is currently the default ([gh-84559](#)). Adding a runtime warning about this was deemed too disruptive as the majority of code is not expected to care. Use the `get_context()` or `set_start_method()` APIs to explicitly specify when your code *requires* `'fork'`. See contexts and start methods.
- `pkgutil`: `pkgutil.find_loader()` and `pkgutil.get_loader()` are deprecated and will be removed in Python 3.14; use `importlib.util.find_spec()` instead. (Contributed by Nikita Sobolev in [gh-97850](#).)
- `pty`: The module has two undocumented `master_open()` and `slave_open()` functions that have been deprecated since Python 2 but only gained a proper `DeprecationWarning` in 3.12. Remove them in 3.14. (Contributed by Soumendra Ganguly and Gregory P. Smith in [gh-85984](#).)
- `os`:
 - 在 Windows 上由 `os.stat()` 和 `os.lstat()` 返回的 `st_ctime` 字段已被弃用。在未来的发布版中, 它们将包含最近的元数据修改时间, 以与其他平台保持一致。目前, 它们仍然包含创建时间, 该值也可通过新的 `st_birthtime` 字段获取。(由 Steve Dower 在 [gh-99726](#) 中贡献。)
 - On POSIX platforms, `os.fork()` can now raise a `DeprecationWarning` when it can detect being called from a multithreaded process. There has always been a fundamental incompatibility with the POSIX platform when doing so. Even if such code *appeared* to work. We added the warning to raise awareness as issues encountered by code doing this are becoming more frequent. See the `os.fork()` documentation for more details along with [this discussion on fork being incompatible with threads](#) for why we're now surfacing this longstanding platform compatibility problem to developers.

When this warning appears due to usage of `multiprocessing` or `concurrent.futures` the fix is to use a different `multiprocessing` start method such as `"spawn"` or `"forkserver"`.

- `shutil`: The `onerror` argument of `shutil.rmtree()` is deprecated and will be removed in Python 3.14. Use `onexc` instead. (Contributed by Irit Katriel in [gh-102828](#).)
- `sqlite3`:
 - 默认适配器和转换器现在已被弃用, 请使用 `sqlite3-adapter-converter-recipes` 并根据你的需要调整它们。(由 Erlend E. Aasland 在 [gh-90016](#) 中贡献。)

- 在 `execute()` 中, 现在当命名占位符与作为 `sequence` 而不是 `dict` 提供的形参一起使用时将发出 `DeprecationWarning`。从 Python 3.14 开始, 当命名占位符与作为序列提供的形参一起使用时将引发 `ProgrammingError`。(由 Erlend E. Aasland 在 [gh-101698](#) 中贡献。)
- `sys`: `sys.last_type`, `sys.last_value` 和 `sys.last_traceback` 字段已被弃用。请改用 `sys.last_exc`。(由 Irit Katriel 在 [gh-102778](#) 中贡献。)
- `tarfile`: 提取 `tar` 归档而不指定 `filter` 的做法已被弃用直到 Python 3.14, 在该版本中 `'data'` 将成为默认过滤器。请参阅 [tarfile-extraction-filter](#) 了解详情。
- `typing`:
 - `typing.Hashable` 和 `typing.Sized` 是 `collections.abc.Hashable` 和 `collections.abc.Sized` 的别名。(根据 [gh-94309](#)。)
 - `typing.ByteString`, deprecated since Python 3.9, now causes a `DeprecationWarning` to be emitted when it is used. (Contributed by Alex Waygood in [gh-91896](#).)
- `xml.etree.ElementTree`: The module now emits `DeprecationWarning` when testing the truth value of an `xml.etree.ElementTree.Element`. Before, the Python implementation emitted `FutureWarning`, and the C implementation emitted nothing. (Contributed by Jacob Walls in [gh-83122](#).)
- The 3-arg signatures (`type`, `value`, `traceback`) of `coroutine throw()`, `generator throw()` and `async generator throw()` are deprecated and may be removed in a future version of Python. Use the single-arg versions of these functions instead. (Contributed by Ofey Chan in [gh-89874](#).)
- 现在当一个模块上的 `__package__` 不同于 `__spec__.parent` 时将引发 `DeprecationWarning` (在之前版本中则为 `ImportWarning`)。 (由 Brett Cannon 在 [gh-65961](#) 中贡献。)
- Setting `__package__` or `__cached__` on a module is deprecated, and will cease to be set or taken into consideration by the import system in Python 3.14. (Contributed by Brett Cannon in [gh-65961](#).)
- The bitwise inversion operator (`~`) on `bool` is deprecated. It will throw an error in Python 3.14. Use `not` for logical negation of bools instead. In the rare case that you really need the bitwise inversion of the underlying `int`, convert to `int` explicitly: `~int(x)`. (Contributed by Tim Hoffmann in [gh-103487](#).)
- Accessing `co_lnotab` on code objects was deprecated in Python 3.10 via [PEP 626](#), but it only got a proper `DeprecationWarning` in 3.12, therefore it will be removed in 3.14. (Contributed by Nikita Sobolev in [gh-101866](#).)

10.1 计划在 Python 3.13 中移除

以下模块和 API 已在之前的 Python 发布版中弃用, 并将在 Python 3.13 中移除。

模块 (参见 [PEP 594](#)):

- `aifc`
- `audioop`
- `cgi`
- `cgitb`
- `chunk`
- `crypt`
- `imghdr`
- `mailcap`
- `msilib`
- `nis`
- `nntplib`

- `ossaudiodev`
- `pipes`
- `sndhdr`
- `spwd`
- `sunau`
- `telnetlib`
- `uu`
- `xdrlib`

其他模块：

- `lib2to3`，以及 **2to3** 程序 ([gh-84540](#))

API:

- `configparser.LegacyInterpolation` ([gh-90765](#))
- `locale.getdefaultlocale()` ([gh-90817](#))
- `turtle.RawTurtle.settiltangle()` ([gh-50096](#))
- `unittest.findTestCases()` ([gh-50096](#))
- `unittest.getTestCaseNames()` ([gh-50096](#))
- `unittest.makeSuite()` ([gh-50096](#))
- `unittest.TestProgram.usageExit()` ([gh-67048](#))
- `webbrowser.MacOSX` ([gh-86421](#))
- `classmethod` 描述器串联 ([gh-89519](#))

10.2 计划在 Python 3.14 中移除

The following APIs have been deprecated and will be removed in Python 3.14.

- `argparse`: The *type*, *choices*, and *metavar* parameters of `argparse.BooleanOptionalAction`
- `ast`:
 - `ast.Num`
 - `ast.Str`
 - `ast.Bytes`
 - `ast.NameConstant`
 - `ast.Ellipsis`
- `asyncio`:
 - `asyncio.MultiLoopChildWatcher`
 - `asyncio.FastChildWatcher`
 - `asyncio.AbstractChildWatcher`
 - `asyncio.SafeChildWatcher`
 - `asyncio.set_child_watcher()`
 - `asyncio.get_child_watcher()`,
 - `asyncio.AbstractEventLoopPolicy.set_child_watcher()`
 - `asyncio.AbstractEventLoopPolicy.get_child_watcher()`

- `collections.abc`: `collections.abc.ByteString`.
- `email`: the *isdst* parameter in `email.utils.localtime()`.
- `importlib.abc`:
 - `importlib.abc.ResourceReader`
 - `importlib.abc.Traversable`
 - `importlib.abc.TraversableResources`
- `itertools`: Support for copy, deepcopy, and pickle operations.
- `pkgutil`:
 - `pkgutil.find_loader()`
 - `pkgutil.get_loader()`.
- `pty`:
 - `pty.master_open()`
 - `pty.slave_open()`
- `shutil`: The *onerror* argument of `shutil.rmtree()`
- `typing`: `typing.ByteString`
- `xml.etree.ElementTree`: Testing the truth value of an `xml.etree.ElementTree.Element`.
- The `__package__` and `__cached__` attributes on module objects.
- The `co_notab` attribute of code objects.

10.3 计划在未来版本中移除

下列 API 在更早的 Python 版本中已被弃用并将被移除，但目前还没有确定它们的移除日期。

- `array` 的 'u' 格式代码 ([gh-57281](#))
- `typing.Text` ([gh-92332](#))
- 目前 Python 接受数字类面值后面紧跟关键字的写法，例如 `0in x, 1or x, 0if 1else 2`。它将允许像 `[0x1for x in y]` 这样令人困惑且模棱两可的表达式 (它可以被解读为 `[0x1 for x in y]` 或者 `[0x1f or x in y]`)。从本发布版开始，如果数字类面值后面紧跟关键字 `and`, `else`, `for`, `if`, `in`, `is` 和 `or` 中的一个将会引发弃用警告。在未来的版本中它将改为语法警告，最终将改为语法错误。([gh-87999](#))

11 移除

11.1 `asynchat` 和 `asyncore`

- 这两个模块已根据 [PEP 594](#) 中的时间表被移除，它们从 Python 3.6 起已被弃用。请改用 `asyncio`。(由 Nikita Sobolev 在 [gh-96580](#) 中贡献。)

11.2 configparser

- configparser 中的几个从 3.2 起已被弃用的名称已根据 [gh-89336](#) 被移除：
 - configparser.ParsingError 不再具有 filename 属性或参数。请改用 source 属性和参数。
 - configparser 不再具有 SafeConfigParser 类。请改用更简短的名称 ConfigParser。
 - configparser.ConfigParser 不再具有 readfp 方法。请改用 read_file()。

11.3 distutils

- 移除了 distutils 包。它已在 Python 3.10 中根据 [PEP 632](#) ”Deprecate distutils module” 被弃用。对于仍然使用 distutils 且无法升级为使用其他工具的项目，可以安装 setuptools 项目：它仍然提供了 distutils。（由 Victor Stinner 在 [gh-92584](#) 中贡献。）

11.4 ensurepip

- 从 ensurepip 中移除了捆绑的 setuptools wheel，并停止在由 venv 创建的环境中安装 setuptools。
pip (>= 22.1) 不再要求在环境中安装 setuptools。基于 setuptools (和基于 distutils) 的包仍然可通过 pip install 来使用，因为 pip 将在它用于构建包的构建环境中提供 setuptools。
在默认情况下由 venv 创建或通过 ensurepip 初始化的环境将不再提供 easy_install, pkg_resources, setuptools 和 distutils 包，因为它们是 setuptools 包的组成部分。对于在运行时依赖这些包的项目，应当将 setuptools 项目声明为依赖项之一并单独安装（通常是使用 pip）。
(由 Pradyun Gedam 在 [gh-95299](#) 中贡献。)

11.5 enum

- 移除了 enum 的 EnumMeta.__getattr__，枚举属性访问已不再需要它。（由 Ethan Furman 在 [gh-95083](#) 中贡献。）

11.6 ftplib

- 移除了 ftplib 的 FTP_TLS.ssl_version 类属性：请改用 context 形参。（由 Victor Stinner 在 [gh-94172](#) 中贡献。）

11.7 gzip

- 移除了 gzip 中 gzip.GzipFile 的 filename 属性，自 Python 2.6 起该属性已被弃用，请改用 name 属性。在可写模式下，如果 filename 属性没有 '.gz' 文件扩展名则会添加它。（由 Victor Stinner 在 [gh-94196](#) 中贡献。）

11.8 hashlib

- 移除了 hashlib 中 `hashlib.pbkdf2_hmac()` 的纯 Python 实现，它在 Python 3.10 中已被弃用。Python 3.10 及更新版本需要 OpenSSL 1.1.1 (PEP 644)：该 OpenSSL 版本提供了 `pbkdf2_hmac()` 的更快速的 C 实现。（由 Victor Stinner 在 [gh-94199](#) 中贡献。）

11.9 importlib

- `importlib` 中许多先前已弃用对象的清理工作现已完成：
 - 对 `module_repr()` 的引用和支持已被移除。（由 Barry Warsaw 在 [gh-97850](#) 中贡献。）
 - `importlib.util.set_package`, `importlib.util.set_loader` 和 `importlib.util.module_for_loader` 均已被移除。（由 Brett Cannon 和 Nikita Sobolev 在 [gh-65961](#) 和 [gh-97850](#) 中贡献。）
 - 对 `find_loader()` 和 `find_module()` API 的支持已被移除。（由 Barry Warsaw 在 [gh-98040](#) 中贡献。）
 - `importlib.abc.Finder`, `pkgutil.ImpImporter` 和 `pkgutil.ImpLoader` 已被移除。（由 Barry Warsaw 在 [gh-98040](#) 中贡献。）

11.10 imp

- `imp` 模块已被移除。（由 Barry Warsaw 在 [gh-98040](#) 中贡献。）

To migrate, consult the following correspondence table:

imp	importlib
<code>imp.NullImporter</code>	将 None 插入到 <code>sys.path_importer_cache</code>
<code>imp.cache_from_source()</code>	<code>importlib.util.cache_from_source()</code>
<code>imp.find_module()</code>	<code>importlib.util.find_spec()</code>
<code>imp.get_magic()</code>	<code>importlib.util.MAGIC_NUMBER</code>
<code>imp.get_suffixes()</code>	<code>importlib.machinery.SOURCE_SUFFIXES</code> , <code>importlib.machinery.EXTENSION_SUFFIXES</code> 和 <code>importlib.machinery.BYTECODE_SUFFIXES</code>
<code>imp.get_tag()</code>	<code>sys.implementation.cache_tag</code>
<code>imp.load_module()</code>	<code>importlib.import_module()</code>
<code>imp.new_module(name)</code>	<code>types.ModuleType(name)</code>
<code>imp.reload()</code>	<code>importlib.reload()</code>
<code>imp.source_from_cache()</code>	<code>importlib.util.source_from_cache()</code>
<code>imp.load_source()</code>	<i>See below</i>

将 `imp.load_source()` 替换为:

```

import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(modname, filename)
    spec = importlib.util.spec_from_file_location(modname, filename,
    ↪ loader=loader)
    module = importlib.util.module_from_spec(spec)
    # The module is always executed and not cached in sys.modules.
    # Uncomment the following line to cache the module.
    # sys.modules[module.__name__] = module
    loader.exec_module(module)
    return module

```

- Remove `imp` functions and attributes with no replacements:

- Undocumented functions:

```

* imp.init_builtin()
* imp.load_compiled()
* imp.load_dynamic()
* imp.load_package()

```

- `imp.lock_held()`, `imp.acquire_lock()`, `imp.release_lock()`: 加锁方案在 Python 3.3 中已改为模块级锁。
- `imp.find_module()` 常量: `SEARCH_ERROR`, `PY_SOURCE`, `PY_COMPILED`, `C_EXTENSION`, `PY_RESOURCE`, `PKG_DIRECTORY`, `C_BUILTIN`, `PY_FROZEN`, `PY_CODERESOURCE`, `IMP_HOOK`。

11.11 io

- 移除了 `io` 中的 `io.OpenWrapper` 和 `_pyio.OpenWrapper`，它们在 Python 3.10 中已被弃用：请改用 `open()`。`open()` (`io.open()`) 函数是一个内置函数。自 Python 3.10 起，`_pyio.open()` 也是一个静态方法。（由 Victor Stinner 在 [gh-94169](#) 中贡献。）

11.12 locale

- 移除了 `locale` 的 `locale.format()` 函数，它在 Python 3.7 中已被弃用：请改用 `locale.format_string()`。（由 Victor Stinner 在 [gh-94226](#) 中贡献。）
- `smtpd`: 该模块已按照 [PEP 594](#) 中的计划表被移除，它在 Python 3.4.7 和 3.5.4 中已被弃用。请改用 `aiosmtpd` PyPI 模块或任何其他基于 `asyncio` Oleg Iarygin 在 [gh-93243](#) 中贡献。）

11.13 sqlite3

- 以下未写入文档的 `sqlite3` 特性，在 Python 3.10 中已被弃用，现在已被移除：

```

- sqlite3.enable_shared_cache()
- sqlite3.OptimizedUnicode

```

如果必须使用共享缓存，请在以 `URI` 模式打开数据库时使用 `cache=shared` 查询参数。

`sqlite3.OptimizedUnicode` 文本工厂函数自 Python 3.3 起已成为 `str` 的一个别名。之前将文本工厂设为 `OptimizedUnicode` 的代码可以显式地使用 `str`，或者依赖同样为 `str` 的默认值。

（由 Erlend E. Aasland 在 [gh-92548](#) 中贡献。）

11.14 ssl

- 移除了 `ssl` 的 `ssl.RAND_pseudo_bytes()` 函数，它在 Python 3.6 中已被弃用：请改用 `os.urandom()` 或 `ssl.RAND_bytes()`。（由 Victor Stinner 在 [gh-94199](#) 中贡献。）
- 移除了 `ssl.match_hostname()` 函数。它在 Python 3.7 中已被弃用。OpenSSL 自 Python 3.7 起将会执行主机名匹配，Python 已不再使用 `ssl.match_hostname()` 函数。（由 Victor Stinner 在 [gh-94199](#) 中贡献。）
- 移除了 `ssl.wrap_socket()` 函数，它在 Python 3.7 中已被弃用：请改为创建一个 `ssl.SSLContext` 对象并调用其 `ssl.SSLContext.wrap_socket` 方法。任何仍然使用 `ssl.wrap_socket()` 的包都不再适用并且是不安全的。该函数既不会发送 SNI TLS 扩展也不会验证服务器主机名。其代码会受到 [CWE-295](#)：不正确的证书验证问题的影响。（由 Victor Stinner 在 [gh-94199](#) 中贡献。）

11.15 unittest

- Remove many long-deprecated unittest features:
 - 一些 `TestCase` 方法的别名：

已弃用的别名	方法名	弃用于
<code>failUnless</code>	<code>assertTrue()</code>	3.1
<code>failIf</code>	<code>assertFalse()</code>	3.1
<code>failUnlessEqual</code>	<code>assertEqual()</code>	3.1
<code>failIfEqual</code>	<code>assertNotEqual()</code>	3.1
<code>failUnlessAlmostEqual</code>	<code>assertAlmostEqual()</code>	3.1
<code>failIfAlmostEqual</code>	<code>assertNotAlmostEqual()</code>	3.1
<code>failUnlessRaises</code>	<code>assertRaises()</code>	3.1
<code>assert_</code>	<code>assertTrue()</code>	3.2
<code>assertEquals</code>	<code>assertEqual()</code>	3.2
<code>assertNotEquals</code>	<code>assertNotEqual()</code>	3.2
<code>assertAlmostEquals</code>	<code>assertAlmostEqual()</code>	3.2
<code>assertNotAlmostEquals</code>	<code>assertNotAlmostEqual()</code>	3.2
<code>assertRegexpMatches</code>	<code>assertRegex()</code>	3.2
<code>assertRaisesRegexp</code>	<code>assertRaisesRegex()</code>	3.2
<code>assertNotRegexpMatches</code>	<code>assertNotRegex()</code>	3.5

您可以使用 <https://github.com/isidentical/teyit> 来自动更新你的单元测试。

- 未写入文档且已不可用的 `TestCase` 方法 `assertDictContainsSubset`。（在 Python 3.2 中已弃用。）
- 未写入文档的 `TestLoader.loadTestsFromModule` 形参 `use_load_tests`。（自 Python 3.2 起已弃用并被忽略。）
- `TextTestResult` 类的一个别名：`_TextTestResult`。（在 Python 3.2 中已弃用。）

(Contributed by Serhiy Storchaka in [gh-89325](#).)

11.16 webbrowser

- Remove support for obsolete browsers from webbrowser. The removed browsers include: Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird, and Firefox versions 35 and below ([gh-102871](#)).

11.17 xml.etree.ElementTree

- 移除了纯 Python 实现的 `ElementTree.Element.copy()` 方法，该方法在 Python 3.10 中已被弃用，请改用 `copy.copy()` 函数。`xml.etree.ElementTree` 的 C 实现没有 `copy()` 方法，只有 `__copy__()` 方法。（由 Victor Stinner 在 [gh-94383](#) 中贡献。）

11.18 zipimport

- 移除了 `zipimport` 的 `find_loader()` 和 `find_module()` 方法，它们在 Python 3.10 中已被弃用：请改用 `find_spec()` 方法。请参阅 [PEP 451](#) 了解相关说明。（由 Victor Stinner 在 [gh-94379](#) 中贡献。）

11.19 其他事项

- Remove the suspicious rule from the documentation Makefile and `Doc/tools/rstlint.py`, both in favor of `sphinx-lint`. (Contributed by Julien Palard in [gh-98179](#).)
- 移除了 `ftplib`、`imaplib`、`poplib` 和 `smtplib` 模块中的 `keyfile` 和 `certfile` 形参数，以及 `http.client` 模块中的 `key_file`、`cert_file` 和 `check_hostname` 形参，它们自 Python 3.6 起都已被弃用。请改用 `context` 形参（在 `imaplib` 中为 `ssl_context` 形参）。（由 Victor Stinner 在 [gh-94172](#) 中贡献。）

12 移植到 Python 3.12

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

- 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、数字和下划线。（由 Serhiy Storchaka 在 [gh-91760](#) 中贡献。）
- Remove `randrange()` functionality deprecated since Python 3.10. Formerly, `randrange(10.0)` losslessly converted to `randrange(10)`. Now, it raises a `TypeError`. Also, the exception raised for non-integer values such as `randrange(10.5)` or `randrange('10')` has been changed from `ValueError` to `TypeError`. This also prevents bugs where `randrange(1e25)` would silently select from a larger range than `randrange(10**25)`. (Originally suggested by Serhiy Storchaka [gh-86388](#).)
- `argparse.ArgumentParser` 将从文件（例如 `fromfile_prefix_chars` 选项）读取参数的编码格式和错误处理句柄从默认的文本编码格式（例如 `locale.getpreferredencoding(False)` 调用）改为 filesystem encoding and error handler。在 Windows 系统中参数文件应使用 UTF-8 而不是 ANSI 代码页来编码。
- Remove the `asyncore`-based `smtpd` module deprecated in Python 3.4.7 and 3.5.4. A recommended replacement is the `asyncio`-based `aiosmtpd` PyPI module.
- `shlex.split()`: 传入 `None` 作为 `s` 参数现在将引发异常，而不是读取 `sys.stdin`。该特性在 Python 3.9 中已被弃用。（由 Victor Stinner 在 [gh-94352](#) 中贡献。）
- `os` 模块不再接受类似字节串的路径，如 `bytearray` 和 `memoryview` 类型：只接受明确的 `bytes` 类型字节串。（由 Victor Stinner 在 [gh-98393](#) 中贡献。）

- `syslog.openlog()` and `syslog.closelog()` now fail if used in subinterpreters. `syslog.syslog()` may still be used in subinterpreters, but now only if `syslog.openlog()` has already been called in the main interpreter. These new restrictions do not apply to the main interpreter, so only a very small set of users might be affected. This change helps with interpreter isolation. Furthermore, `syslog` is a wrapper around process-global resources, which are best managed from the main interpreter. (Contributed by Donghee Na in [gh-99127](#).)
- 未写入文档的 `cached_property()` 的锁定行为已被移除, 因为该行为会在类的所有实例中锁定, 从而导致高锁定争用。这意味着如果两个线程同时运行, 缓存属性获取函数现在可以在单个实例中运行不止一次。对于大多数简单的缓存属性 (例如那些幂等的并且只需根据实例的其他属性计算一个值的属性) 来说这是没有问题的。如果需要同步, 可在缓存属性获取函数中或多线程访问点周围实现锁定操作。
- 现在 `sys._current_exceptions()` 将返回从线程 ID 到异常实例的映射, 而不是到 `(typ, exc, tb)` 元组的映射。(由 Irit Katriel 在 [gh-103176](#) 中贡献。)
- 当使用 `tarfile` 或 `shutil.unpack_archive()` 提取 tar 文件时, 请传入 `filter` 参数来限制可能令人感到意外或危险的特性。请参阅 [tarfile-extraction-filter](#) 了解详情。
- 由于在 [PEP 701](#) 中引入的更改 `tokenize.tokenize()` 和 `tokenize.generate_tokens()` 函数的输出现在发生了改变。这意味着不再为 f-字符串输出 `STRING` 词元而是改为产生 [PEP 701](#) 中描述的词元: 除了用于对表达式组件进行分词的适当词元外现在还有 `FSTRING_START`, `FSTRING_MIDDLE` 和 `FSTRING_END` 会被用于 f-字符串的“字符串”部分。例如对于 f-字符串 `f"start {1+1} end"` 旧版本的分词器会生成:

<code>1,0-1,18:</code>	<code>STRING</code>	<code>'f"start {1+1} end"'</code>
------------------------	---------------------	-----------------------------------

而新版本将生成:

<code>1,0-1,2:</code>	<code>FSTRING_START</code>	<code>'f''</code>
<code>1,2-1,8:</code>	<code>FSTRING_MIDDLE</code>	<code>'start '</code>
<code>1,8-1,9:</code>	<code>OP</code>	<code>'{'</code>
<code>1,9-1,10:</code>	<code>NUMBER</code>	<code>'1'</code>
<code>1,10-1,11:</code>	<code>OP</code>	<code>'+'</code>
<code>1,11-1,12:</code>	<code>NUMBER</code>	<code>'1'</code>
<code>1,12-1,13:</code>	<code>OP</code>	<code>'}'</code>
<code>1,13-1,17:</code>	<code>FSTRING_MIDDLE</code>	<code>' end'</code>
<code>1,17-1,18:</code>	<code>FSTRING_END</code>	<code>'''</code>

此外, 支持 [PEP 701](#) 所需的改变还可能会导致一些细微的行为改变。这些变化包括:

- 在对一些无效 Python 字符如 `!` 进行分词时相应词元的 `type` 属性已从 `ERRORTOKEN` 变为 `OP`。
- 不完整的单行字符串现在也会像不完整的多行字符串一样引发 `tokenize.TokenError`。
- 某些不完整或无效的 Python 代码现在会引发 `tokenize.TokenError` 而不是在执行分词时返回任意的 `ERRORTOKEN` 词元。
- 在同一文件中混合使用制表符和空格作为缩进不再受到支持而是会引发 `TabError`。

13 构建变化

- Python no longer uses `setup.py` to build shared C extension modules. Build parameters like headers and libraries are detected in `configure` script. Extensions are built by `Makefile`. Most extensions use `pkg-config` and fall back to manual detection. (Contributed by Christian Heimes in [gh-93939](#).)
- 现在需要用带有两个形参的 `va_start()`, 如 `va_start(args, format)`, 来构建 Python。现在将不会再调用单个形参的 `va_start()`。(由 Kumar Aditya 在 [gh-93207](#) 中贡献。)
- CPython now uses the `ThinLTO` option as the default link time optimization policy if the Clang compiler accepts the flag. (Contributed by Donghee Na in [gh-89536](#).)

- Add `COMPILEALL_OPTS` variable in Makefile to override `compileall` options (default: `-j0`) in `make install`. Also merged the 3 `compileall` commands into a single command to build `.pyc` files for all optimization levels (0, 1, 2) at once. (Contributed by Victor Stinner in [gh-99289](#).)
- 为 64 位 LoongArch 添加了平台三选项:
 - `loongarch64-linux-gnusr`
 - `loongarch64-linux-gnuf32`
 - `loongarch64-linux-gnu`
 (由 Zhang Na 在 [gh-90656](#) 中贡献。)
- `PYTHON_FOR_REGEN` 现在需要 Python 3.10 或更新版本。
- 现在需要有 `autoconf` 2.71 和 `aclocal` 1.16.4 才能重新生成 `!configure`。(由 Christian Heimes 在 [gh-89886](#) 中贡献。)
- 来自 python.org 的 Windows 版本和 macOS 安装程序现在使用 OpenSSL 3.0。

14 C API 的变化

14.1 新的特性

- **PEP 697**: Introduce the Unstable C API tier, intended for low-level tools like debuggers and JIT compilers. This API may change in each minor release of CPython without deprecation warnings. Its contents are marked by the `PyUnstable_` prefix in names.

代码对象构造器:

- `PyUnstable_Code_New()` (由 `PyCode_New` 改名而来)
- `PyUnstable_Code_NewWithPosOnlyArgs()` (由 `PyCode_NewWithPosOnlyArgs` 改名而来)

代码对象的额外存储 (**PEP 523**):

- `PyUnstable_Eval_RequestCodeExtraIndex()` (由 `_PyEval_RequestCodeExtraIndex` 改名而来)
- `PyUnstable_Code_GetExtra()` (由 `_PyCode_GetExtra` 改名而来)
- `PyUnstable_Code_SetExtra()` (由 `_PyCode_SetExtra` 改名而来)

原有名称将继续可用直到对应的 API 发生改变。

(由 Petr Viktorin 在 [gh-101101](#) 中贡献。)

- **PEP 697**: Add an API for extending types whose instance memory layout is opaque:
 - `PyType_Spec.basicsize` 可以为零或负数, 用于以指定继承或扩展基类的大小。
 - 增加了 `PyObject_GetTypeData()` 和 `PyType_GetTypeDataSize()` 以允许访问特定子类的实例数据。
 - 添加了 `Py_TPFLAGS_ITEMS_AT_END` 和 `PyObject_GetItemData()` 以允许安全地扩展某些可变大小的类型, 包括 `PyType_Type`。
 - 添加了 `Py_RELATIVE_OFFSET` 以允许用特定于子类的结构体来定义成员。

(由 Petr Viktorin 在 [gh-103509](#) 中贡献。)

- Add the new limited C API function `PyType_FromMetaclass()`, which generalizes the existing `PyType_FromModuleAndSpec()` using an additional metaclass argument. (Contributed by Wenzel Jakob in [gh-93012](#).)
- 在受限中添加了用于创建可使用 `vectorcall` 协议来调用的对象的 API:

- Py_TPFLAGS_HAVE_VECTORCALL
- PyVectorcall_NARGS()
- PyVectorcall_Call()
- vectorcallfunc

现在当一个类的 `__call__()` 方法被重新赋值时, 该类的 `Py_TPFLAGS_HAVE_VECTORCALL` 旗标将被移除。这使得 `vectorcall` 可以安全地用于可变类型 (即没有不可变旗标 `Py_TPFLAGS_IMMUTABLETYPE` 的堆类型)。未重载 `tp_call` 的可变类型现在继承了 `Py_TPFLAGS_HAVE_VECTORCALL` 旗标。(由 Petr Viktorin 在 [gh-93274](#) 中贡献。)

新增了 `Py_TPFLAGS_MANAGED_DICT` 和 `Py_TPFLAGS_MANAGED_WEAKREF` 旗标。这将允许扩展类在支持对象 `__dict__` 和弱引用时减少记录消耗, 占用更少内存并加快访问速度。

- 在受限 API 中添加了使用 `vectorcall` 协议执行调用的 API:

- `PyObject_Vectorcall()`
- `PyObject_VectorcallMethod()`
- `Py_VECTORCALL_ARGUMENTS_OFFSET`

这意味着 `vectorcall` 调用协议的传入端和传出端现在都可以在受限 API 中使用。(由 Wenzel Jakob 在 [gh-98586](#) 中贡献。)

- Add two new public functions, `PyEval_SetProfileAllThreads()` and `PyEval_SetTraceAllThreads()`, that allow to set tracing and profiling functions in all running threads in addition to the calling one. (Contributed by Pablo Galindo in [gh-93503](#).)
- Add new function `PyFunction_SetVectorcall()` to the C API which sets the `vectorcall` field of a given `PyFunctionObject`. (Contributed by Andrew Frost in [gh-92257](#).)
- C API 现在允许通过 `PyDict_AddWatcher()`、`PyDict_Watch()` 和相关 API 注册回调, 以便在字典被修改时调用。这主要用于优化解释器、JIT 编译器或调试器。(由 Carl Meyer 在 [gh-91052](#) 中贡献。)
- Add `PyType_AddWatcher()` and `PyType_Watch()` API to register callbacks to receive notification on changes to a type. (Contributed by Carl Meyer in [gh-91051](#).)
- Add `PyCode_AddWatcher()` and `PyCode_ClearWatcher()` APIs to register callbacks to receive notification on creation and destruction of code objects. (Contributed by Itamar Oren in [gh-91054](#).)
- 添加了 `PyFrame_GetVar()` 和 `PyFrame_GetVarString()` 函数用于通过名称来获取帧变量。(由 Victor Stinner 在 [gh-91248](#) 中贡献。)
- 添加 `PyErr_GetRaisedException()` 和 `PyErr_SetRaisedException()` 用于保存和恢复当前异常。这些函数返回并接受单个异常对象, 而不是像现在已弃用的 `PyErr_Fetch()` 和 `PyErr_Restore()` 那样的三个参数。这样不容易出错并且更为高效。(由 Mark Shannon 在 [gh-101578](#) 中贡献。)
- 添加了 `_PyErr_ChainExceptions1`, 它接受一个异常实例, 用于取代旧式 API `_PyErr_ChainExceptions`, 后者现已弃用。(由 Mark Shannon 在 [gh-101578](#) 中贡献。)
- 添加了 `PyException_GetArgs()` 和 `PyException_SetArgs()` 作为便捷函数用于检索和修改传递给异常的构造函数的 `args`。(由 Mark Shannon 在 [gh-101578](#) 中贡献。)
- 添加了 `PyErr_DisplayException()`, 它接受一个异常实例, 用于取代旧式 API `PyErr_Display()`。(由 Irit Katriel 在 [gh-102755](#) 中贡献。)
- **PEP 683**: Introduce *Immortal Objects*, which allows objects to bypass reference counts, and related changes to the C-API:
 - **`_Py_IMMORTAL_REFCNT`**: 定义对象的引用计数 为永生对象。
 - `_Py_IsImmortal` 检测一个对象是否具有永生引用计数。
 - **`PyObject_HEAD_INIT`** 这将把引用计数初始化为 `_Py_IMMORTAL_REFCNT` 当 配 合 `Py_BUILD_CORE` 使用时。

- `SSTATE_INTERNED_IMMORTAL` 一个针对内部 `unicode` 对象的标识符 为永生对象。
- `SSTATE_INTERNED_IMMORTAL_STATIC` 一个针对内部 `unicode` 为永生且静态的对象
- `sys.getunicodeinternedsize` 这将返回总计的 `unicode` objects that have been interned.
This is now needed for `refleak.py` to correctly track reference counts and allocated blocks

(由 Eddie Elizondo 在 [gh-84436](#) 中贡献。)

- **PEP 684:** Add the new `Py_NewInterpreterFromConfig()` function and `PyInterpreterConfig`, which may be used to create sub-interpreters with their own GILs. (See [PEP 684](#): 解释器级 *GIL* for more info.) (Contributed by Eric Snow in [gh-104110](#).)
- 在 3.12 版的受限 C API 中, `Py_INCREF()` 和 `Py_DECREF()` 函数现在使用不透明函数调用的方式实现以隐藏实现细节。(由 Victor Stinner 在 [gh-105387](#) 中贡献。)

14.2 移植到 Python 3.12

- 基于 `Py_UNICODE*` 表示形式的旧式 `Unicode` API 已被移除。请迁移到基于 `UTF-8` 或 `wchar_t*` 的 API。
- `PyArg_ParseTuple()` 等参数解析函数不再支持基于 `Py_UNICODE*` 的格式 (例如 `u`, `z` 等)。请迁移到其他 `Unicode` 格式如 `s`, `z`, `es` 和 `U`。
- `tp_weaklist` 对于所有静态内置类型将始终为 `NULL`。这是 `PyTypeObject` 上的一个内部专属字段, 但我们还是要指出这一变化以防有人碰巧仍然直接访问到该字段。为避免出现中断, 请考虑改用现有的公共 C-API, 或在必要时使用 (仅限内部使用的) 宏 `_PyObject_GET_WEAKREFS_LISTPTR()`。
- 现在这个内部专用的 `PyTypeObject.tp_subclasses` 可能不是一个有效的对象指针。为了反映这一点我们将其类型改为 `void*`。我们提到这一点是为了防止有人碰巧直接访问到这个内部专用字段。

要获取子类的列表, 请调用 Python 方法 `__subclasses__()` (例如使用 `PyObject_CallMethod()`)。

- 在 `PyUnicode_FromFormat()` 和 `PyUnicode_FromFormatV()` 中添加对更多格式选项 (左对齐、八进制、大写十六进制、`intmax_t`、`ptrdiff_t`、`wchar_t` C 字符串、可变宽度和精度) 的支持。(由 Serhiy Storchaka 在 [gh-98836](#) 中贡献。)
- `PyUnicode_FromFormat()` 和 `PyUnicode_FromFormatV()` 中未被识别的格式字符现在会设置一个 `SystemError`。在之前的版本中它会导致格式字符串的所有其他部分被原样复制到结果字符串中, 并丢弃任何额外的参数。(由 Serhiy Storchaka 在 [gh-95781](#) 中贡献。)
- **Fix wrong sign placement in `PyUnicode_FromFormat()` and `PyUnicode_FromFormatV()`.** (Contributed by Philip Georgi in [gh-95504](#).)
- 希望添加 `__dict__` 或弱引用槽位的扩展类应分别使用 `Py_TPFLAGS_MANAGED_DICT` 和 `Py_TPFLAGS_MANAGED_WEAKREF` 来代替 `tp_dictoffset` 和 `tp_weaklistoffset`。目前仍支持使用 `tp_dictoffset` 和 `tp_weaklistoffset`, 但并不完全支持多重继承 ([gh-95589](#)), 而且性能可能会变差。声明了 `Py_TPFLAGS_MANAGED_DICT` 的类应当调用 `_PyObject_VisitManagedDict()` 和 `_PyObject_ClearManagedDict()` 来遍历并清除它们的实例的字典。要清除弱引用, 请像之前一样调用 `PyObject_ClearWeakRefs()`。
- `PyUnicode_FSDecoder()` 函数不再接受类似字节串的路径, 如 `bytearray` 和 `memoryview` 类型: 只接受明确的 `bytes` 类型字节字符串。(由 Victor Stinner 在 [gh-98393](#) 中贡献。)
- `Py_CLEAR`、`Py_SETREF` 和 `Py_XSETREF` 宏现在只会对其参数求值一次。如果参数有附带影响, 这些附带影响将不会再重复。(由 Victor Stinner 在 [gh-98724](#) 中贡献。)
- 解释器的错误指示器现在总是规范化的。这意味着 `PyErr_SetObject()`、`PyErr_SetString()` 以及其他设置错误指示器的函数在保存异常之前都会将其规范化。(由 Mark Shannon 在 [gh-101578](#) 中贡献。)

- `_Py_RefTotal` 已不再具有重要性而保留它只是为了 ABI 的兼容性。请注意，这是一个内部全局变量并且仅在调试版本中可用。如果你碰巧要使用它那么你需要开始使用 `_Py_GetGlobalRefTotal()`。
- 下面的函数将为新创建的类型选择一个合适的元类：

- `PyType_FromSpec()`
- `PyType_FromSpecWithBases()`
- `PyType_FromModuleAndSpec()`

创建具有重载了 `tp_new` 的元类的类的做法已被弃用，在 Python 3.14+ 中将被禁止。请注意这些函数会忽略元类的 `tp_new`，从而可能导致不完整的初始化。

请注意 `PyType_FromMetaclass()` (在 Python 3.12 中新增) 已禁止创建具有重载了 `tp_new` (在 Python 中为 `__new__()`) 的元类的类。

由于 `tp_new` 重载了 “`PyType_From*`” 函数的几乎所有内容，因此两者互不兼容。现有的行为 -- 在创建类型的一些步骤中忽略元类 -- 通常都是不安全的，因为 (元) 类会假定 `tp_new` 已被调用。目前还没有简单通用的绕过方式。以下办法之一可能对你有帮助：

- 如果你控制着元类，请避免在其中使用 `tp_new`：
 - * 如初始化可被跳过，则可以改在 `tp_init` 中完成。
 - * 如果元类不需要从 Python 执行实例化，则使用 `Py_TPFLAGS_DISALLOW_INSTANTIATION` 旗标将其 `tp_new` 设为 `NULL`。这将使其可被 `PyType_From*` 函数接受。
- 避免使用 `PyType_From*` 函数：如果不需要 C 专属的特性（槽位或设置实例大小），请通过调用元类来创建类型。
- 如果你知道可以安全地跳过 `tp_new`，就使用 Python 中的 `warnings.catch_warnings()` 过滤掉弃用警告。
- `PyOS_InputHook` 和 `PyOS_ReadlineFunctionPointer` 将不再在子解释器中被调用。这是因为客户端通常依赖进程级的全局状态（而这些回调没有办法恢复扩展模块状态）。这也避免了扩展程序在不支持（或尚未被加载）的子解释器中运行的情况。请参阅 [gh-104668](#) 了解更多信息。
- `PyLongObject` 对其内部字段进行了修改以提高性能。虽然 `PyLongObject` 的内部字段是私有的，但某些扩展模块会使用它们。内部字段不应再被直接访问，而应改用以 `PyLong_...` 打头的 API 函数。新增了两个暂定 API 函数用于高效访问适配至单个机器字的 `PyLongObject` 的值：
 - `PyUnstable_Long_IsCompact()`
 - `PyUnstable_Long_CompactValue()`
- 通过 `PyMem_SetAllocator()` 设置的自定义分配器现在必须是线程安全的，无论内存域是什么。没有自己的状态的分配器，包括“钩子”将不会受影响。如果你的自定义分配器还不是线程安全的且你需要指导则请创建一个新的 GitHub 问题并抄送给 @ericcsnowcurrently。

14.3 弃用

- 根据 [PEP 699](#) 的要求，`PyDictObject` 中的 `ma_version_tag` 字段对于扩展模块已被弃用。访问该字段将在编译时生成编译器警告。该字段将在 Python 3.14 中移除。（由 Ramvikrams 和 Kumar Aditya 在 [gh-101193](#) 中贡献。PEP 由 Ken Jin 撰写。）
- 已弃用的全局配置变量：
 - `Py_DebugFlag`: 使用 `PyConfig.parser_debug`
 - `Py_VerboseFlag`: 使用 `PyConfig.verbose`
 - `Py_QuietFlag`: 使用 `PyConfig.quiet`
 - `Py_InteractiveFlag`: 使用 `PyConfig.interactive`

- `Py_InspectFlag`: 使用 `PyConfig.inspect`
- `Py_OptimizeFlag`: 使用 `PyConfig.optimization_level`
- `Py_NoSiteFlag`: 使用 `PyConfig.site_import`
- `Py_BytesWarningFlag`: 使用 `PyConfig.bytes_warning`
- `Py_FrozenFlag`: 使用 `PyConfig.pathconfig_warnings`
- `Py_IgnoreEnvironmentFlag`: 使用 `PyConfig.use_environment`
- `Py_DontWriteBytecodeFlag`: 使用 `PyConfig.write_bytecode`
- `Py_NoUserSiteDirectory`: 使用 `PyConfig.user_site_directory`
- `Py_UnbufferedStdioFlag`: 使用 `PyConfig.buffered_stdio`
- `Py_HashRandomizationFlag`: 使用 `PyConfig.use_hash_seed` 和 `PyConfig.hash_seed`
- `Py_IsolatedFlag`: 使用 `PyConfig.isolated`
- `Py_LegacyWindowsFSEncodingFlag`: 使用 `PyPreConfig.legacy_windows_fs_encoding`
- `Py_LegacyWindowsStdioFlag`: 使用 `PyConfig.legacy_windows_stdio`
- `Py_FileSystemDefaultEncoding`: 使用 `PyConfig.filesystem_encoding`
- `Py_HasFileSystemDefaultEncoding`: 使用 `PyConfig.filesystem_encoding`
- `Py_FileSystemDefaultEncodeErrors`: 使用 `PyConfig.filesystem_errors`
- `Py_UTF8Mode`: 使用 `PyPreConfig.utf8_mode` (参见 `Py_PreInitialize()`)

`Py_InitializeFromConfig()` API 应当改为使用 `PyConfig`。(由 Victor Stinner 在 [gh-77782](#) 中贡献。)

- `Creating immutable types with mutable bases is deprecated and will be disabled in Python 3.14.` ([gh-95388](#))
- `The structmember.h header is deprecated, though it continues to be available and there are no plans to remove it.`

Its contents are now available just by including `Python.h`, with a `Py` prefix added if it was missing:

- `PyMemberDef`, `PyMember_GetOne()` 和 `PyMember_SetOne()`
- 类型宏如 `Py_T_INT`, `Py_T_DOUBLE` 等 (之前为 `T_INT`, `T_DOUBLE` 等)
- 旗标 `Py_READONLY` (之前为 `READONLY`) 和 `Py_AUDIT_READ` (之前为全大写形式)

Several items are not exposed from `Python.h`:

- `T_OBJECT` (使用 `Py_T_OBJECT_EX`)
- `T_NONE` (之前未写入文档, 并且相当怪异)
- 不进行任何操作的宏 `WRITE_RESTRICTED`。
- `RESTRICTED` 和 `READ_RESTRICTED` 宏, 等同于 `Py_AUDIT_READ`。
- In some configurations, `<stddef.h>` is not included from `Python.h`. It should be included manually when using `offsetof()`.

已被弃用的头文件将继续以原来的名称提供原来的内容。你的旧代码可以保持不变, 除非额外的包括指令和无命名空间宏会给你带来很大困扰。

(由 Petr Viktorin 在 [gh-47146](#) 中贡献, 基于 Alexander Belopolsky 和 Matthias Braun 在先前的工作。)

- `PyErr_Fetch()` 和 `PyErr_Restore()` 已被弃用。请使用 `PyErr_GetRaisedException()` 和 `PyErr_SetRaisedException()` 代替。(由 Mark Shannon 在 [gh:101578](#) 贡献)。

- `PyErr_Display()` 已被弃用，请改用 `PyErr_DisplayException()`。（由 Irit Katriel 在 [gh-102755](#) 中贡献。）
- `_PyErr_ChainExceptions` 已被弃用。请改用 `_PyErr_ChainExceptions1`。（由 Irit Katriel 在 [gh-102192](#) 中贡献。）
- 使用 `PyType_FromSpec()`，`PyType_FromSpecWithBases()` 或 `PyType_FromModuleAndSpec()` 来创建所属元类重载了 `tp_new` 的类的做法已被弃用。请改为调用相应元类。is deprecated. Call the metaclass instead.

计划在 Python 3.14 中移除

- The `ma_version_tag` field in `PyDictObject` for extension modules ([PEP 699](#); [gh-101193](#)).
- Global configuration variables:
 - `Py_DebugFlag`: 使用 `PyConfig.parser_debug`
 - `Py_VerboseFlag`: 使用 `PyConfig.verbose`
 - `Py_QuietFlag`: 使用 `PyConfig.quiet`
 - `Py_InteractiveFlag`: 使用 `PyConfig.interactive`
 - `Py_InspectFlag`: 使用 `PyConfig.inspect`
 - `Py_OptimizeFlag`: 使用 `PyConfig.optimization_level`
 - `Py_NoSiteFlag`: 使用 `PyConfig.site_import`
 - `Py_BytesWarningFlag`: 使用 `PyConfig.bytes_warning`
 - `Py_FrozenFlag`: 使用 `PyConfig.pathconfig_warnings`
 - `Py_IgnoreEnvironmentFlag`: 使用 `PyConfig.use_environment`
 - `Py_DontWriteBytecodeFlag`: 使用 `PyConfig.write_bytecode`
 - `Py_NoUserSiteDirectory`: 使用 `PyConfig.user_site_directory`
 - `Py_UnbufferedStdioFlag`: 使用 `PyConfig.buffered_stdio`
 - `Py_HashRandomizationFlag`: 使用 `PyConfig.use_hash_seed` 和 `PyConfig.hash_seed`
 - `Py_IsolatedFlag`: 使用 `PyConfig.isolated`
 - `Py_LegacyWindowsFSEncodingFlag`: 使用 `PyPreConfig.legacy_windows_fs_encoding`
 - `Py_LegacyWindowsStdioFlag`: 使用 `PyConfig.legacy_windows_stdio`
 - `Py_FileSystemDefaultEncoding`: 使用 `PyConfig.filesystem_encoding`
 - `Py_HasFileSystemDefaultEncoding`: 使用 `PyConfig.filesystem_encoding`
 - `Py_FileSystemDefaultEncodeErrors`: 使用 `PyConfig.filesystem_errors`
 - `Py_UTF8Mode`: 使用 `PyPreConfig.utf8_mode` (参见 `Py_PreInitialize()`)

The `Py_InitializeFromConfig()` API should be used with `PyConfig` instead.

- Creating immutable types with mutable bases ([gh-95388](#)).

Pending Removal in Python 3.15

- `PyImport_ImportModuleNoBlock()`: use `PyImport_ImportModule()`
- `Py_UNICODE_WIDE` type: use `wchar_t`
- `Py_UNICODE` type: use `wchar_t`
- Python initialization functions:
 - `PySys_ResetWarnOptions()`: clear `sys.warnoptions` and `warnings.filters`
 - `Py_GetExecPrefix()`: get `sys.exec_prefix`
 - `Py_GetPath()`: get `sys.path`
 - `Py_GetPrefix()`: get `sys.prefix`
 - `Py_GetProgramFullPath()`: get `sys.executable`
 - `Py_GetProgramName()`: get `sys.executable`
 - `Py_GetPythonHome()`: get `PyConfig.home` or the `PYTHONHOME` environment variable

计划在未来版本中移除

The following APIs are deprecated and will be removed, although there is currently no date scheduled for their removal.

- `Py_TPFLAGS_HAVE_FINALIZE`: **unneded since Python 3.8**
- `PyErr_Fetch()`: use `PyErr_GetRaisedException()`
- `PyErr_NormalizeException()`: use `PyErr_GetRaisedException()`
- `PyErr_Restore()`: use `PyErr_SetRaisedException()`
- `PyModule_GetFilename()`: use `PyModule_GetFilenameObject()`
- `PyOS_AfterFork()`: use `PyOS_AfterFork_Child()`
- `PySlice_GetIndicesEx()`: use `PySlice_Unpack()` and `PySlice_AdjustIndices()`
- `PyUnicode_AsDecodedObject()`: use `PyCodec_Decode()`
- `PyUnicode_AsDecodedUnicode()`: use `PyCodec_Decode()`
- `PyUnicode_AsEncodedObject()`: use `PyCodec_Encode()`
- `PyUnicode_AsEncodedUnicode()`: use `PyCodec_Encode()`
- `PyUnicode_READY()`: **unneded since Python 3.12**
- `PyErr_Display()`: use `PyErr_DisplayException()`
- `_PyErr_ChainExceptions()`: use `_PyErr_ChainExceptions1`
- `PyBytesObject.ob_shash` member: call `PyObject_Hash()` instead
- `PyDictObject.ma_version_tag` member
- Thread Local Storage (TLS) API:
 - `PyThread_create_key()`: use `PyThread_tss_alloc()`
 - `PyThread_delete_key()`: use `PyThread_tss_free()`
 - `PyThread_set_key_value()`: use `PyThread_tss_set()`
 - `PyThread_get_key_value()`: use `PyThread_tss_get()`
 - `PyThread_delete_key_value()`: use `PyThread_tss_delete()`
 - `PyThread_ReInitTLS()`: **unneded since Python 3.7**

14.4 移除

- Remove the `token.h` header file. There was never any public tokenizer C API. The `token.h` header file was only designed to be used by Python internals. (Contributed by Victor Stinner in [gh-92651](#).)
- 旧式 Unicode API 已被移除。请参阅 [PEP 623](#) 了解详情。for detail.
 - `PyUnicode_WCHAR_KIND`
 - `PyUnicode_AS_UNICODE()`
 - `PyUnicode_AsUnicode()`
 - `PyUnicode_AsUnicodeAndSize()`
 - `PyUnicode_AS_DATA()`
 - `PyUnicode_FromUnicode()`
 - `PyUnicode_GET_SIZE()`
 - `PyUnicode_GetSize()`
 - `PyUnicode_GET_DATA_SIZE()`
- 移除了 `PyUnicode_InternImmortal()` 函数宏。(由 Victor Stinner 在 [gh-85858](#) 中贡献。)
- 从多个标准库模块和测试中移除了 Jython 兼容性处理。(由 Nikita Sobolev 在 [gh-99482](#) 中贡献。)
- 从 `ctypes` 模块移除了 `_use_broken_old_ctypes_structure_semantics_` 旗标。(由 Nikita Sobolev 在 [gh-99285](#) 中贡献。)

索引

非字母

环境变量

PYTHONHOME, 34

PYTHONPERFSUPPORT, 10

P

Python 提高建议

PEP 249, 13

PEP 451, 26

PEP 484, 4, 8

PEP 523, 28

PEP 554, 6

PEP 572, 9

PEP 594, 19, 21, 24

PEP 617, 6

PEP 623, 4, 16, 35

PEP 626, 19

PEP 632, 4, 22

PEP 644, 23

PEP 669, 7

PEP 678, 9

PEP 683, 29

PEP 684, 6, 30

PEP 688, 7

PEP 692, 8

PEP 693, 3

PEP 695, 4, 5, 17

PEP 697, 28

PEP 698, 9

PEP 699, 31, 33

PEP 701, 5, 6, 14, 16, 27

PEP 706, 10

PEP 709, 7, 17

PYTHONHOME, 34

PYTHONPERFSUPPORT, 10