What's New in Python

发行版本 *3.12.3*

A. M. Kuchling

五月 18, 2024

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: 每解释器 GIL . . . 7
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 . . . 8

3 有关类型提示的新增特性 **9**
3.1 PEP 692: 使用 `TypedDict` 进行更精确的 `**kwargs` 类型标注 . . . 9
3.2 PEP 698: 覆盖静态类型的装饰器 . . . 9

4 其他语言特性修改 **10**

5 新增模块 **11**

6 改进的模块 **11**
6.1 array . . . 11
6.2 asyncio . . . 11
6.3 calendar . . . 11
6.4 csv . . . 12
6.5 dis . . . 12
6.6 fractions . . . 12
6.7 importlib.resources . . . 12
6.8 inspect . . . 12
6.9 itertools . . . 12
6.10 math . . . 13
6.11 os . . . 13
6.12 os.path . . . 13
6.13 pathlib . . . 13
6.14 pdb . . . 14

6.15 random . . . 14
6.16 shutil . . . 14
6.17 sqlite3 . . . 14
6.18 statistics . . . 15
6.19 sys . . . 15
6.20 tempfile . . . 15
6.21 threading . . . 15
6.22 tkinter . . . 15
6.23 tokenize . . . 16
6.24 types . . . 16
6.25 typing . . . 16
6.26 unicodedata . . . 17
6.27 unittest . . . 17
6.28 uuid . . . 17

7 性能优化 **17**

8 CPython 字节码的改变 **18**

9 演示和工具 **18**

10 弃用 **18**
10.1 计划在 Python 3.13 中移除 . . . 21
10.2 计划在 Python 3.14 中移除 . . . 22
10.3 Python 3.15 中的待移除功能 . . . 23
10.4 计划在未来版本中移除 . . . 23

11 移除 **23**
11.1 asynchat 和 asyncore . . . 23
11.2 configparser . . . 23
11.3 distutils . . . 24
11.4 ensurepip . . . 24
11.5 enum . . . 24
11.6 ftplib . . . 24
11.7 gzip . . . 24
11.8 hashlib . . . 24
11.9 importlib . . . 25
11.10imp . . . 25
11.11io . . . 26
11.12locale . . . 26
11.13smtpd . . . 26
11.14sqlite3 . . . 26
11.15ssl . . . 27
11.16unittest . . . 27
11.17webbrowser . . . 28
11.18xml.etree.ElementTree . . . 28
11.19zipimport . . . 28
11.20其他事项 . . . 28

12 移植到 Python 3.12 **28**
12.1 Python API 的变化 . . . 28

13 构建的改变 **30**

14 C API 的改变 **30**

14.1 新的特性 30
14.2 移植到 Python 3.12 32
14.3 弃用 34
14.4 移除 37

15 3.12.4 中的重要变化 **38**
15.1 ipaddress 38

索引 **39**

编者
Adam Turner

本文介绍 Python 3.12 相比 3.11 增加的新特性。Python 3.12 已于 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 是 Python 编程语言的最新稳定发布版，包含一系列对语言和标准库的改变。库的改变主要集中在清理已弃用的 API、可用性和正确性等方面。值得注意的是，`distutils` 包已从标准库中移除。`os` 和 `pathlib` 中的文件系统支持增加了许多改进，而且部分模块的性能也获得了提升。

语言的改变主要集中在可用性方面，如 f-字符串的许多限制已被移除，而'Did you mean ...' 提示消息继续得到改进。新的*类型形参语法* 和 `type` 语句提升了 泛型类型和 类型别名配合静态类型检查器使用时的效率。

本文并不试图提供所有新功能的完整规范说明，而是提供一个方便的概览。如需了解完整细节，请参阅相应文档，如 标准库参考和 语言参考。如果你想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。

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*，f-字符串语法的改进

解释器的改进:

- *PEP 684*，单独的每解释器 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。
- 在 uuid 模块中添加了 命令行界面。
- 由于*PEP 701* 中的更改，通过 tokenize 模块生成令牌（token）的速度最多可提高 64%。

安全改进：

- 用来自 HACL* 项目的经过正式验证的代码替代 SHA1, SHA3, SHA2-384, SHA2-512 和 MD5 的内置 hashlib 实现。这些内置实现保留作为仅在当 OpenSSL 未提供它们时使用的回退选项。

C API 的改进：

- *PEP 697*，不稳定 C API 层
- *PEP 683*，永生对象

CPython 实现的改进：

- *PEP 709*，推导式内联化
- 对 Linux perf 性能分析器的 CPython 支持
- 在受支持的平台上实现栈溢出保护

新的类型标注特性：

- *PEP 692*，使用 TypedDict 来标注 **kwargs
- *PEP 698*，typing.override() 装饰器

重要的弃用、移除或限制：

- **PEP 623**: 在 Python 的 C API 中移除 Unicode 对象中的 wstr，使每个 str 对象的大小缩减至少 8 个字节。
- **PEP 632**: 移除 distutils 包。请参阅 迁移指南 了解有关替换其所提供的 API 的建议。第三方 Setuptools 包将继续提供 distutils，如果你在 Python 3.12 及更高版本中仍然需要它的话。
- gh-95299: 不在使用 venv 创建的虚拟环境中预装 setuptools。这意味着 distutils、setuptools、pkg_resources 和 easy_install 默认将不再可用；要访问这些工具请在 激活的虚拟环境中运行 pip install setuptools。
- 移除了 asynchat、asyncore 和 imp 模块，以及一些 unittest.TestCase *方法别名*。

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，并且需要显式声明变化。

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:
        ...
```

(续下页)

(接上页)

```
    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:

```
type IntFunc[**P] = Callable[P, int]  # ParamSpec
type LabeledTuple[*Ts] = tuple[str, *Ts]  # TypeVarTuple
type HashableSequence[T: Hashable] = Sequence[T]  # TypeVar with bound
type IntOrStrSequence[T: (int, str)] = Sequence[T]  # TypeVar with constraints
```

类型别名的值以及通过此语法创建的类型变量的边界和约束仅在需要时才进行求值 (参见 惰性求值)。这意味着类型别名可以引用稍后在文件中定义的其他类型。

通过类型参数列表声明的类型参数在声明的作用域和任何嵌套的作用域内都可见，但在外部作用域内不可见。例如，它们可以用于泛型类的方法的类型注解或类体中。但是，在定义类之后，不能在模块范围中使用它们。有关类型参数的运行时语义的详细描述，请参见 type-params。

为了支持这些作用域定义，引入了一种新的作用域，即 标注作用域。标注作用域的行为在很大程度上类似于函数作用域，但与封闭类作用作用域的交互方式不同。在 Python 3.13 中，标注也将在标注作用域中进行求值。

更多细节请参见 **PEP 695**。

（PEP 由 Eric Traut 撰写。由 Jelle Zijlstra、Eric Traut 和其他人在 gh-103764 中实现。）

2.2 PEP 701：f-字符串的句法形式化

PEP 701 取消了对 f-字符串使用的一些限制。f-字符串内部的表达式部分现在可以是任何有效的 Python 表达式，包括重用了与标记 f-字符串本身相同的引号的字符串、多行表达式、注释、反斜杠以及 unicode 转义序列。让我们详细介绍一下：

- 引号重用：在 Python 3.11 中，重用与标记 f-字符串本身相同的引号会引发 `SyntaxError`，迫使用户使用其他可用的引号（如在 f-字符串使用单引号时使用双引号或三重引号）。在 Python 3.12 中，你现在可以这样做了：

  ```
  >>> songs = ['Take me back to Eden', 'Alkaline', 'Ascensionism']
  >>> f"This is the playlist: {", ".join(songs)}"
  'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
  ```

  请注意，在这一更改之前，对 f-字符串的嵌套方式没有明确的限制，但字符串引号不能在 f-字符串的表达式组件中重复使用，这使得不可能任意嵌套 f-字符串。事实上，这是可以编写的嵌套最多的 f-字符串：

  ```
  >>> f"""{f'''{f'{f"{1+1}"}'}'''}"""
  '2'
  ```

由于现在 f-字符串可以在表达式组件中包含任何有效的 Python 表达式，因此现在可以任意嵌套 f-字符串：

```
>>> f"{f"{f"{f"{f"{1+1}"}"}"}"}"}"
'2'
```

- 多行表达式和注释：在 Python 3.11 中，f-字符串表达式必须在一行中完成定义，即使 f-字符串中的表达式在正常情况下可以跨多行（如在多行中定义的列表字面值），这使得它们更难被读懂。在 Python 3.12 中，你现在可以定义跨越多行的 f-字符串并添加内联注释：

```
>>> f"This is the playlist: {", ".join([
...     'Take me back to Eden',  # My, my, those eyes like fire
...     'Alkaline',              # Not acid nor alkaline
...     'Ascensionism'           # Take to the broken skies at last
... ])}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

- 反斜杠和 unicode 字符：在 Python 3.12 之前，f-字符串表达式不能包含任何 \ 字符。这也影响了 unicode 转义序列 (如 `\N{snowman}`)，因为这些序列包含 `\N` 部分，而这部分以前不能作为 f-字符串表达式组件的一部分。现在，你可以这样定义表达式：

```
>>> 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**。

实现此特性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: 每解释器 GIL

PEP 684 引入了每解释器 GIL，使得现在可以创建带有单独的每解释器 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 用于具有每解释器 GIL 的子解释器的更多示例，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。

（由 Mark Shannon 在 gh-103082 中贡献。）

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):
...         somethin = blech
...
>>> A().foo()
Traceback (most recent call last):
  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
Traceback (most recent call last):
  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用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` 现在支持半精度浮点类型 (”e” 格式代码)。(由 Donghee Na 和 Antoine Pitrou 在 gh-90751 中贡献。)
- `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()` 现在使用 Neumaier 求和算法以改善对浮点数或混合了整数和浮点数时求和运算的准确性和可换算性。(由 Raymond Hettinger 在 gh-100425 中贡献。)
- `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。）
- 添加了 `asyncio.eager_task_factory()` 和 `asyncio.create_eager_task_factory()` 函数以允许在主动型任务执行中选择事件循环，使某些用例的速度提升了 2 至 5 倍。（由 Jacob Bower 和 Itamar Oren 在 gh-102853, gh-104140 和 gh-104138 中贡献。）
- 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。）
- 现在事件循环会针对每个平台使用最佳的可用子监视器 (在受支持的情况下使用 `asyncio.PidfdChildWatcher`，否则使用 `asyncio.ThreadedChildWatcher`)，因此不建议手动配置子监视器。（由 Kumar Aditya 在 gh-94597 中贡献。）
- 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。）
- 添加了 `dis.hasexc` 多项集来表示设置异常处理器的指令。（由 Irit Katriel 在 gh-94216 中贡献。）

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。）
- 将 `importlib.resources.files()` 的第一个形参重命名为 *anchor*。（由 Jason R. Coombs 在 gh-100598 中贡献。）

6.8 inspect

- 增加了 `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。）
- 添加 `inspect.getasyncgenstate()` 和 `inspect.getasyncgenlocals()` 用来确定异步发生器的当前状态。（由 Thomas Krennwallner 在 gh-79940 中贡献。）
- `inspect.getattr_static()` 的性能已得到显著提升。对该函数的大多数调用相比 Python 3.11 至少应有 2 倍的加速。（由 Alex Waygood 在 gh-103193 中贡献。）

6.9 itertools

- 增加了 `itertools.batched()` 用来将数据收集为相同大小的元组，其中最后一个批次的长度可能会比其他批次的短。（由 Raymond Hettinger 在 gh-98363 中贡献。）

6.10 math

- 添加了 `math.sumprod()` 用于计算乘积之和。(由 Raymond Hettinger 在 gh-100485 中贡献。)
- 扩展了 `math.nextafter()` 以包括一个 *steps* 参数用于一次性向上或向下移动多步。(由 Matthias Goergens, Mark Dickinson 和 Raymond Hettinger 在 gh-94906 中贡献。)

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.listdrives()`、`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。)
- As of 3.12.4, `os.mkdir()` and `os.makedirs()` on Windows now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects `tempfile.mkdtemp()` and is a mitigation for CVE-2024-4030. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in gh-118486.)

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。)
- 在 `pathlib.PurePath.relative_to()` 中添加了 *walk_up* 可选形参以允许在结果中插入 ".." 条目；此行为与 `os.path.relpath()` 更为一致。(由 Domenico Ragusa 在 gh-84538 中贡献。)
- 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

- 添加了 `random.binomialvariate()`。（由 Raymond Hettinger 在 gh-81620 中贡献。）
- 在 `random.expovariate()` 中添加默认值 `lambd=1.0` 。（由 Raymond Hettinger 在 gh-100234 中贡献。）

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，但它接受一个异常实例而不是一个 *(typ, val, tb)* 三元组。*onerror* 已被弃用。（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

- 扩展 `statistics.correlation()` 以 `ranked` 方法的形式包括对分级数据的斯皮尔曼相关性计算。(由 Raymond Hettinger 在 gh-95861 中贡献。)

6.19 sys

- 添加了 `sys.monitoring` 命名空间以公开新的*PEP 669* 监控 API。(由 Mark Shannon 在 gh-103082 中贡献。)
- 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。
- As of 3.12.4 on Windows, the default mode `0o700` used by `tempfile.mkdtemp()` now limits access to the new directory due to changes to `os.mkdir()`. This is a mitigation for CVE-2024-4030. (Contributed by Steve Dower in gh-118486.)

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()` 会展平其参数。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就将被视为 "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。对于仅有少量成员的协议的大部分 `isinstance()` 检测相比 3.11 应当至少有 2 倍的加速。不过，对于具有大量成员的协议的 `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

增加了 `--durations` 命令行选项，显示 N 个最慢的测试用例:

```
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)
```

（由 Giampaolo Rodola 在 gh-48330 中贡献。）

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

- 从 Unicode 对象中移除了 `wstr` 和 `wstr_length` 成员。这使得对象大小在 64 位平台上减少了 8 个或 16 个字节。(**PEP 623**)（由 Inada Naoki 在 gh-92536 中贡献。）
- 增加了在构建进程中使用 BOLT 二进制优化器的实验性支持，这将使得性能提升 1-5%。（由 Kevin Modzelewski 在 gh-90536 中贡献并由 Donghee Na 在 gh-101525 中微调。）
- 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。)
- 移除了 PRECALL 指令。(由 Mark Shannon 在 gh-92925 中贡献。)
- 添加了 BINARY_SLICE 和 STORE_SLICE 指令。(由 Mark Shannon 在 gh-94163 中贡献。)
- 添加了 CALL_INTRINSIC_1 指令。(由 Mark Shannon 在 gh-99005 中贡献。)
- 添加了 CALL_INTRINSIC_2 指令。(由 Irit Katriel 在 gh-101799 中贡献。)
- 添加了 CLEANUP_THROW 指令。(由 Brandt Bucher 在 gh-90997 中贡献。)
- 添加了 END_SEND 指令。(由 Mark Shannon 在 gh-103082 中贡献。)
- 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。)
- 添加了 LOAD_FAST_CHECK 指令。(由 Dennis Sweeney 在 gh-93143 中贡献。)
- 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。)
- 添加了 RETURN_CONST 指令。(由 Wenyang Wang 在 gh-101632 中贡献。)

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: 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:
  - 子监视器类 asyncio.MultiLoopChildWatcher, asyncio.FastChildWatcher, asyncio.AbstractChildWatcher 和 asyncio.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。）
- 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: 已弃用 email.utils.localtime() 中的 *isdst* 形参。（由 Alan Williams 在 gh-72346 中贡献。）
- 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: 已弃用对 copy、deepcopy 和 pickle 操作的支持，它们未被写入文档、效率低下、历史上充满问题且缺乏一致性。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: 在 Python 3.14 中，默认的 multiprocessing 启动方法将在 Linux、BSD 和其他非 macOS 的 POSIX 平台上改为更安全的方法，在这些平台上目前默认为 'fork' (gh-84559)。在运行时添加相关警告被认为干扰性太大因为大部分代码都不会在意这个问题。当你的代码 *需要* 'fork' 时请使用 get_context() 或 set_start_method() API 显式地指明。参见 上下文和启动方法。
- pkgutil: pkgutil.find_loader() 和 pkgutil.get_loader() 已被弃用并将在 Python 3.14 中移除；请改用 importlib.util.find_spec()。（由 Nikita Sobolev 在 gh-97850 中贡献。）
- pty: 该模块有两个未写入文档的 master_open() 和 slave_open() 函数自 Python 2 起即已被弃用但直到 3.12 才添加了相应的 DeprecationWarning。它们将在 3.14 中移除。（由 Soumendra Ganguly 和 Gregory P. Smith 在 gh-85984 中贡献。）.)
- 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。)
  - 在 POSIX 平台上，当 `os.fork()` 检测到被多线程的进程调用时现在会引发 `DeprecationWarning`。当在 POSIX 平台上这样做时始终会存在功能上的不兼容性。即使这样的代码 *看起来*有效。我们添加该警告是为了引起人们的注意因为这样做遇到的问题变得越来越多。请参阅 `os.fork()` 文档了解详情并查看 关于 fork 与线程不兼容问题的讨论 以了解 为什么我们现在要向开发者公开这一长期存在的平台兼容性问题。

  当由于使用 `multiprocessing` 或 `concurrent.futures` 而出现此警告时的解决办法是使用其他的 `multiprocessing` 启动方法如 `"spawn"` 或 `"forkserver"`。

- `shutil`: `shutil.rmtree()` 的 *onerror* 参数已被弃用；请改用 *onexc*。(由 Irit Katriel 在 gh-102828 中贡献。)
- `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`，分别为 `collections.abc.Hashable` 和 `collections.abc.Sized` 的别名，现已被弃用。(gh-94309。)
  - `typing.ByteString` 自 Python 3.9 起已被弃用，现在当被使用时将会发出 `DeprecationWarning`。(由 Alex Waygood 在 gh-91896 中贡献。)
- `xml.etree.ElementTree`: 现在该模块在对 `xml.etree.ElementTree.Element` 执行真值测试时将发出 `DeprecationWarning`。在之前，Python 实现会发出 `FutureWarning`，而 C 实现则不会发出任何警告。(由 Jacob Walls 在 gh-83122 中贡献。)
- `coroutine throw()`, `generator throw()` 和 `async generator throw()` 的三参数签名形式 (type, value, traceback) 已被弃用并可能在未来的 Python 版本中移除。请改用这些函数的单参数版本。(由 Ofey Chan 在 gh-89874 中贡献。)
- 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。)
- 在模块上设置 `__package__` 或 `__cached__` 已被弃用，在 Python 3.14 中导入系统将不再设置或考虑这些属性。(由 Brett Cannon 在 gh-65961 中贡献。)
- 对布尔值的位取反运算符 (~) 已被弃用。它在 Python 3.14 中将抛出错误。请改用 `not` 来执行布尔值的逻辑非运算。在你确实需要对下层的 `int` 执行按位取反运算的少数场景下，请显式地将其转换为整数值: `~int(x)`。(由 Tim Hoffmann 在 gh-103487 中贡献。)
- 在代码对象上对 `co_lnotab` 的访问自 Python 3.10 起已通过 **PEP 626** 被弃用，但直到 3.12 才添加了适当的 `DeprecationWarning`，因而它将在 3.14 中被移除。(由 Nikita Sobolev 在 gh-101866 中贡献。)

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.resetlocale()` (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)
- `importlib.resources` 中已弃用的方法:
  - `contents()`

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

  请改用 `importlib.resources.files()`。参见 importlib-resources: Migrating from Legacy (gh-106531)

10.2 计划在 Python 3.14 中移除

以下 API 已被弃用并将在 Python 3.14 中移除。

- `argparse`: `argparse.BooleanOptionalAction` 的 *type*, *choices* 和 *metavar* 形参
- `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`: `email.utils.localtime()` 中的 *isdst* 形参。
- `importlib.abc`:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`
- `itertools`: 对 copy, deepcopy 和 pickle 操作的支持。
- `pkgutil`:
  - `pkgutil.find_loader()`
  - `pkgutil.get_loader()`。

- pty:
  - pty.master_open()
  - pty.slave_open()
- shutil: shutil.rmtree() 的 *onerror* 参数
- typing: typing.ByteString
- xml.etree.ElementTree: 对 xml.etree.ElementTree.Element 的真值测试。
- 模块对象上的 __package__ 和 __cached__ 属性。
- 代码对象的 co_lnotab 属性。

10.3 Python 3.15 中的待移除功能

以下 API 已被弃用并将在 Python 3.15 中移除。

API:

- locale.getdefaultlocale() (gh-90817)

10.4 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 接受数字类字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。）

  要进行迁移，请参考以下对照表：

imp	importlib
`imp.NullImporter`	将 `None` 插入到 `sys.path_importer_cache`
`imp.cache_from_s`	`importlib.util.cache_from_source()`
`imp.find_module(`	`importlib.util.find_spec()`
`imp.get_magic()`	`importlib.util.MAGIC_NUMBER`
`imp.get_suffixes`	`importlib.machinery.SOURCE_SUFFIXES`, `importlib.machinery.EXTENSION_SUFFIXES` 和 `importlib.machinery.BYTECODE_SUFFIXES`
`imp.get_tag()`	`sys.implementation.cache_tag`
`imp.load_module(`	`importlib.import_module()`
`imp.new_module(r`	`types.ModuleType(name)`
`imp.reload()`	`importlib.reload()`
`imp.source_from_`	`importlib.util.source_from_cache()`
`imp.load_source(`	见下文

  将 `imp.load_source()` 替换为：

```
import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(modname, filename)
```

(续下页)

(接上页)

```
    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
```

- 已移除 imp 的函数和属性并且没有替代选项:
  - 未写入文档的函数:
    - 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。)

11.13 smtpd

- smtpd 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 的服务器。(由 Oleg Iarygin 在 gh-93243 中贡献。)

11.14 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。）

11.15 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：应改为创建一个 `ssl.SSLContext` 对象并调用其 `ssl.SSLContext.wrap_socket` 方法。任何仍然使用 `ssl.wrap_socket()` 的包都是已不适用且不安全的。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 (Improper Certificate Validation) 的影响。（由 Victor Stinner 在 gh-94199 中贡献。）

11.16 unittest

- 移除了许多早已弃用的 `unittest` 特性：
  - 一些 `TestCase` 方法的别名：

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

  （由 Serhiy Storchaka 在 gh-89325 中贡献。）

11.17 webbrowser

- 从 webbrowser 移除了对过时浏览器的支持。被移除的浏览器包括：Grail、Mosaic、Netscape、Galeon、Skipstone、Iceape、Firebird 和 Firefox 35 及以下的版本 (gh-102871)。

11.18 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。）

11.19 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。）

11.20 其他事项

- 从文档 Makefile 和 Doc/tools/rstlint.py 中移除了 suspicious 规则，请改用 sphinx-lint。（由 Julien Palard 在 gh-98179 中贡献。）
- 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。）.)
- 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。）

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。）
- 移除了自 Python 3.10 起已被弃用的 randrange() 功能。以前，randrange(10.0) 会无损地转换为 randrange(10)。现在，它将引发 TypeError。此外，对于非整数值如 randrange(10.5) 或 randrange('10') 所引发的异常已从 ValueError 改为 TypeError。这也防止了 randrange(1e25) 会从比 randrange(10**25) 更大的范围中静默选择的问题。（最初由 Serhiy Storchaka 在 gh-86388 中提议。）
- 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器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。

- 移除了在 Python 3.4.7 和 3.5.4 中已被弃用的基于 `asyncore` 的 `smtpd` 模块。推荐的替代是基于 `asyncio` 的 aiosmtpd PyPI 模块。
- `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。)
- 现在 `syslog.openlog()` 和 `syslog.closelog()` 如果在子解释器中使用将失败。`syslog.syslog()` 仍可在子解释器中使用，但前提是 `syslog.openlog()` 已在主解释器中被调用。这些新限制不适用于主解释器，因此只有少数用户可能会受到影响。这一改变有助于实现解释器隔离。此外，`syslog` 是一个针对进程全局资源的包装器，而这些资源最好是由主解释器来管理。(由 Donghee Na 在 gh-99127 中贡献。)
- 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:

```
1,0-1,18:           STRING         'f"start {1+1} end"'
```

  而新版本将生成:

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

  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`。
- 现在 `threading` 模块会预期 `_thread` 模块具有 `_is_main_interpreter` 属性。它是一个不带参数的函数并会在当前解释器为主解释器时返回 `True`。

任何提供了自定义 _thread 模块的库或应用程序都应当提供 _is_main_interpreter()。(参见 gh-112826。)

13 构建的改变

- Python 不再使用 setup.py 来构建共享的 C 扩展模块。头文件和库等编译参数在 configure 脚本中检测。扩展将由 Makefile 来构建。大多数扩展使用 pkg-config 并回退为手动检测。(由 Christian Heimes 在 gh-93939 中贡献。)
- 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。)
- 现在如果 Clang 编译器接受 ThinLTO 选项则 CPython 会将其作为默认的链接时间优化策略。(由 Donghee Na 在 gh-89536 中贡献。)
- 在 Makefile 中添加了 COMPILEALL_OPTS 变量以覆盖 make install 中的 compileall 选项 (默认值: -j0)。并将 3 条 compileall 命令合并为单条命令以便一次性构建所有优化级别 (0, 1, 2) 的.pyc 文件。(由 Victor Stinner 在 gh-99289 中贡献。)
- 为 64 位 LoongArch 添加了平台三选项:
  - loongarch64-linux-gnusf
  - 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 的改变

14.1 新的特性

- **PEP 697**: 引入了 不稳定 C API 层，用于调试器和 JIT 编译器等低层级工具。该 API 可能会在 CPython 的每个次要版本中发生变化而但发出弃用警告。其内容在名称中以 PyUnstable_ 前缀标记。

  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**: 添加了用于扩展实例内存布局不透明的类型的 API:
  - `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。）
- 添加了新的 受限 C API 函数 `PyType_FromMetaclass()`，它使用了额外的 metaclass 参数对现有的 `PyType_FromModuleAndSpec()` 进行了泛化。（由 Wenzel Jakob 在 gh-93012 中贡献。）
- 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。）
- 添加了两个新的公共函数 `PyEval_SetProfileAllThreads()` 和 `PyEval_SetTraceAllThreads()`，允许在调用的同时在所有运行线程中设置追踪和性能分析函数。（由 Pablo Galindo 在 gh-93503 中贡献。）
- 为 C API 添加了新函数 `PyFunction_SetVectorcall()` 用于设置给定 `PyFunctionObject` 的 vectorcall 字段。（由 Andrew Frost 在 gh-92257 中贡献。）
- 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。）
- 添加了 `PyType_AddWatcher()` 和 `PyType_Watch()` API 用于注册回调以接收类型变更通知。（由 Carl Meyer 在 gh-91051 中贡献。）
- 添加了 `PyCode_AddWatcher()` 和 `PyCode_ClearWatcher()` API 用于注册回调以接收代码对象创建和销毁时的通知。（由 Itamar Oren 在 gh-91054 中贡献。）

- 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，后者现已被弃用。(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**: 引入了 永生对象，它允许对象绕过引用计数，并对 C-API 进行相应修改：
  - **_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**
    被管理的对象。现在 refleak.py 需要这样才能正确地追踪引用计数和分配的块

  (由 Eddie Elizondo 在 gh-84436 中贡献。)
- **PEP 684**: 新增了 Py_NewInterpreterFromConfig() 函数和 PyInterpreterConfig，可用于创建具有单独 GIL 的子解释器。(更多信息参见*PEP 684: 每解释器 GIL*。)(由 Eric Snow 在 gh-104110 中贡献。)
- 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。）
- 修复了 PyUnicode_FromFormat() 和 PyUnicode_FromFormatV() 中错误的标志位置。（由 Philip Georgi 在 gh-95504 中贡献。）
- 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用:
  - 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 `@ericsnowcurrently`。

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。）

- 使用可变的基类创建 不可变类型的做法已被弃用并将在 Python 3.14 中被禁用。(gh-95388)
- structmember.h 头文件已被弃用，不过它仍可继续使用也没有计划将其移除。

  现在只需包括 Python.h 即可获得其内容，如果找不到请添加 Py 前缀：

  - 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)

  Python.h 上有几个项目没有暴露：

  - 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。
  - 在某些配置中，Python.h 未包含 <stddef.h> 。使用 offsetof() 时，应手动将其包含在内。

  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

- PyDictObject 中的 ma_version_tag 字段用于扩展模块 (**PEP 699** ; gh-101193)。
- 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 应与 PyConfig 一起使用。
- 创建 immutable types 的可变基础 (gh-95388)。

Python 3.15 中的待移除功能

- PyImport_ImportModuleNoBlock()：使用 PyImport_ImportModule()
- Py_UNICODE_WIDE 类型：使用 wchar_t
- Py_UNICODE 类型：使用 wchar_t
- Python 初始化函数
  - PySys_ResetWarnOptions()：清除 sys.warnoptions 和 warnings.filters
  - Py_GetExecPrefix()：获取 sys.exec_prefix
  - Py_GetPath()：获取 sys.path
  - Py_GetPrefix()：获取 sys.prefix
  - Py_GetProgramFullPath()：获取 sys.executable
  - Py_GetProgramName()：获取 sys.executable
  - Py_GetPythonHome()：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()`：使用 `PyErr_GetRaisedException()`
- `PyErr_NormalizeException()`：使用 `PyErr_GetRaisedException()`
- `PyErr_Restore()`：使用 `PyErr_SetRaisedException()`
- `PyModule_GetFilename()`：使用 `PyModule_GetFilenameObject()`
- `PyOS_AfterFork()`：使用 `PyOS_AfterFork_Child()`
- `PySlice_GetIndicesEx()`： 使用 `PySlice_Unpack()` 和 `PySlice_AdjustIndices()`
- `PyUnicode_AsDecodedObject()`：使用 `PyCodec_Decode()`
- `PyUnicode_AsDecodedUnicode()`：使用 `PyCodec_Decode()`
- `PyUnicode_AsEncodedObject()`：使用 `PyCodec_Encode()`
- `PyUnicode_AsEncodedUnicode()`：使用 `PyCodec_Encode()`
- `PyUnicode_READY()`： 自 Python 3.12 起不再需要
- `PyErr_Display()`：使用 `PyErr_DisplayException()`
- `_PyErr_ChainExceptions()`：使用 `_PyErr_ChainExceptions1`
- `PyBytesObject.ob_shash` 成员：改用 `PyObject_Hash()`
- `PyDictObject.ma_version_tag` 成员
- 线程本地存储 (TLS) API：
  - `PyThread_create_key()`： 使用 `PyThread_tss_alloc()`
  - `PyThread_delete_key()`： 使用 `PyThread_tss_free()`
  - `PyThread_set_key_value()`： 使用 `PyThread_tss_set()`
  - `PyThread_get_key_value()`：使用 `PyThread_tss_get()`
  - `PyThread_delete_key_value()`：使用 `PyThread_tss_delete()`
  - `PyThread_ReInitTLS()`： 自 Python 3.7 起不再需要

14.4 移除

- 移除 `token.h` 头文件。从来就没有任何公开的 C 语言标记程序接口。`token.h` 头文件只是为 Python 内部使用而设计的。(由 Victor Stinner 在 gh-92651 提供)。
- 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。）.)

15 3.12.4 中的重要变化

15.1 ipaddress

- 修正了 `IPv4Address`，`IPv6Address`，`IPv4Network` 和 `IPv6Network` 中的 `is_global` 和 `is_private` 行为。

索引

非字母

环境变量
    PYTHONHOME, 36
    PYTHONPERFSUPPORT, 10

P

Python 增强建议; PEP 249, 14
Python 增强建议; PEP 451, 28
Python 增强建议; PEP 484, 4, 9
Python 增强建议; PEP 523, 30
Python 增强建议; PEP 554, 7
Python 增强建议; PEP 572, 10
Python 增强建议; PEP 594, 21, 23, 26
Python 增强建议; PEP 617, 6
Python 增强建议; PEP 623, 4, 17, 37
Python 增强建议; PEP 626, 20
Python 增强建议; PEP 632, 4, 24
Python 增强建议; PEP 644, 24
Python 增强建议; PEP 669, 7
Python 增强建议; PEP 678, 10
Python 增强建议; PEP 683, 32
Python 增强建议; PEP 684, 7, 32
Python 增强建议; PEP 688, 7
Python 增强建议; PEP 692, 9
Python 增强建议; PEP 693, 3
Python 增强建议; PEP 695, 4, 5, 18
Python 增强建议; PEP 697, 30, 31
Python 增强建议; PEP 698, 9
Python 增强建议; PEP 699, 34, 35
Python 增强建议; PEP 701, 5, 6, 16, 17, 29
Python 增强建议; PEP 706, 10
Python 增强建议; PEP 709, 7, 8, 18
PYTHONHOME, 36
PYTHONPERFSUPPORT, 10