What's New in Python

发布 *3.11.6*

A. M. Kuchling

十月 27, 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 657：回溯信息中标注更详细的错误位置 4
2.2 PEP 654：异常组与 `except*` 5
2.3 PEP 678：可用注释丰富异常 5
2.4 Windows 下的 `py.exe` 启动器改进 5

3 有关类型提示的新增特性 **5**
3.1 PEP 646：可变参数泛型 5
3.2 PEP 655：将单个 `TypedDict` 项标记为必填或非必填项 6
3.3 PEP 673：`Self` 类型 6
3.4 PEP 675：任意字面值字符串类型 7
3.5 PEP 681：数据类变换 7
3.6 未来版本可能不再实现 PEP 563 8

4 其他语言特性修改 **8**

5 其他 CPython 实现的改变 **8**

6 新增模块 **9**

7 改进的模块 **9**
7.1 asyncio 9
7.2 contextlib 10
7.3 dataclasses 10
7.4 datetime 10
7.5 enum 10
7.6 fcntl 11
7.7 fractions 11
7.8 functools 11
7.9 hashlib 11
7.10 IDLE 与 idlelib 12
7.11 inspect 12

7.12 locale . . . 12
7.13 logging . . . 12
7.14 math . . . 12
7.15 operator . . . 13
7.16 os . . . 13
7.17 pathlib . . . 13
7.18 re . . . 13
7.19 shutil . . . 13
7.20 socket . . . 13
7.21 sqlite3 . . . 13
7.22 string . . . 14
7.23 sys . . . 14
7.24 sysconfig . . . 14
7.25 tempfile . . . 14
7.26 threading . . . 15
7.27 time . . . 15
7.28 tkinter . . . 15
7.29 回溯 . . . 15
7.30 typing . . . 15
7.31 unicodedata . . . 16
7.32 unittest . . . 16
7.33 venv . . . 16
7.34 warnings . . . 16
7.35 zipfile . . . 17

8 性能优化 **17**

9 更快的 CPython **17**
9.1 更快的启动 . . . 18
9.2 更快的运行时 . . . 18
9.3 杂项 . . . 20
9.4 常见问题 . . . 20
9.5 关于 . . . 20

10 CPython 字节码的改变 **21**
10.1 新的操作码 . . . 21
10.2 被替换的操作码 . . . 22
10.3 修改/移除的操作码 . . . 22

11 弃用 **23**
11.1 语言/内置对象 . . . 23
11.2 模块 . . . 23
11.3 标准库 . . . 23

12 计划在 Python 3.12 中移除 **25**

13 移除 **26**

14 移植到 Python 3.11 **27**

15 构建的改变 **28**

16 C API 的改变 **29**
16.1 新的特性 . . . 29
16.2 移植到 Python 3.11 . . . 30

16.3 弃用 34
16.4 计划在 Python 3.12 中移除 35
16.5 移除 35

17 3.11.4 中的重要变化 **37**
17.1 tarfile 37

18 3.11.5 中的重要变化 **37**
18.1 OpenSSL 37

索引 **38**

编者 Pablo Galindo Salgado

这篇文章介绍了 Python 3.11 相比 3.10 增加的新特性。Python 3.11 发布于 2022 年 10 月 24 日。要了解更详细的信息，可参阅 更新日志。

1 摘要 -- 发布重点

- Python 3.11 的速度比 Python 3.10 快 10-60%。在平均状况下，在标准基准测试（standard benchmark suite）中可见 1.25 倍的加速效果。更多细节请参见*更快的 CPython* 一节。

新的语法特性：

- *PEP 654*：*异常组与 except**

新的内置特性：

- *PEP 678*：*可用注释丰富异常*

新的标准库模块：

- **PEP 680**: `tomllib` —标准库中对解析 TOML 的支持

解释器的改进:

- *PEP 657*：*回溯信息中标注更详细的错误位置*
- 新增 `-P` 命令行选项以及 `PYTHONSAFEPATH` 环境变量来*禁止自动将潜在的不安全路径前置* 到 `sys.path`

新的类型标注特性：

- *PEP 646*：*可变参数泛型*
- *PEP 655*：*将单个 TypedDict 项标记为必填或非必填项*
- *PEP 673*：*Self 类型*
- *PEP 675*：*任意字面值字符串类型*
- *PEP 681*：*数据类变换*

重要的弃用、移除或限制：

- **PEP 594**: *许多旧标准库模块已被弃用*，并将在 Python 3.13 中移除
- **PEP 624**: *Py_UNICODE 编码器 API 已被移除*
- **PEP 670**: *转换为静态内联函数的宏*

2 新的特性

2.1 PEP 657：回溯信息中标注更详细的错误位置

在打印回溯信息（traceback）时，解释器现在不仅会指出错误所在行，还会进一步指出引发错误的表达式在哪里。例如：

```
Traceback (most recent call last):
  File "distance.py", line 11, in <module>
    print(manhattan_distance(p1, p2))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "distance.py", line 6, in manhattan_distance
    return abs(point_1.x - point_2.x) + abs(point_1.y - point_2.y)
                           ^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'x'
```

先前版本的解释器只会指出该行存在错误，我们不清楚哪个对象是这里的 `None`。在深度嵌套的 `dict` 对象以及连用多个函数调用的场景下，这些增强的错误也会大有裨益：

```
Traceback (most recent call last):
  File "query.py", line 37, in <module>
    magic_arithmetic('foo')
  File "query.py", line 18, in magic_arithmetic
    return add_counts(x) / 25
           ^^^^^^^^^^^^^
  File "query.py", line 24, in add_counts
    return 25 + query_user(user1) + query_user(user2)
                ^^^^^^^^^^^^^^^^
  File "query.py", line 32, in query_user
    return 1 + query_count(db, response['a']['b']['c']['user'], retry=True)
                           ~~~~~~~~~~~~~~~~~~~~^^^^^
TypeError: 'NoneType' object is not subscriptable
```

在复杂的算数表达式中同样有用：

```
Traceback (most recent call last):
  File "calculation.py", line 54, in <module>
    result = (x / y / z) * (a / b / c)
              ~~~~~~^~~
ZeroDivisionError: division by zero
```

此外，增强的回溯信息功能使用的信息通过通用 API 提供，该 API 可用于将 bytecode 指令与源代码位置相关联。可以使用以下方式获取此信息：

- Python 中的 `codeobject.co_positions()` 方法。
- C API 中的 `PyCode_Addr2Location()` 函数。

更多细节请参见 **PEP 657**。（由 Pablo Galindo、Batuhan Taskaya 和 Ammar Askar 在 bpo-43950 中贡献）

备注： 该特性需要在 codeobjects 中存储列位置，这可能会导致解释器内存占用和经过编译的 Python 文件的文件大小略有增加。要避免存储额外的信息同时取消打印额外的回溯信息，请使用 `-X no_debug_ranges` 命令行选项或 `PYTHONNODEBUGRANGES` 环境变量。

2.2 PEP 654：异常组与 except*

PEP 654 引入了若干语言特性，从而让程序能够同时引发和处理多个不相关的异常。内置类型 `ExceptionGroup` 和 `BaseExceptionGroup` 使得将异常划分成组并一起引发成为可能，新添加的 `except*` 是对 `except` 的泛化语法，这一语法能够匹配异常组的子组。

更多细节请参见 **PEP 654**。

（由 Irit Katriel 在 bpo-45292 中贡献，PEP 由 Irit Katriel、Yury Selivanov 和 Guido van Rossum 编写）

2.3 PEP 678：可用注释丰富异常

`add_note()` 方法已被添加到 `BaseException` 中。如果存在引发异常时不可用的上下文信息，使用该方法可以手动附加这些信息来丰富异常。添加的备注会显示在默认的回溯信息中。

更多细节请参见 **PEP 678**。

（由 Irit Katriel 在 bpo-45607 中贡献，PEP 由 Zac Hatfield-Dodds 编写）

2.4 Windows 下的 py.exe 启动器改进

包括在 Python 3.11 中的 launcher 的副本已进行了重大更新。现在它支持 **PEP 514** 所定义的 company/tag 语法即使用 `-V:`*<company>*`/`*<tag>* 参数代替受限的 `-`*<major>*`.`*<minor>*。这允许启动托管在 python.org 上的 `PythonCore` 以外的其他发行版。

当使用 `-V:` 选择器时，可以省略 company 或 tag，此时会搜索所有的安装。例如，`-V:OtherPython/` 会选择 `OtherPython` 所注册的“最佳”标签，而 `-V:3.11` 或 `-V:/3.11` 则会选择标签为 `3.11` 的“最佳”发行版。

在使用旧式的 `-`*<major>*、`-`*<major>*`.`*<minor>*、`-`*<major>*`-`*<bitness>* 或 `-`*<major>*`.`*<minor>*`-`*<bitness>* 参数时，应保留过去版本的所有已有行为，并只选择从 `PythonCore` 发布的版本。不过，`-64` 后缀现在表示“非 32 位”（不一定是 x86-64），因为有多种受支持的 64 位平台。32 位运行时是通过检查运行时的标签是否有 `-32` 后缀来检测的。自 Python 3.5 以来的所有版本都在其 32 位编译中包括了这个后缀。

3 有关类型提示的新增特性

本节介绍了涉及 **PEP 484** 类型提示和 `typing` 模块的主要更改。

3.1 PEP 646：可变参数泛型

之前的 **PEP 484** 引入了 `TypeVar`，其支持创建带单一类型参数的泛型。**PEP 646** 新引入了 `TypeVarTuple`，其支持任意数量的类型的参数化。换言之，`TypeVarTuple` 是可变参数（*variadic*）类型变量，支持可变参数泛型。

该泛型的引入让相当多的代码写法成为可能。特别是在诸如 NumPy 和 TensorFlow 这样的数值计算库中，这种泛型让类数组（array-like）结构类型可以用数组的形状（*shape*）来参数化。这样一来，静态类型检查器就能够在使用这些库的代码中捕获与形状有关的错误了。

更多细节请参见 **PEP 646**。

（由 Matthew Rahtz 在 bpo-43224 中贡献，共同贡献的还有 Serhiy Storchaka 和 Jelle Zijlstra，PEP 由 Mark Mendoza、Matthew Rahtz、Pradeep Kumar Srinivasan 以及 Vincent Siles 编写）

3.2 PEP 655：将单个 TypedDict 项标记为必填或非必填项

Required 和 NotRequired 提供了一种简单明了的方式来标记 TypedDict 中的单个项是否必须存在。而在之前的版本中，这只能通过使用继承来实现。

默认情况下，所有字段仍然是必填的，除非 *total* 参数设置为 False，在这种情况下，默认情况下所有字段则是非必填的。例如，下面指定了一个 TypedDict，其中有一个必填的键和一个非必填的键：

```
class Movie(TypedDict):
   title: str
   year: NotRequired[int]

m1: Movie = {"title": "Black Panther", "year": 2018}  # OK
m2: Movie = {"title": "Star Wars"}  # OK (year is not required)
m3: Movie = {"year": 2022}  # ERROR (missing required field title)
```

而以下的定义和上述定义等价：

```
class Movie(TypedDict, total=False):
   title: Required[str]
   year: int
```

更多细节请参见 **PEP 655**。

（由 David Foster 和 Jelle Zijlstra 在 bpo-47087 中贡献，PEP 由 David Foster 编写）

3.3 PEP 673：Self 类型

新的 Self 注解提供了一种简单而又直观的方法来标注返回其类实例的方法。这一注解的行为与 **PEP 484** 中指定的基于 TypeVar 的方法是一致的，但更简洁、更易于遵循。

常见的用法包括以 classmethod() 形式提供的替代构造函数，以及返回 self 的 __enter__() 方法：

```
class MyLock:
    def __enter__(self) -> Self:
        self.lock()
        return self

    ...

class MyInt:
    @classmethod
    def fromhex(cls, s: str) -> Self:
        return cls(int(s, 16))

    ...
```

Self 也可以用来标注与其封闭类类型相同的方法参数或属性。

更多细节请参见 **PEP 673**。

（由 James Hilton-Balfe 在 bpo-46534 中贡献，PEP 由 Pradeep Kumar Srinivasan 和 James Hilton-Balfe 编写）

3.4 PEP 675：任意字面值字符串类型

新的 `LiteralString` 注解能用于注明函数参数可为任何字面值字符串类型。这允许函数接受任意字面值字符串类型，以及从其他字面值字符串创建的字符串。这样一来，类型检查器就可以强制对此敏感的函数（例如执行 SQL 语句或 shell 命令的函数）只以静态的实参来调用，从而提供对注入攻击的保护。

例如，SQL 查询函数可按照如下方式注解：

```
def run_query(sql: LiteralString) -> ...
    ...

def caller(
    arbitrary_string: str,
    query_string: LiteralString,
    table_name: LiteralString,
) -> None:
    run_query("SELECT * FROM students")       # ok
    run_query(query_string)                   # ok
    run_query("SELECT * FROM " + table_name)  # ok
    run_query(arbitrary_string)               # type checker error
    run_query(                                # type checker error
        f"SELECT * FROM students WHERE name = {arbitrary_string}"
    )
```

请参阅 **PEP 675** 了解详情。

（由 Jelle Zijlstra 在 bpo-47088 中贡献，PEP 由 Pradeep Kumar Srinivasan 和 Graham Bleaney 编写）

3.5 PEP 681：数据类变换

`dataclass_transform` 可用于修饰类、元类或本身是装饰器的函数。使用 `@dataclass_Transform()` 就能让静态类型检查器知道被修饰的对象会在运行时执行对类的变换专业的“魔法”，从而让它具有类似 `dataclass` 的行为。

例如：

```
# The create_model decorator is defined by a library.
@typing.dataclass_transform()
def create_model(cls: Type[T]) -> Type[T]:
    cls.__init__ = ...
    cls.__eq__ = ...
    cls.__ne__ = ...
    return cls

# The create_model decorator can now be used to create new model classes:
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="Eric Idle")
```

更多细节请参见 **PEP 681**。

（由 Jelle Zijlstra 在 gh-91860 中贡献，PEP 由 Erik De Bonte 和 Eric Traut 编写）

3.6 未来版本可能不再实现 PEP 563

原计划随 Python 3.10 发布的 **PEP 563** 延迟注解求值（`from __future__ import annotations` 的 future 语句）已被无限期搁置。更多信息请参见 指导委员会（Steering Council）邮件列表中的讨论 。

4 其他语言特性修改

- 星号解包表达式现在可以在 `for` 语句中使用。（更多细节请参见 bpo-46725）
- 现在，在 异步函数中的推导式内部允许使用异步 推导式。此时，外部推导式隐式地变成了异步推导式。（由 Serhiy Storchaka 在 bpo-33346 中贡献）
- 在 `with` 语句和用于不支持 context manager 协议的对象 `contextlib.ExitStack.enter_context()` 中，以及 `async with` 语句和用于不支持 asynchronous context manager 协议的对象的 `contextlib.AsyncExitStack.enter_async_context()` 中现在会引发 `TypeError` 而不是 `AttributeError`。（由 Serhiy Storchaka 在 bpo-12022 和 bpo-44471 中贡献。）
- 增加了 `object.__getstate__()`，它提供 `__getstate__()` 方法的默认实现。`copy` 并 `pickle` 内置类型 `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet` 和 `datetime.tzinfo` 的子类的实例现在将会拷贝并封存被实现为 槽位的实例属性。此项改变有一个意外的附带影响：它将扰乱少数不使用 `object.__getstate__()` 的现有 Python 项目。请参阅 gh-70766 上近期的评论了解有关此类代码所需处理的讨论。（由 Serhiy Storchaka 在 bpo-26579 中贡献。）
- 增加了 `-P` 命令行选项和 `PYTHONSAFEPATH` 环境变量，它们将禁用当运行脚本时将脚本目录，或者当使用 `-c` 和 `-m` 时将当前目前自动添加到 `sys.path`。这可以确保只有标准库和已安装模块可通过 `import` 导入，而避免无意或恶意地使用本地（且通常为用户可写）的目录屏蔽此类模块。（由 Victor Stinner 在 gh-57684 中贡献。）
- 在 formatspec 中增加了一个 `"z"` 选项用来在舍入到格式精度后强制将负数转为正数。请参阅 **PEP 682** 了解详情。（由 John Belmonte 在 gh-90153 中贡献。）
- `sys.path` 不再接受字节串。对此的支持在 Python 3.2 和 3.6 之间中断过一段时间，但是直到 Python 3.10.0 发布时才被人发现。此外，由于 `-b` 和 `sys.path_importer_cache` 之间的交互，当同时存在 `str` 和 `bytes` 键时，恢复对此的支持会很困难。（由 Thomas Grainger 在 gh-91181 中贡献）

5 其他 CPython 实现的改变

- 实现了用于 `complex` 的 `__complex__()` 和用于 `bytes` 的 `__bytes__()` 特殊方法以支持 `typing.SupportsComplex` 和 `typing.SupportsBytes` 协议。（由 Mark Dickinson 和 Donghee Na 在 bpo-24234 中贡献。）
- 添加了新的内部哈希算法 `siphash13`。它与 `siphash24` 有类似的安全特性，但是对于长输入，它的速度略快。`str`、`bytes` 和其他一些类型现在使用它作为 `hash()` 的默认算法。**PEP 552** 基于哈希的.pyc 文件现在也使用 `siphash13`。（由 Inada Naoki 在 bpo-29410 中贡献）
- 当使用没有参数的 `raise` 语句重新引发活动的异常时，被附加在此异常上的回溯现在始终为 `sys.exc_info()[1].__traceback__`。这意味着在当前 `except` 子句中对回溯的修改将被反映到重新引发的异常。（由 Irit Katriel 在 bpo-45711 中贡献）
- 解释器状态对已处理异常（又名 `exc_info` 或 `_PyErr_StackItem`）的表示现在只有 `exc_value` 字段；`exc_type` 和 `exc_traceback` 已被移除，因为它们可以派生自 `exc_value`。（由 Irit Katriel 在 bpo-45711 中贡献）

- WIndows 安装程序添加了一个新的 命令行选项 `AppendPath`。它的行为类似于 `PrependPath`，但是会追加安装和脚本目录而不是前加。(由 Bastian Neuburger 在 bpo-44934 中贡献)
- 为了使用 `PyConfig.module_search_paths` 初始化 `sys.path`，`PyConfig.module_search_paths_set` 字段现在必须使用 "1" 作初始化，否则，该初始化行为会重新计算路径并替换任何加入到 `module_search_paths` 的值。
- `--help` 选项的输出现在将适应于 50 行/80 列。有关 Python environment variables 和 `-X` 选项的信息可以分别使用 `--help-env` 和 `--help-xoptions` 标志获得，并可以使用新的标志 `--help-all`。(由 Éric Araujo 在 bpo-46142 贡献。)
- 使用十进制以外的底，如 2（二进制）、4、8（八进制）、16（十六进制）、32 以外作为基数在 `int` 和 `str` 之间进行转换，如果字符串形式的数字数量超过一个限制，会抛出 `ValueError`，以避免因算法复杂而导致的潜在拒绝服务攻击。这是对 CVE-2020-10735 的缓解方案。这个限制可以通过环境变量、命令行旗标或 `sys` API 进行配置或禁用。参见 integer string conversion length limitation 文档。默认限制是字符串形式的 4300 位数字。

6 新增模块

- `tomllib`: 用于解析 TOML。请参阅 **PEP 680** 了解详情。(由 Taneli Hukkinen 在 bpo-40059 中贡献。)
- `wsgiref.types`: 用于表态类型检查的 **WSGI** 专属类型。(由 Sebastian Rittau 在 bpo-42012 中贡献。)

7 改进的模块

7.1 asyncio

- 添加了 `TaskGroup` 类，它是一个 异步上下文管理器，可以持有一组任务，等待它们全部完成后才退出。对于新代码，建议使用此类，而不是直接使用 `create_task()` 和 `gather()`。(由 Yury Selivanov 等人在 gh-90908 中贡献。)
- 增加了 `timeout()`，一个用于在异步操作上设置超时的异步上下文管理器。对于新代码推荐用这个来代替直接使用 `wait_for()`。(由 Andrew Svetlov 在 gh-90927 中贡献)
- 增加了 `Runner` 类，该类对外公开了 `run()` 所使用的机制。(由 Andrew Svetlov 在 gh-91218 中贡献。).)
- 为 asyncio 库中的同步化原语添加了 `Barrier` 类，以及相应的 `BrokenBarrierError` 异常。(由 Yves Duprat 和 Andrew Svetlov 在 gh-87518 中贡献。)
- 向 `asyncio.loop.create_connection()` 添加了关键字参数 *all_errors* 以便可以将多个连接错误作为一个 `ExceptionGroup` 来引发。
- 增加了 `asyncio.StreamWriter.start_tls()` 方法用于将现有的基于流的连接升级为 TLS。(由 Ian Good 在 bpo-34975 中贡献。)
- 为事件循环添加了原始数据报套接字函数: `sock_sendto()`，`sock_recvfrom()` 和 `sock_recvfrom_into()`。这些函数在 `SelectorEventLoop` 和 `ProactorEventLoop` 中均有实现。(由 Alex Grönholm 在 bpo-46805 中贡献。).)
- 为 `Task` 添加了 `cancelling()` 和 `uncancel()` 方法。它们主要供内部使用，特别是 `TaskGroup`。

7.2 contextlib

- 增加了非并行安全的 `chdir()` 上下文管理器用来改变当前工作目录并在退出时恢复它。是 `chdir()` 的简单包装器。(由 Filipe Laíns 在 bpo-25625 中贡献)

7.3 dataclasses

- 修改了字段默认的可变性检查，默认仅允许 hashable 而非任何不为 `dict`, `list` 或 `set` 实例的对象。(由 Eric V. Smith 在 bpo-44674 中贡献。)

7.4 datetime

- 增加了 `datetime.UTC`，是 `datetime.timezone.utc` 的便捷别名。(由 Kabir Kwatra 在 gh-91973 中贡献。)
- `datetime.date.fromisoformat()`， `datetime.time.fromisoformat()` 和 `datetime.datetime.fromisoformat()` 现在可以被用来解析大多数 ISO 8601 格式（除了那些支持分数小时和分钟的格式)。(由 Paul Ganssle 在 gh-80010 中贡献。)

7.5 enum

- 将 `EnumMeta` 重命名为 `EnumType` (`EnumMeta` 作为别名保留)。
- 增加了 `StrEnum`，其成员可以（且必须）作为字符串使用。
- 增加了 `ReprEnum`，它只是在为 `__str__()` 和 `__format__()` 方法(供 `str()`, `format()` 和 f-string 使用）返回成员的字面值（而不是名称）时修改了它们的 `__repr__()`。
- 修改了 `Enum.__format__()` (为 `format()`, `str.format()` 和 f-string 的默认值) 以便始终产生与 `Enum.__str__()` 相同的结果：对于继承自 `ReprEnum` 的枚举它将为其成员的值；对于所有其他枚举它将为枚举和成员名称 (例如 `Color.RED`)。
- 将新的 *boundary* 类形参连同其选项添加到 `Flag` 枚举和 `FlagBoundary` 枚举中，以控制超范围旗标值的处理方式。
- 增加了 `verify()` 枚举装饰器和 `EnumCheck` 枚举及其选项，以基于特定约束条件来检查枚举类。
- 增加了 `member()` 和 `nonmember()` 装饰器，用于确保被装饰的对象是/否会被转换为枚举成员。
- 增加了 `property()` 装饰器，它类似于 `property()` 但是专门针对枚举。请使用它来代替 `types.DynamicClassAttribute()`。
- 增加了 `global_enum()` 枚举装饰器，它会调整 `__repr__()` 和 `__str__()` 以将值显示为其模块的成员而不是枚举类的成员。例如，`'re.ASCII'` 是 `re.RegexFlag` 的 `ASCII` 成员而不是 `'RegexFlag.ASCII'`。
- 增强了 `Flag` 以支持针对其成员的 `len()`，迭代和 `in`/`not in`。例如，现在可以使用下面的代码: `len(AFlag(3)) == 2 and list(AFlag(3)) == (AFlag.ONE, AFlag.TWO)`
- 修改了 `Enum` 和 `Flag` 使得成员的定义是在 `__init_subclass__()` 被调用之前；`dir()` 现在将包括来自混入数据类型的方法等。
- 将 `Flag` 修改为只考虑规范的基本值（即二的乘方）而复合值（如 3, 6, 10 等）则被视为别名；逆向旗标将被强制转换为对应的正向旗标。

7.6 fcntl

- 在 FreeBSD 上，`F_DUP2FD` 和 `F_DUP2FD_CLOEXEC` 旗标分别受到支持，前者等价于 `dup2` 用法而后者额外设置了 `FD_CLOEXEC` 旗标。

7.7 fractions

- 支持基于字符串执行 **PEP 515** 网络的 `Fraction` 初始化。(由 Sergey B Kirpichev 在 bpo-44258 中贡献。)
- `Fraction` 现在实现了一个 `__int__` 方法，因而 `isinstance(some_fraction, typing.SupportsInt)` 检测将会通过。(由 Mark Dickinson 在 bpo-44547 中贡献。)

7.8 functools

- `functools.singledispatch()` 现在支持以 `types.UnionType` 和 `typing.Union` 作为 dispatch 参数的标注。:

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
...
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end=" ")
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

(由 Yurii Karabas 在 bpo-46014 中贡献。)

7.9 hashlib

- `hashlib.blake2b()` 和 `hashlib.blake2s()` 现在将优先使用 libb2 而不是 Python 自带的副本。(由 Christian Heimes 在 bpo-47095 中贡献。)
- 包含 SHA3 和 SHAKE 的内部 `_sha3` 模块现在会使用 *tiny_sha3* 而不是 *Keccak Code Package* 来减小代码和二进制文件的大小。`hashlib` 模块将首选来自 OpenSSL 的优化版 SHA3 和 SHAKE 实现。这个改变将只影响不带 OpenSSL 支持的安装版。(由 Christian Heimes 在 bpo-47098 中贡献。)
- 增加了 `hashlib.file_digest()`，一个针对文件或文件类对象高效哈希运算的辅助函数。(由 Christian Heimes 在 gh-89313 中贡献。)

7.10 IDLE 与 idlelib

- 对 `.pyi` 文件应用语法高亮。(由 Alex Waygood 和 Terry Jan Reedy 在 bpo-45447 中贡献。)
- 当附带输入和输出地保存 Shell 时将包括提示符。(由 Terry Jan Reedy 在 gh-95191 中贡献。)

7.11 inspect

- 增加了 `getmembers_static()` 用于返回所有成员而不通过描述器协议触发动态查找。(由 Weipeng Hong 在 bpo-30533 中贡献。)
- 增加了 `ismethodwrapper()` 用于检查某个对象的类型是否为 `MethodWrapperType`。(由 Hakan Çelik 在 bpo-29418 中贡献。)
- 修改了 `inspect` 模块中与帧相关的函数以返回新的 `FrameInfo` 和 `Traceback` 类实例(与之前的 named tuple 风格的接口保持向下兼容),它们包括扩展的 **PEP 657** 位置信息(末尾行编号,列与结束列等)。受影响的函数有:
  - `inspect.getframeinfo()`
  - `inspect.getouterframes()`
  - `inspect.getinnerframes()`,
  - `inspect.stack()`
  - `inspect.trace()`

  (由 Pablo Galindo 在 gh-88116 中贡献。)

7.12 locale

- 增加了 `locale.getencoding()` 以获取当前语言区域编码格式。它类似于 `locale.getpreferredencoding(False)` 但会忽略 Python UTF-8 模式。

7.13 logging

- 增加了 `getLevelNamesMapping()` 以返回一个从日志记录级别名称 (例如 `'CRITICAL'`) 到其对应 levels 值 (例如默认值 `50`) 的映射。(由 Andrei Kulakovin 在 gh-88024 中贡献。)
- 向 `SysLogHandler` 增加了 `createSocket()` 方法以匹配 `SocketHandler.createSocket()`。它将在处理句柄初始化期间以及发出事件时被自动调用,如果没有已激活的套接字的话。(由 Kirill Pinchuk 在 gh-88457 中贡献。)

7.14 math

- 增加了 `math.exp2()`: 返回 2 的 x 次幂。(由 Gideon Mitchell 在 bpo-45917 中贡献。)
- 增加了 `math.cbrt()`: 返回 x 的立方根。(由 Ajith Ramachandran 在 bpo-44357 中贡献。)
- 两个 `math.pow()` 边界情况的行为已改变,以便与 IEEE 754 规范保持一致。`math.pow(0.0, -math.inf)` 和 `math.pow(-0.0, -math.inf)` 等运算现在将返回 `inf`。在此之前它们会引发 `ValueError`。(由 Mark Dickinson 在 bpo-44339 中贡献。)
- 现在 `math.nan` 值将总是可用。(由 Victor Stinner 在 bpo-46917 中贡献。)

7.15 operator

- 增加了一个新函数 `operator.call`，使得 `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)`。（由 Antony Lee 在 bpo-44019 中贡献。）

7.16 os

- 在 Windows 上，`os.urandom()` 现在将使用 `BCryptGenRandom()`，而不是已被弃用的 `CryptGenRandom()`。（由 Donghee Na 在 bpo-44611 中贡献。）

7.17 pathlib

- `glob()` 和 `rglob()` 在 *pattern* 以路径组件分隔符即 `sep` 或 `altsep` 结束时将只返回目录。（由 Eisuke Kawasima 在 bpo-22276 和 bpo-33392 中贡献。）

7.18 re

- 正则表达式现已支持原子化分组 (`(?>...)`) 和占有型数量限定符 (`*+`, `++`, `?+`, `{m,n}+`)。（由 Jeffrey C. Jacobs 和 Serhiy Storchaka 在 bpo-433030 中贡献。）

7.19 shutil

- 在 `shutil.rmtree()` 中添加了可选形参 *dir_fd*。（由 Serhiy Storchaka 在 bpo-46245 中贡献。）

7.20 socket

- 为 NetBSD 添加了 CAN Socket 支持。（由 Thomas Klausner 在 bpo-30512 中贡献。）
- `create_connection()` 具有一个在连接失败的情况下引发包含所有错误而的 `ExceptionGroup` 不是只引发最后的错误的选项。（由 Irit Katriel 在 bpo-29980 中贡献。）

7.21 sqlite3

- 你现在可以通过将 `None` 传给 `set_authorizer()` 来禁用身份验证。(由 Erlend E. Aasland 在 bpo-44491 中贡献。)
- 排序名 `create_collation()` 现在可以包含任意 Unicode 字符。带有无效字符的排序名现在将引发 `UnicodeEncodeError` 而不是 `sqlite3.ProgrammingError`。（由 Erlend E. Aasland 在 bpo-44688 中贡献。）
- 现在 `sqlite3` 异常包括以 `sqlite_errorcode` 代表的 SQLite 扩展错误码和以 `sqlite_errorname` 代表的 SQLite 错误名。（由 Aviv Palivoda, Daniel Shahaf 和 Erlend E. Aasland 在 bpo-16379 和 bpo-24139 中贡献。）
- 向 `sqlite3.Connection` 添加了 `setlimit()` 和 `getlimit()` 用于在连接上设置和获取 SQLite 限制。（由 Erlend E. Aasland 在 bpo-45243 中贡献。）
- 现在 `sqlite3` 会基于兼容底层 SQLite 库的默认线程模式来设置 `sqlite3.threadsafety`。（由 Erlend E. Aasland 在 bpo-45613 中贡献。）

- 现在 `sqlite3` C 回调会在启用了回调回溯的情况下使用不可引发的异常。用户现在可以注册 *不可引发的钩子处理句柄* 来提升其调试体验。(由 Erlend E. Aasland 在 bpo-45828 中贡献。)
- 跨回滚的获取不会再引发 `InterfaceError`。而是改为由 SQLite 库来处理这类情况。(由 Erlend E. Aasland 在 bpo-44092 中贡献。)
- 向 `sqlite3.Connection` 添加了 `serialize()` 和 `deserialize()` 用于序列化和反序列化数据库。(由 Erlend E. Aasland 在 bpo-41930 中贡献。)
- 向 `sqlite3.Connection` 添加了 `create_window_function()` 用于创建聚合窗口函数。(由 Erlend E. Aasland 在 bpo-34916 中贡献。)
- 向 `sqlite3.Connection` 添加了 `blobopen()`。`sqlite3.Blob` 允许对 blob 进行增量 I/O 操作。(由 Aviv Palivoda 和 Erlend E. Aasland 在 bpo-24905 中贡献。)

7.22 string

- 向 `string.Template` 添加了 `get_identifiers()` 和 `is_valid()`，它们分别返回全部的有效占位符，以及是否存在无效占位符。(由 Ben Kehoe 在 gh-90465 中贡献。)

7.23 sys

- `sys.exc_info()` 的 `type` 和 `traceback` 字段现在是派生自 `value` (异常实例)，因此当一个异常在处理期间被修改时，其变化会在后续对 `exc_info()` 的调用结果中反映出来。(由 Irit Katriel 在 bpo-45711 中贡献。)
- 增加了返回激活的异常实例的 `sys.exception()` (等价于 `sys.exc_info()[1]`)。(由 Irit Katriel 在 bpo-46328 中贡献。)
- 增加了 `sys.flags.safe_path` 旗标。(由 Victor Stinner 在 gh-57684 中贡献。)

7.24 sysconfig

- 增加了三个新的 安装方案 (*posix_venv*, *nt_venv* and *venv*) 并将在 Python 创建新虚拟环境或从虚拟环境运行时使用。前两个方案 (*posix_venv* 和 *nt_venv*) 是用于非 Windows 和 Windows 的 OS 专属方案，*venv* 实际上是根据 Python 运行所在的 OS 来确定的前两者之一。这对于要修改 `sysconfig.get_preferred_scheme()` 的下游分发者来说很有用处。创建新虚拟环境的第三方代码应当使用新的 *venv* 安装方案来确定路径，就像 `venv` 所做的那样。(由 Miro Hrončok 在 bpo-45413 中贡献。)

7.25 tempfile

- `SpooledTemporaryFile` 对象现在完整实现了 `io.BufferedIOBase` 或 `io.TextIOBase` 的方法 (取决于具体文件模式)。这使它们能正确地配合接受文件类对象的 API 工作，如压缩文件的模块。(由 Carey Metcalfe 在 gh-70363 中贡献。)

7.26 threading

- 在 Unix 上，如果 `sem_clockwait()` 函数存在于 C 库中 (即 glibc 2.30 及更新的版本)，则 `threading.Lock.acquire()` 方法现在将使用单调时钟 (`time.CLOCK_MONOTONIC`) 来计算超时，而不使用系统时钟 (`time.CLOCK_REALTIME`)，以不受系统时钟修改的影响。(由 Victor Stinner 在 bpo-41710 中贡献。)

7.27 time

- 在 Unix 上，如果有可能，`time.sleep()` 现在将使用 `clock_nanosleep()` 或 `nanosleep()` 函数，其精度为 1 纳秒 (10^{-9} 秒)，而不是使用精度为 1 微秒 (10^{-6} 秒) 的 `select()`。(由 Benjamin Szőke 和 Victor Stinner 在 bpo-21302 中贡献。)
- 在 Windows 8.1 或更新版本上，现在 `time.sleep()` 会使用一个基于 高精度计时器 的可等待计时器，其精度为 100 纳秒 (10^{-7} 秒)。在之前版本中，其精度为 1 毫秒 (10^{-3} 秒)。(由 Benjamin Szőke, Donghee Na, Eryk Sun 和 Victor Stinner 在 bpo-21302 和 bpo-45429 中贡献。)

7.28 tkinter

- 增加了将 Tcl 库的准确版本号作为类似 `sys.version_info` 的命名元组返回的方法 `info_patchlevel()`。(由 Serhiy Storchaka 在 gh-91827 中贡献。)

7.29 回溯

- 增加了 `traceback.StackSummary.format_frame_summary()` 以允许用户重载要在回溯中出现哪些帧，以及要如何格式化它们。(由 Ammar Askar 在 bpo-44569 中贡献。)
- 增加了 `traceback.TracebackException.print()`，该函数可将 `TracebackException` 实例格式化打印到一个文件。(由 Irit Katriel 在 bpo-33809 中贡献。)

7.30 typing

主要的变化，请参阅有关类型提示的新增特性。

- 增加了 `typing.assert_never()` 和 `typing.Never`。`typing.assert_never()` 适用于要求类型检查器确认某一行代码是不可达的。在运行时，它会引发 `AssertionError`。(由 Jelle Zijlstra 在 gh-90633 中贡献。)
- 增加了 `typing.reveal_type()`。它适用于让类型检查器推理出给定表达式的类型。在运行时它会打印所接收的值的类型。(由 Jelle Zijlstra 在 gh-90572 中贡献。)
- 增加了 `typing.assert_type()`。它适用于让类型检查器确认推理出的给定表达式的类型与给定的类型相匹配。在运行时它将简单地返回所接收的值。(由 Jelle Zijlstra 在 gh-90638 中贡献。)
- 现在 `typing.TypedDict` 类型可以是泛型。(由 Samodya Abeysiriwardane 在 gh-89026 中贡献。)
- 现在 `NamedTuple` 类型可以是泛型。(由 Serhiy Storchaka 在 bpo-43923 中贡献。)
- 允许 `typing.Any` 子类化。这适用于避免关联到高度动态类的类型检查器错误，例如 mock 类。(由 Shantanu Jain 在 gh-91154 中贡献。)
- 现在 `typing.final()` 装饰器可在被装饰的对象上设置 `__final__` 属性。(由 Jelle Zijlstra 在 gh-90500 中贡献。)

- `typing.get_overloads()` 函数可被用来内省一个函数的重载。`typing.clear_overloads()` 可被用来清理一个函数所有的重载。(由 Jelle Zijlstra 在 gh-89263 中贡献。)
- 现在 `Protocol` 子类的 `__init__()` 方法将被保留。(由 Adrian Garcia Badarasco 在 gh-88970 中贡献。)
- 空元组类型 (`Tuple[()]`) 的表示形式已被简化。这将影响内省操作，例如 `get_args(Tuple[()])` 现在将被求值为 `()` 而不是 `(())`。(由 Serhiy Storchaka 在 gh-91137 中贡献。)
- 通过移除私有 `typing._type_check` 函数的回调检查放松了类型标注的运行时要求。(由 Gregory Beauregard 在 gh-90802 中贡献。)
- 现在 `typing.get_type_hints()` 支持将字符串求值为 PEP 585 泛型别名中的前向引用。(由 Niklas Rosenstein 在 gh-85542 中贡献。)
- `typing.get_type_hints()` 将不再添加 `Optional` 到形参并以 `None` 作为默认值。(由 Nikita Sobolev 在 gh-90353 中贡献。)
- 现在 `typing.get_type_hints()` 支持与纯字符串化的 `ClassVar` 标注进行求值。(由 Gregory Beauregard 在 gh-90711 中贡献。)
- `typing.no_type_check()` 将不再修改外部类和函数。现在它还会正确地将类方法标记为不进行类型检查。(由 Nikita Sobolev 在 gh-90729 中贡献。)

7.31 unicodedata

- Unicode 数据库已更新到 14.0.0 版。(由 Benjamin Peterson 在 bpo-45190 中贡献。)

7.32 unittest

- 增加了 `TestCase` 类的 `enterContext()` 和 `enterClassContext()` 方法，`IsolatedAsyncioTestCase` 类的 `enterAsyncContext()` 方法和 `unittest.enterModuleContext()` 函数。(由 Serhiy Storchaka 在 bpo-45046 中贡献。)

7.33 venv

- 当新的 Python 虚拟环境被创建时，将使用 *venv* sysconfig 安装方案来确定环境内部的路径。当 Python 在虚拟环境中运行时，同一个安装方案将被设为默认。这意味着下游分发方可以修改默认的 sysconfig 安装方案而不会改变虚拟环境的行为。同样会创建新的虚拟环境的第三方代码也应当这样做。(由 Miro Hrončok 在 bpo-45413 中贡献。)

7.34 warnings

- `warnings.catch_warnings()` 现在接受 `warnings.simplefilter()` 的参数，这提供了一种在局部忽略警告或将其转为错误的更精确方式。(由 Zac Hatfield-Dodds 在 bpo-47074 中贡献。)

7.35 zipfile

- 增加了为在 `ZipFile` 的目录和文件头中读取元数据指定成员名称编码格式的支持。（由 Stephen J. Turnbull 和 Serhiy Storchaka 在 bpo-28080 中贡献。）
- 增加了 `ZipFile.mkdir()` 用于在 ZIP 归档中新建目录。（由 Sam Ezeh 在 gh-49083 中贡献。）
- 为 `zipfile.Path` 增加了 `stem`, `suffix` 和 `suffixes`。（由 Miguel Brito 在 gh-88261 中贡献。）

8 性能优化

本节列出的特定优化均不依赖于*更快的 CPython* 项目，后者将在其专属章节中列出。

- 编译器现在将优化只包含格式代码 `%s`, `%r` 和 `%a` 的字符串字面值中的简单 printf 风格% 格式化并使其速度与对应的 f-string 表达式一样快。（由 Serhiy Storchaka 在 bpo-28307 中贡献。）
- 整除运算 (`//`) 已进行了更好的编译器微调。在 x86-64 上现在将 `int` 除以小于 `2**30` 的值时能够提速 20%。（由 Gregory P. Smith 和 Tim Peters 在 gh-90564 中贡献。）
- `sum()` 现在对小于 `2**30` 的整数运算可提速将近 30%。（由 Stefan Behnel 在 gh-68264 中贡献。）
- 列表大小调整针对常见场景进行了优化，对于 `list.append()` 可提速 ≈15% 而对于简单的 list comprehension 可提速 20-30%。（由 Dennis Sweeney 在 gh-91165 中贡献。）
- 字典在所有键均为 Unicode 对象时将不保存哈希值，以缩减 `dict` 的大小。例如，`sys.getsizeof(dict.fromkeys("abcdefg"))` 在 64 位平台上将从 352 字节缩减为 272 字节（减小 23%）。（由 Inada Naoki 在 bpo-46845 中贡献。）
- 现在使用 `asyncio.DatagramProtocol` 通过 UDP 传输大文件时速度将有成数量级的提升，对于 ≈60 MiB 的文件将可提速 100 倍以上。（由 msoxzw 在 gh-91487 中贡献。）
- 现在 `math` 中的函数 `comb()` 和 `perm()` 对于大参数可提速 ≈10 倍（对于越大的 k 值提速幅度越大）。（由 Serhiy Storchaka 在 bpo-37295 中贡献。）
- 现在 `statistics` 中的函数 `mean()`, `variance()` 和 `stdev()` 将会直接消耗迭代器而不是先将它们转换为 `list`。这将使速度翻倍并能节省大量内存。（由 Raymond Hettinger 在 gh-90415 中贡献。）
- 现在 `unicodedata.normalize()` 将在固定时间内正规化纯 ASCII 字符串。（由 Donghee Na 在 bpo-44987 中贡献。）

9 更快的 CPython

平均而言 CPython 3.11 比 CPython 3.10 快 25%，该数据是用 pyperformance 基准测试套件测得的，基于 Ubuntu Linux 上的 GCC 编译版。根据工作负载的不同，总的提速效果可达 10-60%。

本项目聚焦于 Python 的两个主要领域: *更快的启动* 和*更快的运行时*。本项目未涉及的优化将在*性能优化* 中单独列出。

9.1 更快的启动

冻结导入 / 静态代码对象

Python 会将 bytecode 缓存到 __pycache__ 目录以加快模型加载的速度。

在 3.10 版本时，Python 模块执行类似于这样：

```
Read __pycache__ -> Unmarshal -> Heap allocated code object -> Evaluate
```

在 Python 3.11 中，对 Python 启动具有关键影响的核心模块已被“冻结”。这意味着它们的 codeobjects (及字节码) 将由解释器静态地分配。这使得模块执行过程的步骤减少为:

```
Statically allocated code object -> Evaluate
```

现在 Python 3.11 解释器启动加快了 10-15%。这对使用 Python 的短期运行程序具有显著的影响。

（由 Eric Snow, Guido van Rossum 和 Kumar Aditya 在许多问题事件中贡献。）

9.2 更快的运行时

开销更低、更为惰性的 Python 帧

存放执行信息的 Python 帧会在 Python 调用一个 Python 函数时被自动创建。下面是新帧的优化操作:

- 优化改进了帧创建进程。
- 通过大量重用 C 栈上的帧空间来避免内存分配。
- 将内部帧结构优化为仅包含关键信息。在此之前的帧保存有额外的调试和内存管理信息。

现在旧式的 帧对象仅在调试器或 Python 内省函数如 `sys._getframe()` 和 `inspect.currentframe()` 发出请求时才会被创建。对于大多数用户代码，将不会创建任何帧对象。因此，几乎所有 Python 函数调用都有显著的提速。我们在 pyperformance 中测得了 3-7% 的提速。

（由 Mark Shannon 在 bpo-44590 中贡献。）

内联的 Python 函数调用

在 Python 函数调用期间，Python 将调用一个评测 C 函数来解读该函数的代码。这会有效地将纯 Python 递归限制在 C 栈的安全范围以内。

在 3.11 中，当 CPython 检测到 Python 代码调用了另一个 Python 函数时，它会设置一个新帧，并“跳转”到新帧内部的新代码。这可以避免全部调用 C 解析函数。

大多数 Python 函数调用现在将不消耗任何 C 栈空间，这提升了它们的速度。在简单的递归函数如斐波那契或阶乘函数中，我们测得了 1.7x 的提速。这还意味着递归函数能够递归得更深（如果用户通过 `sys.setrecursionlimit()` 提升了递归限制的话）。我们在 pyperformance 中测得了 1-3% 的提升。

（由 Pablo Galindo 和 Mark Shannon 在 bpo-45256 中贡献。）

PEP 659：专门化自适应解释器

PEP 659 是 Faster CPython 项目的关键部分之一。基本理念在于虽然 Python 是一种动态语言，但大部分代码都存在对象和类型极少发生变化的区域。这一理念被称为 *类型稳定性*。

在运行时，Python 将尝试在所执行的代码中寻找常见模式和类型稳定性。然后 Python 将把当前的操作替换为更加专门化的操作。这种专门化的操作使用仅对这些应用场景/类型来说可用的快速路径，它们的性能通常都会超过其泛用型的对应物。这还带来了名为 *内联缓存* 的另一项理念，即 Python 会将高消耗的操作的结果直接缓存在 bytecode 中。

这个特化程序还会将特定的常见指令对合并为一条超级指令，减少执行期间的开销。

Python 将只特化（会被多次执行的）“热门”代码。这可以防止 Python 在只执行一次的代码上浪费时间。Python 还可以在代码过于动态或用法发生变化时取消特化。特化会定期地尝试，而特化尝试的开销也不高，这使得特化能够适应新的环境改变。

（PEP 由 Mark Shannon 撰写，部分想法由 Stefan Brunthaler 提供。请参阅 **PEP 659** 了解详情。由 Mark Shannon 和 Brandt Bucher 实现，并由 Irit Katriel 和 Dennis Sweeney 提供了额外的帮助。）

运算	形式	专门化	运行加速（最高）	贡献者

双目运算	`x + x`
	`x - x`
	`x * x`
		常见类型如 `int`, `float` 和 `str` 的双目加法、乘法和减法将采用针对其下层类型专门定制的快速路径。	10%	Mark Shannon, Donghee Na, Brandt Bucher, Dennis Sweeney

下标	`a[i]`	对容器类型如 `list`, `tuple` 和 `dict` 的下标操作将直接索引下层数据结构。 对自定义 `__getitem__()` 的下标操作也是采用类似于*内联的 Python 函数调用* 的内联方式。	10-25%	Irit Katriel, Mark Shannon
存储下标操作	`a[i] = z`	类似于上述的下标操作专门化。	10-25%	Dennis Sweeney
调用	`f(arg)` `C(arg)`	对常用内置 (C) 函数和类型如 `len()` 和 `str` 的调用将直接调用其下层 C 版本。这将避免经历内部调用流程。	20%	Mark Shannon, Ken Jin
加载全局变量	`print` `len`	对象在全局/内置命名空间中的索引会被缓存。加载全局和内置变量将不需要命名空间查找过程。	[1]	Mark Shannon
加载属性	`o.attr`	类似于加载全局变量。属性在类/对象命名空间中的索引会被缓存。在大多数情况下，加载属性将不需要命名空间查找过程。	[2]	Mark Shannon
加载要调用的方法	`o.meth()`	方法的实际地址会被缓存。加载方法现在将不需要命名空间查找过程 -- 即使对于具有较长继承链的类来说也是如此。	10-20%	Ken Jin, Mark Shannon
存储属性	`o.attr = z`	类似于加载属性的优化。	2% 的运行效率	Mark Shannon
解包序列	`*seq`	针对常见容器如 `list` 和 `tuple` 进行了专门化。避免内部调用流程。	8%	Brandt Bucher

[1] 类似的优化自 Python 3.8 起即已存在。3.11 针对更多形式进行了专门化并减少了部分开销。

[2] 类似的优化自 Python 3.10 起即已存在。3.11 针对更多形式进行了专门化。此外，所有属性加载都应当通过 bpo-45947 获得了加速。

9.3 杂项

- 现在由于惰性创建的对象命名空间对象需要的内存将会减少。它们的命名空间现在还将更自由地共享键。（由 Mark Shannon 在 bpo-45340 和 bpo-40116 中贡献。）
- 实现了 “零消耗” 的异常，可在未引发任何异常时消除 `try` 语句的开销。（由 Mark Shannon 在 bpo-40222 中贡献。）
- 解释器中更为简洁的异常表示形式使得捕获异常所需的时间减少了大约 10%。（由 Irit Katriel 在 bpo-45711 中贡献。）
- `re` 的正则表达式匹配引擎已被部分重构，现在会在受支持的平台上使用已计算的 goto（或 “线程式代码”）。因此，Python 3.11 执行 pyperformance 正则表达式基准测试 相比 Python 3.10 提速了 10%。（由 Brandt Bucher 在 gh-91404 中贡献。）

9.4 常见问题

我要如何编写代码以便应用这些加速？

请编写遵循常见最佳实践的具有 Python 风格的代码；你不需要修改你的代码。CPython 加速计划会针对我们观察到的常见代码模式进行优化。

CPython 3.11 会使用更多内存吗？

可能不会；我们预期内存占用的增加相比 3.10 不会超过 20%。这是通过上文提及的帧对象和对象字典内存优化来平衡的。

我没有发现我的运行负载有任何加速。为什么？

特定代码将不会有明显的收益。如果你的代码大部时间消耗在 I/O 操作上，或者像 NumPy 那样大部分计算是在 C 扩展库中进行的就将如此。目前这个项目将只针对纯 Python 的运行负载。

此外，pyperformance 分数是一个几何平均值。即使在 pyperformance 基准测试内部，特定的基准测试也略有放缓，但其他的基准测试则有将近 2x 的加速！

是否有 JIT 编译器？

没有。我们还在探索其他优化方式。

9.5 关于

CPython 加速项目探索针对 CPython 的优化。项目主团队由 Microsoft 提供资助来支持全职工作。Pablo Galindo Salgado 还由 Bloomberg LP 提供资助来兼职该项目。此外，还有许多贡献者是来自社区的志愿者。

10 CPython 字节码的改变

字节码现在包含内联缓存条目，它采用新增的 CACHE 指令形式。许多操作码都预期带有确切数量的缓存，并指示解释器在运行时跳过它们。被填充的缓存看起来可以像是任意指令，因此在读取或修改包含加速的数据的原始自适应字节码时应当格外小心。

10.1 新的操作码

- ASYNC_GEN_WRAP, RETURN_GENERATOR 和 SEND，用于生成器和协程。
- COPY_FREE_VARS，这可以避免需要特别的调用方代码来关闭。
- JUMP_BACKWARD_NO_INTERRUPT，用于某些不希望处理中断的循环。
- MAKE_CELL，用于创建 cell-objects。
- CHECK_EG_MATCH 和 PREP_RERAISE_STAR，用于处理在 **PEP 654** 中增加的新异常组和 *except**。
- PUSH_EXC_INFO，用于异常处理句柄。
- RESUME，空操作，用于内部追踪、调试和优化检查。

10.2 被替换的操作码

被替换的操作码	新增的操作码	备注
BINARY_* INPLACE_*	BINARY_OP	用单个操作码替换所有数值类双目/原地操作码
CALL_FUNCTION CALL_FUNCTION_KW CALL_METHOD	CALL KW_NAMES PRECALL PUSH_NULL	对方法的参数变换与关键字参数的处理进行解偶；允许更好的调用特化
DUP_TOP DUP_TOP_TWO ROT_TWO ROT_THREE ROT_FOUR ROT_N	COPY SWAP	栈操纵指令
JUMP_IF_NOT_EXC_MATCH	CHECK_EXC_MATCH	现在会执行检查但不会跳转
JUMP_ABSOLUTE POP_JUMP_IF_FALSE POP_JUMP_IF_TRUE	JUMP_BACKWARD POP_JUMP_BACKWARD_IF_* POP_JUMP_FORWARD_IF_*	参见[3]；针对每个方向的 TRUE, FALSE, NONE 和 NOT_NONE 变种
SETUP_WITH SETUP_ASYNC_WITH	BEFORE_WITH	with 代码块设置

10.3 修改/移除的操作码

- 修改 MATCH_CLASS 和 MATCH_KEYS 为不再推入额外的布尔值来指示成功/失败。而是在失败时推入 None 来代替由被提取值组成的元组。
- 修改配合异常使用的操作码以反映它们现在是由栈上的一个条目而非三个条目代表 (参见 gh-89874)。
- 移除了 COPY_DICT_WITHOUT_KEYS, GEN_START, POP_BLOCK, SETUP_FINALLY 和 YIELD_FROM。

[3] 所有跳转操作码现在都是相对的，包括现有的 JUMP_IF_TRUE_OR_POP 和 JUMP_IF_FALSE_OR_POP。该参数现在是相对当前指令的偏移量而不是绝对位置。

11 弃用

本小节列出了已在 Python 3.11 中弃用的 Python API。

已弃用的 C API 将单独列出。

11.1 语言/内置对象

- 串连 `classmethod` 描述器（在 bpo-19072 中引入）现已被弃用。它不能再被用来包装其他描述器如 `property`。该特性的核心设计存在缺陷并导致了许多下游问题。要“穿过”一个 `classmethod`，请考虑使用在 Python 3.10 中添加的 `__wrapped__` 属性。（由 Raymond Hettinger 在 gh-89519 中贡献。）
- 数值大于 `0o377` (十进制的 255) 的八进制转义符会产生 `DeprecationWarning`。在未来的 Python 版本中，这将引发 `SyntaxWarning` 并最终改为 `SyntaxError`。（由 Serhiy Storchaka 在 gh-81548 中贡献。）
- 现在从 `int()` 至 `__trunc__()` 的委托已被弃用。当 `type(a)` 实现了 `__trunc__()` 但未实现 `__int__()` 或 `__index__()` 时调用 `int(a)` 现在将引发 `DeprecationWarning`。（由 Zackery Spytz 在 bpo-44977 中贡献。）

11.2 模块

- **PEP 594** 使得以下模块被弃用并将在 Python 3.13 中被移除:

`aifc`	`chunk`	`msilib`	`pipes`	`telnetlib`
`audioop`	`crypt`	`nis`	`sndhdr`	`uu`
`cgi`	`imghdr`	`nntplib`	`spwd`	`xdrlib`
`cgitb`	`mailcap`	`ossaudiodev`	`sunau`	

（由 Brett Cannon 在 bpo-47061 以及 Victor Stinner 在 gh-68966 中贡献。）

- 至少从 Python 3.6 起 `asynchat`, `asyncore` 和 `smtpd` 模块已被弃用。它们的文档和弃用警告现在已更新为提示它们将在 Python 3.12 中被移除。（由 Hugo van Kemenade 在 bpo-47022 中贡献。）
- `lib2to3` 包和 2to3 工具现已被弃用并可能无法解析 Python 3.10 或更新版本。参见引入新 PEG 解析器的 **PEP 617** 了解详情。（由 Victor Stinner 在 bpo-40360 中贡献。）
- 未写入文档的模块 `sre_compile`, `sre_constants` 和 `sre_parse` 现已被弃用。（由 Serhiy Storchaka 在 bpo-47152 中贡献。）

11.3 标准库

- `configparser` 的下列部分自 Python 3.2 起已被弃用。现在它们的弃用警告已更新为提示它们将在 Python 3.12 中被移除:
  - `configparser.SafeConfigParser` 类
  - `configparser.ParsingError.filename` 特征属性
  - `configparser.RawConfigParser.readfp()` 方法

  （由 Hugo van Kemenade 在 bpo-45173 中贡献。）

- `configparser.LegacyInterpolation` 自 Python 3.2 起已在文档字符串中被弃用，并未在 `configparser` 文档中列出。现在它将发出 `DeprecationWarning` 并将在 Python 3.13 中被移除。请改用 `configparser.BasicInterpolation` 或 `configparser.ExtendedInterpolation`。(由 Hugo van Kemenade 在 bpo-46607 中贡献。)
- 较旧的 `importlib.resources` 函数集合已被弃用而改用在 Python 3.9 中添加的替代物并将在未来的 Python 版本中被移除，因为它们不支持位于 package 子目录下的资源:
  - `importlib.resources.contents()`
  - `importlib.resources.is_resource()`
  - `importlib.resources.open_binary()`
  - `importlib.resources.open_text()`
  - `importlib.resources.read_binary()`
  - `importlib.resources.read_text()`
  - `importlib.resources.path()`
- The `locale.getdefaultlocale()` function is deprecated and will be removed in Python 3.15. Use `locale.setlocale()`, `locale.getpreferredencoding(False)` and `locale.getlocale()` functions instead. (Contributed by Victor Stinner in gh-90817.)
- `locale.resetlocale()` 函数已被弃用并将在 Python 3.13 中移除。请改用 `locale.setlocale(locale.LC_ALL, "")`。(由 Victor Stinner 在 gh-90817 中贡献。)
- 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。现在只有 ASCII 数字序列会被接受作为数字引用，而 `bytes` 模式和替换字符串中的分组名称只能包含 ASCII 字母、数字和下划线。目前对于违反这些规则的语法将会引发弃用警告。(由 Serhiy Storchaka 在 gh-91760 中贡献。)
- 在 `re` 模块中，`re.template()` 函数和相应的 `re.TEMPLATE` 和 `re.T` 旗标已被弃用，因为它们未被写入文档并缺少明显的目的。它们将在 Python 3.13 中移除。(由 Serhiy Storchaka 和 Miro Hrončok 在 gh-92728 由贡献。)
- `turtle.settiltangle()` 自 Python 3.1 起已被弃用；现在它会发出弃用警告并将在 Python 3.13 中移除。请改用 `turtle.tiltangle()` (该函数在此前被错误地标记为已弃用，现在它的文档字符串已被修正)。(由 Hugo van Kemenade 在 bpo-45837 中贡献。)
- `typing.Text`，它的存在只是为了在 Python 2 和 Python 3 代码之间提供兼容性支持，现在已被弃用。目前尚无移除它的计划，但推荐用户在任何可能的地方改用 `str`。(由 Alex Waygood 在 gh-92332 中贡献。)
- 用于构造 `typing.TypedDict` 类型的关键字参数语法现在已被弃用。将在 Python 3.13 中移除对它的支持。(由 Jingchen Ye 在 gh-90224 中贡献。)
- `webbrowser.MacOSX` 已被弃用并将在 Python 3.13 中移除。它未经测试，未写入文档，也未被 `webbrowser` 本身所使用。(由 Donghee Na 在 bpo-42255 中贡献。)
- 从 `TestCase` 和 `IsolatedAsyncioTestCase` 测试方法返回一个值（默认的 `None` 以外的值）的行为现在已被弃用。
- 已弃用下列未正式写入文档的 `unittest` 函数，计划在 Python 3.13 中移除:
  - `unittest.findTestCases()`
  - `unittest.makeSuite()`
  - `unittest.getTestCaseNames()`

  请改用 `TestLoader` 方法:
  - `unittest.TestLoader.loadTestsFromModule()`

- unittest.TestLoader.loadTestsFromTestCase()
- unittest.TestLoader.getTestCaseNames()

（由 Erlend E. Aasland 在 bpo-5846 中贡献。）

- usageExit() 被标记为已弃用，将在 3.13 中被移除。（由 Carlos Damázio 在 gh-67048 中贡献。）

12 计划在 Python 3.12 中移除

以下 Python API 已在之前的 Python 发布版中弃用，并将在 Python 3.12 中移除。

C API 的移除计划将单独列出。

- asynchat 模块
- asyncore 模块
- 整个 distutils 包
- imp 模块
- typing.io 命名空间
- typing.re 命名空间
- cgi.log()
- importlib.find_loader()
- importlib.abc.Loader.module_repr()
- importlib.abc.MetaPathFinder.find_module()
- importlib.abc.PathEntryFinder.find_loader()
- importlib.abc.PathEntryFinder.find_module()
- importlib.machinery.BuiltinImporter.find_module()
- importlib.machinery.BuiltinLoader.module_repr()
- importlib.machinery.FileFinder.find_loader()
- importlib.machinery.FileFinder.find_module()
- importlib.machinery.FrozenImporter.find_module()
- importlib.machinery.FrozenLoader.module_repr()
- importlib.machinery.PathFinder.find_module()
- importlib.machinery.WindowsRegistryFinder.find_module()
- importlib.util.module_for_loader()
- importlib.util.set_loader_wrapper()
- importlib.util.set_package_wrapper()
- pkgutil.ImpImporter
- pkgutil.ImpLoader
- pathlib.Path.link_to()
- sqlite3.enable_shared_cache()

- `sqlite3.OptimizedUnicode()`
- `PYTHONTHREADDEBUG` 环境变量
- The following deprecated aliases in `unittest` 中的下列已弃用别名:

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

13 移除

本小节列出了已在 Python 3.11 中移除的 Python API。

已移除的 C API 将单独列出。

- 移除了允许旧式基于生成器的协程兼容 `async` / `await` 代码的 `@asyncio.coroutine()` decorator。该函数自 Python 3.8 起已被弃用并且原定在 Python 3.10 中移除。请改用 `async def`。(由 Illia Volochii 在 bpo-43216 中贡献。)
- 移除了用于在调试模式下包装旧式基于生成器的协程对象的 `asyncio.coroutines.CoroWrapper`。(由 Illia Volochii 在 bpo-43216 中贡献。)
- 出于显著的安全性考量，自 Python 3.9 起已被禁用的 `asyncio.loop.create_datagram_endpoint()` 的 *reuse_address* 形参现在已彻底移除。这是因为在 UDP 中套接字选项 `SO_REUSEADDR` 的行为。(由 Hugo van Kemenade 在 bpo-45129 中贡献。)
- 移除了自 Python 3.9 起已弃用的 `binhex` 模块。并移除了相关联的同样已弃用的 `binascii` 函数:
  - `binascii.a2b_hqx()`
  - `binascii.b2a_hqx()`
  - `binascii.rlecode_hqx()`
  - `binascii.rldecode_hqx()`

  `binascii.crc_hqx()` 函数仍然可用。

  (由 Victor Stinner 在 bpo-45085 中贡献。)
- 移除了自 Python 3.9 起已弃用的 `distutils bdist_msi` 命令。请改用 `bdist_wheel` (wheel 包)。(由 Hugo van Kemenade 在 bpo-45124 中贡献。)

- 移除了自 Python 3.9 起已弃用的 `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` 和 `fileinput.FileInput` 的 `__getitem__()` 方法。(由 Hugo van Kemenade 在 bpo-45132 中贡献。)
- 移除了已弃用的 `gettext` 函数 `lgettext()`, `ldgettext()`, `lngettext()` 和 `ldngettext()`。并移除了 `bind_textdomain_codeset()` 函数、`NullTranslations.output_charset()` 和 `NullTranslations.set_output_charset()` 方法，以及 `translation()` 和 `install()` 的 *codeset* 形参，因为它们仅被用于 `l*gettext()` 函数。(由 Donghee Na 和 Serhiy Storchaka 在 bpo-44235 中贡献。)
- 已从 `inspect` 模块中移除:
  - `getargspec()` 函数自 Python 3.0 起已被弃用；请改用 `inspect.signature()` 或 `inspect.getfullargspec()`。
  - `formatargspec()` 函数自 Python 3.5 起已被弃用；请改用 `inspect.signature()` 函数或直接使用 `inspect.Signature` 对象。
  - 未写入文档的 `Signature.from_builtin()` 和 `Signature.from_function()` 方法自 Python 3.5 起已被弃用；请改用 `Signature.from_callable()` 方法。

  (由 Hugo van Kemenade 在 bpo-45320 中贡献。)
- 从 `pathlib.PurePath` 中移除了 `__class_getitem__()` 方法，因为它从未被使用而是在之前版本中误添加的。(由 Nikita Sobolev 在 bpo-46483 中贡献。)
- 移除了 `smtpd` 模块中的 `MailmanProxy` 类，因为它在没有外部 `mailman` 包时是无法使用的。(由 Donghee Na 在 bpo-35800 中贡献。)。
- 移除了 `_tkinter.TkappType` 中已被弃用的 `split()` 方法。(由 Erlend E. Aasland 在 bpo-38371 中贡献。)
- 从 `unittest` 发现中移除了命名空间包支持。它在 Python 3.4 中引入但自 Python 3.7 起已不可用。(由 Inada Naoki 在 bpo-23882 中贡献。)
- 移除了未写入文档的私有 `float.__set_format__()` 方法，之前在 Python 3.7 中名为 `float.__setformat__()`。其文档字符串已写明：“你应该不需要使用此函数。它的存在主要是用于 Python 的测试套件。”(由 Victor Stinner 在 bpo-46852 中贡献。)
- `--experimental-isolated-subinterpreters` 配置旗标 (和相应的 `EXPERIMENTAL_ISOLATED_SUBINTERPRETERS` 宏) 已被移除。
- Pynche --- The Pythonically Natural Color and Hue Editor --- 已被移出 `Tools/scripts` 并将脱离 Python 源代码树独立开发。

14 移植到 Python 3.11

本节列出了先前描述的更改以及 Python API 中可能需要修改你的 Python 代码的其他错误修正。

针对 C API 的移植说明将单独列出。

- `open()`, `io.open()`, `codecs.open()` 和 `fileinput.FileInput` 的文件模式中不再接受 `'U'` (“通用换行符”)。在 Python 3 中，“通用换行符”模式会在文件以文本模式打开时默认被使用，而 `'U'` 旗标自 Python 3.3 起已被弃用。这些函数的 newline 形参将控制如何使用通用换行符。(由 Victor Stinner 在 bpo-37330 中贡献。)
- 现在 `ast.AST` 节点位置在提供给 `compile()` 和其他相关函数时会进行验证。如果检测到无效位置，将会引发 `ValueError`。(由 Pablo Galindo 在 gh-93351 中提供。)
- 继在 Python 3.8 中弃用后，已禁止向 `asyncio.loop.set_default_executor()` 传入非 `concurrent.futures.ThreadPoolExecutor` 执行器。(由 Illia Volochii 在 bpo-43234 中贡献。)

- `calendar`: 在未指定语言区域的情况下，`calendar.LocaleTextCalendar` 和 `calendar.LocaleHTMLCalendar` 类现在会使用 `locale.getlocale()`，而不是使用 `locale.getdefaultlocale()`。(由 Victor Stinner 在 bpo-46659 中贡献。)
- 现在 `pdb` 模块会使用 `'UTF-8'` 编码来读取 `.pdbrc` 配置文件。(由 Srinivas Reddy Thatiparthy (□□□□□□□□□□□ □□□□□□ □□□□□□□□□□) 在 bpo-41137 中贡献。)
- `random.sample()` 的 *population* 形参必须是一个序列，不再支持将 `set` 自动转换为 `list`。此外，如果样本大小大于总体大小，将会引发 `ValueError`。(由 Raymond Hettinger 在 bpo-40465 中贡献。)
- 移除了 `random.shuffle()` 的 *random* 可选形参。在之前版本中重排操作是使用任意随机函数；现在，将始终使用 `random.random()` (之前的默认值)。
- 在 `re` re-syntax 中，全局内联旗标 (例如 `(?i)`) 现在只能在正则表达式的开头使用。自 Python 3.6 起在别处使用这些旗标的做法已被弃用。(由 Serhiy Storchaka 在 bpo-47066 中贡献)。
- 在 `re` 模块中，修复了几个长期存在的错误，在极少数情况下，这些错误可能会导致捕获分组得到错误的结果。因此，这可能会改变这些情况下的捕获输出。(由 Ma Lin 在 bpo-35859 中贡献。)

15 构建的改变

- CPython 现已具有 **PEP 11 Tier 3 support** 以便交叉编译至 WebAssembly 平台 Emscripten (`wasm32-unknown-emscripten` 即浏览器版 Python) 和 WebAssembly System Interface (WASI) (`wasm32-unknown-wasi`)。此计划的灵感来自前人的工作如 Pyodide。这些平台提供了 POSIX API 的有限子集；与网络、进程、线程、信号、mmap 和用户/组相关的 Python 标准库特性和模块将不可用或无法正常工作。(Emscripten 由 Christian Heimes 和 Ethan Smith 在 gh-84461 中贡献，WASI 由 Christian Heimes 在 gh-90473 中贡献；平台的推进在 gh-95085 中追踪。)
- 构建 CPython 现在需要:
  - C11 编译器和标准库。可选的 C11 特性 不是必须的。(由 Victor Stinner 在 bpo-46656, bpo-45440 和 bpo-46640 中贡献。)
  - 对 IEEE 754 浮点数的支持。(由 Victor Stinner 在 bpo-46917 中贡献。)
- `Py_NO_NAN` 宏已被移除。由于 CPython 现在要求 IEEE 754 浮点数，NaN 值将始终可用。(由 Victor Stinner 在 bpo-46656 中贡献。)
- `tkinter` 包现在需要 Tcl/Tk 8.5.12 或更新的版本。(由 Serhiy Storchaka 在 bpo-46996 中贡献。)
- 大多数标准库扩展模块的构建依赖、编译器旗标和链接器旗标现在将由 **`configure`** 来检测。libffi, libnsl, libsqlite3, zlib, bzip2, liblzma, libcrypt, Tcl/Tk 和 uuid 旗标将由 pkg-config (如果可用) 来检测。`tkinter` 现在需要由 pkg-config 命令来检测 Tcl/Tk 标头和库的开发设置。(由 Christian Heimes 和 Erlend Egeberg Aasland 在 bpo-45847, bpo-45747 和 bpo-45763 中贡献。)
- libpython 不再与 libcrypt 链接。(由 Mike Gilbert 在 bpo-45433 中贡献。)
- 现在 CPython 可以通过向 `--with-lto` 传入 `thin`，即 `--with-lto=thin` 在编译时启用 ThinLTO 选项。(由 Donghee Na 和 Brett Holman 在 bpo-44340 中贡献。)
- 现在可以禁用对象结构体的自由列表。新的 **`configure`** 选项 `--without-freelists` 可用于禁用除空元组单例之外的所有自由列表。(由 Christian Heimes 在 bpo-45522 中贡献。)
- `Modules/Setup` 和 `Modules/makesetup` 已获得改进并进行绑定。扩展模块现在可以通过 `makesetup` 来构建。除部分测试模块外所有模块都可以静态链接到主二进制文件或库中。(由 Brett Cannon 和 Christian Heimes 在 bpo-45548, bpo-45570, bpo-45571 和 bpo-43974 中贡献。)

备注: 使用环境变量 `TCLTK_CFLAGS` 和 `TCLTK_LIBS` 来手动指定 Tcl/Tk 头文件和库的位置。**`configure`** 选项 `--with-tcltk-includes` 和 `--with-tcltk-libs` 已被移除。

在 RHEL 7 和 CentOS 7 上开发包将不提供 tcl.pc 和 tk.pc；请使用 TCLTK_LIBS="-ltk8.5 -ltkstub8.5 -ltcl8.5"。Misc/rhel7 目录包含 .pc 文件以及如何使用 RHEL 7 和 CentOS 7 的 Tcl/Tk 和 OpenSSL 构建 Python 的说明。

- CPython 现在将默认使用 30 比特位的数字来实现 Python int。之前版本中，在 SIZEOF_VOID_P >= 8 的平台上默认使用 30 比特位数字，否则使用 15 位数字。仍然有可能通过配置脚本的 --enable-big-digits 选项或 PC/pyconfig.h 中的 PYLONG_BITS_IN_DIGIT 变量（适用于 Windows）显式地要求使用 15 比特位数字，但该选项可能会在未来某个时候被移除。（由 Mark Dickinson 在 bpo-45569 中贡献。）

16 C API 的改变

16.1 新的特性

- 新增了 PyType_GetName() 函数用来获取类型的简短名称。（由 Hai Shi 在 bpo-42035 中贡献。）
- 新增了 PyType_GetQualName() 函数用来获取类型的限定名称。（由 Hai Shi 在 bpo-42035 中贡献。）
- 在受限的 C API 中新增了 PyThreadState_EnterTracing() 和 PyThreadState_LeaveTracing() 函数用来挂起和恢复追踪和性能分析。（由 Victor Stinner 在 bpo-43760 中贡献。）
- 增加了 Py_Version 常量，其中的值与 PY_VERSION_HEX 相同。（由 Gabriele N. Tornetta 在 bpo-43931 中贡献。）
- Py_buffer 及其 API 现在是受限 API 和稳定 ABI 的组成部分:
  - PyObject_CheckBuffer()
  - PyObject_GetBuffer()
  - PyBuffer_GetPointer()
  - PyBuffer_SizeFromFormat()
  - PyBuffer_ToContiguous()
  - PyBuffer_FromContiguous()
  - PyObject_CopyData()
  - PyBuffer_IsContiguous()
  - PyBuffer_FillContiguousStrides()
  - PyBuffer_FillInfo()
  - PyBuffer_Release()
  - PyMemoryView_FromBuffer()
  - bf_getbuffer 和 bf_releasebuffer 类型槽位

  （由 Christian Heimes 在 bpo-45459 中贡献。）
- 增加了 PyType_GetModuleByDef() 函数，用于在无法直接获取信息的情况下 (通过 PyCMethod) 获取方法定义所在的模块。（由 Petr Viktorin 在 bpo-46613 中贡献。）
- 添加了用于打包和解包 C double (序列化和反序列化) 的新函数: PyFloat_Pack2(), PyFloat_Pack4(), PyFloat_Pack8(), PyFloat_Unpack2(), PyFloat_Unpack4() 和 PyFloat_Unpack8()。（由 Victor Stinner 在 bpo-46906 中贡献。）

- 添加了用于获取帧对象属性的新函数: `PyFrame_GetBuiltins()`, `PyFrame_GetGenerator()`, `PyFrame_GetGlobals()`, `PyFrame_GetLasti()`。
- 新增了两个用于获取和设置活动异常实例的函数: `PyErr_GetHandledException()` 和 `PyErr_SetHandledException()`。这两个函数是 `PyErr_SetExcInfo()` 和 `PyErr_GetExcInfo()` 的替代品，后者使用传统的 3 元组表示异常。（由 Irit Katriel 在 bpo-46343 中贡献。）
- 添加了 `PyConfig.safe_path` 成员。（由 Victor Stinner 在 gh-57684 中贡献。）.)

16.2 移植到 Python 3.11

- 部分宏已被转换为静态内联函数以避免 宏陷阱。这项改变对用户来说应该是基本无感的，因为替代函数会将其参数强制转换为预期的类型以避免静态类型检查导致的编译器警告。但是，当受限 C API 被设为 >=3.11 时，将不会执行这些强制转换，调用方将需要自行将参数强制转换为其预期的类型。请参阅 **PEP 670** 了解详情。（由 Victor Stinner 和 Erlend E. Aasland 在 gh-89653 中贡献。）
- `PyErr_SetExcInfo()` 不再使用 `type` 和 `traceback` 参数，解释器现在将从异常实例（即 `value` 参数）中获取这些值。该函数仍会偷取对所有三个参数的引用。（由 Irit Katriel 在 bpo-45711 中贡献。）
- `PyErr_GetExcInfo()` 现在将从异常实例（即 `value` 字段）获取结果的 `type` 和 `traceback` 字段。（由 Irit Katriel 在 bpo-45711 中贡献。）
- `_frozen` 新增了 `is_package` 字段用来指明冻结模块是否为包。之前，是将 `size` 字段设置负值作为指示符。现在 `size` 将只使用非负值。（由 Kumar Aditya 在 bpo-46608 中贡献。）
- 现在 `_PyFrameEvalFunction()` 接受 `_PyInterpreterFrame*` 作为其第二个形参，而不是 `PyFrameObject*`。请参阅 **PEP 523** 了解如何使用此函数指针类型的更多细节。
- 现在 `PyCode_New()` 和 `PyCode_NewWithPosOnlyArgs()` 接受一个额外的 `exception_table` 参数。如果可能，应当必须使用这些函数。获取自定义的代码对象：使用编译器创建一个代码对象，然后使用 `replace` 方法得到一个经过修改的版本。
- `PyCodeObject` 不再具有 `co_code`, `co_varnames`, `co_cellvars` 和 `co_freevars` 字段。请分别改用 `PyCode_GetCode()`, `PyCode_GetVarnames()`, `PyCode_GetCellvars()` 和 `PyCode_GetFreevars()` 通过 C API 来访问它们。（由 Brandt Bucher 在 bpo-46841 以及 Ken Jin 在 gh-92154 和 gh-94936 中贡献。）
- 旧的垃圾桶宏 (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`) 现在已被弃用。它们应该由新的宏 `Py_TRASHCAN_BEGIN` 和 `Py_TRASHCAN_END` 代替。

  带有旧版宏的 tp_dealloc 函数，例如:

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

  应当按照以下方式迁移到新版宏:

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
```

(下页继续)

```
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    Py_TRASHCAN_END
}
```

请注意 Py_TRASHCAN_BEGIN 的第二个参数应该是它所属的取消分配函数。

要在同一代码库中支持旧版本的 Python，可以定义以下的宏并在整个代码中使用它们 (版权声明：这些宏是从 mypy 代码库中拷贝的):

```
#if PY_VERSION_HEX >= 0x03080000
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_BEGIN(op, dealloc)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_END
#else
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_SAFE_BEGIN(op)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_SAFE_END(op)
#endif
```

- 现在如果一个类型定义了 Py_TPFLAGS_HAVE_GC 旗标但没有遍历函数 (PyTypeObject.tp_traverse) 则 PyType_Ready() 函数将引发一个错误。（由 Victor Stinner 在 bpo-44263 中贡献。）
- 带有 Py_TPFLAGS_IMMUTABLETYPE 旗标的堆类型现在可以继承 **PEP 590** vectorcall 协议。在之前版本中，这只适用于 静态类型。（由 Erlend E. Aasland 在 bpo-43908 中贡献。）
- 由于 Py_TYPE() 已改为内联静态函数，因此 Py_TYPE(obj) = new_type 必须换成 Py_SET_TYPE(obj, new_type): 参见 Py_SET_TYPE() 函数（自 Python 3.9 起可用）。为保持向下兼容，可以使用这个宏:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_TYPE)
static inline void _Py_SET_TYPE(PyObject *ob, PyTypeObject *type)
{ ob->ob_type = type; }
#define Py_SET_TYPE(ob, type) _Py_SET_TYPE((PyObject*)(ob), type)
#endif
```

（由 Victor Stinner 在 bpo-39573 中贡献。）

- 由于 Py_SIZE() 已改为内联静态函数，因此 Py_SIZE(obj) = new_size 必须换成 Py_SET_SIZE(obj, new_size): 参见 Py_SET_SIZE() 函数（自 Python 3.9 起可用）。为保持向下兼容，可以使用这个宏:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_SIZE)
static inline void _Py_SET_SIZE(PyVarObject *ob, Py_ssize_t size)
{ ob->ob_size = size; }
#define Py_SET_SIZE(ob, size) _Py_SET_SIZE((PyVarObject*)(ob), size)
#endif
```

（由 Victor Stinner 在 bpo-39573 中贡献。）

- 在 Py_LIMITED_API 宏被设为 0x030b0000 (Python 3.11) 或更高版本时，<Python.h> 将不再包含头文件 <stdlib.h>, <stdio.h>, <errno.h> 和 <string.h>。C 扩展应在 #include <Python.h> 之后显式地包括头文件。（由 Victor Stinner 在 bpo-45434 中贡献。）
- 非受限 API 文件 cellobject.h, classobject.h, code.h, context.h, funcobject.h, genobject.h 和 longintrepr.h 已被移至 Include/cpython 目录。此外，还移除了 eval.h 头文件。这些文件不能被直接包括，因为它们已经被包括在 Python.h 中了：参见 包括文件。如果它们已被直接包括，请考虑改为包括 Python.h。（由 Victor Stinner 在 bpo-35134 中贡献。）

- `PyUnicode_CHECK_INTERNED()` 宏已被排除在受限 C API 之外。它从未在那里被使用，因为它使用了受限 C API 中不可用的内部结构体。（由 Victor Stinner 在 bpo-46007 中贡献。）
- 以下帧函数和类型现在可通过 `#include <Python.h>` 直接使用，不再需要添加 `#include <frameobject.h>`:
  - `PyFrame_Check()`
  - `PyFrame_GetBack()`
  - `PyFrame_GetBuiltins()`
  - `PyFrame_GetGenerator()`
  - `PyFrame_GetGlobals()`
  - `PyFrame_GetLasti()`
  - `PyFrame_GetLocals()`
  - `PyFrame_Type`

  （由 Victor Stinner 在 gh-93937 中贡献。）
- `PyFrameObject` 结构体成员已从公有 C API 中被移除。

  虽然文档指出 `PyFrameObject` 字段可能随时更改，但这些字段长期以来一直保持稳定，并在多个流行的扩展中使用。

  在 Python 3.11 中，为了优化性能，对帧结构进行了重组。一些字段被完全删除，因为它们属于旧实现的细节。

  `PyFrameObject` 字段:
  - `f_back`: 使用 `PyFrame_GetBack()`。
  - `f_blockstack`: 已移除。
  - `f_builtins`: 使用 `PyFrame_GetBuiltins()`。
  - `f_code`: 使用 `PyFrame_GetCode()`。
  - `f_gen`: 使用 `PyFrame_GetGenerator()`。
  - `f_globals`: 使用 `PyFrame_GetGlobals()`。
  - `f_iblock`: 已移除。
  - `f_lasti`: 使用 `PyFrame_GetLasti()`。使用 `f_lasti` 并带有 `PyCode_Addr2Line()` 的代码应当改用 `PyFrame_GetLineNumber()`；它可能会更快。
  - `f_lineno`: 使用 `PyFrame_GetLineNumber()`
  - `f_locals`: 使用 `PyFrame_GetLocals()`。
  - `f_stackdepth`: 已移除。
  - `f_state`: 无公共 API (重命名为 `f_frame.f_state`)。
  - `f_trace`: 无公共 API。
  - `f_trace_lines`: 使用 `PyObject_GetAttrString((PyObject*)frame, "f_trace_lines")`。
  - `f_trace_opcodes`: 使用 `PyObject_GetAttrString((PyObject*)frame, "f_trace_opcodes")`。
  - `f_localsplus`: 无公共 API (重命名为 `f_frame.localsplus`)。

- f_valuestack: 已移除。

现在 Python 帧对象是惰性地创建的。一个附带影响是 f_back 成员不可直接访问，因为它现在的值也是惰性地计算的。必须改为调用 PyFrame_GetBack() 函数。

直接访问 f_locals 的调试器必须改为调用 PyFrame_GetLocals()。它们不再需要调用 PyFrame_FastToLocalsWithError() 或 PyFrame_LocalsToFast()，实际上它们不应该调用这些函数。现在帧所需要的更新将由虚拟机来管理。

在 Python 3.8 及更旧版本上定义 PyFrame_GetCode() 的代码:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyCodeObject* PyFrame_GetCode(PyFrameObject *frame)
{
    Py_INCREF(frame->f_code);
    return frame->f_code;
}
#endif
```

在 Python 3.8 及更旧版本上定义 PyFrame_GetBack() 的代码:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyFrame_GetBack(PyFrameObject *frame)
{
    Py_XINCREF(frame->f_back);
    return frame->f_back;
}
#endif
```

或者使用 pythoncapi_compat 项目 在更旧版本的 Python 上获取这些函数。

- PyThreadState 结构体成员的变化:
  - frame: 已被移除，请使用 PyThreadState_GetFrame() (由 bpo-40429 添加到 Python 3.9 的函数)。警告：该函数返回一个 strong reference，需要调用 Py_XDECREF()。
  - tracing: 已被更改，请使用 PyThreadState_EnterTracing() 和 PyThreadState_LeaveTracing() (由:issue:*43760* 添加到 Python 3.11 的函数)。
  - recursion_depth: 已被移除，请使用 (tstate->recursion_limit - tstate->recursion_remaining) 代替。
  - stackcheck_counter: 已移除。

在 Python 3.8 或更旧版本中定义 PyThreadState_GetFrame() 的代码:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyThreadState_GetFrame(PyThreadState *tstate)
{
    Py_XINCREF(tstate->frame);
    return tstate->frame;
}
#endif
```

在 Python 3.10 或更旧版本中定义 PyThreadState_EnterTracing() 和 PyThreadState_LeaveTracing() 的代码:

```
#if PY_VERSION_HEX < 0x030B00A2
static inline void PyThreadState_EnterTracing(PyThreadState *tstate)
{
```

(下页继续)

```
    tstate->tracing++;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = 0;
#else
    tstate->use_tracing = 0;
#endif
}

static inline void PyThreadState_LeaveTracing(PyThreadState *tstate)
{
    int use_tracing = (tstate->c_tracefunc != NULL || tstate->c_profilefunc !=␣
↪NULL);
    tstate->tracing--;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = use_tracing;
#else
    tstate->use_tracing = use_tracing;
#endif
}
#endif
```

或者使用 pythoncapi-compat 项目 在旧版的 Python 函数上获取这些函数。

- 推荐发行方使用优化的 Blake2 库 libb2 来构建 Python。
- 现在初始化时 `PyConfig.module_search_paths_set` 字段必须设为 1 以使用 `PyConfig.module_search_paths` 来初始化 `sys.path`。否则，初始化将重新计算路径并替换任何加入到 `module_search_paths` 的值。
- `PyConfig_Read()` 将不会再计算初始搜索路径，并且不会将任何值填充到 `PyConfig.module_search_paths`。要计算默认路径再修改它们，请结束初始化并使用 `PySys_GetObject()` 来将 `sys.path` 提取为一个 Python 列表对象并直接修改它。

16.3 弃用

- 弃用以下配置 Python 初始化的函数:
  - `PySys_AddWarnOptionUnicode()`
  - `PySys_AddWarnOption()`
  - `PySys_AddXOption()`
  - `PySys_HasWarnOptions()`
  - `PySys_SetArgvEx()`
  - `PySys_SetArgv()`
  - `PySys_SetPath()`
  - `Py_SetPath()`
  - `Py_SetProgramName()`
  - `Py_SetPythonHome()`
  - `Py_SetStandardStreamEncoding()`
  - `_Py_SetProgramFullPath()`

改用新的 Python 初始化配置的 `PyConfig` API (**PEP 587**)。(由 Victor Stinner 在 gh-88279 中贡献。)

- 弃用 `PyBytesObject` 的 `ob_shash` 成员。改用 `PyObject_Hash()`。(由 Inada Naoki 在 bpo-46864 中贡献。).)

16.4 计划在 Python 3.12 中移除

以下 C API 在早期 Python 发行版中已经弃用，将在 Python 3.12 中移除。

- `PyUnicode_AS_DATA()`
- `PyUnicode_AS_UNICODE()`
- `PyUnicode_AsUnicodeAndSize()`
- `PyUnicode_AsUnicode()`
- `PyUnicode_FromUnicode()`
- `PyUnicode_GET_DATA_SIZE()`
- `PyUnicode_GET_SIZE()`
- `PyUnicode_GetSize()`
- `PyUnicode_IS_COMPACT()`
- `PyUnicode_IS_READY()`
- `PyUnicode_READY()`
- `PyUnicode_WSTR_LENGTH()`
- `_PyUnicode_AsUnicode()`
- `PyUnicode_WCHAR_KIND`
- `PyUnicodeObject`
- `PyUnicode_InternImmortal()`

16.5 移除

- `PyFrame_BlockSetup()` 和 `PyFrame_BlockPop()` 已被移除。(由 Mark Shannon 在 bpo-40222 中贡献。)
- 移除了下列使用 `errno` 变量的数学宏:
  - `Py_ADJUST_ERANGE1()`
  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  (由 Victor Stinner 在 bpo-45412 中贡献。)
- 移除 `Py_UNICODE_COPY()` 和 `Py_UNICODE_FILL()` 宏，它们自 Python 3.3 起已被弃用。改用 `PyUnicode_CopyCharacters()` 或 `memcpy()` (`wchar_t*` 字符串) 和 `PyUnicode_Fill()` 函数。(由 Victor Stinner 在 bpo-41123 中贡献。)

- 移除 `pystrhex.h` 头文件。它只包含私有函数。C 扩展应当只包括主 `<Python.h>` 头文件。(由 Victor Stinner 在 bpo-45434 中贡献。)
- 移除了 `Py_FORCE_DOUBLE()` 宏，它曾经由 `Py_IS_INFINITY()` 宏使用。(由 Victor Stinner 在 bpo-45440 贡献。)
- 以下项目在 `Py_LIMITED_API` 定义时不再可用:
  - `PyMarshal_WriteLongToFile()`
  - `PyMarshal_WriteObjectToFile()`
  - `PyMarshal_ReadObjectFromString()`
  - `PyMarshal_WriteObjectToString()`
  - `Py_MARSHAL_VERSION` 宏

  这些不是 受限 API 的组成部分。

  (由 Victor Stinner 在 bpo-45474 中贡献。)
- 将 `PyWeakref_GET_OBJECT()` 排除在受限 C API 之外。由于 `PyWeakReference` 结构体在受限 C API 中是不透明的因此它从未正确发挥作用。(由 Victor Stinner 在 d:issue:*35134* 中贡献。)
- 移除了 `PyHeapType_GET_MEMBERS()` 宏。它错误地暴露在公开的 C API 中，且只能由 Python 在内部使用。请使用 `PyTypeObject.tp_members` 作为替代。(由 Victor Stinner 在 bpo-40170 贡献。)
- 移除了 `HAVE_PY_SET_53BIT_PRECISION` 宏 (移动到了内部 C API)。(由 Victor Stinner 在 bpo-45412 贡献。)
- 移除了 `Py_UNICODE` 编码器 API，它们从 Python 3.3 起已经弃用，很少使用，而且相对于推荐的替代品来说，效率很低。

  被移除的函数有:
  - `PyUnicode_Encode()`
  - `PyUnicode_EncodeASCII()`
  - `PyUnicode_EncodeLatin1()`
  - `PyUnicode_EncodeUTF7()`
  - `PyUnicode_EncodeUTF8()`
  - `PyUnicode_EncodeUTF16()`
  - `PyUnicode_EncodeUTF32()`
  - `PyUnicode_EncodeUnicodeEscape()`
  - `PyUnicode_EncodeRawUnicodeEscape()`
  - `PyUnicode_EncodeCharmap()`
  - `PyUnicode_TranslateCharmap()`
  - `PyUnicode_EncodeDecimal()`
  - `PyUnicode_TransformDecimalToASCII()`

  请参阅 **PEP 624** 了解细节以及 **迁移指引**。(由 Inada Naoki 在 bpo-44029 中贡献。)

17 3.11.4 中的重要变化

17.1 tarfile

- `tarfile` 和 `shutil.unpack_archive()` 中的提取方法有一个新的 *filter* 参数，该参数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。有关详细信息请参阅 tarfile-extraction-filter。在 Python 3.12 中，不带 *filter* 参数的用法将显示 `DeprecationWarning`。在 Python 3.14 中，默认值将切换为 `'data'`。（由 Petr Viktorin 在 **PEP 706** 中贡献。）

18 3.11.5 中的重要变化

18.1 OpenSSL

- 来自 python.org 的 Windows 版本和 macOS 安装程序现在使用 OpenSSL 3.0。

索引

非字母

环境变量
    PYTHONNODEBUGRANGES, 4
    PYTHONSAFEPATH, 3, 8
    PYTHONTHREADDEBUG, 26

P

Python 提高建议
    PEP 11, 28
    PEP 11#tier-3, 28
    PEP 484, 5
    PEP 484#annotating-instance-and-class-methods, 6
    PEP 514, 5
    PEP 515, 11
    PEP 523, 30
    PEP 552, 8
    PEP 563, 8
    PEP 587, 35
    PEP 590, 31
    PEP 594, 3, 23
    PEP 617, 23
    PEP 624, 3, 36
    PEP 624#alternative-apis, 36
    PEP 646, 5
    PEP 654, 5, 21
    PEP 655, 6
    PEP 657, 4, 12
    PEP 659, 19
    PEP 670, 3, 30
    PEP 673, 6
    PEP 675, 7
    PEP 678, 5
    PEP 680, 3, 9
    PEP 681, 7
    PEP 682, 8
    PEP 706, 37
    PEP 3333, 9
PYTHONNODEBUGRANGES, 4
PYTHONSAFEPATH, 3, 8
PYTHONTHREADDEBUG, 26