What's New in Python

發🄵 *3.8.2*

A. M. Kuchling

4 月 29, 2020

Python Software Foundation
Email: docs@python.org

Contents

编者 Raymond Hettinger

本文解释了 Python 3.8 相比 3.7 的新增特性。完整的详情可参阅 更新日志。

Python 3.8 已于 2019 年 10 月 14 日发布。

1 摘要 - 发布重点

2 新的特性

2.1 赋值表达式

新增的语法 := 可在表达式内部为变量赋值。它被昵称为“海象运算符”因为它很像是 海象的眼睛和长牙。

在这个示例中，赋值表达式可以避免调用 len() 两次:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

类似的益处还可出现在正则表达式匹配中需要使用两次匹配对象的情况中，一次检测用于匹配是否发生，另一次用于提取子分组:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

此运算符也适用于配合 while 循环计算一个值来检测循环是否终止，而同一个值又在循环体中再次被使用的情况:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

另一个值得介绍的用例出现于列表推导式中，在筛选条件中计算一个值，而同一个值又在表达式中需要被使用:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

请尽量将海象运算符的使用限制在清晰的场合中，以降低复杂性并提升可读性。

请参阅 **PEP 572** 了解详情。

(由 Morehouse 在 bpo-35224 中贡献。)

2.2 仅限位置形参

新增了一个函数形参语法 / 用来指明某些函数形参必须使用仅限位置而非关键字参数的形式。这种标记语法与通过 help() 所显示的使用 Larry Hastings 的 Argument Clinic 工具标记的 C 函数相同。

在下面的例子中，形参 *a* 和 *b* 为仅限位置形参，*c* 或 *d* 可以是位置形参或关键字形参，而 *e* 或 *f* 要求为关键字形参:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

以下均为合法的调用:

```
f(10, 20, 30, d=40, e=50, f=60)
```

但是，以下均为不合法的调用:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a keyword argument
f(10, 20, 30, 40, 50, f=60)           # e must be a keyword argument
```

这种标记形式的一个用例是它允许纯 Python 函数完整模拟现有的用 C 代码编写的函数的行为。例如，内置的 `divmod()` 函数不接受关键字参数:

```
def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

另一个用例是在不需要形参名称时排除关键字参数。例如，内置的 `len()` 函数的签名为 `len(obj, /)`。这可以排除如下这种笨拙的调用形式:

```
len(obj='hello')  # The "obj" keyword argument impairs readability
```

另一个益处是将形参标记为仅限位置形参将允许在未来修改形参名而不会破坏客户的代码。例如，在 `statistics` 模块中，形参名 *dist* 在未来可能被修改。这使得以下函数描述成为可能:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

由于在 / 左侧的形参不会被公开为可用关键字，其他形参名仍可在 `**kwargs` 中使用:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
>>> f(10, 20, a=1, b=2, c=3)         # a and b are used in two ways
10 20 {'a': 1, 'b': 2, 'c': 3}
```

这极大地简化了需要接受任意关键字参数的函数和方法的实现。例如，以下是一段摘自 `collections` 模块的代码:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Note "iterable" is a possible keyword argument
```

请参阅 **PEP 570** 了解详情。

(由 Pablo Galindo 在 bpo-36540 中贡献。)

2.3 用于已编译字节码文件的并行文件系统缓存

新增的 `PYTHONPYCACHEPREFIX` 设置 (也可使用 `-X pycache_prefix`) 可将隐式的字节码缓存配置为使用单独的并行文件系统树，而不是默认的每个源代码目录下的 `__pycache__` 子目录。

缓存的位置会在 `sys.pycache_prefix` 中报告 (`None` 表示默认位置即 `__pycache__` 子目录)。

(由 Carl Meyer 在 bpo-33499 中贡献。)

2.4 调试构建使用与发布构建相同的 ABI

Python 现在不论是以发布模式还是调试模式进行构建都将使用相同的 ABI。在 Unix 上，当 Python 以调试模式构建时，现在将可以加载以发布模式构建的 C 扩展和使用稳定版 ABI 构建的 C 扩展。

发布构建和调试构建现在都是 ABI 兼容的：定义 `Py_DEBUG` 宏不会再启用 `Py_TRACE_REFS` 宏，它引入了唯一的 ABI 不兼容性。`Py_TRACE_REFS` 宏添加了 `sys.getobjects()` 函数和 `PYTHONDUMPREFS` 环境变量，它可以使用新的 `./configure --with-trace-refs` 构建选项来设置。(由 Victor Stinner 在 bpo-36465 中贡献。)

在 Unix 上，C 扩展不会再被链接到 libpython，但 Android 和 Cygwin 例外。现在静态链接的 Python 将可以加载使用共享库 Python 构建的 C 扩展。(由 Victor Stinner 在 bpo-21536 中贡献。)

在 Unix 上，当 Python 以调试模式构建时，导入操作现在也会查找在发布模式下编译的 C 扩展以及使用稳定版 ABI 编译的 C 扩展。（由 Victor Stinner 在 bpo-36722 中贡献。）

要将 Python 嵌入到一个应用中，必须将新增的 `--embed` 选项传给 `python3-config --libs --embed` 以获得 `-lpython3.8` (将应用链接到 libpython)。要同时支持 3.8 和旧版本，请先尝试 `python3-config --libs --embed` 并在此命令失败时回退到 `python3-config --libs` (即不带 `--embed`)。

增加一个 pkg-config `python-3.8-embed` 模块用来将 Python 嵌入到一个应用中：`pkg-config python-3.8-embed --libs` 包含 `-lpython3.8`。要同时支持 3.8 和旧版本，请先尝试 `pkg-config python-X.Y-embed --libs` 并在此命令失败时回退到 `pkg-config python-X.Y --libs` (即不带 `--embed`) （请将 `X.Y` 替换为 Python 版本号）。

另一方面，`pkg-config python3.8 --libs` 不再包含 `-lpython3.8`。C 扩展不可被链接到 libpython (但 Android 和 Cygwin 例外，这两者的情况由脚本处理)；此改变是故意被设为向下不兼容的。（由 Victor Stinner 在 bpo-36721 中贡献。）

2.5 f-字符串支持 = 用于自动记录表达式和调试文档

增加 = 说明符用于 f-string。形式为 `f'{expr=}'` 的 f-字符串将扩展表示为表达式文本，加一个等于号，再加表达式的求值结果。例如：

```
>>> user = 'eric_idle'
>>> member_since = date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31)"
```

通常的 f-字符串格式说明符允许更细致地控制所要显示的表达式结果：

```
>>> delta = date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

= 说明符将输出整个表达式，以便详细演示计算过程：

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

（由 Eric V. Smith 和 Larry Hastings 在 bpo-36817 中贡献。）

2.6 PEP 578: Python 运行时审核钩子

此 PEP 添加了审核钩子和已验证开放钩子。两者在 Python 与本机代码中均可用。允许以纯 Python 代码编写的应用和框架利用额外的通知，同时允许嵌入开发人员或系统管理员部署始终启用审核的 Python 版本。

请参阅 **PEP 578** 了解详情。

2.7 PEP 587: Python 初始化配置

PEP 587 增加了一个新的 C API 用来配置 Python 初始化，提供对整个配置过程的更细致控制以及更好的错误报告。

新的结构：

- `PyConfig`
- `PyPreConfig`
- `PyStatus`

- `PyWideStringList`

新的函数：

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`
- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`
- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`
- `PyStatus_Exit()`
- `PyStatus_IsError()`
- `PyStatus_IsExit()`
- `PyStatus_NoMemory()`
- `PyStatus_Ok()`
- `PyWideStringList_Append()`
- `PyWideStringList_Insert()`
- `Py_BytesMain()`
- `Py_ExitStatusException()`
- `Py_InitializeFromConfig()`
- `Py_PreInitialize()`
- `Py_PreInitializeFromArgs()`
- `Py_PreInitializeFromBytesArgs()`
- `Py_RunMain()`

此 PEP 还为这些内部结构添加了 `_PyRuntimeState.preconfig` (`PyPreConfig` 类型) 和 `PyInterpreterState.config` (`PyConfig` 类型) 字段。`PyInterpreterState.config` 成为新的引用配置，替代全局配置变量和其他私有变量。

请参阅 Python 初始化配置获取详细文档。

请参阅 **PEP 587** 了解详情。

(由 Victor Stinner 在 bpo-36763 中贡献。)

2.8 Vectorcall: 用于 CPython 的快速调用协议

添加"vectorcall" 协议到 Python/C API。它的目标是对已被应用于许多类的现有优化进行正式化。任何实现可调用对象的扩展类型均可使用此协议。

此特性目前为暂定状态，计划在 Python 3.9 将其完全公开。

请参阅 **PEP 590** 了解详情。

(由 Jeroen Demeyer 和 Mark Shannon 在 bpo-36974 中贡献。)

2.9 具有外部数据缓冲区的 pickle 协议 5

当使用 `pickle` 在 Python 进程间传输大量数据以充分发挥多核或多机处理的优势时，非常重要一点是通过减少内存拷贝来优化传输效率，并可能应用一些定制技巧例如针对特定数据的压缩。

`pickle` 协议 5 引入了对于外部缓冲区的支持，这样 **PEP 3118** 兼容的数据可以与主 pickle 流分开进行传输，这是由通信层来确定的。

请参阅 **PEP 574** 了解详情。

(由 Antoine Pitrou 在 bpo-36785 中贡献。)

3 其他语言特性修改

- 在之前版本中 `continue` 语句不允许在 `finally` 子句中使用，这是因为具体实现存在一个问题。在 Python 3.8 中此限制已被取消。(由 Serhiy Storchaka 在 bpo-32489 中贡献。)
- `bool`, `int` 和 `fractions.Fraction` 类型现在都有一个 `as_integer_ratio()` 方法，与 `float` 和 `decimal.Decimal` 中的已有方法类似。这个微小的 API 扩展使得 `numerator, denominator = x.as_integer_ratio()` 这样的写法在多种数字类型上通用成为可能。(由 Lisa Roach 在 bpo-33073 和 Raymond Hettinger 在 bpo-37819 中贡献。)
- `int`, `float` 和 `complex` 的构造器现在会使用 `__index__()` 特殊方法，如果该方法可用而对应的方法 method `__int__()`, `__float__()` 或 `__complex__()` 方法不可用的话。(由 Serhiy Storchaka 在 bpo-20092 中贡献。)
- 添加 `\N{name}` 转义符在 正则表达式中的支持:

```
>>> notice = 'Copyright © 2019'
>>> copyright_year_pattern = re.compile(r'\N{copyright sign}\s*(\d{4})')
>>> int(copyright_year_pattern.search(notice).group(1))
2019
```

  (由 Jonathan Eunice 和 Serhiy Storchaka 在 bpo-30688 中贡献。)
- 现在 dict 和 dictview 可以使用 `reversed()` 按插入顺序反向迭代。(由 Rémi Lapeyre 在 bpo-33462 中贡献。)
- 在函数调用中允许使用的关键字名称语法受到进一步的限制。特别地，`f((keyword)=arg)` 不再被允许。关键字参数赋值形式的左侧绝不允许一般标识符以外的其他内容。(由 Benjamin Peterson 在 bpo-34641 中贡献。)
- 在 `yield` 和 `return` 语句中的一般可迭代对象解包不再要求加圆括号。这使得 *yield* 和 *return* 的语法与正常的赋值语法更为一致:

```
>>> def parse(family):
        lastname, *members = family.split()
        return lastname.upper(), *members

>>> parse('simpsons homer marge bart lisa sally')
('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'sally')
```

（由 David Cuthbert 和 Jordan Chapman 在 bpo-32117 中贡献。）

- 当类似 `[(10, 20) (30, 40)]` 这样在代码中少了一个逗号时，编译器将显示 `SyntaxWarning` 并附带更有帮助的提示。这相比原来用 `TypeError` 来提示第一个元组是不可调用的更容易被理解。（由 Serhiy Storchaka 在 bpo-15248 中贡献。）

- `datetime.date` 或 `datetime.datetime` 和 `datetime.timedelta` 对象之间的算术运算现在将返回相应子类的实例而不是基类的实例。这也会影响到在具体实现中（直接或间接地）使用了 `datetime.timedelta` 算术运算的返回类型，例如 `astimezone()`。（由 Paul Ganssle 在 bpo-32417 中贡献。）

- 当 Python 解释器通过 Ctrl-C (SIGINT) 被中断并且所产生的 `KeyboardInterrupt` 异常未被捕获，Python 进程现在会通过一个 SIGINT 信号或是使得发起调用的进程能检测到它是由 Ctrl-C 操作杀死的正确退出代码来退出。POSIX 和 Windows 上的终端会相应地使用此代码在交互式会话中终止脚本。（由 Google 的 Gregory P. Smith 在 bpo-1054041 中贡献。）

- 某些高级编程风格要求为现有的函数更新 `types.CodeType` 对象。由于代码对象是不可变的，需要基于现有代码对象模型创建一个新的代码对象。使用 19 个形参将会相当繁琐。现在，新的 `replace()` 方法使得通过少量修改的形参创建克隆对象成为可能。

  下面是一个修改 `statistics.mean()` 函数来防止 *data* 形参被用作关键字参数的例子:

```
>>> from statistics import mean
>>> mean(data=[10, 20, 90])
40
>>> mean.__code__ = mean.__code__.replace(co_posonlyargcount=1)
>>> mean(data=[10, 20, 90])
Traceback (most recent call last):
  ...
TypeError: mean() got some positional-only arguments passed as keyword␣
↪arguments: 'data'
```

（由 Victor Stinner 在 bpo-37032 中贡献。）

- 对于整数，现在 `pow()` 函数的三参数形式在底数与模数不可约的情况下允许指数为负值。随后它会在指数为 `-1` 时计算底数的模乘逆元，并对其他负指数计算反模的适当幂次。例如，要计算 38 模 137 的 模乘逆元 则可写为:

```
>>> pow(38, -1, 137)
119
>>> 119 * 38 % 137
1
```

模乘逆元在求解 线性丢番图方程 会被用到。例如，想要求出 `4258𝑥 + 147𝑦 = 369` 的整数解，首先应重写为 `4258𝑥 ≡ 369 (mod 147)` 然后求解:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

（由 Mark Dickinson 在 bpo-36027 中贡献。）

- 字典推导式已与字典字面值实现同步，会先计算键再计算值:

```
>>> # Dict comprehension
>>> cast = {input('role? '): input('actor? ') for i in range(2)}
role? King Arthur
actor? Chapman
role? Black Knight
actor? Cleese

>>> # Dict literal
>>> cast = {input('role? '): input('actor? ')}
```

（下页继续）

(繼續上一頁)

```
role? Sir Robin
actor? Eric Idle
```

对执行顺序的保证对赋值表达式来说很有用，因为在键表达式中赋值的变量将可在值表达式中被使用:

```
>>> names = ['Martin von Löwis', 'Łukasz Langa', 'Walter Dörwald']
>>> {(n := normalize('NFC', name)).casefold() : n for name in names}
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

（由 Jörn Heissler 在 bpo-35224 中贡献。）

- `object.__reduce__()` 方法现在可返回长度为二至六个元素的元组。之前的上限为五个。新增的第六个可选元素是签名为 `(obj, state)` 的可调用对象。这样就允许直接控制特定对象的状态更新。如果元素值不为 *None*，该可调用对象将优先于对象的 `__setstate__()` 方法。（由 Pierre Glaser 和 Olivier Grisel 在 bpo-35900 中贡献。）

4 新增模块

- 新增的 `importlib.metadata` 模块提供了从第三方包读取元数据的（临时）支持。例如，它可以提取一个已安装软件包的版本号、入口点列表等等:

```
>>> # Note following example requires that the popular "requests"
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, requires, files
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER'),
 PackagePath('requests-2.22.0.dist-info/LICENSE'),
 PackagePath('requests-2.22.0.dist-info/METADATA'),
 PackagePath('requests-2.22.0.dist-info/RECORD'),
 PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

（由 Barry Warsaw 和 Jason R. Coombs 在 bpo-34632 中贡献）。

5 改进的模块

5.1 ast

AST 节点现在具有 `end_lineno` 和 `end_col_offset` 属性，它们给出节点结束的精确位置。（这只适用于具有 `lineno` 和 `col_offset` 属性的节点。）

新增函数 `ast.get_source_segment()` 返回指定 AST 节点的源代码。

（由 Ivan Levkivskyi 在 bpo-33416 中贡献。）

`ast.parse()` 函数具有一些新的旗标:

- `type_comments=True` 导致其返回与特定 AST 节点相关联的 **PEP 484** 和 **PEP 526** 类型注释文本；
- `mode='func_type'` 可被用于解析 **PEP 484** ” 签名类型注释” (为函数定义 AST 节点而返回)；

- `feature_version=(3, N)` 允许指定一个更早的 Python 3 版本。例如，`feature_version=(3, 4)` 将把 `async` 和 `await` 视为非保留字。

（由 Guido van Rossum 在 bpo-35766 中贡献。）

5.2 asyncio

`asyncio.run()` 已经从暂定状态晋级为稳定 API。此函数可被用于执行一个 coroutine 并返回结果，同时自动管理事件循环。例如:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

这大致等价于:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

实际的实现要更复杂许多。因此 `asyncio.run()` 应该作为运行 asyncio 程序的首选方式。

（由 Yury Selivanov 在 bpo-32314 中贡献。）

运行 `python -m asyncio` 将启动一个原生异步 REPL。这允许快速体验具有最高层级 `await` 的代码。这时不再需要直接调用 `asyncio.run()`，因为此操作会在每次发起调用时产生一个新事件循环:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

（由 Yury Selivanov 在 bpo-37028 中贡献。）

异常 `asyncio.CancelledError` 现在继承自 `BaseException` 而不是 `Exception`。（由 Yury Selivanov 在 bpo-32528 中贡献。）

在 Windows 上，现在默认的事件循环为 `ProactorEventLoop`。（由 Victor Stinner 在 bpo-34687 中贡献。）

`ProactorEventLoop` 现在也支持 UDP。（由 Adam Meily 和 Andrew Svetlov 在 bpo-29883 中贡献。）

`ProactorEventLoop` 现在可通过 `KeyboardInterrupt` ("CTRL+C") 来中断。（由 Vladimir Matveev 在 bpo-23057 中贡献。）

添加了 `asyncio.Task.get_coro()` 用来获取 `asyncio.Task` 中的已包装协程。（由 Alex Grönholm 在 bpo-36999 中贡献。）

asyncio 任务现在可以被命名，或者是通过将 `name` 关键字参数传给 `asyncio.create_task()` 或 `create_task()` 事件循环方法，或者是通过在任务对象上调用 `set_name()` 方法。任务名称在 `asyncio.Task` 的 `repr()` 输出中可见，并且还可以使用 `get_name()` 方法来获取。(由 Alex Grönholm 在 bpo-34270 中贡献。)

将对 Happy Eyeballs 的支持添加到 `asyncio.loop.create_connection()`。要指定此行为，已增加了两个新的形参: *happy_eyeballs_delay* 和 *interleave*。Happy Eyeballs 算法可提升支持 IPv4 和 IPv6 的应用的响应速度，具体做法是尝试同时使用两者进行连接。(由 twisteroid ambassador 在 bpo-33530 中贡献。)

5.3 builtins

内置的 `compile()` 已改进为可接受 `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` 旗标。当传入此新旗标时，`compile()` 将允许通常被视为无效语法的最高层级 `await`, `async for` 和 `async with` 构造。此后将可返回带有 `CO_COROUTINE` 旗标的异步代码对象。(由 Matthias Bussonnier 在 bpo-34616 中贡献。)

5.4 collections

`collections.namedtuple()` 的 `_asdict()` 方法现在将返回 `dict` 而不是 `collections.OrderedDict`。此项更改是由于普通字典自 Python 3.7 起已保证具有确定的元素顺序。如果还需要 `OrderedDict` 的额外特性，建议的解决方案是将结果转换为需要的类型: `OrderedDict(nt._asdict())`。(由 Raymond Hettinger 在 bpo-35864 中贡献。)

5.5 cProfile

`cProfile.Profile` 类现在可被用作上下文管理器。在运行时对一个代码块实现性能分析:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(由 Scott Sanderson 在 bpo-29235 中贡献。)

5.6 csv

`csv.DictReader` 现在将返回 `dict` 而不是 `collections.OrderedDict`。此工具现在会更快速且消耗更少内存同时仍然保留字段顺序。(由 Michael Selik 在 bpo-34003 中贡献。)

5.7 curses

添加了一个新变量用于保存下层 ncurses 库的结构版信息: `ncurses_version`。(由 Serhiy Storchaka 在 bpo-31680 中贡献。)

5.8 ctypes

在 Windows 上，CDLL 及其子类现在接受 *winmode* 形参来指定用于底层 LoadLibraryEx 调用的旗标。默认旗标被设为仅加载来自可信任位置的 DLL 依赖项，包括 DLL 的存放路径（如果加载初始 DLL 时使用了完整或部分路径）以及通过 add_dll_directory() 添加的路径。（由 Steve Dower 在 bpo-36085 中贡献。）

5.9 datetime

添加了新的替代构造器 datetime.date.fromisocalendar() 和 datetime.datetime.fromisocalendar()，它们分别基于 ISO 年份、周序号和周内日序号来构造 date 和 datetime 对象；这两者分别是其所对应类中 isocalendar 方法的逆操作。（由 Paul Ganssle 在 bpo-36004 中贡献。）

5.10 functools

functools.lru_cache() 现在可直接作为装饰器而不是作为返回装饰器的函数。因此这两种写法现在都被支持:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

（由 Raymond Hettinger 在 bpo-36772 中贡献。）

添加了新的 functools.cached_property() 装饰器，用于在实例生命周期内缓存的已计算特征属性。

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

（由 Carl Meyer 在 bpo-21145 中贡献）

添加了新的 functools.singledispatchmethod() 装饰器可使用 single dispatch 将方法转换为 泛型函数:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)
```

（下页继续）

(繼續上一頁)

```
    @discard.register(list)
    def _(self, tasks):
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in targets]
```

(由 Ethan Smith 在 bpo-32380 中贡献)

5.11 gc

`get_objects()` 现在能接受一个可选的 *generation* 形参来指定一个用于获取对象的生成器。(由 Pablo Galindo 在 bpo-36016 中贡献。)

5.12 gettext

添加了 `pgettext()` 及其变化形式。(由 Franz Glasner, Éric Araujo 和 Cheryl Sabella 在 bpo-2504 中贡献。)

5.13 gzip

添加 *mtime* 形参到 `gzip.compress()` 用于可重现的输出。(由 Guo Ci Teo 在 bpo-34898 中贡献。)

对于特定类型的无效或已损坏 gzip 文件现在将引发 `BadGzipFile` 而不是 `OSError`。(由 Filip Gruszczyński, Michele Orrù 和 Zackery Spytz 在 bpo-6584 中贡献。)

5.14 IDLE 与 idlelib

超过 N 行(默认值为 50)的输出将被折叠为一个按钮。N 可以在 Settings 对话框的 General 页的 PyShell 部分中进行修改。数量较少但是超长的行可以通过在输出上右击来折叠。被折叠的输出可通过双击按钮来展开，或是通过右击按钮来放入剪贴板或是单独的窗口。(由 Tal Einat 在 bpo-1529353 中贡献。)

在 Run 菜单中增加了"Run Customized" 以使用自定义设置来运行模块。输入的任何命令行参数都会被加入 sys.argv。它们在下次自定义运行时会再次显示在窗体中。用户也可以禁用通常的 Shell 主模块重启。(由 Cheryl Sabella, Terry Jan Reedy 等人在 bpo-5680 和 bpo-37627 中贡献。)

在 IDLE 编辑器窗口中增加了可选的行号。窗口打开时默认不显示行号，除非在配置对话框的 General 选项卡中特别设置。已打开窗口中的行号可以在 Options 菜单中显示和隐藏。(由 Tal Einat 和 Saimadhav Heblikar 在 bpo-17535 中贡献。)

现在会使用 OS 本机编码格式在 Python 字符串和 Tcl 对象间进行转换。这允许在 IDLE 中处理 emoji 和其他非 BMP 字符。这些字符将可被显示或是从剪贴板复制和粘贴。字符串从 Tcl 到 Python 的来回转换现在不会再发生失败。(过去八年有许多人都为此付出过努力，问题最终由 Serhiy Storchaka 在 bpo-13153 中解决。)

上述修改已被反向移植到 3.7 维护发行版中。

5.15 inspect

`inspect.getdoc()` 函数现在可以找到 `__slots__` 的文档字符串，如果该属性是一个元素值为文档字符串的 `dict` 的话。这提供了类似于目前已有的 `property()`，`classmethod()` 和 `staticmethod()` 等函数的文档选项:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to one decimal place',
                 'duration': 'in seconds, rounded up to an integer'}
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

（由 Raymond Hettinger 在 bpo-36326 中贡献。）

5.16 io

在开发模式 (`-X env`) 和调试构建中，`io.IOBase` 终结器现在会在 `close()` 方法失败时将异常写入日志。发生的异常在发布构建中默认会被静默忽略。（由 Victor Stinner 在 bpo-18748 中贡献。）

5.17 itertools

`itertools.accumulate()` 函数增加了可选的 *initial* 关键字参数用来指定一个初始值:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

（由 Lisa Roach 在 bpo-34659 中贡献。）

5.18 json.tool

添加选项 `--json-lines` 用于将每个输入行解析为单独的 JSON 对象。（由 Weipeng Hong 在 bpo-31553 中贡献。）

5.19 logging

为 `logging.basicConfig()` 添加了 *force* 关键字参数，当设为真值时，关联到根日志记录器的任何现有处理程序都将在执行由其他参数所指定的配置之前被移除并关闭。

这解决了一个长期存在的问题。当一个日志处理器或 *basicConfig()* 被调用时，对 *basicConfig()* 的后续调用会被静默地忽略。这导致使用交互提示符或 Jupyter 笔记本更新、试验或讲解各种日志配置选项变得相当困难。

（由 Raymond Hettinger 提议，由 Dong-hee Na 实现，并由 Vinay Sajip 在 bpo-33897 中完成审核。）

5.20 math

添加了新的函数 `math.dist()` 用于计算两点之间的欧几里得距离。(由 Raymond Hettinger 在 bpo-33089 中贡献。)

扩展了 `math.hypot()` 函数以便处理更多的维度。之前它仅支持 2-D 的情况。(由 Raymond Hettinger 在 bpo-33089 中贡献。)

添加了新的函数 `math.prod()` 作为的 `sum()` 同类，该函数返回'start' 值 (默认值: 1) 乘以一个数字可迭代对象的积:

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(由 Pablo Galindo 在 bpo-35606 中贡献。)

添加了两个新的组合函数 `math.perm()` 和 `math.comb()`:

```
>>> math.perm(10, 3)     # Permutations of 10 things taken 3 at a time
720
>>> math.comb(10, 3)     # Combinations of 10 things taken 3 at a time
120
```

(由 Yash Aggarwal, Keller Fuchs, Serhiy Storchaka 和 Raymond Hettinger 在 bpo-37128, bpo-37178 和 bpo-35431 中贡献。)

添加了一个新函数 `math.isqrt()` 用于计算精确整数平方根而无需转换为浮点数。该新函数支持任意大整数。它的执行速度比 `floor(sqrt(n))` 快但是比 `math.sqrt()` 慢:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)          # correct
650320426
>>> floor(sqrt(s - 1))    # incorrect
650320427
```

(由 Mark Dickinson 在 bpo-36887 中贡献。)

函数 `math.factorial()` 不再接受非整数类参数。(由 Pablo Galindo 在 bpo-33083 中贡献。)

5.21 mmap

`mmap.mmap` 类现在具有一个 `madvise()` 方法用于访问 `madvise()` 系统调用。(由 Zackery Spytz 在 bpo-32941 中贡献。)

5.22 multiprocessing

添加了新的 `multiprocessing.shared_memory` 模块。(由 Davin Potts 在 bpo-35813 中贡献。)

在 macOS 上，现在默认使用的启动方式是 *spawn* 启动方式。(由 Victor Stinner 在 bpo-33725 中贡献。)

5.23 os

在 Windows 上添加了新函数 `add_dll_directory()` 用于在导入扩展模块或使用 `ctypes` 加载 DLL 时为本机依赖提供额外搜索路径。(由 Steve Dower 在 bpo-36085 中贡献。)

添加了新的 `os.memfd_create()` 函数用于包装 `memfd_create()` 系统调用。(由 Zackery Spytz 和 Christian Heimes 在 bpo-26836 中贡献。)

在 Windows 上，大部分用于处理重解析点,(包括符号链接和目录连接) 的手动逻辑已被委托给操作系统。特别地，`os.stat()` 现在将会遍历操作系统所支持的任何内容，而 `os.lstat()` 将只打开被标识为“名称代理”的重解析点，而其要由 `os.stat()` 打开其他的重解析点。在所有情况下，`stat_result.st_mode` 将只为符号链接而非其他种类的重解析点设置 `S_IFLNK`。要标识其他种类的重解析点，请检查新的 `stat_result.st_reparse_tag` 属性。

在 Windows 上，`os.readlink()` 现在能够读取目录连接。请注意 `islink()` 会对目录连接返回 `False`，因此首先检查 `islink` 的代码将连续把连接视为目录，而会处理 `os.readlink()` 所引发错误的代码现在会把连接视为链接。

(由 Steve Dower 在 bpo-37834 中贡献。)

5.24 os.path

返回布尔值结果的 `os.path` 函数例如 `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()`, 以及 `ismount()` 现在对于包含在 OS 层级无法表示的字符或字节的路径将会返回 `False` 而不是引发 `ValueError` 或其子类 `UnicodeEncodeError` 和 `UnicodeDecodeError`。(由 Serhiy Storchaka 在 bpo-33721 中贡献。)

`expanduser()` 在 Windows 上现在改用 `USERPROFILE` 环境变量而不再使用 `HOME`，后者通常不会为一般用户账户设置。(由 Anthony Sottile 在 bpo-36264 中贡献。)

`isdir()` 在 Windows 上将不再为不存在的目录的链接返回 `True`。

`realpath()` 在 Windows 上现在会识别重解析点，包括符号链接和目录连接。

(由 Steve Dower 在 bpo-37834 中贡献。)

5.25 pathlib

返回布尔值结果的 `pathlib.Path` 方法例如 `exists()`, `is_dir()`, `is_file()`, `is_mount()`, `is_symlink()`, `is_block_device()`, `is_char_device()`, `is_fifo()`, `is_socket()` 现在对于包含在 OS 层级无法表示的字符或字节的路径将会返回 `False` 而不是引发 `ValueError` 或其子类 `UnicodeEncodeError`。(由 Serhiy Storchaka 在 bpo-33721 中贡献。)

添加了 `pathlib.Path.link_to()` 用于创建指向某个路径的硬链接。(由 Joannah Nanjekye 在 bpo-26978 中贡献。)

5.26 pickle

`pickle` 扩展子类化针对 C 优化的 `Pickler` 现在可通过定义特殊的 `reducer_override()` 方法来重载函数和类的封存逻辑。(由 Pierre Glaser 和 Olivier Grisel 在 bpo-35900 中贡献。)

5.27 plistlib

添加了新的 `plistlib.UID` 并启动了对读取和写入经过 NSKeyedArchiver 编码的二进制 plists 的支持。（由 Jon Janzen 在 bpo-26707 中贡献。）

5.28 pprint

`pprint` 模块为一些函数添加了 *sort_dicts* 形参。默认情况下，这些函数会继续在渲染或打印之前对字典进行排序。但是，如果 *sort_dicts* 设为假值，则字典将保持键插入时的顺序。这在调试期间与 JSON 输入进行比较时会很有用。

除此之外，还增加了一个方便的新函数 `pprint.pp()`，它类似于 `pprint.pprint()` 但它的 *sort_dicts* 默认为 `False`:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', destination='output.txt')
>>> pp(d, width=40)                  # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)              # Keys sorted alphabetically
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

（由 Rémi Lapeyre 在 bpo-30670 中贡献。）

5.29 py_compile

`py_compile.compile()` 现在支持静默模式。（由 Joannah Nanjekye 在 bpo-22640 中贡献。）

5.30 shlex

新增了 `shlex.join()` 函数作为 `shlex.split()` 的逆操作。（由 Bo Bayles 在 bpo-32102 中贡献。）

5.31 shutil

`shutil.copytree()` 现在接受新的 `dirs_exist_ok` 关键字参数。（由 Josh Bronson 在 bpo-20849 中贡献。）

`shutil.make_archive()` 现在对新的归档默认使用 modern pax (POSIX.1-2001) 格式以提升可移植性和标准一致性，此特性继承自对 `tarfile` 模块的相应更改。（由 C.A.M. Gerlach 在 bpo-30661 中贡献。）

`shutil.rmtree()` 在 Windows 上现在会移除目录连接而不会递归地先移除其中的内容。（由 Steve Dower 在 bpo-37834 中贡献。）

5.32 socket

添加了便捷的 `create_server()` 和 `has_dualstack_ipv6()` 函数以自动化在创建服务器套接字时通常情况下所必须的任务，包括在同一套接字中同时接受 IPv4 和 IPv6 连接。（由 Giampaolo Rodolà 在 bpo-17561 中贡献。）

`socket.if_nameindex()`, `socket.if_nametoindex()` 和 `socket.if_indextoname()` 函数已经在 Windows 上实现。（由 Zackery Spytz 在 bpo-37007 中贡献。）

5.33 ssl

增加了 `post_handshake_auth` 和 `verify_client_post_handshake()` 分别启用和初始化 TLS 1.3 握手后验证。（由 Christian Heimes 在 bpo-34670 中贡献。）

5.34 statistics

添加了 `statistics.fmean()` 作为 `statistics.mean()` 的更快速的浮点数版版本。（由 Raymond Hettinger 和 Steven D'Aprano 在 bpo-35904 中贡献。）

添加了 `statistics.geometric_mean()` （由 Raymond Hettinger 在 bpo-27181 中贡献。）

添加了 `statistics.multimode()` 用于返回最常见值的列表。（由 Raymond Hettinger 在 bpo-35892 中贡献。）

添加了 `statistics.quantiles()` 用于将数据或分布划分为多个等概率区间（例如四分位、十分位或百分位）。（由 Raymond Hettinger 在 bpo-36546 中贡献。）

添加了 `statistics.NormalDist` 用于创建和操纵随机变量的正态分布。（由 Raymond Hettinger 在 bpo-36018 中贡献。）

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3, 2, 7, 14])
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)           # Chance of being under 3 degrees
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degrees
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
>>> temperature_feb += el_niño       # Add in a climate effect
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32     # Convert to Fahrenheit
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)       # Generate random samples
[7.672102882379219, 12.000027119750287, 4.647488369766392]
```

5.35 sys

添加了新的 sys.unraisablehook() 函数，可被重载以便控制如何处理“不可引发的异常”。它会在发生了一个异常但 Python 没有办法处理时被调用。例如，当一个析构器在垃圾回收时 (gc.collect()) 所引发的异常。(由 Victor Stinner 在 bpo-36829 中贡献。)

5.36 tarfile

tarfile 模块现在对新的归档默认使用 modern pax (POSIX.1-2001) 格式而不再是之前的 GNU 专属格式。这通过标准化和可扩展格式的统一编码 (UTF-8) 提升了跨平台可移植性，还提供了其他一些益处。(由 C.A.M. Gerlach 在 bpo-36268 中贡献。)

5.37 threading

添加了新的 threading.excepthook() 函数用来处理未捕获的 threading.Thread.run() 异常。它可被重载以便控制如何处理未捕获的 threading.Thread.run() 异常。(由 Victor Stinner 在 bpo-1230540 中贡献。)

添加了新的 threading.get_native_id() 函数以及 threading.Thread 类的 native_id 属性。它们会返回内核所分配给当前线程的原生整数线程 ID。此特性仅在特定平台上可用，参见 get_native_id 了解详情。(由 Jake Tesler 在 bpo-36084 中贡献。)

5.38 tokenize

当提供不带末尾新行的输入时，tokenize 模块现在会隐式地添加 NEWLINE 形符。此行为现在已与 C 词法分析器的内部行为相匹配。(由 Ammar Askar 在 bpo-33899 中贡献。)

5.39 tkinter

在 tkinter.Spinbox 中添加了方法 selection_from(), selection_present(), selection_range() 和 selection_to()。(由 Juliette Monsel 在 bpo-34829 中贡献。)

在 tkinter.Canvas 类中添加了方法 moveto()。(由 Juliette Monsel 在 bpo-23831 中贡献。)

tkinter.PhotoImage 类现在具有 transparency_get() 和 transparency_set() 方法。(由 Zackery Spytz 在 bpo-25451 中贡献。)

5.40 time

为 macOS 10.12 添加了新的时钟 CLOCK_UPTIME_RAW。(由 Joannah Nanjekye 在 bpo-35702 中贡献。)

5.41 typing

typing 模块加入了一些新特性：

- 一个带有键专属类型的字典类型。参见 **PEP 589** 和 typing.TypedDict。TypedDict 只使用字符串作为键。默认情况下每个键都要求提供。指定”total=False” 以允许键作为可选项：

```
class Location(TypedDict, total=False):
    lat_long: tuple
    grid_square: str
    xy_coordinate: tuple
```

- Literal 类型。参见 **PEP 586** 和 typing.Literal。Literal 类型指明一个形参或返回值被限定为一个或多个特定的字面值：

```
def get_status(port: int) -> Literal['connected', 'disconnected']:
    ...
```

- ”Final” 变量、函数、方法和类。参见 **PEP 591**, `typing.Final` 和 `typing.final()`。final 限定符会指示静态类型检查器限制进行子类化、重载或重新赋值:

```
pi: Final[float] = 3.1415926536
```

- 协议定义。参见 **PEP 544**, `typing.Protocol` 和 `typing.runtime_checkable()`。简单的 ABC 例如 `typing.SupportsInt` 现在是 `Protocol` 的子类。
- 新的协议类 `typing.SupportsIndex`。
- 新的函数 `typing.get_origin()` 和 `typing.get_args()`。

5.42 unicodedata

`unicodedata` 模块现在已升级为使用 Unicode 12.1.0 发布版。

新的函数 `is_normalized()` 可被用来验证字符串是否为特定正规形式，通常会比实际进行字符串正规化要快得多。(由 Max Belanger, David Euresti 和 Greg Price 在 bpo-32285 和 bpo-37966 中贡献。)

5.43 unittest

添加了 `AsyncMock` 以支持异步版本的 `Mock`。同时也添加了相应的断言函数用于测试。(由 Lisa Roach 在 bpo-26467 中贡献。)

unittest 添加了 `addModuleCleanup()` 和 `addClassCleanup()` 以支持对 `setUpModule()` 和 `setUpClass()` 进行清理。(由 Lisa Roach 在 bpo-24412 中贡献。)

一些模拟断言函数现在也会在失败时打印一个实际调用列表。(由 Petter Strandmark 在 bpo-35047 中贡献。)

`unittest` 模块已支持通过 `unittest.IsolatedAsyncioTestCase` 来使用协程作为测试用例。(由 Andrew Svetlov 在 bpo-32972 中贡献。)

示例:

```
import unittest


class TestRequest(unittest.IsolatedAsyncioTestCase):

    async def asyncSetUp(self):
        self.connection = await AsyncConnection()

    async def test_get(self):
        response = await self.connection.get("https://example.com")
        self.assertEqual(response.status_code, 200)

    async def asyncTearDown(self):
        await self.connection.close()


if __name__ == "__main__":
    unittest.main()
```

5.44 venv

现在 venv 在所有平台上都会包含 Activate.ps1 脚本用于在 PowerShell Core 6.1 下激活虚拟环境。(由 Brett Cannon 在 bpo-32718 中贡献。)

5.45 weakref

由 weakref.proxy() 返回的代理对象现在除其他算术运算符外也支持矩阵乘法运算符 @ 和 @=。(由 Mark Dickinson 在 bpo-36669 中贡献。)

5.46 xml

作为对 DTD 和外部实体检索的缓解，在默认情况下 xml.dom.minidom 和 xml.sax 模块不再处理外部实体。(由 Christian Heimes 在 bpo-17239 中贡献。)

xml.etree.ElementTree 模块中的 .find*() 方法支持通配符搜索例如 {*}tag，此搜索会忽略命名空间以及返回给定命名空间中所有标签的 {namespace}*。(由 Stefan Behnel 在 bpo-28238 中贡献。)

xml.etree.ElementTree 模块提供了实现 C14N 2.0 的新函数 –xml.etree.ElementTree.canonicalize()。(由 Stefan Behnel 在 bpo-13611 中贡献。)

xml.etree.ElementTree.XMLParser 的目标对象可通过新的回调方法 start_ns() 和 end_ns() 来接受命名空间声明事件。此外，xml.etree.ElementTree.TreeBuilder 目标可被配置为处理有关注释和处理指令事件以将它们包含在所生成的树当中。(由 Stefan Behnel 在 bpo-36676 和 bpo-36673 中贡献。)

5.47 xmlrpc

xmlrpc.client.ServerProxy 现在支持可选的 *headers* 关键字参数作为随同每次请求发送的 HTTP 标头序列。此特征的作用之一是使得从默认的基础认证升级到更快速的会话认证成为可能。(由 Cédric Krier 在 bpo-35153 中贡献。)

6 性能优化

- subprocess 模块现在能在某些情况下使用 os.posix_spawn() 函数以获得更好的性能。目前，它的使用仅限 macOS 和 Linux（使用 glibc 2.24 或更新版本），并要求满足以下条件：
  - *close_fds* 为假值；
  - *preexec_fn*, *pass_fds*, *cwd* 和 *start_new_session* 形参未设置；
  - *executable* 路径包含一个目录。

  (由 Joannah Nanjekye 和 Victor Stinner 在 bpo-35537 中贡献。)
- shutil.copyfile()，shutil.copy()，shutil.copy2()，shutil.copytree() 和 shutil.move() 在 Linux 和 macOS 上会使用平台专属的"fast-copy" 系统调用以提高效率。"fast-copy" 意味着拷贝操作发生于内核中，从而避免在进行"outfd.write(infd.read())" 等操作时使用 Python 中的用户空间缓冲区。在 Windows 上 shutil.copyfile() 会使用更大的默认缓冲区(1 MiB 而不是 16 KiB) 并且使用基于 memoryview() 的 shutil.copyfileobj() 版本。在同一分区内拷贝一个 512 MiB 文件的速度提升在 Linux 上约为 +26%，在 macOS 上为 +50%，在 Windows 上为 +40%。此外还将消耗更少的 CPU 周期。参见 shutil-platform-dependent-efficient-copy-operations 一节。(由 Giampaolo Rodolà 在 bpo-33671 中贡献。)
- shutil.copytree() 会根据其所用缓存的 os.stat() 值使用 os.scandir() 函数及所有拷贝函数。拷贝一个包含 8000 文件的目录的速度提升在 Linux 上约为 +9%，在 Windows 上为 +20%，对于 Windows SMB 共享目录则为 +30%。此外 os.stat() 系统调用的次数也减少了 38%，使得

shutil.copytree() 在网络文件系统上会特别快速。（由 Giampaolo Rodolà 在 bpo-33695 中贡献。）

- pickle 模块使用的默认协议现在为 Protocol 4，最早在 Python 3.4 中被引入。它提供了比自 Python 3.0 起可用的 Protocol 3 更好的性能和更小的数据尺寸。
- 从 PyGC_Head 移除了一个 Py_ssize_t 成员。所有跟踪 GC 的对象（例如 tuple, list, dict）大小减少了 4 或 8 字节。（由 Inada Naoki 在 bpo-33597 中贡献。）
- uuid.UUID 现在会使用 __slots__ 以减少内存足迹。（由 Wouter Bolsterlee 和 Tal Einat 在 bpo-30977 中贡献。）
- operator.itemgetter() 的性能提升了 33%。优化了参数处理，并为常见的在元组中单个非负整数索引的情况新增了一条快速路径（这是标准库中的典型用例）。（由 Raymond Hettinger 在 bpo-35664 中贡献。
- 加快了在 collections.namedtuple() 中的字段查找。它们现在的速度快了两倍以上，成为 Python 中最快的实例变量查找形式。（由 Raymond Hettinger, Pablo Galindo 和 Joe Jevnik, Serhiy Storchaka 在 bpo-32492 中贡献。）
- 如果输入的可迭代对象的长度已知 (即输入对象实现了 __len__)，list 构造器不会过度分配内部项缓冲区。这使得所创建的列表资源占用平均减少了 12%。（由 Raymond Hettinger 和 Pablo Galindo 在 bpo-33234 中贡献。）
- 类变量写入速度加倍。当一个非冗余属性被更新时，之前存在一个更新 slots 的非必要调用。（由 Stefan Behnel, Pablo Galindo Salgado, Raymond Hettinger, Neil Schemenauer, 和 Serhiy Storchaka 在 bpo-36012 中贡献。）
- 减少了传递给许多内置函数和方法的参数转换的开销。这使得某些简单内置函数和方法的速度提升了 20--50%。（由 Serhiy Storchaka 在 bpo-23867, bpo-35582 和 bpo-36127 中贡献。）
- LOAD_GLOBAL 指令现在会使用新的"per opcode cache" 机制。它的速度现在提升了大约 40%。（由 Yury Selivanov 和 Inada Naoki 在 bpo-26219 中贡献。）

7 构建和 C API 的改变

- 默认的 sys.abiflags 成为一个空字符串：pymalloc 的 m 旗标不再有意义（无论是否启用 pymalloc 构建都是兼容 ABI 的）因此已被移除。（由 Victor Stinner 在 bpo-36707 中贡献。）

  改变的例子：

  - 只会安装 python3.8 程序，不再有 python3.8m 程序。
  - 只会安装 python3.8-config 脚本，不再有 python3.8m-config 脚本。
  - m 旗标已经从动态库文件名后缀中移除：包括标准库中的扩展模块以及第三方包所产生和安装的模块例如从 PyPI 下载的模块。以 Linux 为例，Python 3.7 的后缀 .cpython-37m-x86_64-linux-gnu.so 在 Python 3.8 中改为 .cpython-38-x86_64-linux-gnu.so。
- 重新组织了所有头文件以更好地区分不同种类的 API：
  - Include/*.h 应为可移植的公有稳定版 C API。
  - Include/cpython/*.h 应为 CPython 专属的不稳定版 C API；公有 API，部分私有 API 附加 _Py or _PY 前缀。
  - Include/internal/*.h 应为 CPython 特别专属的私有内部 C API。此 API 不具备向下兼容保证并且不应在 CPython 以外被使用。它们的公开仅适用于特别限定的需求例如调试器和性能分析等必须直接访问 CPython 内部数据而不通过调用函数的应用。此 API 现在是通过 make install 安装的。

  （由 Victor Stinner 在 bpo-35134 和 bpo-35081 中贡献，相关工作由 Eric Snow 在 Python 3.7 中发起。）
- 某些宏已被转换为静态内联函数：形参类型和返回类型定义良好，它们不再会有与宏相关的问题，变量具有局部作用域。例如：

  - `Py_INCREF()`, `Py_DECREF()`
  - `Py_XINCREF()`, `Py_XDECREF()`
  - `PyObject_INIT()`, `PyObject_INIT_VAR()`
  - 私有函数: `_PyObject_GC_TRACK()`, `_PyObject_GC_UNTRACK()`, `_Py_Dealloc()`

  (由 Victor Stinner 在 bpo-35059 中贡献。)

- `PyByteArray_Init()` 和 `PyByteArray_Fini()` 函数已被移除。它们自 Python 2.7.4 和 Python 3.2.0 起就没有任何用处，被排除在受限 API (稳定版 ABI) 之外，并且未被写入文档。(由 Victor Stinner 在 bpo-35713 中贡献。)
- `PyExceptionClass_Name()` 的结果类型现在是 `const char *` 而非 `char *`。(由 Serhiy Storchaka 在 bpo-33818 中贡献。)
- `Modules/Setup.dist` 与 `Modules/Setup` 两者的共存已被移除。之前在更新 CPython 源码树时，开发者必须手动拷贝 `Modules/Setup.dist` (在源码树内) 到 `Modules/Setup` (在构建树内) 以反映上游的任何改变。旧特性对打包者来说有一点益处，但代价是对追踪 CPython 开发进程的开发者造成经常性的麻烦，因为忘记拷贝该文件可能导致构建失败。

  现在构建系统总是会从源码树内的 `Modules/Setup` 读取数据。建议希望定制该文件的开发者在 CPython 的一个 git 分叉或补丁文件中维护他们的更改，就如他们对源码树做任何其他改变时一样。

  (由 Antoine Pitrou 在 bpo-32430 中贡献。)
- 将 Python 数字转换为 C 整型的函数例如 `PyLong_AsLong()` 以及带有 `'i'` 之类整型转换格式单元的参数解析函数例如 `PyArg_ParseTuple()` 现在如果可能将会使用 `__index__()` 特殊方法而不是 `__int__()`。对于带有 `__int__()` 方法但没有 `__index__()` 方法的对象 (例如 `Decimal` 和 `Fraction`) 将会发出弃用警告。对于实现了 `__index__()` 的对象 `PyNumber_Check()` 现在将返回 `1`。`PyNumber_Long()`, `PyNumber_Float()` 和 `PyFloat_AsDouble()` 现在如果可能也将会使用 `__index__()` 方法。(由 Serhiy Storchaka 在 bpo-36048 和 bpo-20092 中贡献。)
- 堆分配类型对象现在将增加它们在 `PyObject_Init()` (及其对应的宏 `PyObject_INIT`) 中的引用计数而不是在 `PyType_GenericAlloc()` 中。修改实例分配或中止分配的类型可能需要进行调整。(由 Elizondo 在 bpo-35810 中贡献。)
- 新增函数 `PyCode_NewWithPosOnlyArgs()` 允许创建代码对象例如 `PyCode_New()`，但带有一个额外的 *posonlyargcount* 形参以指明仅限位置参数的数量。(由 Pablo Galindo 在 bpo-37221 中贡献。)
- `Py_SetPath()` 现在会将 `sys.executable` 设为程序完整路径 (`Py_GetProgramFullPath()`) 而不是程序名称 (`Py_GetProgramName()`)。(由 Victor Stinner 在 bpo-38234 中贡献。)

8 弃用

- distutils 的 `bdist_wininst` 命令现在已弃用，请改用 `bdist_wheel` (wheel 包)。(由 Victor Stinner 在 bpo-37481 中贡献。)
- `ElementTree` 模块中的已弃用方法 `getchildren()` 和 `getiterator()` 现在会引发 `DeprecationWarning` 而不是 `PendingDeprecationWarning`。它们将在 Python 3.9 中被移除。(由 Serhiy Storchaka 在 bpo-29209 中贡献。)
- 将不是 `concurrent.futures.ThreadPoolExecutor` 的实例的对象传给 `loop.set_default_executor()` 已被弃用，并将在 Python 3.9 中被禁止。(由 Elvis Pranskevichus 在 bpo-34075 中贡献。)
- `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` 和 `fileinput.FileInput` 的 `__getitem__()` 方法已被弃用。

  这些方法的实现会忽略它们的 *index* 形参，并改为返回下一条目。(由 Berker Peksag 在 bpo-9372 中贡献。)

- typing.NamedTuple 类已弃用了 _field_types 属性而改用具有同样信息的 __annotations__ 属性。(由 Raymond Hettinger 在 bpo-36320 中贡献。)
- ast 类 Num, Str, Bytes, NameConstant 和 Ellipsis 被视为已弃用并将在未来的 Python 版本中被移除。应当改用 Constant。(由 Serhiy Storchaka 在 bpo-32892 中贡献。)
- ast.NodeVisitor 的方法 visit_Num(), visit_Str(), visit_Bytes(), visit_NameConstant() 和 visit_Ellipsis() 现在已被弃用，并且在未来的 Python 版本中将不再被调用。添加 visit_Constant() 方法来处理所有常量节点。(由 Serhiy Storchaka 在 bpo-36917 中贡献。)
- asyncio.coroutine() decorator 已弃用并将在 3.10 版中被移除。原有的 @asyncio.coroutine 请改用 async def 来替代。(由 Andrew Svetlov 在 bpo-36921 中贡献。)
- 在 asyncio 中，*loop* 参数的显式传递已被弃用并且到 3.10 版时将会在以下函数中被移除: asyncio.sleep(), asyncio.gather(), asyncio.shield(), asyncio.wait_for(), asyncio.wait(), asyncio.as_completed(), asyncio.Task, asyncio.Lock, asyncio.Event, asyncio.Condition, asyncio.Semaphore, asyncio.BoundedSemaphore, asyncio.Queue, asyncio.create_subprocess_exec() 以及 asyncio.create_subprocess_shell()。
- 将协程对象显式传递给 asyncio.wait() 的做法已被弃用并且将在 3.11 版中被移除。(由 Yury Selivanov 在 bpo-34790 中贡献。)
- gettext 模块中的下列函数和方法已被弃用: lgettext(), ldgettext(), lngettext() 和 ldngettext()。它们返回已编码的字节串，如果转换后的字符串存在编码问题你将可能遭遇预期之外的异常。更好的做法是使用 Python 3 中返回 Unicode 字符串的操作作为替代。前面的函数都早已不宜使用。

  函数 bind_textdomain_codeset()，方法 output_charset() 和 set_output_charset()，以及函数 translation() 和 install() 的 *codeset* 形参也忆被弃用，因为它们仅适用于 l*gettext() 函数。(由 Serhiy Storchaka 在 bpo-33710 中贡献。)
- threading.Thread 的 isAlive() 方法已弃用。(由 Dong-hee Na 在 bpo-35283 中贡献。)
- 许多接受整数参数的内置和扩展模块的函数现在将对传入 Decimal, Fraction 以及任何其他可被转换为整数但会丢失精度（即具有 __int__() 方法但没有 __index__() 方法）的对象发出弃用警告。在未来的版本中则将报错。(由 Serhiy Storchaka 在 bpo-36048 中贡献。)
- 以下参数作为关键字参数传递的方式已被弃用:
  - functools.partialmethod(), weakref.finalize(), profile.Profile.runcall(), cProfile.Profile.runcall(), bdb.Bdb.runcall(), trace.Trace.runfunc() 与 curses.wrapper() 中的 *func*。
  - unittest.TestCase.addCleanup() 中的 *function*。
  - concurrent.futures.ThreadPoolExecutor 和 concurrent.futures.ProcessPoolExecutor 的 submit() 方法中的 *fn*。
  - contextlib.ExitStack.callback(), contextlib.AsyncExitStack.callback() 和 contextlib.AsyncExitStack.push_async_callback() 中的 *callback*。
  - multiprocessing.managers.Server 和 multiprocessing.managers.SharedMemoryServer 的 create() 方法中的 *c* 和 *typeid*。
  - weakref.finalize() 中的 *obj*。

  在未来版本的 Python 中，它们将成为 仅限位置参数。(由 Serhiy Storchaka 在 bpo-36492 中贡献。)

9 API 与特性的移除

下列特性与 API 已从 Python 3.8 中移除：

- 自 Python 3.3 起，从 `collections` 导入 ABC 的做法已弃用，而应当从 `collections.abc` 执行导入。从 collections 导入的操作在 3.8 中已标记为被移除，但将推迟到 3.9 中实施。(参见 bpo-36952。)
- `macpath` 模块，在 Python 3.7 中弃用，现已被移除。(由 Victor Stinner 在 bpo-35471 中贡献。)
- 函数 `platform.popen()` 已被移除，它自 Python 3.3 起就已被弃用：请改用 `os.popen()`。(由 Victor Stinner 在 bpo-35345 中贡献。)
- 函数 `time.clock()` 已被移除，它自 Python 3.3 起就已被弃用：请根据具体需求改用 `time.perf_counter()` 或 `time.process_time()` 以获得具有良好定义的行为。(由 Matthias Bussonnier 在 bpo-36895 中贡献。)
- `pyvenv` 脚本已被移除，推荐改用 `python3.8 -m venv` 来帮助消除容易混淆 `pyvenv` 脚本所关联的 Python 解释器这一问题。(由 Brett Cannon 在 bpo-25427 中贡献。)
- `parse_qs`, `parse_qsl` 和 `escape` 已从 `cgi` 模块中被移除。这些函数自 Python 3.2 或更早就已被弃用。它们应改为从 `urllib.parse` 和 `html` 模块导入。
- `filemode` 函数已从 `tarfile` 模块中被移除。该函数未被写入文档，自 Python 3.3 起就已弃用。
- `XMLParser` 构造器不再接受 *html* 参数。它从来没有任何作用并在 Python 3.4 中已被弃用。所有其他形参现在都是 仅限关键字参数。(由 Serhiy Storchaka 在 bpo-29209 中贡献。)
- `XMLParser` 的 `doctype()` 方法已被移除。(由 Serhiy Storchaka 在 bpo-29209 中贡献。)
- "unicode_internal" 编解码器已被移除。(由 Inada Naoki 在 bpo-36297 中贡献。)
- `sqlite3` 模块的 `Cache` 和 `Statement` 对象已不再公开给用户。(由 Aviv Palivoda 在 bpo-30262 中贡献。)
- `fileinput.input()` 和 `fileinput.FileInput()` 中自 Python 3.6 起就已被忽略并弃用的 `bufsize` 关键字参数已被移除。由 Matthias Bussonnier 在 bpo-36952 中贡献。)
- 在 Python 3.7 中弃用的函数 `sys.set_coroutine_wrapper()` 和 `sys.get_coroutine_wrapper()` 已被移除。(由 Matthias Bussonnier 在 bpo-36933 中贡献。)

10 移植到 Python 3.8

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.

10.1 Python 行为的改变

- yield 表达式（包括 `yield` 和 `yield from` 子句）现在不允许在推导式和生成器表达式中使用（但 `for` 子句最左边的可迭代对象表达式除外）。(由 Serhiy Storchaka 在 bpo-10544 中贡献。)
- 当标识号检测 (`is` 和 `is not`) 与特定类型的字面值 (例如字符串和数字) 一同使用时编译器现在会产生 `SyntaxWarning`。这在 CPython 中通常是可行的，但并不被语言定义所保证。该警告会建议用户改用相等性检测 (== and `!=`)。(由 Serhiy Storchaka 在 bpo-34850 中贡献。)
- CPython 解释器在某些情况下可以忽略异常。在 Python 3.8 中这种情况会更少发生。特别地，从类型字典获取属性时引发的异常不会再被忽略。(由 Serhiy Storchaka 在 bpo-35459 中贡献。)
- 从内置类型 `bool`, `int`, `float`, `complex` 和标准库的一些类中移除了 `__str__` 实现。它们现在会从 `object` 继承 `__str__()`。作为结果，在这些类的子类中定义 `__repr__()` 方法将会影响它们的字符串表示。(由 Serhiy Storchaka 在 bpo-36793 中贡献。)
- 在 AIX 上，`sys.platform` 将不再包含主要版本号。它将总是 `'aix'` 而不再是 `'aix3'` .. `'aix7'`。由于较旧的 Python 版本会包含该版本号，因此推荐总是使用 `sys.platform.startswith('aix')`。(由 M. Felt 在 bpo-36588 中贡献。)

- 现在 PyEval_AcquireLock() 和 PyEval_AcquireThread() 如果当解释器终结化时被调用将会终结当前线程，以使它们与 PyEval_RestoreThread(),Py_END_ALLOW_THREADS() 以及 PyGILState_Ensure() 保持一致。如果不想要这样的行为，请通过检测 _Py_IsFinalizing() 或 sys.is_finalizing() 来保护该调用。（由 Joannah Nanjekye 在 bpo-36475 中贡献。）

10.2 更改的 Python API

- 在 Windows 上 os.getcwdb() 函数现在会使用 UTF-8 编码格式而不是 ANSI 代码页：请参看 **PEP 529** 了解具体原因。该函数在 Windows 上不再被弃用。（由 Victor Stinner 在 bpo-37412 中贡献。）
- 现在 subprocess.Popen 在某些情况下会使用 os.posix_spawn() 以获得更好的性能。在适用于 Linux 的 Windows 子系统和 QEMU 用户模拟器上，使用 os.posix_spawn() 的 Popen 构造器不会再因为“找不到程序”这样的错误引发异常。而是让子进程失败并返回一个非零的 returncode。（由 Joannah Nanjekye 和 Victor Stinner 在 bpo-35537 中贡献。）
- The *preexec_fn* argument of * subprocess.Popen is no longer compatible with subinterpreters. The use of the parameter in a subinterpreter now raises RuntimeError. (Contributed by Eric Snow in bpo-34651, modified by Christian Heimes in bpo-37951.)
- imap.IMAP4.logout() 方法不会再静默地忽略任意异常。（由 Victor Stinner 在 bpo-36348 中贡献。）
- 函数 platform.popen() 已被移除，它自 Python 3.3 起就已被弃用：请改用 os.popen()。（由 Victor Stinner 在 bpo-35345 中贡献。）
- 当传入多模数据时 statistics.mode() 函数不会再引发异常。它将改为返回在输入数据中遇到的第一个模式。（由 Raymond Hettinger 在 bpo-35892 中贡献。）
- tkinter.ttk.Treeview 类的 selection() 方法不再接受参数。带参数调用该方法来改变选择在 Python 3.6 中已弃用。请使用专门方法例如 selection_set() 来改变选择。（由 Serhiy Storchaka 在 bpo-31508 中贡献。）
- xml.dom.minidom 的 writexml(), toxml() 和 toprettyxml() 方法以及 xml.etree 的 write() 方法现在会保留用户指定的属性顺序。（由 Diego Rojas 和 Raymond Hettinger 在 bpo-34160 中贡献。）
- 附带 'r' 旗标打开的 dbm.dumb 数据库现在将是只读的。如果数据库不存在，附带 'r' 和 'w' 旗标的 dbm.dumb.open() 不会再创建数据库。（由 Serhiy Storchaka 在 bpo-32749 中贡献。）
- 在 XMLParser 的子类中定义的 doctype() 方法将不会再被调用，并将导致发出 RuntimeWarning 而不是 DeprecationWarning。请在目标上定义 doctype() 方法来处理 XML doctype 声明。（由 Serhiy Storchaka 在 bpo-29209 中贡献。）
- 现在当自定义元类未在传给 type.__new__ 的命名空间中提供 __classcell__ 入口时将引发 RuntimeError。在 Python 3.6--3.7 中是则是引发 DeprecationWarning。（由 Serhiy Storchaka 在 bpo-23722 中贡献。）
- cProfile.Profile 类现在可被用作上下文管理器。（由 Scott Sanderson 在 bpo-29235 中贡献。）
- shutil.copyfile(), shutil.copy(), shutil.copy2(), shutil.copytree() 和 shutil.move() 会使用平台专属的”fast-copy” 系统调用（参见 shutil-platform-dependent-efficient-copy-operations 一节）。
- shutil.copyfile() 在 Windows 上的默认缓冲区大小从 16 KiB 改为 1 MiB。
- PyGC_Head 结构已被完全改变。所有接触到该结构的代码都应当被重写。（参见 bpo-33597。）
- PyInterpreterState 结构已被移入”internal” 头文件（特别是 Include/internal/pycore_pystate.h）。不透明的 PyInterpreterState 作为仅有 API（以及稳定版 ABI）的一部分仍然可用。文档指明该结构的任何字段都不是公有的，因此我们希望没人在使用它们。但是，如果你确实依赖其中某一个或更多个私有字段并且没有其他替代选项，则请开一个 BPO 问题。我们将尽力帮助你进行调整（可能包括向公有 API 添加访问器函数）。（参见 bpo-35886。）

- 现在所有平台下的 `mmap.flush()` 方法都会在成功时返回 `None` 并在错误时引发异常。之前它的行为取决于具体平台：Windows 下会在成功时返回非零值；在失败时返回零。Unix 下会在成功时返回零；在失败时引发错误。（由 Berker Peksag 在 bpo-2122 中贡献。）
- `xml.dom.minidom` 和 `xml.sax` 模块默认将不再处理外部实体。（由 Christian Heimes 在 bpo-17239 中贡献。）
- 从只读的 dbm 数据库 (`dbm.dumb`, `dbm.gnu` 或 `dbm.ndbm`) 删除键将会引发 `error` (`dbm.dumb.error`, `dbm.gnu.error` 或 `dbm.ndbm.error`) 而不是 `KeyError`。（由 Xiang Zhang 在 bpo-33106 中贡献。）
- 简化了字面值的 AST。所有常量将被表示为 `ast.Constant` 的实例。实例化旧类 `Num`, `Str`, `Bytes`, `NameConstant` 和 `Ellipsis` 都将返回 `Constant` 的实例。（由 Serhiy Storchaka 在 bpo-32892 中贡献。）
- `expanduser()` 在 Windows 上现在改用 `USERPROFILE` 环境变量而不再使用 `HOME`，后者通常不会为一般用户账户设置。（由 Anthony Sottile 在 bpo-36264 中贡献。）
- 异常 `asyncio.CancelledError` 现在继承自 `BaseException` 而不是 `Exception`。（由 Yury Selivanov 在 bpo-32528 中贡献。）
- 当使用 `asyncio.Task` 的实例时，函数 `asyncio.wait_for()` 现在会正确地等待撤销。在此之前当达到 *timeout* 时，它会被撤销并立即返回。（由 Elvis Pranskevichus 在 bpo-32751 中贡献。）
- 当将'socket' 作为 *name* 形参传入时，函数 `asyncio.BaseTransport.get_extra_info()` 现在会返回一个可安全使用的套接字对象。（由 Yury Selivanov 在 bpo-37027 中贡献。）
- `asyncio.BufferedProtocol` 已经晋级为稳定 API。
- 在 Windows 上对扩展模块的 DLL 依赖以及通过 `ctypes` 加载的 DLL 的解析现在将更为安全。只有系统路径、包含相信 DLL 或 PYD 文件的路径以及通过 `add_dll_directory()` 添加的目录才会被作为加载时依赖的搜索位置。特别地，`PATH` 和当前工作目录将不再被使用，对它们的修改将不再对正常的 DLL 解析产生影响。如果你的应用依赖于这些机制，你应当先检查 `add_dll_directory()`，如果它存在就用它在加载你的库时添加你的 DLL 目录。请注意 Windows 7 用户还需要确保 Windows 更新包 KB2533623 已安装（这一点也会由安装器进行验证）。（由 Steve Dower 在 bpo-36085 中贡献。）
- 关联到 pgen 的头文件和函数在其被纯 Python 实现取代后已被移除。（由 Pablo Galindo 在 bpo-36623 中贡献。）
- `types.CodeType` 在构造器的第二个位置新增了一个形参 (*posonlyargcount*) 以支持在 **PEP 570** 中定义的仅限位置参数。第一个参数 (*argcount*) 现在表示位置参数的总数量 (包括仅限位置参数)。`types.CodeType` 中新增的 `replace()` 方法可用于让代码支持 future 特性。

10.3 C API 中的改变

- `PyCompilerFlags` 结构体添加了一个新的 *cf_feature_version* 字段。它应当被初始化为 `PY_MINOR_VERSION`。该字段默认会被忽略，当且仅当 `PyCF_ONLY_AST` 在 *cf_flags* 中设置旗标时才会被使用。（由 Guido van Rossum 在 bpo-35766 中贡献。）
- `PyEval_ReInitThreads()` 函数已从 C API 中移除。它不应当被显式地调用；请改用 `PyOS_AfterFork_Child()`。（由 Victor Stinner 在 bpo-36728 中贡献。）
- 在 Unix 上，C 扩展不会再被链接到 libpython，但 Android 和 Cygwin 例外。当 Python 被嵌入时，`libpython` 不可使用 `RTLD_LOCAL` 加载，而要改用 `RTLD_GLOBAL`。之前使用 `RTLD_LOCAL` 已经不可能加载未链接到 `libpython` 的 C 扩展了，例如通过 `Modules/Setup` 的 `*shared*` 部分构建的标准库 C 扩展。（由 Victor Stinner 在 bpo-21536 中贡献。）
- 在解析或构建值时（例如 `PyArg_ParseTuple()`, `Py_BuildValue()`, `PyObject_CallFunction()` 等等）使用形如 `#` 的格式而不定义 `PY_SSIZE_T_CLEAN` 现在将会引发 `DeprecationWarning`。它将在 3.10 或 4.0 中被移除。请参阅 arg-parsing 了解详情。（由 Inada Naoki 在 bpo-36381 中贡献。）
- 堆分配类型的实例（例如使用 `PyType_FromSpec()` 创建的实例）会保存一个对其类型对象的引用。提升这些类型对象引用计数的操作已从 `PyType_GenericAlloc()` 移至更低层级的

函数 PyObject_Init() 和 PyObject_INIT()。这使用通过 This makes types created through PyType_FromSpec() 所创建类型的行为与管理代码中的其他类保持一致。

静态分配类型将不受影响。

在大部分情况下，这应该都不会有附带影响。但是，在分配实例后手动提升引用计数的类型（也许是为了绕过漏洞）现在可能永远不会被销毁。要避免这种情况，这些类需要在实例撤销分配期间在类型对象上调用 Py_DECREF。

要正确地将这些类型移植到 3.8，请应用以下修改：

- 在分配实例之后在类型对象上移除 Py_INCREF —— 如果有的话。这可以发生在调用 PyObject_New(), PyObject_NewVar(), PyObject_GC_New(), PyObject_GC_NewVar() 或任何其他使用 PyObject_Init() 或 PyObject_INIT() 的自定义分配器之后。

  示例:

```
static foo_struct *
foo_new(PyObject *type) {
    foo_struct *foo = PyObject_GC_New(foo_struct, (PyTypeObject *) type);
    if (foo == NULL)
        return NULL;
#if PY_VERSION_HEX < 0x03080000
    // Workaround for Python issue 35810; no longer necessary in Python 3.8
    PY_INCREF(type)
#endif
    return foo;
}
```

- 确保所有堆分配类型的自定义 tp_dealloc 函数会减少类型的引用计数。

  示例:

```
static void
foo_dealloc(foo_struct *instance) {
    PyObject *type = Py_TYPE(instance);
    PyObject_GC_Del(instance);
#if PY_VERSION_HEX >= 0x03080000
    // This was not needed before Python 3.8 (Python issue 35810)
    Py_DECREF(type);
#endif
}
```

（由 Eddie Elizondo 在 bpo-35810 中贡献。）

- Py_DEPRECATED() 宏已经针对 MSVC 实现。这个宏现在必须放在符号名称之前。

  示例:

```
Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);
```

（由 Zackery Spytz 在 bpo-33407 中贡献。）

- 解释器将不再假装支持跨发布版本的扩展类型二进制兼容性。由第三方扩展模块所导出的 PyTypeObject 应该具有当前 Python 版本所要求的所有空位，包括 tp_finalize (Py_TPFLAGS_HAVE_FINALIZE 不会再在读取 tp_finalize 之前被检查)。

（由 Antoine Pitrou 在 bpo-32388 中贡献。）

- PyCode_New() 在第二个位置添加了新的形参 (*posonlyargcount*) 以支持 **PEP 570**，指明仅限位置参数的数量。

- 函数 PyNode_AddChild() 和 PyParser_AddToken() 现在接受两个额外的 int 参数 *end_lineno* 和 *end_col_offset*。

- 允许 MinGW 工具直接链接到 `python38.dll` 的 `libpython38.a` 文件已不再包含于标准的 Windows 分发包中。如果你需要此文件，可使用 `gendef` 和 `dlltool` 工具来生成它，这些工具是 MinGW binutils 包的一部分：

```
gendef - python38.dll > tmp.def
dlltool --dllname python38.dll --def tmp.def --output-lib libpython38.a
```

  已安装的 `pythonXY.dll` 所在位置将取决于安装选项以及 Windows 的版本和语言。请参阅 using-on-windows 了解更多信息。该结果库应当放在与 `pythonXY.lib` 相同的目录下，这通常是你的 Python 安装路径下的 `libs` 目录。

  (由 Steve Dower 在 bpo-37351 中贡献。)

10.4 CPython 字节码的改变

- 解释器循环已通过将块堆栈展开逻辑移入编译器获得了简化。编译器现在会发出显式指令来调整值堆栈并为 `break`, `continue` 和 `return` 调用清除代码。

  移除了操作码 `BREAK_LOOP`, `CONTINUE_LOOP`, `SETUP_LOOP` 和 `SETUP_EXCEPT`。添加了新的操作码 `ROT_FOUR`, `BEGIN_FINALLY`, `CALL_FINALLY` 和 `POP_FINALLY`。修改了 `END_FINALLY` 和 `WITH_CLEANUP_START` 的行为。

  (由 Mark Shannon, Antoine Pitrou 和 Serhiy Storchaka 在 bpo-17611 中贡献。)

- 添加了新的操作码 `END_ASYNC_FOR` 用于处理当等待 `async for` 循环的下一项时引发的异常。(由 Serhiy Storchaka 在 bpo-33041 中贡献。)

- `MAP_ADD` 现在会预期值为栈的第一个元素而键为第二个元素。作出此改变以使得字典推导式能如 **PEP 572** 所提议的那样，键总是会在值之前被求值。(由 Jörn Heissler 在 bpo-35224 中贡献。)

10.5 演示和工具

添加了一个检测脚本用于对访问变量的不同方式进行计时: `Tools/scripts/var_access_benchmark.py`。(由 Raymond Hettinger 在 bpo-35884 中贡献。)

以下是自 Python 3.3 以来性能提升情况的总结:

```
Python version                       3.3     3.4     3.5     3.6     3.7     3.8
--------------                       ---     ---     ---     ---     ---     ---

Variable and attribute read access:
    read_local                       4.0     7.1     7.1     5.4     5.1     3.9
    read_nonlocal                    5.3     7.1     8.1     5.8     5.4     4.4
    read_global                     13.3    15.5    19.0    14.3    13.6     7.6
    read_builtin                    20.0    21.1    21.6    18.5    19.0     7.5
    read_classvar_from_class        20.5    25.6    26.5    20.7    19.5    18.4
    read_classvar_from_instance     18.5    22.8    23.5    18.8    17.1    16.4
    read_instancevar                26.8    32.4    33.1    28.0    26.3    25.4
    read_instancevar_slots          23.7    27.8    31.3    20.8    20.8    20.2
    read_namedtuple                 68.5    73.8    57.5    45.0    46.8    18.4
    read_boundmethod                29.8    37.6    37.9    29.6    26.9    27.7

Variable and attribute write access:
    write_local                      4.6     8.7     9.3     5.5     5.3     4.3
    write_nonlocal                   7.3    10.5    11.1     5.6     5.5     4.7
    write_global                    15.9    19.7    21.2    18.0    18.0    15.8
    write_classvar                  81.9    92.9    96.0   104.6   102.1    39.2
    write_instancevar               36.4    44.6    45.8    40.0    38.9    35.5
    write_instancevar_slots         28.7    35.6    36.1    27.3    26.6    25.7

Data structure read access:
```

(下页继续)

(繼續上一頁)

```
    read_list                      19.2     24.2     24.5     20.8     20.8     19.0
    read_deque                     19.9     24.7     25.5     20.2     20.6     19.8
    read_dict                      19.7     24.3     25.7     22.3     23.0     21.0
    read_strdict                   17.9     22.6     24.3     19.5     21.2     18.9

Data structure write access:
    write_list                     21.2     27.1     28.5     22.5     21.6     20.0
    write_deque                    23.8     28.7     30.1     22.7     21.8     23.5
    write_dict                     25.9     31.4     33.3     29.3     29.2     24.7
    write_strdict                  22.9     28.4     29.9     27.5     25.2     23.1

Stack (or queue) operations:
    list_append_pop               144.2     93.4    112.7     75.4     74.2     50.8
    deque_append_pop               30.4     43.5     57.0     49.4     49.2     42.5
    deque_append_popleft           30.8     43.7     57.3     49.7     49.7     42.8

Timing loop:
    loop_overhead                   0.3      0.5      0.6      0.4      0.3      0.3
```

基准测试是在 Intel® Core™ i7-4960HQ 处理器 上运行从 python.org 获取的 macOS 64 位版得到的数据。基准测试脚本显示时间以纳秒为单位。

11 Python 3.8.1 中的重要变化

出于重要的安全性考量，`asyncio.loop.create_datagram_endpoint()` 的 *reuse_address* 形参不再被支持。这是由 UDP 中的套接字选项 `SO_REUSEADDR` 的行为导致的。更多细节请参阅 `loop.create_datagram_endpoint()` 的文档。(由 Kyle Stanley, Antoine Pitrou 和 Yury Selivanov 在 bpo-37228 中贡献。。)

12 Python 3.8.2 中的重要变化

修复了 `shutil.copytree()` 的 `ignore` 回调中的回归问题。现在该参数的类型已改回 str 和 List[str]。(由 Manuel Barkhau 和 Giampaolo Rodola 在 bpo-39390 中贡献。)

索引

非字母

H

P

U