

API 和 ABI 版本管理

构建时版本常量

CPython 在下列宏中公开其版本号。请注意这对应于 **构建** 所用的版本代码。请查看 [Py_Version](#) 获取在 **运行时** 所用的版本。

请参阅 [C API 的稳定性](#) 查看跨版本的 API 和 ABI 稳定情。

PY_MAJOR_VERSION

3 (3.4.1a2 中的第一段)。

PY_MINOR_VERSION

4 (3.4.1a2 中的第二段)。

PY_MICRO_VERSION

1 (3.4.1a2 中第三段的数字)。

PY_RELEASE_LEVEL

a (3.4.1a2 中第3段的字母)。可能为 0xA 即 alpha, 0xB 即 beta, 0xC 即 release candidate 或 0xF 即 final。

PY_RELEASE_SERIAL

2 (3.4.1a2 中的末尾数字)。零代表最终发布版。

PY_VERSION_HEX

Python 版本号被编码为一个整数。请查看 [Py_PACK_FULL_VERSION\(\)](#) 了解编码细节。

可将其用于数字比较，例如 `#if PY_VERSION_HEX >= ...`。

运行时版本

`const unsigned long Py_Version`

属于 [稳定ABI](#) 自 3.11 版起

Python 运行时版本号被编码为一个整数常量。请查看 [Py_PACK_FULL_VERSION\(\)](#) 了解编码细节。这包含了在运行时使用的 Python 版本。

可将其用于数字比较，例如 `if (Py_Version >= ...)`。

Added in version 3.11.

比特位打包宏

uint32_t **Py_PACK_FULL_VERSION**(int major, int minor, int micro, int release_level, int release_serial)

属于 [稳定ABI](#) 自 3.14 版起

返回给定的版本，编码为一个具有如下结构的 32 位整数：

参数	位编号	位掩码	位移	示例值	
				3.4.1a2	3.10.0
major	8	0xFF000000	24	0x03	0x03
minor	8	0x00FF0000	16	0x04	0x0A
micro	8	0x0000FF00	8	0x01	0x00
release_level	4	0x000000F0	4	0xA	0xF
release_serial	4	0x0000000F	0	0x2	0x0

例如:

版本	Py_PACK_FULL_VERSION 参数	已编码版本
3.4.1a2	(3, 4, 1, 0xA, 2)	0x030401a2
3.10.0	(3, 10, 0, 0xF, 0)	0x030a00f0

参数中超范围的比特位将被忽略。也就是说，该宏可以被定义为：

```
#ifndef Py_PACK_FULL_VERSION
#define Py_PACK_FULL_VERSION(X, Y, Z, LEVEL, SERIAL) ( \
    (((X) & 0xff) << 24) | \
    (((Y) & 0xff) << 16) | \
    (((Z) & 0xff) << 8) | \
    (((LEVEL) & 0xf) << 4) | \
    (((SERIAL) & 0xf) << 0))
#endif
```

Py_PACK_FULL_VERSION 本质上是一个宏，主要在 #if 指令中使用，但也可作为导出的函数使用。

Added in version 3.14.

uint32_t **Py_PACK_VERSION**(int major, int minor)

属于 [稳定ABI](#) 自 3.14 版起

等价于 Py_PACK_FULL_VERSION(major, minor, 0, 0, 0)。其结果不与任何 Python 发布版对应，但在数字比较中很有用处。

Added in version 3.14.