引用计数

本节介绍的函数和宏被用于管理 Python 对象的引用计数。

Py_ssize_t **Py_REFCNT**(PyObject *o)

属于 稳定 ABI 自 3.14 版起.

获取 Python 对象 *o* 的引用计数。

请注意返回的值可能并不真正反映实际持有的对象引用数。 例如，有些对象属于 immortal 对象并具有并不反映实际引用数的非常高的 refcount 值。 因此，除了 0 或 1 这两个值，不要依赖返回值的准确性。

使用 `Py_SET_REFCNT()` 函数来设置一个对象引用计数。

> **备注:** 在 Python 的 自由线程 构建版中，返回 1 并不足以确定是否能安全地将 *o* 视为不可被其他线程访问。 对于此场景请改用 `PyUnstable_Object_IsUniquelyReferenced()`。
>
> 另请参阅 `PyUnstable_Object_IsUniqueReferencedTemporary()` 函数。

在 3.10 版本发生变更: `Py_REFCNT()` 被改为内联的静态函数。

在 3.11 版本发生变更: 形参类型不再是 `const PyObject*`。

void **Py_SET_REFCNT**(PyObject *o, Py_ssize_t refcnt)

将对象 *o* 的引用计数器设为 *refcnt*。

在 启用自由线程的 Python 编译版 中，如果 *refcnt* 大于 `UINT32_MAX`，该对象将被设为 immortal 对象。

此函数对 immortal 对象没有效果。

Added in version 3.9.

在 3.12 版本发生变更: 永生对象不会被修改。

void **Py_INCREF**(PyObject *o)

表示为对象 *o* 获取一个新的 strong reference，指明该对象正在被使用且不应被销毁。

此函数对 immortal 对象没有效果。

此函数通常被用来将 borrowed reference 原地转换为 strong reference。 `Py_NewRef()` 函数可被用来创建新的 strong reference。

当对象使用完毕后，可调用 `Py_DECREF()` 来释放它。

此对象必须不为 `NULL`；如果你不能确定它不为 `NULL`，请使用 `Py_XINCREF()`。

不要预期此函数会以任何方式实际地改变 *o*。 至少对 **某些对象** 来说，此函数将没有任何效果。

> *在 3.12 版本发生变更:* 永生对象不会被修改。

`void` **Py_XINCREF**(`PyObject` *o)

与 `Py_INCREF()` 类似，但对象 *o* 可以为 `NULL`，在这种情况下此函数将没有任何效果。

另请参阅 `Py_XNewRef()`。

`PyObject` ***Py_NewRef**(`PyObject` *o)

属于 稳定 ABI 自 3.10 版起.

为对象创建一个新的 strong reference: 在 *o* 上调用 `Py_INCREF()` 并返回对象 *o*。

当不再需要这个 strong reference 时，应当在其上调用 `Py_DECREF()` 来释放引用。

对象 *o* 必须不为 `NULL`；如果 *o* 可以为 `NULL` 则应改用 `Py_XNewRef()`。

例如:

```
Py_INCREF(obj);
self->attr = obj;
```

可以写成:

```
self->attr = Py_NewRef(obj);
```

另请参阅 `Py_INCREF()`。

> *Added in version 3.10.*

`PyObject` ***Py_XNewRef**(`PyObject` *o)

属于 稳定 ABI 自 3.10 版起.

类似于 `Py_NewRef()`，但对象 *o* 可以为 NULL。

如果对象 *o* 为 `NULL`，该函数也将返回 `NULL`。

> *Added in version 3.10.*

`void` **Py_DECREF**(`PyObject` *o)

释放一个指向对象 *o* 的 strong reference，表明该引用不再被使用。

此函数对 immortal 对象没有效果。

当最后一个 strong reference 被释放时 (即对象的引用计数变为 0)，将会唤起该对象所属类型的 deallocation 函数 (它必须不为 `NULL`)。

此函数通常被用于在退出作用域之前删除一个 strong reference。

此对象必须不为 `NULL`；如果你不能确定它不为 `NULL`，请使用 `Py_XDECREF()`。

不要预期此函数会以任何方式实际地改变 *o*。 至少对 **某些对象** 来说，此函数将没有任何效果。

警告： 释放函数会导致任意 Python 代码被唤起（例如当一个带有 `__del__()` 方法的类实例被释放时就是如此）。 虽然这些代码中的异常不会被传播，但被执行的代码能够自由访问所有 Python 全局变量。 这意味着在调用 `Py_DECREF()` 之前任何可通过全局变量获取的对象都应该处于完好的状态。 例如，从一个列表中删除对象的代码应该将被删除对象的引用拷贝到一个临时变量中，更新列表数据结构，然后再为临时变量调用 `Py_DECREF()`。

> *在 3.12 版本发生变更:* 永生对象不会被修改。

`void` **Py_XDECREF**(`PyObject` *o)

与 `Py_DECREF()` 类似，但对象 *o* 可以为 `NULL`，在这种情况下此函数将没有任何效果。 来自 `Py_DECREF()` 的警告同样适用于此处。

`void` **Py_CLEAR**(`PyObject` *o)

释放一个指向对象 *o* 的 strong reference。 对象可以为 `NULL`，在此情况下该宏将没有任何效果；在其他情况下其效果与 `Py_DECREF()` 相同，区别在于其参数也会被设为 `NULL`。 针对 `Py_DECREF()` 的警告不适用于所传递的对象，因为该宏会细心地使用一个临时变量并在释放引用之前将参数设为 `NULL`。

当需要释放指向一个在垃圾回收期间可能被会遍历的对象的引用时使用该宏是一个好主意。

> *在 3.12 版本发生变更:* 该宏参数现在只会被求值一次。 如果该参数具有附带影响，它们将不会再被复制。

`void` **Py_IncRef**(`PyObject` *o)

属于 稳定 ABI.

表示获取一个指向对象 *o* 的新 strong reference。 `Py_XINCREF()` 的函数版本。 它可被用于 Python 的运行时动态嵌入。

`void` **Py_DecRef**(`PyObject` *o)

属于 稳定 ABI.

释放一个指向对象 *o* 的 strong reference。 `Py_XDECREF()` 的函数版本。 它可被用于 Python 的运行时动态嵌入。

Py_SETREF(`dst, src`)

该宏可安全地释放一个指向对象 *dst* 的 strong reference，并将 *dst* 设为 *src*。

在 `Py_CLEAR()` 的情况中，这样"直观"的代码可能会是致命的:

```
Py_DECREF(dst);
dst = src;
```

安全的方式是这样:

```
Py_SETREF(dst, src);
```

这样使得在释放对旧 *dst* 值的引用 *之前* 将 *dst* 设为 *src*，从而让任何因 *dst* 被去除而触发的代码不再相信 *dst* 指向一个有效的对象。

> *Added in version 3.6.*

> *在 3.12 版本发生变更:* 该宏参数现在只会被求值一次。 如果某个参数具有附带影响，它们将不会再被复制。

Py_XSETREF(dst, src)

使用 Py_XDECREF() 代替 Py_DECREF() 的 Py_SETREF 宏的变种。

> *Added in version 3.6.*

> *在 3.12 版本发生变更:* 该宏参数现在只会被求值一次。 如果某个参数具有附带影响，它们将不会再被复制。