🄵釋 (annotation) 最佳實踐

發🄵 *3.12.3*

Guido van Rossum and the Python development team

5 月 02, 2024

Python Software Foundation
Email: docs@python.org

Contents

作者
Larry Hastings

摘要

本文件旨在封裝 (encapsulate) 使用🄵釋字典 (annotations dicts) 的最佳實踐。如果你寫 Python 程式碼🄵在調查 Python 物件上的 `__annotations__`，我們鼓勵你遵循下面描述的准則。

本文件分🄵四個部分：在 Python 3.10 及更高版本中存取物件🄵釋的最佳實踐、在 Python 3.9 及更早版本中存取物件🄵釋的最佳實踐、適用於任何 Python 版本 `__annotations__` 的最佳實踐，以及 `__annotations__` 的奇🄵之處。

請注意，本文件是特🄵🄵明 `__annotations__` 的使用，而非如何使用🄵釋。如果你正在尋找如何在你的程式碼中使用「型🄵提示 (type hint)」的資訊，請查🄵模組 (module) `typing`。

1 在 Python 3.10 及更高版本中存取物件的🅵釋字典

Python 3.10 在標准函式庫中新增了一個新函式：`inspect.get_annotations()`。在 Python 3.10 及更高版本中，呼叫此函式是存取任何支援🅵釋的物件的🅵釋字典的最佳實踐。此函式也可以🅵你「取消字串化 (un-stringize)」字串化🅵釋。

若由於某種原因 `inspect.get_annotations()` 對你的場合不可行，你可以手動存取 `__annotations__` 資料成員。Python 3.10 中的最佳實踐也已經改變：從 Python 3.10 開始，保證 `o.__annotations__` 始終適用於 Python 函式、類🅵 (class) 和模組。如果你確定正在檢查的物件是這三個特定物件之一，你可以簡單地使用 `o.__annotations__` 來取得物件的🅵釋字典。

但是，其他型🅵的 callable（可呼叫物件）（例如，由 `functools.partial()` 建立的 callable）可能🅵有定義 `__annotations__` 屬性 (attribute)。當存取可能未知的物件的 `__annotations__` 時，Python 3.10 及更高版本中的最佳實踐是使用三個參數呼叫 `getattr()`，例如 `getattr(o, '__annotations__', None)`。

在 Python 3.10 之前，存取未定義🅵釋但具有🅵釋的父類🅵的類🅵上的 `__annotations__` 將傳回父類🅵的 `__annotations__`。在 Python 3.10 及更高版本中，子類🅵的🅵釋將會是一個空字典。

2 在 Python 3.9 及更早版本中存取物件的🅵釋字典

在 Python 3.9 及更早版本中，存取物件的🅵釋字典比新版本🅵雜得多。問題出在於這些舊版 Python 中有設計缺陷，特🅵是與類🅵🅵釋有關的設計缺陷。

存取其他物件（如函式、其他 callable 和模組）的🅵釋字典的最佳實踐與 3.10 的最佳實踐相同，假設你🅵有呼叫 `inspect.get_annotations()`：你應該使用三個：參數 `getattr()` 來存取物件的 `__annotations__` 屬性。

不幸的是，這不是類🅵的最佳實踐。問題是，由於 `__annotations__` 在類🅵上是選填的 (optional)，🅵且因🅵類🅵可以從其基底類🅵 (base class) 繼承屬性，所以存取類🅵的 `__annotations__` 屬性可能會無意中回傳基底類🅵的🅵釋字典。舉例來🅵：

```
class Base:
    a: int = 3
    b: str = 'abc'

class Derived(Base):
    pass

print(Derived.__annotations__)
```

這將印出 (print) 來自 `Base` 的🅵釋字典，而不是 `Derived`。

如果你正在檢查的物件是一個類🅵 (`isinstance(o, type)`)，你的程式碼將必須有一個單獨的程式碼路徑。在這種情🅵下，最佳實踐依賴 Python 3.9 及之前版本的實作細節 (implementation detail)：如果一個類🅵定義了🅵釋，它們將儲存在該類🅵的 `__dict__` 字典中。由於類🅵可能定義了🅵釋，也可能🅵有定義，因此最佳實踐是在類🅵字典上呼叫 `get` 方法。

總而言之，以下是一些範例程式碼，可以安全地存取 Python 3.9 及先前版本中任意物件上的 `__annotations__` 屬性：

```
if isinstance(o, type):
    ann = o.__dict__.get('__annotations__', None)
else:
    ann = getattr(o, '__annotations__', None)
```

運行此程式碼後，`ann` 應該是字典或 `None`。我們鼓勵你在進一步檢查之前使用 `isinstance()` 仔細檢查 `ann` 的型🅵。

請注意，某些外來 (exotic) 或格式錯誤 (malform) 的型🅵物件可能🅵有 `__dict__` 屬性，因此🅵了額外的安全，你可能還希望使用 `getattr()` 來存取 `__dict__`。

3 手動取消字串化🄵釋

在某些🄵釋可能被「字串化」的情🄵下，🄵且你希望評估這些字串以🄵生它們表示的 Python 值，最好呼叫 `inspect.get_annotations()` 來🄵你完成這項工作。

如果你使用的是 Python 3.9 或更早版本，或者由於某種原因你無法使用 `inspect.get_annotations()`，則需要🄵🄵其邏輯。我們鼓勵你檢查目前 Python 版本中 `inspect.get_annotations()` 的實作🄵遵循類似的方法。

簡而言之，如果你希望評估任意物件 `o` 上的字串化🄵釋：

- 如果 `o` 是一個模組，則在呼叫 `eval()` 時使用 `o.__dict__` 作🄵全域變數。
- 如果 `o` 是一個類🄵，當呼叫 `eval()` 時，則使用 `sys.modules[o.__module__].__dict__` 作🄵全域變數，使用 `dict(vars(o))` 作🄵區域變數。
- 如果 `o` 是使用 `functools.update_wrapper()`、`functools.wraps()` 或 `functools.partial()` 包裝的 callable，請依據需求，透過存取 `o.__wrapped__` 或 `o.func` 來🄵代解開它，直到找到根解包函式。
- 如果 `o` 是 callable（但不是類🄵），則在呼叫 `eval()` 時使用 `o.__globals__` 作🄵全域變數。

然而，🄵非所有用作🄵釋的字串值都可以透過 `eval()` 成功轉🄵🄵 Python 值。理論上，字串值可以包含任何有效的字串，🄵且在實踐中，型🄵提示存在有效的用例，需要使用特定「無法」評估的字串值進行🄵釋。例如：

- 在 Python 3.10 支援 **PEP 604** 聯合型🄵 (union type) `|` 之前使用它。
- Runtime 中不需要的定義，僅在 `typing.TYPE_CHECKING` 🄵 true 時匯入。

如果 `eval()` 嘗試計算這類型的值，它將失敗🄵引發例外。因此，在設計使用🄵釋的函式庫 API 時，建議僅在呼叫者 (caller) 明確請求時嘗試評估字串值。

4 任何 Python 版本中 `__annotations__` 的最佳實踐

- 你應該避免直接指派給物件的 `__annotations__` 成員。讓 Python 管理設定 `__annotations__`。
- 如果你直接指派給物件的 `__annotations__` 成員，則應始終將其設🄵 `dict` 物件。
- 如果直接存取物件的 `__annotations__` 成員，則應在嘗試檢查其🄵容之前確保它是字典。
- 你應該避免修改 `__annotations__` 字典。
- 你應該避免🄵除物件的 `__annotations__` 屬性。

5 `__annotations__` 奇🄵之處

在 Python 3 的所有版本中，如果🄵有在該物件上定義🄵釋，則函式物件會延遲建立 (lazy-create) 🄵釋字典。你可以使用 `del fn.__annotations__` 🄵除 `__annotations__` 屬性，但如果你隨後存取 `fn.__annotations__`，該物件將建立一個新的空字典，它將作🄵🄵釋儲存🄵傳回。在函式延遲建立🄵釋字典之前🄵除函式上的🄵釋將🄵出 `AttributeError`；連續兩次使用 `del fn.__annotations__` 保證總是🄵出 `AttributeError`。

上一段的所有🄵容也適用於 Python 3.10 及更高版本中的類🄵和模組物件。

在 Python 3 的所有版本中，你可以將函式物件上的 `__annotations__` 設定🄵 `None`。但是，隨後使用 `fn.__annotations__` 存取該物件上的🄵釋將根據本節第一段的🄵容延遲建立一個空字典。對於任何 Python 版本中的模組和類🄵來🄵，情🄵🄵非如此；這些物件允許將 `__annotations__` 設定🄵任何 Python 值，🄵且將保留設定的任何值。

如果 Python 🄵你字串化你的🄵釋（使用 `from __future__ import annotations`），🄵且你指定一個字串作🄵🄵釋，則該字串本身將被引用。實際上，🄵釋被引用了兩次。例如：

```python
from __future__ import annotations
def foo(a: "str"): pass

print(foo.__annotations__)
```

這會印出 `{'a': "'str'"}`。這不應該被認🅵是一個「奇🅵的事」，他在這🅵被簡單提及，因🅵他可能會讓人意想不到。

索引

P