10.12 导入模块的同时修改模块¶

问题¶

你想给某个已存在模块中的函数添加装饰器。 不过，前提是这个模块已经被导入并且被使用过。

解决方案¶

这里问题的本质就是你想在模块被加载时执行某个动作。 可能是你想在一个模块被加载时触发某个回调函数来通知你。

这个问题可以使用10.11小节中同样的导入钩子机制来实现。下面是一个可能的方案：

```
# postimport.py
import importlib
import sys
from collections import defaultdict

_post_import_hooks = defaultdict(list)

class PostImportFinder:
    def __init__(self):
        self._skip = set()

    def find_module(self, fullname, path=None):
        if fullname in self._skip:
            return None
        self._skip.add(fullname)
        return PostImportLoader(self)

class PostImportLoader:
    def __init__(self, finder):
        self._finder = finder

    def load_module(self, fullname):
        importlib.import_module(fullname)
        module = sys.modules[fullname]
        for func in _post_import_hooks[fullname]:
            func(module)
        self._finder._skip.remove(fullname)
        return module

def when_imported(fullname):
    def decorate(func):
        if fullname in sys.modules:
            func(sys.modules[fullname])
        else:
            _post_import_hooks[fullname].append(func)
        return func
    return decorate

sys.meta_path.insert(0, PostImportFinder())
```

这样，你就可以使用 `when_imported()` 装饰器了，例如：

```
>>> from postimport import when_imported
```

```
>>> from postimport import when_imported
>>> @when_imported('threading')
... def warn_threads(mod):
...     print('Threads? Are you crazy?')
...
>>>
>>> import threading
Threads? Are you crazy?
>>>
```

作为一个更实际的例子，你可能想在已存在的定义上面添加装饰器，如下所示：

```
from functools import wraps
from postimport import when_imported

def logged(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Calling', func.__name__, args, kwargs)
        return func(*args, **kwargs)
    return wrapper

# Example
@when_imported('math')
def add_logging(mod):
    mod.cos = logged(mod.cos)
    mod.sin = logged(mod.sin)
```

讨论¶

本节技术依赖于10.11小节中讲述过的导入钩子，并稍作修改。

`@when_imported` 装饰器的作用是注册在导入时被激活的处理器函数。 该装饰器检查sys.modules来查看模块是否真的已经被加载了。 如果是的话，该处理器被立即调用。不然，处理器被添加到 `_post_import_hooks` 字典中的一个列表中去。 `_post_import_hooks` 的作用就是收集所有的为每个模块注册的处理器对象。 一个模块可以注册多个处理器。

要让模块导入后触发添加的动作， `PostImportFinder` 类被设置为sys.meta_path第一个元素。 它会捕获所有模块导入操作。

本节中的 `PostImportFinder` 的作用并不是加载模块，而是自带导入完成后触发相应的动作。 实际的导入被委派给位于sys.meta_path中的其他查找器。 `PostImportLoader` 类中的 `imp.import_module()` 函数被递归的调用。 为了避免陷入无线循环， `PostImportFinder` 保持了一个所有被加载过的模块集合。 如果一个模块名存在就会直接被忽略掉。

当一个模块被 `imp.import_module()` 加载后， 所有在_post_import_hooks被注册的处理器被调用，使用新加载模块作为一个参数。

有一点需要注意的是本机不适用于那些通过 `imp.reload()` 被显式加载的模块。 也就是说，如果你加载一个之前已被加载过的模块，那么导入处理器将不会再被触发。 另外，要是你从sys.modules中删除模块然后再重新导入，处理器又会再一次触发。

更多关于导入后钩子信息请参考 PEP 369.