由浅入深：Python 中如何实现自动导入缺失的库?

Python开发者 2019-12-30

以下文章来源于Python猫 ，作者豌豆花下猫

Python猫

分享Python进阶、Python哲学、文章翻译、资源工具等内容

（给Python开发者加星标，提升Python技能）

> 来源： 豌豆花下猫

在写 Python 项目的时候，我们可能经常会遇到导入模块失败的错误：`ImportError: No module named 'xxx'`或者`ModuleNotFoundError: No module named 'xxx'`。

导入失败问题，通常分为两种：一种是导入自己写的模块（即以 .py 为后缀的文件），另一种是导入三方库。本文主要讨论第二种情况，今后有机会，我们再详细讨论其它的相关话题。

解决导入 Python 库失败的问题，其实关键是在运行环境中装上缺失的库（注意是否是虚拟环境），或者使用恰当的替代方案。这个问题又分为三种情况：

一、单个模块中缺失的库

在编写代码的时候，如果我们需要使用某个三方库（如 requests），但不确定实际运行的环境是否装了它，那么可以这样：

```
try:
    import requests
except ImportError:
    import os
    os.system('pip install requests')
    import requests
```

这样写的效果是，如果找不到 requests 库，就先安装，再导入。

在某些开源项目中，我们可能还会看到如下的写法（以 json 为例）：

```
try:
```

```
    import simplejson as json
except ImportError:
    import json
```

这样写的效果是，优先导入三方库 simplejson，如果找不到，那就使用内置的标准库 json。

这种写法的好处是不需要导入额外的库，但它有个缺点，即需要保证那两个库在使用上是兼容的，如果在标准库中找不到替代的库，那就不可行了。

如果真找不到兼容的标准库，也可以自己写一个模块（如 my_json.py），实现想要的东西，然后在 except 语句中导入它。

```
try:
    import simplejson as json
except ImportError:
    import my_json as json
```

二、整个项目中缺失的库

以上的思路是针对开发中的项目，但是它有几个不足：1、在代码中对每个可能缺失的三方库都 pip install，并不可取；2、某个三方库无法被标准库或自己手写的库替代，该怎么办？3、已成型的项目，不允许做这些修改怎么办？

所以这里的问题是：**有一个项目，想要部署到新的机器上，它涉及很多三方库，但是机器上都没有预装，该怎么办？**

对于一个合规的项目，按照约定，通常它会包含一个“**requirements.txt** ”文件，记录了该项目的所有依赖库及其所需的版本号。这是在项目发布前，使用命令`pip freeze > requirements.txt` 生成的。

使用命令`pip install -r requirements.txt` （在该文件所在目录执行，或在命令中写全文件的路径），就能自动把所有的依赖库给装上。

但是，如果项目不合规，或者由于其它倒霉的原因，我们没有这样的文件，又该如何是好？

一个笨方法就是，把项目跑起来，等它出错，遇到一个导库失败，就手动装一个，然后再跑一遍项目，遇到导库失败就装一下，如此循环……（此处省略 1 万句脏话）……

三、自动导入任意缺失的库

有没有一种更好的可以自动导入缺失的库的方法呢？

在不修改原有的代码的情况下，在不需要“requirements.txt”文件的情况下，有没有办法自动导入所需要的库呢？

当然有！先看看效果：

```
C:\Users\yunpoyue\PycharmProjects\untitled>python
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tornado
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'tornado'
>>> import autoinstall
>>> import tornado
Installing tornado
Collecting tornado
  Using cached https://files.pythonhosted.org/packages/35/0b/bcd855847d58600627b17c64535567a871731e81f0f75d2065f72fe58671/tornado-6.0.3-cp36-cp36m-win_amd64.whl
Installing collected packages: tornado
Successfully installed tornado-6.0.3
```

我们以 tornado 为例，第一步操作可看出，我们没有装过 tornado，经过第二步操作后，再次导入 tornado 时，程序会帮我们自动下载并安装好 tornado，所以不再报错。

autoinstall 是我们手写的模块，代码如下：

```
# 以下代码在 python 3.6.1 版本验证通过
import sys
import os
from importlib import import_module


class AutoInstall():
    _loaded = set()

    @classmethod
    def find_spec(cls, name, path, target=None):
            if path is None and name not in cls._loaded:
                cls._loaded.add(name)
                print("Installing", name)
                try:
                    result = os.system('pip install {}'.format(name))
                    if result == 0:
                        return import_module(name)
                except Exception as e:
                    print("Failed", e)
            return None

sys.meta_path.append(AutoInstall)
```

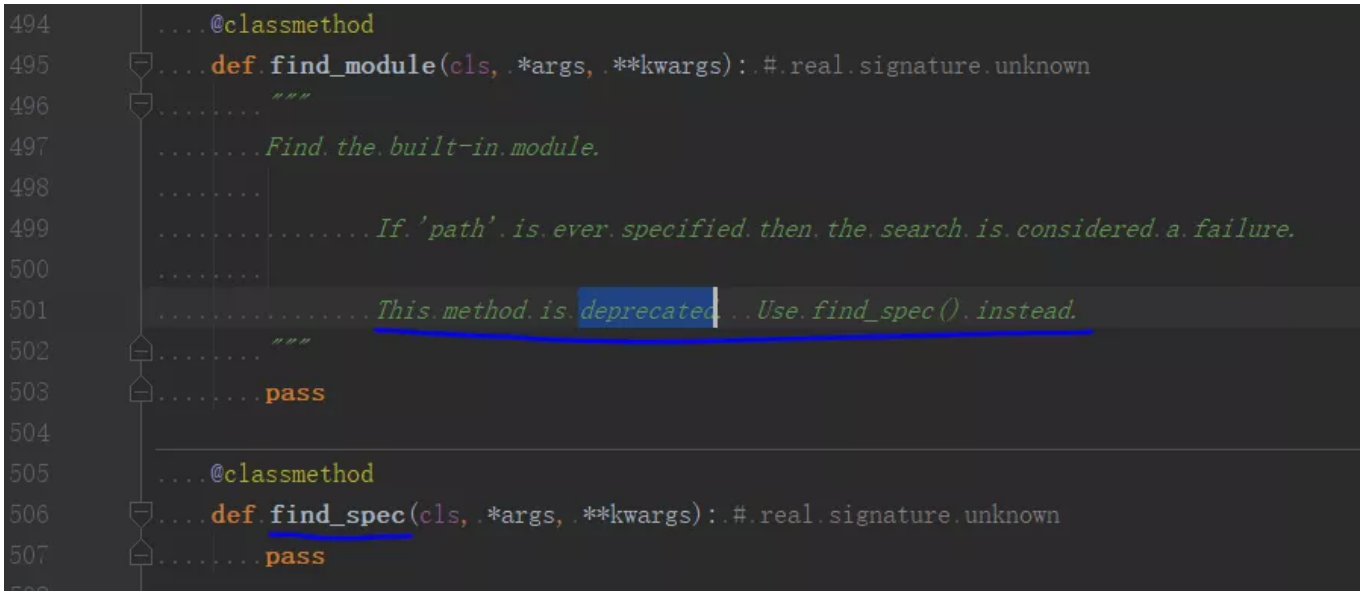
这段代码中使用了`sys.meta_path`，我们先打印一下，看看它是个什么东西?

```
>>> import sys
>>> sys.meta_path
[<class '_frozen_importlib.BuiltinImporter'>, <class '_frozen_importlib.FrozenImporter'>,
 <class '_frozen_importlib_external.PathFinder'>, <class 'autoinstall.AutoInstall'>]
>>>
```

Python 3 的 import 机制在查找过程中，大致顺序如下:

- 在 sys.modules 中查找，它缓存了所有已导入的模块
- 在 sys.meta_path 中查找，它支持自定义的加载器
- 在 sys.path 中查找，它记录了一些库所在的目录名
- 若未找到，抛出`ImportError`异常

其中要注意，sys.meta_path 在不同的 Python 版本中有所差异，比如它在 Python 2 与 Python 3 中差异很大；在较新的 Python 3 版本（3.4+）中，自定义的加载器需要实现`find_spec`方法，而早期的版本用的则是`find_module`。



```
    @classmethod
    def find_module(cls, *args, **kwargs): # real signature unknown
        """
        Find the built-in module.

                If 'path' is ever specified then the search is considered a failure.

                This method is deprecated.  Use find_spec() instead.
        """
        pass

    @classmethod
    def find_spec(cls, *args, **kwargs): # real signature unknown
        pass
```

以上代码是一个自定义的类库加载器 AutoInstall，可以实现自动导入三方库的目的。需要说明一下，这种方法会“劫持”所有新导入的库，破坏原有的导入方式，因此也可能出现一些奇奇怪怪的问题，敬请留意。

sys.meta_path 属于 Python 探针的一种运用。探针，即 `import hook`，是 Python 几乎不受人关注的机制，但它可以做很多事，例如加载网络上的库、在导入模块时对模块进行修改、自动安装缺失库、上传审计信息、延迟加载等等。

限于篇幅，我们不再详细展开了。最后小结一下：

- 可以用 try...except 方式，实现简单的三方库导入或者替换
- 已知全部缺失的依赖库时（如 requirements.txt），可以手动安装
- 利用 sys.meta_path，可以自动导入任意的缺失库

参考资料：

https://github.com/liuchang0812/slides/tree/master/pycon2015cn
http://blog.konghy.cn/2016/10/25/python-import-hook
https://docs.python.org/3/library/sys.html#sys.meta_path

推荐阅读 （点击标题可跳转阅读）

详解 Python 3.8 的海象运算符：大幅提高程序执行效率

Python 中 -m 的典型用法、原理解析与发展演变